Understanding Problem Difficulty in Reinforcement Learning: A Study of Goal-Oriented MDPs and Solution Methods

by

Christopher Beeler

A thesis submitted to the University of Ottawa in partial fulfillment of the requirements for the degree of

Doctorate in Philosophy Mathematics and Statistics[1]

Department of Mathematics and Statistics

Faculty of Science

University of Ottawa

Supervisors: Dr. Isaac Tamblyn
& Dr. Maia Fraser

May 2026

[1]The Ph.D. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

"Battle not with monsters,
lest ye become a monster,
and if you gaze into the abyss,
the abyss gazes also into you."

-Friedrich Nietzsche

Abstract

Due to the rise of optimal control problems in modern life and their increasing complexity, this thesis analyzes various properties of these problems and how they relate to the difficulty of the problem itself to help increase understanding of the space. Through the lenses of learning theory and topology, it discusses how concepts like sample complexity, topological complexity, and path homotopy contribute to characterizing problem difficulty. Through graphical and topologically equivalent representations, methods are presented for bounding sample complexity for various goal-oriented MDPs, with a strong focus on separated-path MDPs.

Using various model environments as specific examples of interest, three different methods of solving these types of problems are presented along with discussions on why the above properties affect those solutions. These environments serve as playgrounds for analyses based on partial observability, topologically complex navigation, and hierarchical frameworks.

Author's Declaration

I hereby declare that this thesis consists of original work of which I have authored. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research. I further authorize University of Ottawa to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research. I understand that my thesis will be made electronically available to the public.

Statement of Contributions

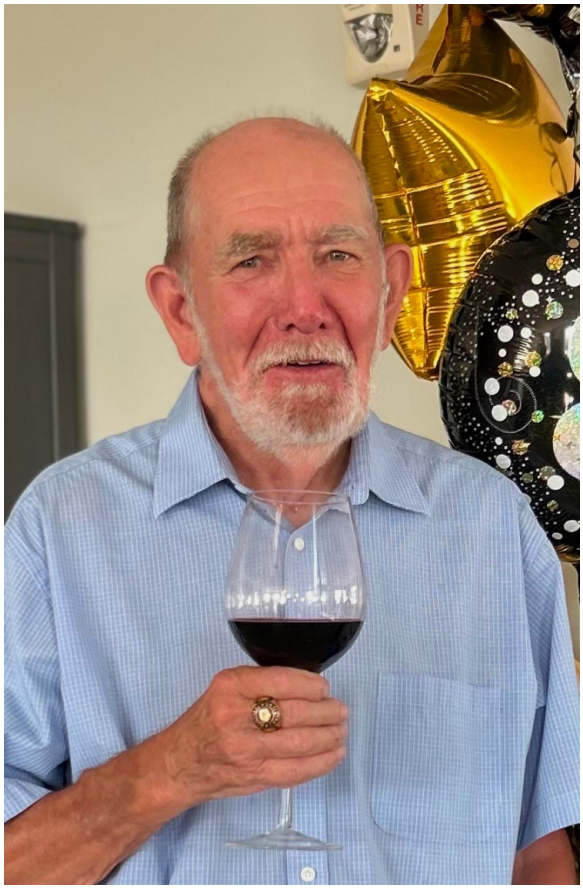
Some of the work in this thesis was completed as co-authored works, all of which are primary-authored by myself. These works primarily exist here in the form they were published in, with irrelevant components excluded and additional discussions added.

Chapter 4 consists of work from "Chris Beeler, Uladzimir Yahorau, Rory Coles, Kyle Mills, Stephen Whitelam, and Isaac Tamblyn. Optimizing thermodynamic trajectories using evolutionary and gradient-based reinforcement learning. *Physical Review E*, 104(6):064128, 2021" [1]. The evolutionary learning aspects were heavily discussed in "Chris Beeler. *Perpetually playing physics*. University of Ontario Institute of Technology (Canada), 2019" [2] and thus are only presented summarized here as needed with novel discussions added. The development of the environment was done by myself with assistance from U. Yahorau, R. Coles, and K. Mills while the writing was done by myself with assistance by K. Mills, S. Whitelam, and I. Tamblyn. The remainder of this work pertaining to the RL aspects and additional discussions were performed solely by myself.

Chapter 5 consists of the entirety of the work from "Chris Beeler, Xinkai Li, Colin Bellinger, Mark Crowley, Maia Fraser, and Isaac Tamblyn. Dynamic programming with incomplete information to overcome navigational uncertainty in POMDPs. Proceedings of the Canadian Conference on Artificial Intelligence, 2024" [3]. The environment was developed by myself with assistance from X. Li, and all other experiments

and writing for this work were performed by myself with guidance from C. Bellinger, M. Crowley, M. Fraser, and I. Tamblyn. The additional discussions in this chapter not presented in the published article were done by myself for the purpose of this thesis.

Chapter 6 consists of work from "Chris Beeler, Sriram Ganapathi Subramanian, Kyle Sprague, Mark Baula, Nouha Chatti, Amanuel Dawit, Xinkai Li, Nicholas Paquin, Mitchell Shahen, Zihan Yang, Xinkai Li, Mark Crowley, Isaac Tamblyn. ChemGymRL: A customizable interactive framework for reinforcement learning for digital chemistry. Digital Discovery, 3(4):742–758, 2024" [4]. I contributed to the conceptualization of this work, development of the ChemGymRL code used, source for background chemistry knowledge, development of project documentation, creation of figures, and writing of the published article. S. Subramanian contributed to the conceptualization of this work, development of the ChemGymRL code used, source for background RL knowledge, RL experimentation, and writing of the published article. K. Sprague contributed to the development of the ChemGymRL code used, development of project documentation, source of background RL knowledge, RL experimentation, creation of figures, and writing of the published article. N. Chatti contributed to the conceptualization of this work and source for background RL knowledge. M. Shanen, N. Paquin, M. Baula, Z. Yang, and X. Li contributed to the development of the ChemGymRL code used. A. Dawit contributed to the development of project documentation. C. Bellinger, M. Crowley, and I. Tamblyn served as the principle investigators for this work, contributing to all aspects listed above. The parts of this published article that are not particularly relevant to this work and/or were performed without the involvement of myself have been omitted from this thesis.



Dedication

In loving memory of my grandfather, Donald J. MacAulay, who taught me that you are never too old to keep learning. This thesis is dedicated to all the memories we had that will stay with me forever and all the examples he set that allowed me to reach this point.

He asked to keep his memory alive by continuing his legacy of making people feel good—through kind words and deeds, smiles, and, most especially, hugs.

Acknowledgements

The research in this thesis is the product of over half a decades worth of work, and therefore would not be possible without the many groups of people in my life, both old and new.

I would like to start by thanking my supervisors, Isaac Tamblyn and Maia Fraser, for providing the opportunity, resources, and guidance to conduct this research, even through the many twists and turns it took to get here.

To my fellow students at uOttawa who became friends along the way, thank you for the countless discussions, both academic and not, that gave me a reason to take a well needed break from work. Gavin and Christoff; our virtual coffee breaks were always great motivation for the rest of the day. Jane, Alice, Cameron, and César; Friday nights at NFG was one of my favourite ways to end the week when I was in Ottawa. I sometimes wonder if they miss our extremely loud debates about math.

To my former fellow students from UOIT, who unexpectedly have since become lifelong friends, thank you for continually listening to me vent about the many problems that arose since you moved on. Austin and Wesley; impromptu "office hours" helped me get through many late-night writing sessions. Liz, Dan, Martin, and Amber; our hangouts, both LOTR-based and otherwise, always helped raise my spirits from the exhaustion of all the long days.

To Rory and Josh, who have become two of my closest friends during this time,

thank you for the many enjoyable distractions that were necessary in keeping me optimistic in this long journey. Introducing me to the world of TTRPGs has opened Pandora's box, providing the motivation to finish this thesis simply so I can invest more time into the hobby I did not know I needed.

To Minka and Buddy, the best dogs anyone could have asked for, thank you for the much needed exercise and emotional support during the pandemic.

Last and most importantly, thank you to Renée for encouraging and supporting me all this time. You did not sign up for another 6 years of grad school but I appreciate your patience and sticking with me until the end.

Contents

List of Figures

List of Tables

Abbreviations

AI Artificial Intelligence.

DP Dynamic Programming.

GOMDP Goal-Oriented MDP.

HMM Hidden Markov Model.

MDP Markov Decision Process.

PAC Probably Approximately Correct.

PDF Probability Density Function.

PMF Probability Mass Function.

POMDP Partially Observed Markov Decision Process.

PPO Proximal Policy Optimization.

RL Reinforcement Learning.

SL Supervised Learning.

Chapter 1

Introduction

Optimal control problems, which seek to compute control policies that optimize a given objective function subject to system dynamics and constraints, have become extremely popular in recent years, and their solutions—especially through Reinforcement Learning (RL), an area of Artificial Intelligence (AI) often used for learning optimal control—are now widely applied. While early RL methods—particularly tabular and linear approaches—were relatively simple and interpretable due to their explicit state–action representations [6], the introduction of deep neural networks as function approximators has led to increasingly complex RL algorithms. These deep RL methods are often treated as black-box models, as the high-dimensional and non-linear representations learned by neural networks make it difficult to interpret their decision-making processes [7, 8]. Initially, video games and robotics were the most well-known applications of RL methods, however, this category has since expanded to include things with real-world tasks like self-driving cars, chat-bots, healthcare, and more. This makes the need for studying RL and its foundations all the more urgent, as the impact of these systems continues to grow. For instance, in self-driving vehicles navigating through complex urban environments, understanding why an RL

agent struggles with decision-making under certain conditions can lead to safer and more reliable algorithms.

With the rising impact and importance of the possible applications in this field, the need to understand the problems and solutions becomes crucial. Specifically, the research in this thesis investigates how certain structural properties of the Markov Decision Process (MDP) class of problems contribute to the learning challenges faced by RL algorithms and how mathematical simplifications can enhance interpretability. The outcomes of this research not only illuminate factors influencing problem complexity, but also propose methods that can be adapted to practical RL applications, potentially improving safety and decision-making reliability in areas such as autonomous navigation and healthcare.

The objective of this work is to analyze what properties of a problem contribute to differences in learning difficulty, and to investigate how a rigorous application of relatively simple mathematical models—compared to the highly parameterized function approximators commonly used in deep RL—can help make these effects more explicit. The results are often presented using navigation-type problems, as these are intuitive to describe and visualize while still exhibiting non-trivial solution complexity. Although these examples are domain-specific, many of the underlying observations extend to a broader class of sequential decision-making problems.

By identifying problem properties that influence learning complexity, this work aims to improve understanding of why certain RL problems are more challenging than others. Additionally, the use of more traditional mathematical formulations can support clearer analysis and facilitate reproducibility, complementing empirical approaches commonly used in modern RL research. This thesis presents these concepts in a way that they can be adapted and applied to various other problems in the field. The work here will primarily be focused on MDPs, which are a specific

type of optimal control problem defined in Chapter 2, and in some cases, an expansion is made to Partially Observed Markov Decision Processs (POMDPs), which is a generalization of MDPs where the state is not fully observable. The solutions to these problems are found analytically through an iterative optimization method called Dynamic Programming (DP) or approximated with AI through various forms of RL. The problems themselves are mostly analyzed through information availability, topology, and sample complexity. While the goal of this work is to increase the community understanding of the connection between optimal control problems and their AI solutions, it emphasizes mathematical structure and interpretability whereas scaling law research focuses on large-scale empirical regularities [9–12].

1.1 A Brief History of MDPs and RL

While the popularity of RL has skyrocketed in the last decade, the journey began over a century ago in the early 1900s with the formalization of the stochastic processes known as Markov chains [13]. A stochastic process is a collection of random variables indexed by time. In the case of a discrete-time process, this formalism models sequences of random events. A discrete Markov chain is a special case where the probability of the next event depends only on the current event (the Markov property), formally defined by

$$\Pr(X_{t+1}|X_0, X_1, \ldots, X_t) = \Pr(X_{t+1}|X_t),$$

i.e. the probability of an event at time $t+1$ depends only on the event at time t, rather than the full history.

MDPs are a standard framework for modelling sequential decision-making prob-

lems. An MDP consists of a set of states, a set of actions that act on the states (via state-to-state transition probabilities), and scalar rewards that quantify the benefit or detriment of each transition. The goal in an MDP is to maximize some form of cumulative reward. More precisely, one seeks a *policy*, i.e. a choice of action to take at each state, such that the associated cumulative reward will be maximized. While MDPs are closely related to Markov chains, the relationship is such that fixing a specific type of policy (a stationary Markovian policy) in an MDP yields a Markov chain with an associated reward structure. One approach to finding such an optimal policy is the iterative optimization method DP which is utilized to determine the value of a state. The Bellman equation [14]—introduced in the 1950s—is an equation concerning the value (or optimality) of a state, and solutions to the equation are found using DP through the value iteration algorithm [15]. The solutions from this process are then used to determine what the optimal decisions are through the policy iteration algorithm [16].

While DP finds analytical solutions using the defined properties of an MDP, it is a method that requires full knowledge of the transition probabilities and rewards. By contrast, RL finds approximate solutions through sampling experiences which is useful when the characteristic properties for DP are not known or available. In this same time period, the exploration versus exploitation trade-off was introduced by Robbins [17], which has become a fundamental concept in RL. Handling this balance of exploring the system to gain new knowledge and exploiting the knowledge already gained is its own field of research within RL [6, 18–21].

Meanwhile, neuroscience has progressed in understanding how neurons function in the brain [22] and how animals learn from rewards and punishments [23,24]. The ideas of learning through reward, combined with the value and policy iteration algorithms, led to some of the first reinforcement learning algorithms in the late 1980s, such as

TD-learning [25] and Q-learning [26, 27] which adapt/learn from experiences and use that to plan decisions based on the rewards and punishments acquired through those experiences. With the advancements of artificial neurons [28, 29], and the ability for them to learn through backpropagation [30], artificial neural networks made their way into RL in the 1990s to approximate value functions in complex problems [31]. It was not for another 20 years before RL was swept up in the big boom of deep learning when an RL agent was able to outperform humans in dynamic, complex video games by using deep neural networks to learn [7].

1.1.1 Specific Relevant Works

Building on the foundational work discussed above, more recent advancements have addressed specific challenges in learning efficiency and exploration strategies, as discussed below. Much of the early work on studying the difficulty of MDPs is related to sample complexity, describing the minimum number of experiences an agent needs to learn effectively. Kearns and Singh [32] constructed a polynomial lower bound on the time to achieve near-optimal RL performance based on the return convergence rates (horizons) and a learning algorithm that, while theoretically optimal, is realistically limited and impractical for modern problems. Brafman and Tennenholtz [33] improve on the previous algorithm, making it more general, but still not very practical. Kakade [34] further expands this by providing upper bounds for the algorithms mentioned before based on the size of the state-action spaces with Azar et. al. [35, 36] and Dann and Brunskill [37] tightening these bounds further. Jiang and Agarwal [38] have since shown that these sample complexity bounds dependence on horizons is not as significant as previously presented.

Outside of sample complexity, much of the remaining work in this area focuses on performance limits of RL agents (i.e. regret bounds). This is in some sense a

dual formulation that describes how well an agent can perform given the amount of experience they have. Similarly, for sample complexity; Jaksch, Ortner, and Auer [39] present an upper bound on regret based on the size of the state-action spaces, however they also include the *diameter* of these spaces as well, taking into account the connectivity of the problem. More recent work such as Osband et. al. [21] and Jin et. al. [40] have continued to present newer bounds, however they are algorithm specific.

The sample complexity and regret bounds mentioned above are directly relevant to the area of interest in this thesis, however they primarily focus on horizons and the size of the state-action spaces, which are only part of what defines these problems. As much of the RL community is focused on solving problems, it is less directly stated where the difficulties lay outside of the ones mentioned above. It becomes more clear when looking at what changes are made to existing algorithms in order to solve new problems.

One common challenge researchers have encountered is "how should one explore a problem when learning how to solve it?" Bellemare et. al. [41] introduced new way of measuring exploration in higher-dimensional spaces, helping estimate the density of states, which gives an idea of what states have been explored less than others. Osband et. al. [20] instead uses a group of models learning asynchronously, or ensemble, to measure the overall uncertainty as a way to determine which states need to be visited more. Fortunado et. al. [42] injects adaptable noise into the models themselves to increase stochasticity as a way of encouraging exploration to solve problems with scarce feedback.

Another large challenge in solving these types of problems is connecting the dynamics with the feedback received (reward signal). Schulman et. al. [43] designed an algorithm that was designed to tackle the challenge of learning from complex

system dynamics, handling the association of reward across non-linear connections. Pathak et. al. [44] used intrinsic rewards to encourage curiosity-driven exploration as a method of overcoming sparse/delayed reward signals where little to no feedback is provided while learning. To handle complex reward systems, van Seijen et. al. [45] introduced a hybrid reward architecture that breaks down the reward signals into various components, allowing one to learn each component individually.

Since AlphaZero [46] demonstrated that an agent could master complex games like chess and Go through self-play—learning entirely from experience rather than human provided examples—it is regarded as a fully autonomous learning model. This approach differs fundamentally from earlier systems such as DeepBlue and AlphaGo, which relied on human knowledge and examples from experts to guide learning. While this may suggest that effective learning should avoid human-imposed biases, previous work has shown that they can help progress the learning process in certain cases. For example, Ng, Harada, and Russel [47] showed that reward shaping, or modifying the feedback system to guide learning, can accelerate finding optimal solutions by offering more consistent feedback, creating an easier path to finding the optimal solution. Nachum et. al. [48] expanded on this by having learning the best way to shape the reward function a problem itself, making it more automated and less experience influenced. On a similar note, Sutton, Precup, and Singh [49] showed that reducing the problem into a simpler, less-detailed version can make learning the more detailed problem easier. Bengio et. al. [50] advanced this idea by showing that starting with solving an easier version of the problem and then making it more difficult can be a more effective learning approach. On the opposite end, Codevilla et. al. [51] made advancements in autonomous driving by using human examples to learn the relationship between sensory information and controls, creating a foundation to begin at for learning the overall problem.

While the examples listed above make solving their respective problems more efficient, the main issue with (most of) them is that the approaches introduced involve some level of human-biased decision. Tricks like those are often required because solving these problems analytically has become computationally intractable, meaning the analytical solutions are often discarded. However, the *method* used to find analytical solutions (DP) has remained the foundation for these approximation methods. Watkins and Dayan's [26] and Powell's [52] methods are both ways of finding approximations that mimic the behaviour of finding an analytic solution and are the basis of many of the modern algorithms used to solve the problems discussed in Chapter 2.

There has been plenty of work on *solving* MDPs and POMDPs, however the majority of this work uses purely black-box AI methods, with little to no incorporation of more traditional mathematics. Within this work, there have been very few studies actually *analyzing* the problems themselves and what makes one problem more complex to learn over others. Many scientists have an intuition as to what makes some problems more difficult, however this knowledge is not generally utilized in the optimal control field.

This lack of understanding mentioned above may not necessarily seem like an issue as progress is still being made in the field without it. In an age where AI solutions are becoming more and more commonplace in everyday lives though, it is important to understand why they behave the way they do and what their decision making limitations might be. With the area of self-driving cars, a mistake in decision making could lead to serious consequences such as car accidents that cause damages and/or injuries. In healthcare, a mistake in decision making could result in a misdiagnosis of a patient and in chat-bots, a mistake could lead to the spread of misinformation (and has [53]). On the other hand, designing systems so they will be easier for higher-level agents to learn to control can improve the stability of these larger AI systems.

1.2 Thesis Overview

This chapter has described the scope of this thesis and provided the motivations for this area of research, along with a high-level overview of the work done. The subsequent chapters go into more detail about the necessary background, methods used, results produced, and the possible directions these results could lead to.

Chapter 2 formally defines what MDPs are and the various types of them. The definitions of each type of MDP are expanded to include the more general case of POMDPs. The general methods of solving these problems, DP for analytic solutions and RL for learned approximations, are also introduced and discussed.

Chapter 3 then analyzes the properties of MDPs, with a primary focus on goal-oriented, or navigational-type problems, presented as a generalized Goal-Oriented MDP (GOMDP). The tools of this analysis feature concepts from learning theory (sample complexity) and topology (topological complexity, homotopies). With these new analysis tools and knowledge, Chapter 4 exploring the simple heat engine environments known as Carnot heat engines [1], explaining the reasoning for the results obtained.

This discussion is briefly continued in Chapter 5, with a shift to focusing on problems that deal with navigating from point A to B. Investigating the level of difficulty in these problems, this chapter also highlights how to leverage known information about the system to approximately solve such problems with DP. Chapter 6 expands some of this analysis to a chemical laboratory setting and presents some preliminary RL results.

Everything presented in Chapters 3-6 is brought together and wrapped up in Chapter 7, along with some of the potential directions this work could lead to but were out of the scope of this thesis. Note, some work was required as validation for

other results or methods presented in the other chapters but is not necessarily directly related to the overall theme. Chapter 8 contains that work, along with other related work that was done but not used elsewhere in this thesis.

Chapter 2

Markov Decision Processes

and How to Solve Them

MDPs are a set of optimal control problems that are often used as models for tasks focused on decision making. They consist of a set of states a system can exist in, actions that one can take to modify those states, some dynamics which govern the outcomes of how actions modify the states, and a feedback structure that specifies how beneficial each transition is. More importantly, the dynamics of these problems depend only on the current state of the system, rather than the history of events, a version of the Markov property. This simplicity makes MDPs an attractive framework to focus on in the optimal control field.

At a high level, sequential decision-making problems are commonly modelled either as MDPs, in which the system state is fully observable, or as POMDPs, in which the state is observed indirectly through a (potentially stochastic) observation process. While these are often treated as two separate modelling frameworks, POMDPs can be understood as a generalization of MDPs: an MDP is recovered as the special case in which the observation process is deterministic and reveals the state exactly, corre-

sponding to the identity mapping. Each of these frameworks contains finer subclasses of problems, which will be discussed throughout this chapter.

The game of *chess* is a classic example of a problem that can be modelled as an MDP. In a strict game theory setting, chess is a two-person combinatorial game where both players are assumed to operate under optimal play. However, from the perspective of a single player, it can be modelled as an MDP by treating the board and the opponent together as the environment. Since the opponent's strategy is usually unknown and they may not play optimally, their decisions act as probabilistic transitions from the player's perspective. Furthermore, the entire board and all pieces are on display at all times, meaning the player always observes the state of the game directly. While the action space $\mathcal{A}$ formally contains all possible legal moves in chess, in reality, the set of allowed actions is highly state-dependent (i.e., only a minuscule fraction of all possible moves are legal from any given state). In practice, RL implementations typically mask illegal actions so the agent can only select from the valid subset at any given time. On the other hand, the game of *poker* is a POMDP. While each player knows what cards they have in their hand, they do not know the order of the cards left in the deck, nor do they know what cards are even there. By *observing* the cards in play, which are the cards not in the deck, a player can determine probabilities of what the deck looks like, but they cannot gain more than this partial observation of the state.

When a MDP [54] is paired with a certain method of choosing actions, it results in a Markov chain (that is labelled by a set of rewards). Similarly, when a POMDP [55] is paired with a certain method of choosing actions, it results in a Hidden Markov Model (HMM) (also labelled by a set of rewards), where an HMM is simply a Markov chain whose state is only observable through a (possibly stochastic) mapping. Although the dynamics in these two systems are fundamentally the same, the information available

to the decision maker is what changes. This chapter will formally define MDPs and POMDPs (following Puterman [54] and Krishnamurthy [55]), introduce the different types of problems, and describe how they are generally solved (following Bellman [15] and Krishnamurthy [55] for DP and Sutton [6] and Krishnamurthy [55] for RL).

2.1 Markov Decision Processes

For some set $\mathcal{S}$ in which random variables S_i, $i \in \mathbb{N}$ take values ($\mathcal{S}$ may be finite or infinite in general; the finite case is assumed here for exposition), we say the discrete stochastic process S_0, S_1, S_2, ..., is a *Markov chain* if for all $t \in \{0, 1, 2, \ldots\}$, we have

$$p(S_{t+1} = s_{t+1} | S_t = s_t, S_{t-1} = s_{t-1}, \ldots, S_1 = s_1) = p(S_{t+1} = s_{t+1} | S_t = s_t), \tag{2.1}$$

for all $s_0, s_1, \ldots, s_t, s_{t+1} \in \mathcal{S}$. This condition is known as the *Markov property* [54]: the next step of the chain only depends on the previous step. The trajectory of how the chain arrived at the last step has no influence on what the next step is.

If the conditional probability $p(s_{t+1}|s_t)$ does not depend on t, then the Markov chain is time-invariant. That is, for all $t \in \{0, 1, 2, \ldots\}$ we have

$$p(S_{t+1} = j | S_t = i) = p(S_2 = j | S_1 = i), \tag{2.2}$$

for all $i, j \in \mathcal{S}$. Unless otherwise stated, all Markov chains considered here are assumed to be time-invariant (also known as stationary).

If some sequence $\{S_i\}$ is a Markov chain, S_t is called the state at time t. A time-invariant Markov chain is characterized by its initial state and a probability transition matrix $P = [P_{ij}]$ for $i, j \in \mathcal{S}$ where $P_{ij} = p(S_{t+1} = j | S_t = i)$. If $p(S_t)$ is the Probability Mass Function (PMF) of the random variable S_t at time t, represented

as a row vector over $\mathcal{S}$, then the PMF at time $t+1$ evaluated at state s_{t+1} is

$$p(S_{t+1} = s_{t+1}) = \sum_{s_t \in \mathcal{S}} p(S_t = s_t) P_{s_t s_{t+1}}. \tag{2.3}$$

Building upon the Markov chain, an MDP introduces actions and rewards. In general, an MDP is characterized by the quadruple $(\mathcal{S}, \mathcal{A}, P, r)$, defined by:

1. A state space, $\mathcal{S}$, where $s \in \mathcal{S}$ is a state. Both finite and continuous state spaces are discussed here.

2. An action space, $\mathcal{A}$, where $a \in \mathcal{A}$ is an action. Primarily finite action spaces are discussed here, although we briefly discuss the continuous action space case in Chapter 5.

3. State-to-state transitions specified by conditional distributions $P(s'|s, a)$, that define the probability of transitioning from some state $s \in \mathcal{S}$ to another state $s' \in \mathcal{S}$ for each action $a \in \mathcal{A}$.

4. A scalar reward, $r(s, a, s')$, assigned to each transition from state s to s' by action $a \in \mathcal{A}$ for all $s, s' \in \mathcal{S}$.

As mentioned before, pairing an MDP with a certain method of choosing actions, i.e. a Markovian (historyless) stationary policy, results in a Markov chain where each transition (s_t, a_t, s_{t+1}) is labelled by $r(s_t, a_t, s_{t+1})$, where s_t is the state at time t, a_t is the action chosen at time t, and s_{t+1} is the state at $t+1$ resulting from choosing the action a_t at s_t. $\mathcal{S}$, the state space, denotes the set of possible states for the Markov chain, and $\mathcal{A}$, the action space, denotes the set of possible actions. An *agent* (or controller) is the decision-making entity that interacts with the environment; it observes the state $s_t \in \mathcal{S}$ of the Markov chain at time t and takes an action $a_t \in \mathcal{A}$.

Each action a determines a transition matrix $P(a)$, which in turn determines the probability that the Markov chain moves to a particular state s_{t+1} at time $t+1$. The trajectory of states and actions an agent takes in an MDP up to time t is the information set, $\mathcal{I}_t = \{s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t\}$.

At each time t, the agent uses the information set, to choose an action $a_t = \pi_t(\mathcal{I}_t)$ using some policy π_t. A policy is a mapping from the information set to the action space, and an optimal policy is one that returns an optimal (maximal) cumulative reward when followed. The sequence of policies used from time 0 to $T-1$ is denoted by $\boldsymbol{\pi} = (\pi_0, \pi_1, \ldots, \pi_{T-1})$.

The type of policy described above is a general policy, where at each time t, the action a_t is chosen according to a probability distribution $\pi_t(\mathcal{I}_t)$. Given the Markovian nature of MDPs, there are also randomized Markovian policies, where at each time t, the action a_t is chosen according to a probability distribution $\pi_t(s_t)$, and deterministic Markovian policies, where at each time t, the action a_t is chosen deterministically according to $\pi_t(s_t)$. An important feature of the MDP problem is that it suffices to only consider deterministic Markovian policies to achieve the maximum [55]. The methods of searching over these policies for the optimal solution are outlined in Sections 2.2 and 2.3.

The goal of the agent in an MDP is to find, or best approximate, an optimal policy that maximizes cumulative reward (or equivalently minimizes cumulative cost) when the policy is followed. The *cumulative reward* (often used interchangeably with *total return*) is the sum of all scalar rewards received by the agent over the course of an episode or trajectory, which may be weighted by a discount factor in some cases. Because the environment transitions and rewards may be stochastic, the objective is formally defined as maximizing the *expected return*, which is the probabilistically weighted average of the cumulative rewards over all possible trajectories. The most

common types of these problems are outlined in the following sections, starting with the simplest.

2.1.1 Discrete State Finite Horizon MDP

As the name implies, for a discrete state MDP, the general state space $\mathcal{S}$ is a discrete (finite) space, $\mathcal{S} = \{1, 2, \ldots, S\}$. Let $t = 0, 1, \ldots, T$ be discrete time for some finite time horizon T. Building directly on the general MDP quadruple $(\mathcal{S}, \mathcal{A}, P, r)$, a discrete state finite-horizon MDP specifically requires:

1. A finite state space $\mathcal{S}$ and finite action space $\mathcal{A} = \{1, 2, \ldots, A\}$.

2. The state-to-state transitions are given by a set of $S \times S$ transition probability matrices, $P(a, t)$, with elements

$$P_{ss'}(a, t) = p(s_{t+1} = s' \mid s_t = s, a_t = a), \tag{2.4}$$

   for each action $a \in \mathcal{A}$ and time $t \in \{0, 1, \ldots, T-1\}$, where $s, s' \in \mathcal{S}$.

3. The scalar reward $r(s, a, s', t)$ depends on the current state s, action a, resulting state s', and time $t \in \{0, 1, \ldots, T-1\}$.

4. A terminal reward, $r_T(s)$, is specified for each state $s \in \mathcal{S}$ at the final time T.

Note that each row $P_s(a, t)$ of the transition probability matrix defines a PMF (represented as a row vector) over the resulting states $\mathcal{S}$ for a given state s, action a, and time t. A finite horizon MDP model is described by the quintuple

$$(\mathcal{S}, \mathcal{A}, P_{ss'}(a, t), r(s, a, s', t), r_T(s)), \tag{2.5}$$

where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, and $t \in \{0, 1, \ldots, T-1\}$. For MDPs, the agent observes the state s_t exactly. The scenario in which this is not true is discussed in Section 2.4. The next subsection introduces continuous state spaces.

2.1.2 Continuous State MDP

Consider the case where $\mathcal{S}$ is a measurable subset of $\mathbb{R}^n$. Previously, the state-to-state transition probability function P was a PMF and could be represented as a matrix for each action, where for some $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $t \in \{0, 1, \ldots, T\}$, $P_s(a, t)$ was a vector of probabilities over the discrete set $\mathcal{S}$. However, now P is a Probability Density Function (PDF), where $P_s(a, t)$ is a continuous distribution of probabilities over the continuous set $\mathcal{S}$.

With these changes, this MDP is still defined by the quintuple given above. In both of these cases, there is a finite number of steps (or horizon), where the chain ends after the agent acts T times. The next subsection introduces an infinite horizon.

2.1.3 Discounted Infinite Horizon MDP

Now consider the case where the horizon is infinite, i.e. $T = \infty$. The transition probabilities and rewards are assumed to not be explicit functions of time, and therefore there is no terminal reward. The discounted infinite horizon MDP model is instead defined as

$$(\mathcal{S}, \mathcal{A}, P_{ss'}(a), r(s, a, s'), \gamma), \tag{2.6}$$

for $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$, where $\gamma \in [0, 1)$ is a discount factor. The rewards acquired by the agent at time t are $\gamma^t r(s_t, a_t, s_{t+1})$, where the convention $0^0 = 1$ is adopted so that the immediate reward is always included. When $\gamma = 0$, this means that only immediate rewards are important in the problem and as $\gamma \to 1$, the more important

future rewards become.

Just as the definition of discrete state finite horizon MDPs was adapted for continuous state spaces, this definition of infinite horizon MDPs can also be adapted for continuous state spaces in the same way, giving us the four main types of MDPs.

With the problem space defined, we now need a method of solving them, that is, a way to find the optimal policies. As mentioned earlier, there are two main approaches for solving MDPs: DP and RL; introduced in the following two sections. DP is a method designed to find analytically optimal solutions by utilizing each component in the tuple that defines an MDP. However, in many realistic scenarios we do not have access to all of these fundamental components and therefore cannot necessarily find analytical solutions. RL on the other hand is a method designed to find approximately optimal solutions by means of sampling the MDPs transitions and rewards through exploration.

2.2 Dynamic Programming

DP is an optimization method that focuses on breaking down a problem into subtasks. The solutions are optimized on the subtasks, starting with the smallest problems that have fewer/no prerequisites. This reduces the search space required to solve the general tasks that depend on those smaller ones. For example, suppose that your overall task is to build a car. This can be broken down into smaller tasks, such as building individual parts. Once the individual parts are constructed, they can be brought together to form the car instead of trying to assemble the entire car from the beginning. Similarly, suppose that you are playing a game with a finite number of turns and want to determine the best move to make. To determine how good each possible move is on the first turn, you need to know how good each possible move

on the second turn is, and so on, until the final turn of the game. Thus, one can work backward through this path, starting at the optimal position to be in on the final turn, to determine the best move to make on the first turn to get there. The remainder of this approach then becomes: How do we find the optimal solution for each task? A couple of these search methods are outlined in this section.

2.2.1 Enumerative Solutions

The simplest form of search method are enumerative solutions, often referred to as solving with brute force. These are ones that list every possible set of rules, or policies, for an agent to follow, calculate the effect of each one for the desired purpose, and then select the one best suited for the problem. Obviously, this can become computationally intensive as the set of rules increases in size. In a simple game like *tic-tac-toe*, there are less than 1000 unique states the game can be in (when accounting for symmetry). With less than nine possible actions to choose from after the first turn, there are only several thousand sets of solutions to enumerate, which is not many by computational standards, making the enumerative approach feasible. This is not true for the game of *chess*, as indicated by Shannon [56], who showed that there are $\approx 10^{120}$ possible outcomes, significantly too many to solve with enumeration.

Assuming a constant initial state, in a purely deterministic scenario, a policy can be simply defined by a sequence of actions to take, significantly reducing the required ruleset because each action always leads to the exact same resulting state. In a stochastic case, however, the outcome of the first action conditions the choice for the second action; therefore, a ruleset must specify an action for every possible outcome. Suppose that we have a two-step problem with two actions, a_1, a_2, and two possible resulting states at step 1, denoted by (1) and (2). In the deterministic case, an action always leads to one specific state, so we only need to enumerate

the sequence of actions: a_1a_1, a_1a_2, a_2a_1, a_2a_2. However, in the stochastic case, the first action could lead to either state (1) or (2), so our policy must cover both possibilities. The rulesets must account for the action taken and the resulting state before the next action: $a_1(1)a_1$, $a_1(2)a_1$, $a_1(1)a_2$, $a_1(2)a_2$, $a_2(1)a_1$, $a_2(2)a_1$, $a_2(1)a_2$, $a_2(2)a_2$. This combinatorial explosion of rulesets makes enumeration impractical for stochastic problems.

2.2.2 Functional Equation Solutions

The *functional equation approach* is an alternative method to finding an optimal policy. The assumption in this method is that the systems involved in these problems are *Markovian*, i.e. the past history is not important when determining an action. Rather than enumerating every possible solution, we define a functional that describes the immediate reward one can get at the current state and what future returns one can get moving forward.

For example, consider the allocation of resources problem, where one needs to allocate resources into two categories at each step. Let $x_t \in [0, x_0]$ denote the amount of resources available at step t, where x_0 is the initial resource. Upon choosing an allocation $y_t \in [0, x_t]$, the resources available at the next step evolve according to $x_{t+1} = ay_t + b(x_t - y_t)$, where $a, b \in [0, 1)$ represent the consumption or decay rates. Let $r(x_t, y_t) = g(y_t) + h(x_t - y_t)$ be the immediate reward function, where $g, h : [0, x_0] \to \mathbb{R}$ are functions representing the profit of the allocations y_t and $x_t - y_t$ respectively. Assuming x represents the available resource at any generic step, the functional equation is

$$f(x) = \sup_{0 \le y \le x} [g(y) + h(x - y) + f(ay + b(x - y))]. \tag{2.7}$$

The action here is to choose a value y for the allocations y and $x-y$, so the goal is to choose a value y such that the total return is maximized, and hence the supremum over all possible y. $f(ay + b(x - y))$ is then how much return we can get from the leftover resources after allocating y and $x - y$.

Let us denote the state of the system at step t by s_t, an action by a_t, the set of actions by $\mathcal{A}$, and the immediate return function by $r(s_t, a_t, s_{t+1})$. The functional equation approach consists in solving for $f : \mathcal{S} \to \mathbb{R}$ the recursion defined as

$$f(s_t) = \sup_{a_t \in \mathcal{A}} \mathbb{E}\left[g(s_t, a_t, f(s_{t+1}))\right], \tag{2.8}$$

where $f : \mathcal{S} \to \mathbb{R}$ represents a measure of the objective starting from state s_t, and g is some function that evaluates the state, action, and the value of the resulting state s_{t+1}. An optimal policy is now defined as one that chooses the actions that attain the supremum.

In general, it is less the value of $f(s)$ that is of interest, but the argument a that satisfies the supremum (or maximum) within the functional as that tells us what *behaviour* is optimal. As we will see, this generic DP formulation directly sets up the standard models used in RL. The most well known example of this approach is Bellman equation, where $g(s_t, a_t, f(s_{t+1})) = r(s_t, a_t, s_{t+1}) + \gamma f(s_{t+1})$, as shown in the next section.

2.2.3 Bellman's Equation

Enumerative solutions are fairly impractical, therefore, they are rarely implemented. For MDPs, the functional approach is implemented more often instead as they have a natural functional for optimization. This application of DP is known as Bellman's equation and the method of using it to find an optimal policy is known as Bellman's

dynamic programming algorithm [14].

In terms of MDPs, for a state $s \in \mathcal{S}$ and set of actions $\mathcal{A}$, the *value* of s is given by the recursive Bellman equation

$$V_t(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}(a,t) \left[r(s,a,s',t) + V_{t+1}(s') \right], \tag{2.9}$$

for finite horizons and time t, where r is a reward function and $P_{ss'}(a,t)$ is the transition probability, and

$$V(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}(a) \left[r(s,a,s') + \gamma V(s') \right], \tag{2.10}$$

for infinite horizons, where r is a reward function and γ is a discount factor. The value of a state is equivalent to its optimality, which determines how advantageous that state is to be for acquiring future rewards. This tells us how much reward we can acquire from that state, how much expected reward we can acquire from the next state, etc.

Using the value functions above, the optimal policy π^* is defined as

$$\pi_t^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}(a,t) \left[r(s,a,s',t) + V_{t+1}(s') \right], \tag{2.11}$$

for finite horizons and

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}(a) \left[r(s,a,s') + \gamma V(s') \right], \tag{2.12}$$

for infinite horizons. The policy π^* obtained is optimal in the sense that an agent following it maximizes the expected cumulative reward. Furthermore, if some other policy π is optimal, it must also satisfy Equation 2.9 or 2.10. The value function can

be conditioned by some deterministic policy π as

$$V_\pi(s,t) = \sum_{s' \in \mathcal{S}} P_{ss'}(\pi_t(s),t)\left[r(s,\pi_t(s),s',t) + V_\pi(s',t+1)\right], \tag{2.13}$$

for finite horizons and

$$V_\pi(s) = \sum_{s' \in \mathcal{S}} P_{ss'}(\pi(s))\left[r(s,\pi(s),s') + \gamma V_\pi(s')\right], \tag{2.14}$$

for infinite horizons. If V_π satisfies Equation 2.9 or 2.10, then π is optimal.

Bellman's equation shows us what the value function and optimal policy should look like. The next subsection outlines how to compute the value function that satisfies the Bellman equation for the types of MDPs discussed in Section 2.1.

2.2.4 Solving MDPs with DP

For any MDP, the value function used to determine the optimal policy can be computed using Bellman's dynamic programming algorithm. This is done by defining an initial value function and recursively optimizing it.

Discrete State Finite Horizon MDP

The optimal policy $\boldsymbol{\pi}^*$ for the discrete state MDP problem with a finite horizon T is obtained by setting $V_T(s) = r_T(s)$ and recursively defining

$$\begin{aligned} V_t(s) &= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}(a,t)\left[r(s,a,s',t) + V_{t+1}(s')\right] \\ \pi_t^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P_{ss'}(a,t)\left[r(s,a,s',t) + V_{t+1}(s')\right], \end{aligned} \tag{2.15}$$

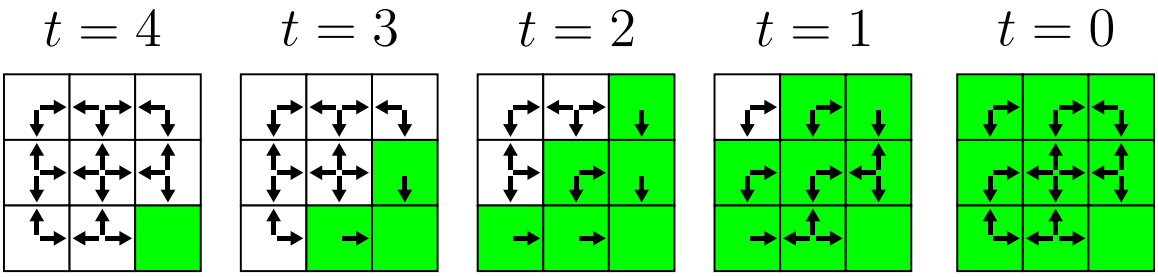


Figure 2.1: A simple visual example of Equation 2.15. This MDP has a horizon of 4 and consists of 9 states connected as a 3×3 grid with up to 4 deterministic actions available at each state (one for each neighbour in the grid layout) except the bottom right state which has no actions. $r_4(s) = 1$ for the bottom right state and 0 otherwise. $r(s, a, s', t) = 0$ for all states, actions, resulting states, and $t < 4$. The green squares represent where $V_t(s) = 1$. Note that for $t = 0$ and $t = 1$, some actions may appear sub-optimal, but because the agent has sufficient time remaining, taking a detour before moving to a state with maximal value at $t+1$ still results in optimal behaviour (i.e., a maximal expected return).

for $t = T-1, \ldots, 0$. $V_T(s)$, the value of the terminal state, is then the terminal reward acquired if s is the terminal state. $V_{T-1}(s)$ then is the reward that can be acquired at state s plus the *expected* terminal reward based on the probability distribution of the states where s can transition. The value function found in this algorithm is an analytic solution to Equation 2.9. A simple visual example of this algorithm is shown in Figure 2.1.

Continuous State MDP

Bellman's dynamic programming algorithm for discrete state MDPs is shown in Equation 2.15 can be used for continuous state MDPs by adapting the expectation from a sum over a PMF to an integration over a PDF. The optimal policy for the continuous state MDP problem with a finite horizon T is obtained by setting $V_T(s) = r_T(s)$ and

recursively defining

$$\begin{aligned} V_t(s) &= \max_{a \in \mathcal{A}} \int_{\mathcal{S}} P_{s'}(s,a,t) \left[r(s,a,s',t) + V_{t+1}(s') \right] ds', \\ \pi_t^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}} \int_{\mathcal{S}} P_{s'}(s,a,t) \left[r(s,a,s',t) + V_{t+1}(s') \right] ds', \end{aligned} \tag{2.16}$$

for $t = N-1, \ldots, 0$. Other than this small change, the principles of this algorithm are the same as before, and the value function found is also an analytic solution to Equation 2.9 for the continuous-state case.

Discounted Infinite Horizon MDP

For an infinite horizon discounted cost MDP with discount factor $\gamma \in [0, 1)$, we can define the optimal value function and policy as

$$\begin{aligned} V(s) &= \max_{a \in \mathcal{A}} \mathcal{Q}(s,a) \\ \pi^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}} \mathcal{Q}(s,a) \end{aligned} \tag{2.17}$$

where $\mathcal{Q}$ is the Q-function, defined as

$$\mathcal{Q}(s,a) = \sum_{s'} P_{ss'}(a) \left[r(s,a,s') + \gamma V(s') \right]. \tag{2.18}$$

However, unlike the previous two cases, we cannot simply initialize V_T and recursively work our way backward to V_0 as T is infinite in this case. Instead, an approximation must be used which is iteratively updated until it converges to the analytic solution.

The value iteration algorithm is used to compute the true value function V. Set-

ting $V_0(s) = 0$, then for $n = 1, 2, \ldots$, we have

$$\begin{aligned} V_n(i) &= \max_{a \in \mathcal{A}} \mathcal{Q}_n(s, a), \quad \pi_n^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \mathcal{Q}_t(s, a), \\ \mathcal{Q}_n(s, a) &= \sum_{s'} P_{ss'}(a) \left[r(s, a, s') + \gamma V_{n-1}(s') \right]. \end{aligned} \tag{2.19}$$

Unlike in the finite-horizon case, the optimized value function is only obtained after infinite optimization steps. However, we can determine an upper bound on the approximation error at finite iterations [55].

Consider the value iteration algorithm with discount factor γ and N iterations. Then for some $\epsilon > 0$,

$$\max_{s \in \mathcal{S}} |V_N(s) - V_{N-1}(s)| \leq \epsilon, \tag{2.20}$$

implies that

$$\max_{s \in \mathcal{S}} |V_N(s) - V(s)| \leq \frac{\epsilon \gamma}{1 - \gamma}. \tag{2.21}$$

Just as before with the definition, this solution of infinite horizon MDPs can also be adapted for continuous state spaces by replacing the sum over the state-to-state PMF with an integration over the state-to-state PDF in Equation 2.19. Similarly, the upper bound on approximation error can be adapted by replacing the maximum over the discrete state space with a supremum over the continuous state space in Equations 2.20 and 2.21.

As mentioned before, the solutions outlined in this section so far have made a crucial assumption: All the parameters that define an MDP are known. This, however, is not always the case, especially in many real-world problems. We don't always necessarily know the state-to-state transition probability or reward functions. In these cases, the DP approaches described here cannot be used. The next section will discuss methods for handling these scenarios.

2.3 Reinforcement Learning

DP assumes that the MDP model is completely known to the agent or the person performing the optimization, i.e. each parameter of the tuple is known. RL, however, focuses on learning-based algorithms for estimating the optimal policy when these parameters are not known. This section will focus on the two main types of RL—value-based (i.e. Q-learning) and policy gradient-based—and how they are applied to the types of MDPs discussed previously.

2.3.1 Q-Learning

While the value function $V(s)$ represents the value of being in a given state, the Q-function $\mathcal{Q}(s,a)$—defined similarly to but not necessarily exactly as the one in Equation 2.18—represents the value of taking a particular action in a given state. Q-learning is the process of learning an approximation $\hat{\mathcal{Q}}$ to the Q-function through experiences.

Discrete State Finite Horizon MDP

Let $(\mathcal{S}, \mathcal{A}, P_{ss'}(a,t), r(s,a,s',t), r_T(s))$ describe some discrete state MDP with states $s, s' \in \mathcal{S}$, action $a \in \mathcal{A}$, and time $t \in \{0, 1, \cdots, T\}$ with finite horizon T. Bellman's dynamic programming equation for a discrete state finite horizon MDP is

$$V_t(s) = \max_{a \in \mathcal{A}} \mathcal{Q}_t(s,a), \tag{2.22}$$

for $s \in \mathcal{S}$. For each state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$, the Q-function is

$$\mathcal{Q}_t(s,a) = \sum_{s' \in \mathcal{S}} P_{ss'}(a,t) \left[r(s,a,s',t) + V_{t+1}(s') \right]. \tag{2.23}$$

Let $\hat{\mathcal{Q}}_t^n$ be the nth iteration Q-function estimate at time t and

$$f(\hat{\mathcal{Q}}_t^n) = r(s_n, a_n, s_{n+1}, t) + \max_{a \in \mathcal{A}} \hat{\mathcal{Q}}_t(s_{n+1}, a) - \hat{\mathcal{Q}}_t^n(s_n, a_n), \tag{2.24}$$

be the Q-learning error function, which has an expected value of 0 for the true Q-function. The Q-function estimate at iteration n can then be updated by

$$\hat{\mathcal{Q}}_t^{n+1}(s_t, a_t) = \hat{\mathcal{Q}}_t^n(s_t, a_t) + \alpha f(\hat{\mathcal{Q}}_t^n), \tag{2.25}$$

where α is the learning rate, which determines how much one wants an experience to change their approximation. The learning rate can be fixed, or determined by some schedule or equation, such as in the Robbins-Monro algorithm, which tunes the learning rate at time t by

$$\alpha_n = \frac{\alpha}{\text{Visit}(s, a, t)}, \tag{2.26}$$

where $\alpha > 0$ is the initial learning rate and $\text{Visit}(s, a, t)$ is the number of times the state-action pair (s, a) has been visited at time t after n iterations.

Continuous State Finite Horizon MDP

Unlike DP, Q-learning does not extend well to the continuous state case. While it is not practical, or necessarily useful, the Q-function in the continuous case can be defined as

$$\mathcal{Q}_t(s, a) = \int_{\mathcal{S}} P_{s'}(s, a, t) \left[r(s, a, s', t) + V_{t+1}(s') \right] ds'. \tag{2.27}$$

The Q-learning error function would remain the same as Equation 2.25, meaning we would require infinite update iterations to improve $\hat{\mathcal{Q}}$ once for each state-action pair. Thus, Q-learning is not a viable approach for continuous state MDPs. There exist methods based on Q-learning that can be used for continuous state spaces, such as

Deep Q-Networks, which use a function that approximates the Q-function and is able to interpret the Q-values between various experienced state-action pairs. A few such approaches are discussed in Chapter 6, but are not entirely relevant to the majority of other chapters.

Discounted Infinite Horizon MDP

Although Q-learning does not adapt well for continuous state spaces, it does work well for infinite horizons. Let $(\mathcal{S}, \mathcal{A}, P_{s'}(s,a), r(s,a,s'), \gamma)$ describe some discrete state infinite-horizon MDP with states $s, s' \in \mathcal{S}$, action $a \in \mathcal{A}$, and discount factor $\gamma \in [0,1)$. Bellman's dynamic programming equation and Q-function for a discrete state infinite horizon MDP are given in Equation 2.19. Let $\hat{\mathcal{Q}}^n$ be the nth iteration Q-function estimate and

$$f(\hat{\mathcal{Q}}^n) = r(s_n, a_n, s_{n+1}) + \gamma \max_{a \in \mathcal{A}} \hat{\mathcal{Q}}(s_{n+1}, a) - \hat{\mathcal{Q}}^n(s_n, a_n), \tag{2.28}$$

be the Q-learning error function, which has an expected value of 0 for the true Q-function. The Q-function estimate at iteration n can then be updated using Equation 2.25.

Ultimately, this approach estimates the value function by *learning* the Q-function and then the policy is just determined based on the value function as in the DP approach. The next subsection discusses how to directly approximate the policy instead.

2.3.2 Policy Gradient Reinforcement Learning

The purpose of Q-learning is to estimate the value function and use that to determine a policy. Policy gradient algorithms instead focus on estimating the policy directly.

Let π_θ be a policy π parameterized by θ. The goal of policy gradient algorithms is to maximize the expected cumulative reward with respect to θ. In other words, the goal is to find θ such that Equation 2.13/2.14 satisfy Equation 2.9/2.10. In terms of the policy gradient approach, the parameters for the optimal policies are computed by

$$\begin{aligned} \theta^* &= \underset{\theta \in \Theta}{\operatorname{argmax}} V_{\pi_\theta}(s) \\ \Theta &= \{\pi_\theta(s,a) \geq 0, \sum_{a \in \mathcal{A}} \pi_\theta(s,a) = 1 \forall s \in \int\}. \end{aligned} \tag{2.29}$$

Computing V_{π_θ} this way requires knowing the underlying dynamics of the MDP, which we assume are not known when using RL. For this reason, policy gradient RL focuses on estimating the gradient to follow when optimizing θ.

Solving MDPs with Policy Gradient RL

Estimating the policy can be done by using the stochastic gradient algorithm

$$\theta_{n+1} = \theta_n + \alpha \hat{\nabla}_\theta R_n(\pi_{\theta_n}), \tag{2.30}$$

where α is some learning rate, $R_n(\pi_{\theta_n})$ is the cumulative reward observed by the agent when using policy π_{θ_n}, and $\hat{\nabla}_\theta R_n(\pi_{\theta_n})$ denotes the estimated gradient of the cumulative reward evaluated at π_{θ_n}.

Recall that for both discrete and continuous state finite horizon MDPs, the cumulative reward for the ith sample trajectory is given by

$$R_i(\theta_i) = \sum_{t=0}^{T-1} r(s_t^i, a_t^i, s_{t+1}^i, t) + r_T(s_T^i), \tag{2.31}$$

where s_t^i is the state experienced and a_t^i is the action chosen at time t in the ith sample trajectory using the policy parameterized by θ_i with $\pi_{\theta_i}(s,a,t) = p(a|s,t,\theta_i)$.

For infinite horizon MDPs the cumulative reward is instead

$$R_i(\theta_i) = \sum_{t=0}^{\infty} \gamma^t r(s_t^i, a_t^i, s_{t+1}^i), \tag{2.32}$$

where $\gamma \in [0, 1)$ is the discount factor.

The gradient of the cumulative reward can be estimated using the REINFORCE policy gradient estimator [57] as

$$\nabla_\theta R_n(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(s_t^i, a_t^i, t)) \right) \left(\sum_{t=0}^{T-1} r(s_t^i, a_t^i, s_{t+1}^i, t) + r_T(s_T^i) \right), \tag{2.33}$$

for the finite horizon case, and

$$\nabla_\theta R_n(\theta) \approx \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log(\pi_\theta(s_t^i, a_t^i, t)) \right) \left(\sum_{t=0}^{\infty} \gamma^t r(s_t^i, a_t^i, s_{t+1}^i, t) \right), \tag{2.34}$$

for the infinite horizon case. However, a truly infinite horizon is impractical for a learning-based algorithm, so an artificial finite horizon is generally used for computational purposes.

The solutions outlined in this section so far have been for the main types of *MDPs*. As mentioned above, MDPs are fully observable, which means that the state can be observed directly without obstruction. However, this is not always the case, especially in real-world problems. The next section introduces Partially Observable MDPs where the agent does not observe the state of the system, and how DP and RL solutions can be adapted for them.

2.4 POMDPs

Whereas an MDP is a controlled Markov chain, a POMDP is a controlled HMM, where an HMM consists of an Markov chain observed via some noisy (or limited) observation process. The HMM filter is a function that computes the posterior distribution, ρ_t, of the state, which is known as the belief state, discussed in more detail in Section 2.4.5. In a POMDP, the agent must use the belief state to choose an action, rather than the state itself, because it does not directly observe it.

2.4.1 Discrete State Finite Horizon POMDP

Let $t = 0, 1, \ldots, T$ be discrete time for some finite time horizon T as in Section 2.1.1. A discrete state finite-horizon POMDP model consists of:

1. A state space, $\mathcal{S} = \{1, 2, \ldots, S\}$, where $s_t \in \mathcal{S}$ is the state of the Markov chain at time $t = 0, 1, \ldots, T$.

2. An action space, $\mathcal{A} = \{1, 2, \ldots, A\}$, where $a_t \in \mathcal{A}$ is the action chosen at time $t = 0, 1, \ldots, T-1$.

3. An observation space, $\mathcal{O}$, (with finite or infinite size) where $o_t \in \mathcal{O}$ is the observation recorded at time $t = 0, 1, \ldots, T$.

4. An $S \times S$ transition probability matrix, $P(a, t)$, with elements

$$P_{ss'}(a, t) = p(s_{t+1} = s' | s_t = s, a_t = a), \tag{2.35}$$

for each action $a \in \mathcal{A}$ and time $t \in \{0, 1, \ldots, T-1\}$, where $s, s' \in \mathcal{S}$.

5. An observation PMF (or PDF for continuous observation spaces), $B(a,t)$ with elements

$$B_{so}(a,t) = p(o_{t+1} = o | s_{t+1} = s, a_t = a), \tag{2.36}$$

for each action $a \in \mathcal{A}$ and time $t \in \{0, 1, \ldots, T-1\}$, where $s \in \mathcal{S}$, $o \in \mathcal{O}$.

6. A scalar reward, $r_t(s,a,t)$, acquired by the agent for each state $s \in \mathcal{S}$, action $a \in \mathcal{A}$, and time $t \in \{0, 1, \ldots, T-1\}$.

7. A terminal reward, $r_T(s)$, for each state $s \in \mathcal{S}$.

A finite horizon POMDP model is then described by the septuple

$$(\mathcal{S}, \mathcal{A}, \mathcal{O}, P_{ss'}(a,t), B_{so}(a,t), r_t(s,a,t), r_T(s)), \tag{2.37}$$

where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, $o \in \mathcal{O}$, and $t \in \{0, 1, \ldots, T-1\}$. The belief state in this case is a PMF over $\mathcal{S}$.

2.4.2 Continuous State POMDP

As with Section 2.1.2, continuous state POMDPs are the case where $\mathcal{S}$ is a measurable subset of $\mathbb{R}^n$. The state-to-state transition probability function and belief state are PDFs over the continuous state space in this case instead of PMFs over a discrete state space. With these changes, the continuous state POMDP is still defined by the septuple given above.

2.4.3 Discounted Infinite Horizon POMDP

Once again, consider the case where the horizon is infinite, i.e. $T = \infty$. The discounted infinite horizon cost POMDP is instead described by the septuple

$$(\mathcal{S}, \mathcal{A}, \mathcal{O}, P_{ss'}(a), B_{so}(a), r(s, a, s'), \gamma), \tag{2.38}$$

for $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, and $o \in \mathcal{O}$ where $\gamma \in [0, 1)$ is a discount factor as defined previously.

Just as the definition of discrete state finite horizon POMDPs were adapted for continuous state spaces, this definition of infinite horizon POMDPs can also be adapted for continuous state spaces in the same way, giving us the same four main types of POMDPs as we had with MDPs. However, with POMDPs, there is a fifth case to consider where actions control the observation probabilities rather than the state-to-state transition probabilities. This additional type of POMDP is discussed next.

2.4.4 Controlled Sensing POMDP

Statistical signal processing focuses on extracting signals from noisy measurements. A controlled sensing problem is one where various types of sensors can be used to measure a process. Typically, each sensor has a use cost associated with it. Rather than choosing which actions to take, the problem is as follows: Which sensor should the agent choose at each time step?

Controlled sensing POMDPs are POMDPs that include some actions that can be used to reduce uncertainty. The goal is still to maximize cumulative reward, however in order to do so, agents *may* need to take advantage of the ability to reduce

uncertainty. Consider the scenario of an agent wandering through a maze. If the agent knows their exact position at all times, with the goal of getting to the exit, then this is an MDP. If the agent has to estimate their position based on some observations of its surroundings and movement, with the goal of getting to the exit, then this is a POMDP. Now imagine if in that last scenario the agent can pay a fee to look at a map that tells them which section of the maze they are in. This action does not necessarily change where the agent is in the maze, but it helps them to figure out where they are.

Let $\mathcal{A} = \{1, 2, \ldots, A\}$ be a set of sensors available to measure the state of a finite-state Markov chain. In general, for controlled sensing, actions can affect both state transition and observation probabilities, but we will only consider the case where actions do not affect the state-to-state transitions here. For example, using a specific radar to measure information related to an aircraft does not affect the dynamics of that aircraft.

For controlled sensing problems, it becomes easier to think of the problem in terms of cost, rather than reward, where cost is just negative reward, i.e. $c(\cdot) = -r(\cdot)$. The controlled sensing cost at time t is then expressed as

$$C(s_t, a_t, t) = c(s_t, a_t, t) + d(s_t, \rho_t, a_t, t) \tag{2.39}$$

where $c(s_t, a_t, t)$ is the instantaneous cost and $d(s_t, \rho_t, a_t, t)$ is the performance loss of using sensor a_t at state s_t. The instantaneous cost generally represents a monetary cost of using the sensor, either in the direct form of payment or in the indirect form of the wear and on the instrument. The sensor performance loss instead models the error when the sensor is used. Typically, an accurate sensor would have a high instantaneous cost but small performance loss.

With the new problem space defined, we now need a method to solve them, that is, a way to find the optimal policies. The two main approaches introduced in Sections 2.2 and 2.3 for solving MDPs are adapted for POMDPs below.

2.4.5 Solving POMDPs

For an MDP, the optimal policy was Markovian and the optimal action chosen at time t was $a_t = \pi_t^*(s_t)$. However, for a POMDP, the optimal action is $a_t = \pi_t^*(\mathcal{I}_t)$ since s_t is not known. Therefore, at each time t, the agent must use all available information, denoted by $\mathcal{I}_t = \{\rho_0, a_0, o_1, \ldots, a_{t-1}, o_t\}$, to choose an action $a_t = \pi_t(\mathcal{I}_t)$ following the policy π_t. The sequence of policies used from time 0 to time $T-1$ is denoted by $\boldsymbol{\pi} = (\pi_0, \pi_1, \ldots, \pi_{T-1})$. However, as $\mathcal{I}_t$ increases in dimension with respect to t, a sufficient statistic that does not grow with t would be useful to obtain, such as a probabilistic estimate over the state space.

Belief State Formulation

The posterior distribution of a Markov chain given the information set, $\mathcal{I}_t$, is defined as

$$\rho_t(s) = p(s_k = s|\mathcal{I}_t), \tag{2.40}$$

for $s \in \mathcal{S}$. For discrete state POMDPs, with either finite or infinite horizons, the S-dimensional probability vector $\rho_t = [\rho_t(1), \rho_t(2), \ldots, \rho_t(S)]^{\mathrm{T}}$ is known as the *belief state* (or information state). The belief state at time t is calculated as $\rho_t =$

$\tau(\rho_{t-1}, o_t, a_{t-1})$, where

$$\begin{aligned}\tau(\rho, o, a) &= \frac{B_o(a,t)P^{\mathrm{T}}(a,t)\rho}{\sigma(\rho, o, a)} \\ \sigma(\rho, o, a) &= \mathbf{1}_S^{\mathrm{T}} B_o(a,t)P^{\mathrm{T}}(a,t)\rho \\ B_o(a,t) &= \mathrm{diag}(B_{1o}(a,t), B_{2o}(a,t), \ldots, B_{So}(a,t)),\end{aligned} \tag{2.41}$$

with states $1, 2, \ldots, S$ denoting the enumerated states in $\mathcal{S}$. The $S-1$ dimensional unit simplex

$$\Delta(S) := \{\rho \in \mathbb{R}^S | \mathbf{1}^{\mathrm{T}}\rho = 1, 0 \leq \rho(s) \leq 1, s \in \mathcal{S}\}, \tag{2.42}$$

is the belief space. For example, $\Delta(2)$ is the one-dimensional simplex, equivalent to a line, $\Delta(3)$ is the two-dimensional simplex, equivalent to an equilateral triangle, and $\Delta(4)$ is the three-dimensional simplex, equivalent to a tetrahedron.

For continuous state POMDPs, with either finite and infinite horizons, the belief state at time t is instead a PDF over $\mathcal{S}$ and is calculated as $\rho_t = \tau(\rho_{t-1}, o_t, a_{t-1})$, where

$$\begin{aligned}\tau(\rho, o, a) &= \frac{B_o(a,t)\int_{\mathcal{S}} P_s(a,t)\rho(s)ds}{\sigma(\rho, o, a)} \\ B_o(a) &= p(o_{t+1} = o | s_{t+1} = s, a_t = a) \\ \sigma(\rho, o, a) &= \int_{\mathcal{S}} B_o(a)ds \int_{\mathcal{S}} P_s(a)\rho(s)ds,\end{aligned} \tag{2.43}$$

where ρ_t, $B_o(a_t)$, and $P_{s_t}(a_t)$ are all PDFs over $\mathcal{S}$ for s_{t+1}. In this case, the set of distributions

$$\Delta(S) := \left\{\rho : \mathbb{R}^S \to \mathbb{R}_{\geq 0} \middle| \int_{\mathbb{R}^n} \rho(s)ds = 1\right\}, \tag{2.44}$$

is the belief space.

With a method of computing the belief state defined, we can now adapt the DP solutions discussed above for POMDPs.

Solving POMDPs with DP

For a finite horizon POMDP, the value function

$$V_{\boldsymbol{\pi}}(\rho_0) = E_{\boldsymbol{\pi}}\left[\sum_{t=0}^{T-1} r_t(s_t, \pi_t(\mathcal{I}_t), t) + r_T(s_T)\middle|\rho_0\right], \tag{2.45}$$

is the expected cumulative reward acquired by using the policy $\boldsymbol{\pi}$ up to time T, given the initial distribution ρ_0 of the Markov chain, where the expectation is taken with respect to the probability distribution induced by $\mathcal{I}_T$. The goal of the agent is to determine the optimal policy defined as

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi}}{\operatorname{argmax}}\, V_{\boldsymbol{\pi}}(\rho_0). \tag{2.46}$$

Since policies in a POMDP operate on the belief state in this formulation, the problem becomes more similar to an MDP; therefore, the dynamic programming equation for the optimal policy for them can be used. The optimal policy $\boldsymbol{\pi}^*$ for a finite horizon POMDP can be obtained by setting $V_T(\rho) = r_T^{\mathrm{T}}\rho$ and recursively defining

$$\begin{aligned} V_t(\rho) &= \max_{a\in\mathcal{A}}\left[r(a,t)\rho + \sum_{o\in\mathcal{O}} V_{t+1}(\tau(\rho,o,a))\sigma(\rho,o,a)\right] \\ \pi_t^* &= \underset{a\in\mathcal{A}}{\operatorname{argmax}}\left[r(a,t)\rho + \sum_{o\in\mathcal{O}} V_{t+1}(\tau(\rho,o,a))\sigma(\rho,o,a)\right], \end{aligned} \tag{2.47}$$

for $t = T-1, \ldots, 0$, where

$$\begin{aligned} r(a,t) &= [r(1,a,t), r(2,a,t), \ldots, r(S,a,t)] \\ r_T &= [r_T(1), r_T(2), \ldots, r_T(S)], \end{aligned} \tag{2.48}$$

for the enumerated states $1, 2, \ldots, S$ in $\mathcal{S}$ with $S = |\mathcal{S}|$.

As with continuous state MDPs, the Bellman's equation value function, $V(\rho)$ is computed by setting $V_0(\rho)$, then for $n = 1, 2, \ldots$, we have

$$\begin{aligned} V_n(\rho) &= \max_a Q_n(\rho, a), \quad \boldsymbol{\pi}^* = \operatorname*{argmax}_a Q_n(\rho, a), \\ Q_n(\rho, a) &= \int_{\mathcal{S}} \int_{\mathcal{S}} P_{s'}(s, a_t, t) r(s, a_t, s', t) \rho_t(s) ds' ds + \int_{\mathcal{O}} V_{n-1}(\tau(\rho, o, a)) \sigma(\rho, o, a) do. \end{aligned} \tag{2.49}$$

For a stationary policy $\pi : \Delta(S) \to \mathcal{A}$ and initial belief $\rho_0 \in \Delta(S)$, the discounted infinite horizon expected reward is

$$V_\pi(\rho_0) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(\rho_t), s_{t+1}) \right]. \tag{2.50}$$

As before, the optimal expected cumulative reward is achieved by a stationary deterministic Markovian policy π^*. The value function $V(\rho)$ and optimal policy $\pi^*(\rho)$ satisfy Bellman's equation and are given by

$$\begin{aligned} V(\pi) &= \min_{a \in \mathcal{A}} \mathcal{Q}(\rho, a), \quad \pi^*(\rho) = \operatorname*{argmin}_{a \in \mathcal{A}} \mathcal{Q}(\rho, a), \\ Q(\rho, a) &= r(a)^{\mathrm{T}} \rho + \gamma \sum_{o \in \mathcal{O}} V(\tau(\rho, o, a)) \sigma(\rho, o, a), \end{aligned} \tag{2.51}$$

where $\tau(\rho, o, a)$ and $\sigma(\rho, o, a)$ are the HMM filter and normalization factor defined previously.

As with MDPs, the value iteration algorithm is used to calculate the value function of the Bellman equation, $V(\rho)$. Setting $V_0(\rho) = 0$, then for $n = 1, 2, \ldots$, we have

$$\begin{aligned} V_n(\rho) &= \min_{a \in \mathcal{A}} \mathcal{Q}_n(\rho, a), \quad \pi_n^*(\rho) = \operatorname*{argmin}_{a \in \mathcal{A}} \mathcal{Q}_n(\rho), \\ \mathcal{Q}_n &= r(a)^{\mathrm{T}} \rho + \gamma \sum_{o \in \mathcal{O}} V_{n-1}(\tau(\rho, o, a)) \sigma(\rho, o, a). \end{aligned} \tag{2.52}$$

From Theorem 7.6.3 in [55], for $\epsilon > 0$ and discount factor $\gamma \in [0, 1)$, if

$$\sup_{\rho \in \Delta(S)} |V_n(\rho) - V_{n-1}(\rho)| \leq \epsilon, \tag{2.53}$$

holds, then

$$\sup_{\rho \in \Delta(S)} |V_n(\rho) - V(\rho)| \leq \frac{\epsilon\gamma}{1-\gamma}. \tag{2.54}$$

This gives an upper bound on the approximation error when the value function is practically computed with finite iterations.

In terms of the belief state for controlled sensing POMDPs, the sensing costs can be expressed as

$$\begin{aligned} C(\rho_t, a_t) &= \sum_{s \in \mathcal{S}} (c(s, a_t, t) + d(s, \rho_t, a_t))\rho_t(s) \\ C_T(\rho_t) &= \sum_{s \in \mathcal{S}} (c_T(s, a_t, t) + d_T(s, \rho_t))\rho_t(s) \end{aligned} \tag{2.55}$$

The finite horizon value function becomes

$$V_{\boldsymbol{\pi}} = E\left[\sum_{t=0}^{T-1} C(\rho_t, a_t, t) + C_T(\rho_T) \middle| \rho_0\right], \tag{2.56}$$

where the optimal policy can be found using Bellman's dynamic programming algorithm as before.

Solving POMDPs with RL

Q-learning *can* be used for POMDPs, but it is a suboptimal approach. Given the belief state ρ, the POMDP policy is computed as

$$\pi(\rho) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s \in \mathcal{S}} \rho(s)\mathcal{Q}(s, a). \tag{2.57}$$

This method assumes that any uncertainty in the state disappears after each action, which would require complete knowledge of the POMDP model, however, the assumption here is that the POMDP model is not necessarily specified. Policy gradient approaches are used more commonly instead.

Now consider the POMDP described by $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P_{ss'}(a), B_{so}(a), r(s, a, s'), \rho)$. As $P_{ss'}(a)$ and $B_{so}(a)$ are not known to the agent, the belief state ρ_t cannot be computed at each time t as before. For policy gradient approximations in POMDPs, a reduced information set, $\mathcal{I}_t = \{a_0, o_1, \ldots, a_{t-1}, o_t\}$, must be used in Equation 2.29 instead. The main issue that using the belief state over the information set solved before was the ever increasing size of the information set. Therefore, a sliding window approximation of the information can be used as a replacement, defined as

$$\mathcal{I}_t^k = \{a_{t-k}, o_{t-k+1}, \ldots, a_{t-1}, o_t\}, \tag{2.58}$$

where $0 \leq k \leq t$ is an integer.

With POMDPs defined and methods of solving them introduced, it is important to see how these types of problems can be used for real-world problems. Section 2.4.6 discusses a real-world application of POMDPs and using DP to solve them in the field of airborne collision avoidance.

2.4.6 Next-Generation Airborne Collision Avoidance System

As an example of the real-world applicability of this topic, this section will use next-generation airborne collision avoidance system [5] as an example. The goal of that paper was to upgrade the Traffic Alert and Collision Avoidance System (TCAS) to adapt to the upcoming major changes to the airspace. The difficulties in devising such a system include

- Accuracy and precision of physical sensors.
- Variation in pilot behaviour and aircraft dynamics.
- Balance of competing objectives (safety and operational).

The Airborne Collision Avoidance System (ACAS) X replaces this beacon-based surveillance system with a GPS-based one.

TCAS operates with four main components: airborne surveillance, safety logic, vertical alerts, and a pilot interface. When TCAS observes a potential threat, the pilot is given an audible warning. In the event that an avoidance manoeuvre is required, the system then issues a resolution advisory for the pilot to either ascend or descend, along with a specific or range of speeds to do so at. The system will also adjust the advisory as the situation progresses. The resolution advisories are strictly for vertical manoeuvres and not horizontal. Once a threat has been avoided, the system will declare it.

Using linear extrapolation, TCAS estimates the minimum distance that will occur between the two aircrafts and the time until that distance is achieved. If both are small, then the system will alert the pilot, choose the vertical direction that allows the most separation, and begin modelling trajectories using a set of various change in altitude speeds; assuming a 5 second pilot response delay and an acceleration rate of $0.25g$.

Instead of using an ad hoc rule-based system, ACAS X uses an optimized numeric lookup table. The logic numerical look-up table takes a probabilistic dynamic model and a multi-objective utility model as input, and is optimized using dynamic programming. The probabilistic dynamic model is a statistical representation of where the aircraft will be in the future, and the multi-objective utility model represents the safety and operational objectives of the system. The system receives sensor measure-

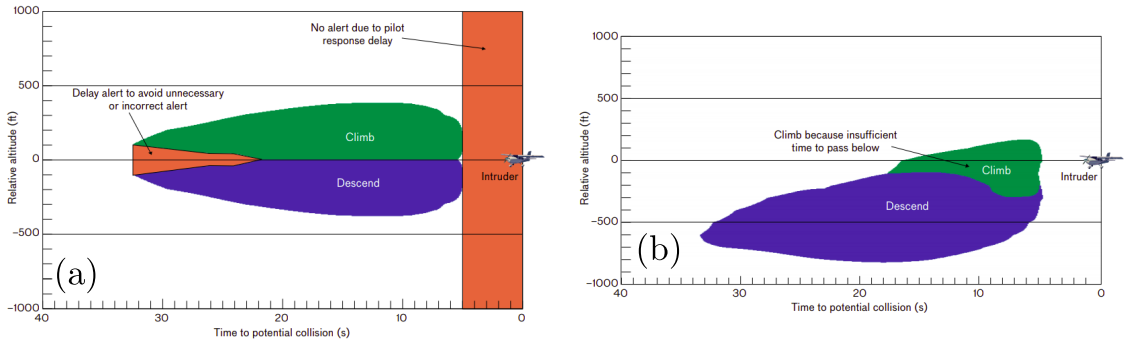


Figure 2.2: The optimal resolution advisory as a function of time and relative altitude where (a) the ACAS X aircraft and intruder are both level and (b) the ACAS X aircraft is climbing at a rate of 1500 ft/min and the intruding aircraft is level. Figure from Kochenderfer et. al. [5].

ments every second, which, along with the probabilistic dynamic and probabilistic sensor models, allows it to construct a distribution of the aircraft state. This state corresponds to an entry in the look-up table, which determines whether an advisory is necessary and, if so, the necessary rate to change the aircraft altitude.

This collision avoidance problem is formulated as a POMDP. The state is the status of the potential threat (such as relative position and velocity). The observations are the distributions of these potential threats, estimated using the probabilistic dynamic and probabilistic sensor models. The actions are the possible resolution advisories available. There are large costs for near (or actual) midair collisions and small costs for issuing resolution advisories with additional costs for issuing reversing or magnifying the previous advisory. The goal of this POMDP is to minimize these costs. An example of the resolution advisories is shown in Figure 2.2 (figure taken from Kochenderfer et. al. [5]).

To construct a look-up table for ACAS X to use, the state space is discretized. In a specific state, probabilistic models are used to construct a weighted average cost of the outcome for taking each action. Starting with the costs of being in the states at the time of expected collision, the expected costs achieved can be computed for the

previous time step, repeating until reaching the initial time.

Compared to TCAS, ACAS X reduces collision risk by 47%, overall alert rate by 40%, and improves safety by 54%. The density of advisories given by ACAS X is much lower than that of TCAS in their example of a New York City municipality. In the same example scenario, not only does TCAS give more advisories than ACAS X, but it also gives sequentially contradictory advisories. Although both systems safely avoided a near-middle-air collision, ACAS X clearly did so more efficiently.

This example outlines how a real-world problem can be constructed as a POMDP along with how important features such as safety measures and certainty can be incorporated into the cost (or reward) function.

This chapter has laid the groundwork for POMDPs, how to solve them, and why they are important. A key component of these is the connectivity of the state space. Therefore, it is important to have tools that allow us to analyze these components. The next chapter introduces generalized GOMDPs and utilizes sample and topological complexity to analyze the difficulty of these decision problems, establishing a framework to explain why some optimal control problems are more difficult to learn than others.

Chapter 3

A Measure of Navigational Decision Problem Complexity

The focus of this chapter is to gain an understanding of what properties of a problem in RL relate to its complexity, where the complexity of a problem is a measure of how difficult it is to solve. More specifically, the type of problems this chapter will focus on are generalized GOMDPs, defined in Section 3.2.

The task in an MDP is to find the policy that maximizes expected return under the given transition dynamics and reward function. When these are fully known, DP methods can be used. When they are partially or entirely unknown—as in many real-world scenarios—RL methods approximate the optimal policy through experience and interaction with the environment.

The concept of sample complexity, a cornerstone of learning theory in Supervised Learning (SL), provides a robust framework for quantifying the difficulty of RL tasks. In SL, sample complexity is defined as the minimum number of training samples sufficient to guarantee, with high probability, that a model achieves a target level of accuracy. Unlike computational complexity, which measures the algorithmic time or

space resources needed for computation, sample complexity measures the number of data points—expressed as sample size—required to extract sufficient information from a dataset to construct a generalizable model. By analogy, the sample complexity of an RL task can be defined as the minimum number of experiences (state-action-reward transitions) required to estimate, with high probability, an approximately optimal policy for the underlying MDP. This chapter utilizes both sample complexity and topological complexity to characterize how specific environmental properties of an MDP influence the efficiency with which an RL agent converges on an optimal policy.

3.1 Probably Approximately Correct Learning

The Probably Approximately Correct (PAC)-learning framework formalizes learnability by relating accuracy, confidence, and sample size [58]. In its classical formulation, PAC-learning is concerned with binary classification, where the label set is $\mathcal{Y} = \{0, 1\}$ and the task is evaluated using the 0-1 loss function (where $L(y, y') = 0$ if $y = y'$ and 1 otherwise). For a set of examples $\mathcal{X}$, a concept is a mapping $c : \mathcal{X} \to \mathcal{Y}$, and $\mathcal{C}$ is a chosen set of concepts. The learner considers a fixed hypothesis set $\mathcal{H}$ from which to choose a hypothesis $h : \mathcal{X} \to \mathcal{Y}$.

Given a concept $c \in \mathcal{C}$ and a sample $S = (x_1, x_2, \ldots, x_n)$ drawn independent and identically distributed from $\mathcal{X}$ according to some unknown distribution $\mathcal{D}$, along with their true labels $(c(x_1), \ldots, c(x_n))$, the goal is to select a hypothesis $h \in \mathcal{H}$ that minimizes the generalization error

$$R(h) = \mathop{\mathbb{P}}_{x \sim \mathcal{D}} \left[h(x) \neq c(x) \right]. \tag{3.1}$$

A learning algorithm $A : (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{H}$ is any rule that maps a set of training

examples to a hypothesis. Note that a learning rule is not restricted to empirical risk minimization; it can be any mapping, though empirical risk minimization (which simply minimizes the error on the training sample S) is the most common approach.

In the classical realizable case (where the true concept c belongs to $\mathcal{H}$), a consistent learner can achieve zero error. In the agnostic case (where c may not be in $\mathcal{H}$), the goal is to find a hypothesis whose error is close to $\inf_{h\in\mathcal{H}} R(h)$.

A concept class $\mathcal{C}$ is said to be PAC-learnable if there exists a learning algorithm A and a polynomial function $poly(\cdot,\cdot)$ such that for any $0 < \epsilon, \delta < 1$, and for any distribution $\mathcal{D}$, the algorithm requires a sample size $n \geq poly(1/\epsilon, 1/\delta)$ to ensure:

$$\mathop{\mathbb{P}}_{S\sim\mathcal{D}^n}\left[R(h) - \inf_{h'\in\mathcal{H}} R(h') \leq \epsilon\right] \geq 1-\delta. \tag{3.2}$$

In modern applications, such as RL, we often move beyond the classical binary definition to a more generalized framework. In this general setting, $\mathcal{Y}$ can be arbitrary, and the generalization error is defined with respect to a general loss function L:

$$R(h) = \mathop{\mathbb{E}}_{x\sim\mathcal{D}}\left[L(h(x), c(x))\right]. \tag{3.3}$$

As $\mathcal{D}$ and c are generally inaccessible, the learner instead estimates this using the empirical error

$$\hat{R}_S(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), c(x_i)). \tag{3.4}$$

In this broader context, let T be a sample complexity function mapping a distribution $\mathcal{D}$, an accuracy parameter $\epsilon \in (0,1)$, and a confidence parameter $\delta \in (0,1)$ to a required sample size threshold in $\mathbb{N}$. We define an (ϵ, δ, T)-correct learning algorithm (aligning with the notation used later by Azar et al. [35]) as an algorithm A that, given a sample size $n \geq T(\mathcal{D}, \epsilon, \delta)$, selects a hypothesis with an excess risk of at most ϵ with

probability at least $1-\delta$. If the sample complexity bound $T(\mathcal{D}, \epsilon, \delta)$ can be uniformly bounded by a function $T(\epsilon, \delta)$ that is independent of the underlying distribution $\mathcal{D}$, then $\mathcal{C}$ is learnable by $\mathcal{H}$. If T is only bounded for a specific distribution $\mathcal{D}$, it is learnable *with respect to* $\mathcal{D}$.

3.2 Goal-Oriented MDP

Recall from Section 2.1 that the core of an MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r)$, which additional elements such as terminal rewards, horizon, or discount factors depending on the type of MDP. GOMDPs [59] are a class of MDPs that—in addition to the usual MDP tuple—are described by the condition that the cost function is non-negative (i.e. the reward function is non-positive) and there are a subset of absorbing states (i.e. no outgoing transitions) called goal-states, where the cost of being in a goal-state is zero [60]. As all actions keep the agent in a goal-state once reached and there is no cost to stay in a goal-state, these goal-states are effectively equivalent to terminal states. This is a more generalized version of MDPs that use the action-penalty representation [61] which requires the reward of any action at each state to be -1, rather than non-positive. In contrast, there are MDPs that use the goal-reward representation [61] which requires the reward for entering a goal-state to be +1 and otherwise zero. As a common generalization of GOMDPs and the goal-reward representation, we define Generalized GOMDPs as a class of MDPs that—in addition to the usual MDP tuple—satisfies the following conditions:

1. a non-empty set of goal-states $\mathcal{G} \subset \mathcal{S}$;

2. an infinite horizon;

3. $r(s, a, s') \leq 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $s' \in \mathcal{S} \setminus \mathcal{G}$;

4. $r(s, a, g) \geq 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, and $g \in \mathcal{G}$.

Furthermore, we define a generalized GOMDP to be continuous if it also satisfies the following additional conditions:

5. $\mathcal{S}$ and $\mathcal{A}$ are compact subsets of $\mathbb{R}^n$;

6. $r(s, a, s')$ is a continuous function from $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \setminus \mathcal{G}$ to $\mathbb{R}$.

While these MDPs can have either deterministic or stochastic actions, this chapter will mainly focus on the deterministic case.

This representation allows for a more realistic real-world reward structure. Neither the reward structure of GOMDPs nor the goal-reward representation differentiate between goal-states, meaning the agent only needs to determine which goal-state costs less to reach. By allowing for variation in the reward assigned to entering a goal-state, generalized GOMDPs have the added difficulty of taking into account both the varying costs to reach different goal-states, the varying rewards for entering them, and balancing these between multiple goal-states.

3.2.1 Measuring Goal-Directed Success

Consider some state s and goal state g in a GOMDP. When evaluating the performance of policies on s relative to the optimal behaviour, they can be separated into one of two groups: those that do not generate trajectories from s to g and those that do. For those in the former "unsuccessful" group, it is a matter of measuring how far their trajectories are from reaching g. For those in the latter "successful" group however, it is a matter of measuring how far their trajectories are from the optimal one(s), which is not necessarily as simple as comparing their total returns.

The total return some policy acquires in a GOMDP measures how "successful" that policy is in a sense, but it does not measure how far the policy is from optimal,

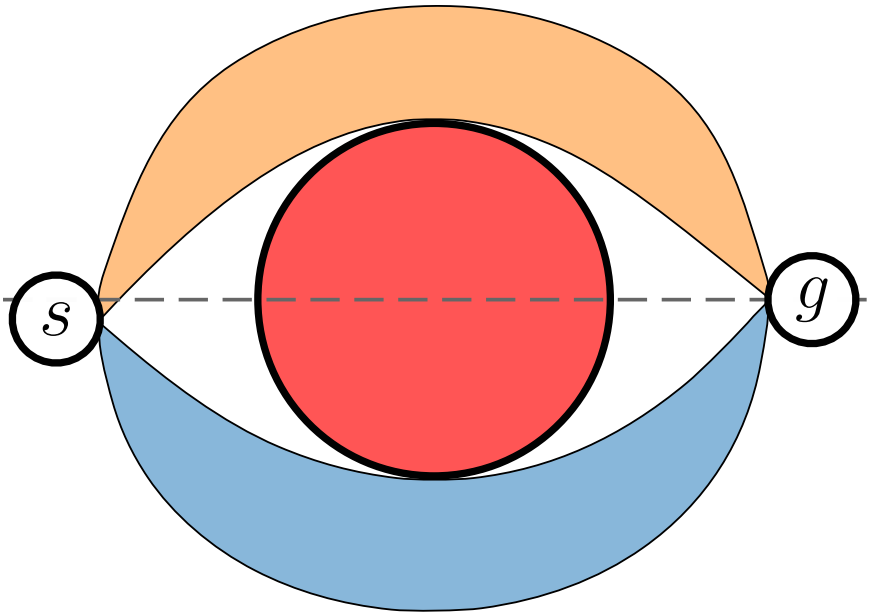


Figure 3.1: Two sets of trajectories that have zero overlap such that no path in one can be continuously deformed to a path in the due to the red obstacle (or topological hole) in the middle. The optimal trajectory in each of the two groups have very similar maximal returns in this example GOMDP. Note that the misalignment of the starting positions is intentional: the blue region contains slightly shorter paths than the orange region because of this shift. While measuring the value of these paths would indicate they have similar optimality, topologically they are very different and not homotopic.

i.e. the difference in its behaviour compared to that of the optimal policy. Figure 3.1 shows two different sets of trajectories such that the optimal one in each set would achieve only slightly different returns in a GOMDP. While the returns would indicate these policies perform very similarly, there is no "easy" way for an agent to transform the sub-optimal one to the optimal one through small perturbations.

Situations like this are the result of the complexity of the space being navigated. The more complex a landscape is, the more likely these separations exist in its trajectory space. The next section discusses how the complexity of these navigational spaces can be measured.

3.3 Topological Complexity

Topological complexity is a measurement of the "navigational discontinuities" in motion planning [62]. For a connected topological space $\mathcal{S}$, the motion planning problem consists of finding a continuous path in $\mathcal{S}$ from s to s' for each $(s, s') \in \mathcal{S} \times \mathcal{S}$. We assume $\mathcal{S}$ is connected, for simplicity. Let $\mathcal{PS}$ be the set of continuous paths in $\mathcal{S}$, equipped with the compact-open topology, and denote $f : \mathcal{S} \times \mathcal{S} \to \mathcal{PS}$ a set of paths between any two states in $\mathcal{S}$, i.e. a solution to the motion planning problem. If f is continuous, then it is called a continuous motion planning on $\mathcal{S}$. A continuous motion planning f exists if and only if $\mathcal{S}$ is contractible [62].

The topological complexity, denoted $TC(\mathcal{S})$, is defined as the minimal number k such that $\mathcal{S} \times \mathcal{S}$ can be covered by k open subsets, where there exists a continuous motion planning in each open subset. If $\mathcal{S}$ is contractible, then $TC(\mathcal{S}) = 1$ [62]. For example, the topological complexity of a disk is 1 because a disk is contractible. If $\mathcal{S}$ is not contractible, then $TC(\mathcal{S}) > 1$. For example, the topological complexities of a circle and Klein bottle are 2 [62] and 4 [63] respectively. Topological complexity is closely related to the Lyusternik-Schnirelmann category of a space, denoted $\text{cat}(\mathcal{S})$, which is the minimal number of open sets needed to cover $\mathcal{S}$ such that each set is contractible within $\mathcal{S}$. While $\text{cat}(\mathcal{S})$ measures the topological complexity of the space itself, $TC(\mathcal{S})$ measures the complexity of motion planning within it. They are fundamentally related by the bounds $\text{cat}(\mathcal{S}) \leq TC(\mathcal{S}) < \text{cat}(\mathcal{S} \times \mathcal{S})$ [62]. While computationally impractical, an algorithm for computing the topological complexity of a space can be found in Appendix 8.1.

Let M be a continuous generalized GOMDP with state space $\mathcal{S}$ and goal states $\mathcal{G}$. We define the motion planning problem for generalized GOMDPs to consist of finding a continuous path in $\mathcal{S}$ from s to g for each $s \in \mathcal{S}$ and some $g \in \mathcal{G}$. We define

a policy π to be successful in M if it is a solution to the generalized GOMDP motion planning problem. By definition of generalized GOMDPs, an optimal policy π^* not only maximizes total return but is also a solution to the generalized GOMDP motion planning problem.

3.3.1 Path Homotopy

Let $\mathcal{X}$ and $\mathcal{Y}$ be two topological spaces. For two continuous functions $f, h : \mathcal{X} \to \mathcal{Y}$, a homotopy between f and h is defined to be a continuous function $H : \mathcal{X} \times [0, 1] \to \mathcal{Y}$ such that $H(x, 0) = f(x)$ and $H(x, 1) = h(x)$ for all $x \in \mathcal{X}$. f and h are said to be homotopic if there exists a homotopy between them. A homotopy class of f is defined to be the set of all functions homotopic to f.

Now consider the case where f and h are paths between x_0 and x_1 for $x_0, x_1 \in \mathcal{X}$, i.e. $f, h : [0, 1] \to \mathcal{X}$ are continuous functions where $f(0) = h(0) = x_0$ and $f(1) = h(1) = x_1$. A path homotopy between f and h is defined to be a continuous function $H : [0, 1] \times [0, 1] \to \mathcal{X}$ such that $H(i, 0) = f(i)$ and $H(i, 1) = h(i)$ for $i \in [0, 1]$, and additionally $H(0, t) = x_0$ and $H(1, t) = x_1$ for all $t \in [0, 1]$ (i.e., the endpoints remain fixed throughout the deformation). f and h are said to be homotopic paths if there exists a path homotopy between them.

Let M be a continuous generalized GOMDP with state space $\mathcal{S}$ and set of goal-states $\mathcal{G}$. For some initial state $s_0 \in \mathcal{S}$ and goal-state $g \in \mathcal{G}$, consider two successful policies π_1 and π_2. In the deterministic case, these two policies each produce a path in $\mathcal{S}$ from s_0 to g. If they are homotopic paths, then there exists a path homotopy that continuously deforms one path into the other. In the context of learning, the deformation of one path into the other would be similar to making a series of small perturbations to one policy to deform it into the other, where the size of perturbation corresponds to how "close" the resulting policy is in the space of all policies. In con-

trast, if these two paths are not homotopic, then no such series of local perturbations exists—at some point in the sequence, a non-local (larger) perturbation is required to "jump" between homotopy classes. Note that if $\mathcal{S}$ is contractible (i.e. has a topological complexity of 1), then all paths between any two given states are homotopic paths.

This section is meant to give insight on how topological complexity and path homotopy classes can be used to describe the navigational landscape of a GOMDP. The next section concludes this chapter with an analysis of the sample complexity on separated-path MDPs, which are are a special class of problem where paths are segregated, much like the homotopy classes discussed here.

3.4 Sample Complexity of Separated-Path MDPs

Azar et. al. [35] provide the following lower bound on the sample complexity for *single-choice separated-path MDPs* (defined below) with identical transition probabilities. There exists some constants ϵ_0, δ_0, and a set of MDPs $\mathcal{M}$, such that for all $\epsilon \in (0, \epsilon_0)$, $\delta \in (0, \delta_0/N)$, and every (ϵ, δ, T)-correct RL algorithm on the set $\mathcal{M}$, the number of transitions needs to be at least

$$T = \lceil \frac{N}{8100(1-\gamma)^3\epsilon^2} \log\left(\frac{N}{6\delta}\right) \rceil. \tag{3.5}$$

The set of MDPs used for the proof of this bound will be denoted as single-choice separated-path MDPs, which is defined as follows: The set of MDPs $\mathcal{M}$ that have a state-action space of cardinality $N = 3KL$, where K and L are positive integers such that for each $M \in \mathcal{M}$, the state space $\mathcal{X}$ consists of three smaller sets $\mathcal{S}$, $\mathcal{Y}^1$ and $\mathcal{Y}^2$. $\mathcal{S}$ consists of K states, each of which has a set of deterministic actions

$\mathcal{A} = \{a_1, a_2, \ldots, a_L\}$, whereas the states in $\mathcal{Y}^1$ and $\mathcal{Y}^2$ are single-action states, i.e. no choice involved. Taking the action $a \in \mathcal{A}$ from any state $s \in \mathcal{S}$ moves to the state $y_1(s,a) \in \mathcal{Y}^1$ with probability 1. The single action available at each state in $\mathcal{Y}^1$ is a self-transition governed by the transition probability p_M. Specifically, the transition probabilities from every $y_1(s,a) \in \mathcal{Y}^1$ is p_M to itself and $1 - p_M$ to the corresponding state $y_2(s,a) \in \mathcal{Y}^2$ where all states in $\mathcal{Y}^2$ are goal-states. In general, only $0 \leq p_M < 1$ is required in this formulation, however the PAC-learning bounds presented below require $0.5 < p_M < 1$ to apply (due to the Chernoff bound conditions used in the derivation). Each $y_1(s,a) \in \mathcal{Y}^1$ and $y_2(s,a) \in \mathcal{Y}^2$ is distinct for each $s \in \mathcal{S}$ and $a \in \mathcal{A}$, thus $|\mathcal{Y}^1| = |\mathcal{Y}^2| = KL$. The reward is $+1$ for arriving in any state in $\mathcal{Y}^1$ (including from itself) and 0 elsewhere. For any $s \in \mathcal{S}$, the optimal Q-function for this set of MDPs is

$$Q^*(s,a) = \gamma V(y_1(s,a)) = \frac{\gamma}{1 - \gamma p_M}, \tag{3.6}$$

where γ is the discount factor and $V(y_1(s,a))$ denotes the value of the intermediate state reached after taking action a from s.

To show the wide applicability of this class of problem, they can be related back to GOMDPs. $\mathcal{S}$ is the set of states where paths diverge into different homotopy classes (which may or may not coincide with the set of initial states in a GOMDP), $\mathcal{Y}^2$ is the set of goal-states, and $\mathcal{Y}^1$ represents the path homotopy classes between $\mathcal{S}$ and $\mathcal{Y}^2$. The value for each state in $\mathcal{Y}^1$, inversely determined by the transition probability, represents the optimality of that path homotopy class. For each state $s \in \mathcal{S}$, each of the L actions represent choosing a path, or homotopy class of paths to follow, analogous to options in RL, where an option is picking a set of actions rather than a single action. The sample complexity of these problems then represents how difficult learning the value of these path homotopy classes.

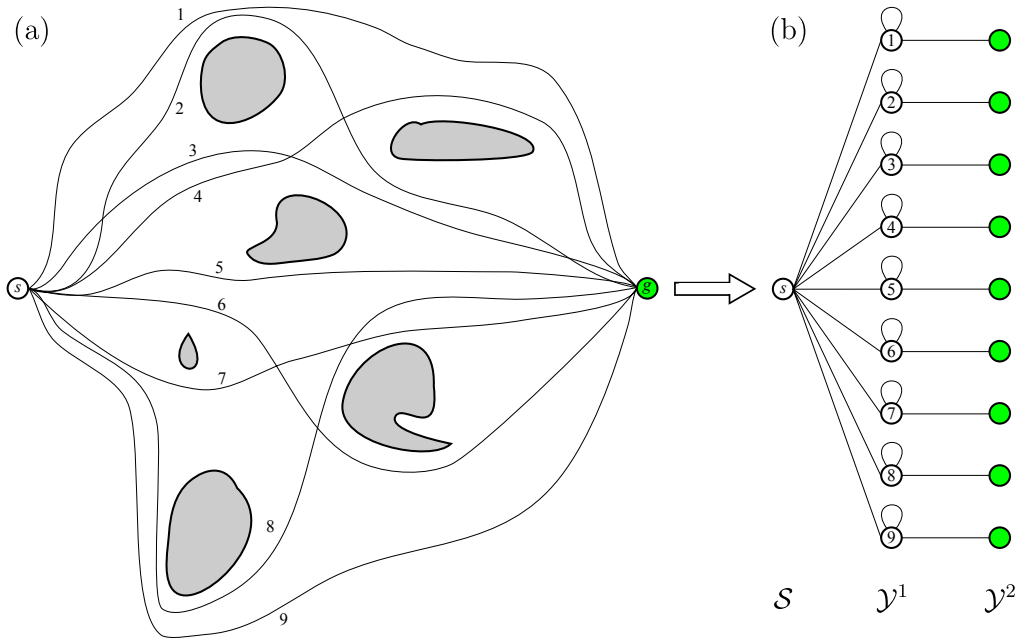


Figure 3.2: Correspondence between a GOMDP and a separated-path MDP. (a) A GOMDP with initial state s and goal-state g. Obstacles (shaded regions) create topological holes in the state space, resulting in multiple non-homotopic path classes (numbered 1–$L = 9$) between s and g. (b) The corresponding separated-path MDP. The initial state $s \in \mathcal{S}$ has L actions, each leading to a distinct state in $\mathcal{Y}^1$ that represents one of the homotopy classes from (a). All paths in $\mathcal{Y}^1$ eventually reach the goal set $\mathcal{Y}^2$.

Remark (Connection to Topological Complexity)**.** *The separated-path MDP structure provides a discrete approximation to the continuous topological setting described in Section 3.3. Specifically, the number of actions L at each state $s \in \mathcal{S}$ corresponds to the number of distinct homotopy classes of paths from s to the goal set $\mathcal{G}$. In a continuous generalized GOMDP with topological complexity k, at least k fundamentally different regions of the path space that require separate learning would be expected. The sample complexity bounds derived below thus scale with L, reflecting the intuition that more topologically complex environments (with more homotopy classes) require more samples to learn. An example of this comparison between GOMDPs and separated-path MDPs is shown in Figure 3.2*

The class of MDPs used by Azar et. al. [35] in their current state is not yet appropriate for this representation as each "path homotopy class" is equivalent—identical transition probabilities p_M for each $y_1(s,a) \in \mathcal{Y}^1$ means equivalent values. While this class of MDPs is relatively simple and quite a specific use case, the next few sections generalize these results. First, the results are generalized to the class of single-choice separated-path MDPs where each $y_1(s,a) \in \mathcal{Y}^1$ has its own transition probability $p(s,a)$ that are not necessarily equal for all states. With the generalized transition probabilities, the results are then adapted for the sample complexity of the value function and the optimal policy, instead of just the Q-function. Lastly, these results are generalized further for the class of single-choice separated-path POMDPs.

3.4.1 Generalizing the Sample Complexity Bound

Let M be a single-choice separated-path MDP with the transition probabilities for each $y_1(s,a) \in \mathcal{Y}^1$ given by p_{sa}. Let T_{sa} be the number of times the state-action pair (s,a) has been observed. From Lemma 8 in Azar et. al. [35], for $s \in \mathcal{S}$, $a \in \mathcal{A}$, and some $\epsilon > 0$, we have

$$\Pr(|Q^*(s,a) - Q_{T_{sa}}(s,a)| > \epsilon) > \frac{1}{6} e^{-\frac{\alpha^2 T_{sa}}{p_{sa}(1-p_{sa})}}, \tag{3.7}$$

where $\alpha = 2(1-\gamma p_{sa})^2 \epsilon / \gamma^2$. Simplifying this gives

$$\Pr(|Q^*(s,a) - Q_{T_{sa}}(s,a)| > \epsilon) > \frac{1}{6} e^{-\frac{1024(1-\gamma p_{sa})^4 \epsilon^2 T_{sa}}{9 p_{sa}(1-p_{sa})\gamma^4}} \tag{3.8}$$

$$\geq \frac{1}{6} e^{-c(\gamma_0)(1-\gamma)^3 \epsilon^2 T_{sa}} \tag{3.9}$$

where the right side of the inequality (Equation 3.9) arises when $p_{sa} = (4\gamma - 1)/3\gamma$ for $0.4 = \gamma_0 < \gamma$ with $c(\gamma_0) = 1024/(9\gamma_0^2(4\gamma_0 - 1))$, which is the one used when

deriving Equation 3.5. Note, Azar et. al. [35] chooses $c(\gamma_0) = 8100$ which satisfies this inequality. We will focus on the first inequality (Equation 3.8) as it pertains to a generic transition probability which only requires $p_{sa} > 0.5$ due to the Chernoff bound conditions.

For $(s, a) \in \mathcal{S} \times \mathcal{A}$, we define

$$\xi_{sa}(\epsilon, \delta) = \frac{9p_{sa}(1 - p_{sa})\gamma^4}{1024(1 - \gamma p_{sa})^4\epsilon^2} \log\left(\frac{1}{6\delta}\right), \tag{3.10}$$

such that if $T_{sa} < \xi_{sa}(\epsilon, \delta)$, then

$$\Pr(|Q^*(s, a) - Q_{T_{sa}}(s, a)| > \epsilon) > \delta. \tag{3.11}$$

By using $\xi_M(\epsilon, \delta)$ in place of $\xi(\epsilon, \delta)$ in Lemma 11 from Azar et. al. [35] where

$$\xi_M(\epsilon, \delta) = \min_{(s,a)\in\mathcal{S}\times\mathcal{A}} \xi_{sa}(\epsilon, \delta), \tag{3.12}$$

we have the generalized lower bound on sample complexity of the Q-function for a single-choice separated-path MDP M is given by

$$T = \lceil N\xi_M\left(\epsilon, \frac{\delta}{12N}\right)\rceil. \tag{3.13}$$

Remark. *The preceding derivation relies on Lemma 8 and Lemma 11 from Azar et. al. [35]. For convenience, these lemmas are restated in Appendix 8.2.*

Known State Distribution

To acquire the lower bound above, Lemma 11 from Azar et. al. [35] assumes nothing about the distribution over $\mathcal{S}$ and thus constructs the bound using the highest number

of transition samples such that it holds regardless of how the samples are distributed. To strengthen the lower bound, the distribution of states is incorporated.

Let P be the distribution over $\mathcal{S}$ such that P_s be the probability that s is the initial state. For $s \in \mathcal{S}$, define

$$\xi_s(\epsilon, \delta) = \sum_{a \in \mathcal{A}} \xi_{sa}(\epsilon, \delta) \tag{3.14}$$

as the minimum number of times s needs to be visited to satisfy Equation 3.10 for all $a \in \mathcal{A}$. Let $\eta > 0$ be a threshold parameter (with the same units as sample count). For $\epsilon, \delta, \delta' > 0$, applying Hoeffding's inequality gives

$$\Pr(n_s \leq \xi_s(\epsilon, \delta) - \eta) = \Pr(n_s - tP_s \leq \xi_s(\epsilon, \delta) - \eta - tP_s) \leq e^{-\frac{2(tP_s - (\xi_s(\epsilon,\delta) - \eta))^2}{t}}, \tag{3.15}$$

where n_s is the number of times s has been visited and t is the total number of samples. This bound is valid when $tP_s \geq \xi_s(\epsilon, \delta) - \eta$ (otherwise the probability is trivially 1). Thus, if

$$\begin{aligned} t \leq \xi_P(\epsilon, \delta, \delta', \eta) =& \frac{4P_s(\xi_s(\epsilon, \delta) - \eta) + \log(1/\delta')}{4P_s^2} \\ &+ \frac{\sqrt{(\log(1/\delta') + 4P_s(\xi_s(\epsilon, \delta) - \eta))^2 - 16P_s^2(\xi_s(\epsilon, \delta) - \eta)^2}}{4P_s^2}, \end{aligned} \tag{3.16}$$

then $\Pr(n_s \leq \xi_s(\epsilon, \delta) - \eta) > \delta'$. Note that solving for the bound in Equation 3.16 leads to multiple quadratic solutions, however as it is a lower bound, we use the smallest of those solutions. In practice, one can choose $\eta = 1$ (a single sample threshold) without loss of generality.

To ensure this holds for all s, we have

$$\Pr(\exists s \in \mathcal{S} \text{ s.t. } n_s \leq \xi_s(\epsilon, \delta) - \eta) \leq \sum_{s \in \mathcal{S}} \Pr(n_s \leq \xi_s(\epsilon, \delta) - \eta), \tag{3.17}$$

by the union bound. Using $\delta' = \delta/K$ for Equation 3.16 in combination with Equation 3.17, we have if

$$T \geq \max_{s \in \mathcal{S}} \left[\xi_P(\epsilon, \delta, \delta/K, \eta)\right]. \tag{3.18}$$

then

$$\Pr(\exists s \in \mathcal{S} \text{ s.t. } n_s \leq \xi_s(\epsilon, \delta) - \eta) \leq \sum_{s \in \mathcal{S}} \frac{\delta}{K} = \delta. \tag{3.19}$$

In other words, if T does not satisfy the above bound, then with probability of at least δ, there exists at least one state such that Equation 3.16 is not satisfied. Therefore, if the initial state distribution P is known, we have the lower bound on sample complexity of the Q-function (with respect to P) for a single-choice separated-path MDP is given by

$$T = \lceil \max_{s \in \mathcal{S}} \left[\xi_P(\epsilon, \delta, \delta/K, \eta)\right] \rceil. \tag{3.20}$$

Next we adapt these results to the value function.

3.4.2 Sample Complexity of the Value Function

Consider the class of single-choice separated-path MDPs described above. For $s \in \mathcal{S}$, the value of s is

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a) = \max_{a \in \mathcal{A}} \frac{\gamma}{1 - \gamma p_{sa}}. \tag{3.21}$$

Let a_s be the optimal action at s, i.e. $V^*(s) = Q^*(s, a_s)$. In order for our estimate $V_t(s)$ to be PAC (for some $\epsilon > 0$), we require both $Q_t(s, a_s)$ to be PAC (for the same ϵ) and $Q_t(s, a)$ to be PAC (for some $\epsilon' > \epsilon$) for all $a \in \mathcal{A} \setminus \{a_s\}$. We require

$Q_t(s, a_s)$ to have the accuracy desired for $V_t(s)$, however, we only require $Q_t(s, a)$ for all $a \in \mathcal{A} \setminus \{a_s\}$ to be accurate enough to say $V_t(s) \neq Q_t(s, a)$ for those actions.

Let T_{sa} be the number of times the state-action pair (s, a) has been observed. For $Q_{T_{sa_s}}(s, a_s)$, we have

$$\Pr(\left|Q^*(s, a_s) - Q_{T_{sa_s}}(s, a_s)\right| > \epsilon) > \delta, \tag{3.22}$$

if $T_{sa_s} < \xi_{sa_s}(\epsilon, \delta)$ with ξ_{sa_a} from Equations 3.7 & 3.10. The choice of α in Equations 3.7 ensures that if one was instead learning in some MDP M' arbitrarily close to M, the random event of acquiring a PAC estimate of the Q-function in M' does not overlap with the random event of acquiring a PAC estimate of the Q-function in M. For each $a \in \mathcal{A} \setminus \{a_s\}$, define $\alpha_{sa} = 2(1 - \gamma p_{sa})^2 \epsilon_{sa}/\gamma^2$ and choose $\epsilon_{sa} > 0$, such that $p_{sa_s} - p_{sa} = \alpha_{sa}$. Note, as a_s is the optimal action at s, $p_{sa_s} > p_{sa}$, hence we will always have $\epsilon_{sa} > \epsilon$. This gives us

$$\Pr(|Q^*(s, a) - Q_{T_{sa}}(s, a)| > \epsilon_{sa}) > \delta, \tag{3.23}$$

if $T_{sa} < \xi_{sa}(\epsilon_{sa}, \delta)$. This combined with

$$Q^*(s, a) + \epsilon_{sa} < Q^*(s, a_s) - \epsilon_{sa} < Q^*(s, a_s) - \epsilon, \tag{3.24}$$

for $\epsilon < \epsilon_{sa}$ for all $a \in \mathcal{A} \setminus \{a_s\}$ we have

$$\Pr(|V^*(s) - V_{T_s}(s)| > \epsilon) > \delta, \tag{3.25}$$

if $T_s < \xi'_s(\epsilon, \delta) = \xi_{sa_s}(\epsilon, \delta) + \sum_{a \in \mathcal{A} \setminus \{a_s\}} \xi_{sa}(\epsilon_{sa}, \delta)$, where T_s is the number of times the state s has been observed. Note that $\xi'_s(\epsilon, \delta) < L\xi_{sa}(\epsilon, \delta)$.

Using T_s rather than T_{sa} means we now only need to consider the K states in $\mathcal{S}$ with $t < \xi'_s(\epsilon, \delta)$ rather than the $N = KL$ state-action pairs with $t < \xi_{sa}(\epsilon, \delta)$. Using Lemma 11 from Azar et. al. [35] in this way, we have the lower bound on sample complexity for the value function of a single-choice separated-path MDP is given by

$$T = \lceil K\xi_M \left(\epsilon, \frac{\delta}{4K}\right) \rceil, \tag{3.26}$$

where

$$\xi'_M(\epsilon, \delta) = \min_{s \in \mathcal{S}} \xi'_s(\epsilon, \delta). \tag{3.27}$$

To extend this to the case where the distribution of initial states is known, we plug $\xi'_s(\epsilon, \delta)$ into Equation 3.16, thus if

$$\begin{aligned} t \leq \xi'_P(\epsilon, \delta, \delta', \eta) =& \frac{4P_s(\xi'_s(\epsilon, \delta) - \eta) + \log(1/\delta')}{4P_s^2} \\ &+ \frac{\sqrt{(\log(1/\delta') + 4P_s(\xi'_s(\epsilon, \delta) - \eta))^2 - 16P_s^2(\xi'_s(\epsilon, \delta) - \eta)^2}}{4P_s^2}, \end{aligned} \tag{3.28}$$

then $\Pr(n_s \leq \xi'_s(\epsilon, \delta) - \eta) > \delta'$. Using Equation 3.17 with Equation 3.28, we have if

$$T \geq \max_{s \in \mathcal{S}} \left[\xi'_P(\epsilon, \delta, \delta/K, \eta)\right]. \tag{3.29}$$

then

$$\Pr(\exists s \in \mathcal{S} \text{ s.t. } n_s \leq \xi'_s(\epsilon, \delta) - \eta) \leq \sum_{s \in \mathcal{S}} \frac{\delta}{K} = \delta. \tag{3.30}$$

Therefore, if the initial state distribution P is known, we have the lower bound on the sample complexity of the value function (with respect to P) for a single-choice separated-path MDP is given by

$$T = \lceil \max_{s \in \mathcal{S}} \left[\xi'_P(\epsilon, \delta, \delta/K, \eta)\right] \rceil. \tag{3.31}$$

Next we can adapt this to learning the optimal policy function instead of the value function.

3.4.3 Sample Complexity of the Optimal Policy

Consider the class of single-choice separated-path MDPs described above. For $s \in \mathcal{S}$, the optimal action at s is given by the optimal policy

$$\pi^*(s) = a_s = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q^*(s,a) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \frac{\gamma}{1-\gamma p_{sa}}. \tag{3.32}$$

For estimating the value function, we needed both the estimate of the Q-function at the optimal action to be PAC and the estimate of the Q-function at the optimal action to be the maximum over the action space. For estimating the optimal policy, we only require the latter. Therefore, we only require our estimate of the Q-function to be accurate enough such that $\max_{a \in \mathcal{A}} Q_t(s,a) = Q_t(s,a_s)$.

Let $\bar{a}_s$ be the next best action at state s, i.e. $\bar{a}_s = \operatorname{argmax}_{a \in \mathcal{A} \setminus \{a_s\}} Q^*(s,a)$. By relaxing Equations 3.24 & 3.25 and for any $0 < \epsilon \leq \epsilon_{s\bar{a}_s}$, we have

$$\Pr(\pi^*(s) = \pi_{T_s}(s)) > \delta, \tag{3.33}$$

if $T_s < \xi'_s(\epsilon_{s\bar{a}_s}, \delta)$, where T_s is the number of times the state s has been observed. Note that $\xi'_s(\epsilon_{s\bar{a}_s}, \delta) < \xi'_s(\epsilon, \delta)$.

Combined with Equation 3.26, we have the lower bound on sample complexity for the optimal policy of a single-choice separated-path MDP is given by

$$T = \lceil K \xi''_M \left(\frac{\delta}{4K} \right) \rceil, \tag{3.34}$$

where

$$\xi''_M(\delta) = \min_{s \in \mathcal{S}} \xi'_s(\epsilon_{s\bar{a}_s}, \delta). \tag{3.35}$$

Note that the specific choice of ϵ is irrelevant as the policy is a classification model, where the value and Q-functions are regression models.

To extend this to the case where the distribution of initial states is known, we simply need to replace ϵ with $\epsilon_{s\bar{a}_s}$ in Equation 3.28, giving us

$$T = \lceil \max_{s \in \mathcal{S}} [\xi'_P(\epsilon_{s\bar{a}_s}, \delta, \delta/K)] \rceil, \tag{3.36}$$

as the sample complexity for learning the optimal policy with respect to P.

Lastly, we adapt the previous results to the partially observable setting.

3.4.4 Sample Complexity of POMDPs

Consider the class of single-choice separated-path MDPs described above, with the additional constraint that the exact current state $s \in \mathcal{S}$ is not known. Let $\rho = \{\rho_s | s \in \mathcal{S}\}$ be the belief state where $\rho_s = \Pr(s' = s|\mathcal{I})$ for some information set $\mathcal{I}$ when s' is the true state. We have

$$Q^*(\rho, a) = \sum_{s \in \mathcal{S}} \rho_s Q^*(s, a) \tag{3.37}$$

as the optimal Q-function for the belief state-action pair (ρ, a). In order for our estimate $Q_t(\rho, a)$ to be PAC (for some $\epsilon > 0$), we require both $Q_t(s, a)$ for all s to be PAC (for some $\epsilon' > \epsilon$), and for the belief state to be sufficiently sampled.

Let $T_{\rho a}$ be the number of times the belief state-action pair (ρ, a) has been observed. For some $\epsilon > 0$, define $\epsilon_s = \epsilon / K\rho_s$. We have

$$\left|Q^*(\rho, a) - Q_{T_{\rho a}}(\rho, a)\right| \leq \sum_{s \in \mathcal{S}} \rho_s \left|Q^*(s, a) - Q_{T_{\rho a}}(s, a)\right| \leq \epsilon \tag{3.38}$$

if $|Q^*(s,a) - Q_t(s,a)| \leq \epsilon_s$ for all $s \in \mathcal{S}$.

To ensure the belief state-action pair (ρ, a) has been sampled sufficiently, for $T_{\rho a}$ transitions we have

$$\Pr\left(|T_{\rho a}\rho_s - n_s| \leq \frac{\epsilon}{K\rho_s}\right) \geq 1 - 2e^{-\frac{\epsilon^2 t}{2K^2\rho_s^3(1-\rho_s)}}, \tag{3.39}$$

using Hoeffding's inequality, where n_s is the number of times the $T_{\rho a}$ transitions at the belief state-action pair (ρ, a) corresponded to state s. If

$$T_{\rho a} < \zeta_{\rho a}(\epsilon, \delta) = \frac{2K^2 \max_{s\in\mathcal{S}}[\rho_s^3(1-\rho_s)]}{\epsilon^2} \log\left(\frac{2}{\delta}\right) \tag{3.40}$$

for some $\delta > 0$, then

$$\Pr\left(|T_{\rho a}\rho_s - n_s| > \frac{\epsilon}{2\rho_s}\right) > \delta \tag{3.41}$$

for all s.

With belief state ρ, taking action a results in reaching state $y_1(s,a)$ with probability ρ_s, where rewards are generated from sampling Bernoulli distributions with probabilities p_{sa}. Although a mixture of Bernoulli distributions is not itself a Bernoulli distribution in general, the *expected* transition probability is $p_{\rho a} = \sum_{s\in\mathcal{S}} \rho_s p_{sa}$. For the purpose of deriving a lower bound, we consider an "effective" placeholder state s' with corresponding transition state $y_1(s', a)$ that has transition probability $p_{\rho a}$. This provides a lower bound because the actual variance of the mixture distribution is at least as large as that of a single Bernoulli with the same mean. If $T_{(s',a)} < \xi_{s'a}(\epsilon, \delta)$, then

$$\Pr(|Q^*(s', a) - Q_t(s', a)| > \epsilon) > \delta, \tag{3.42}$$

from Equation 3.8.

Combining Equations 3.40 & 3.10, we have if

$$T_{(\rho,a)} < \xi'_{\rho a}(\epsilon, \delta) = \max\left[\xi_{s'a}(\epsilon, \delta), \zeta_{\rho a}(\epsilon, \delta)\right], \tag{3.43}$$

then

$$\Pr(|Q^*(\rho, a) - Q_t(\rho, a)| > \epsilon) > \delta. \tag{3.44}$$

For sufficiently small $0 < \delta << 1$ and large $0 << \gamma < 1$ with

$$\max_{s \in \mathcal{S}} \rho_s^3(1 - \rho_s) \propto \frac{p(1-p)\gamma^4}{N^2(1-\gamma p)^4\delta}, \tag{3.45}$$

we have $\xi'_{\rho a}(\epsilon, \delta) = \xi_{s'a}(\epsilon, \delta)$. This means that the complexity of the POMDP is less than the complexity of the uncertainty in the belief state in this scenario. However, in all other cases we have $\xi'_{\rho a}(\epsilon, \delta) = \zeta_{\rho a}(\epsilon, \delta)$. This is because even though the belief state is a distribution over many states, the transition dynamics of the problem are equivalent to the averaged case, as discussed above. Thus, as the uncertainty in the belief state increases—and therefore the sample complexity of sampling the distribution of states—the complexity of the averaged dynamics does not increase with it.

The framework developed in this chapter provides a way to characterize goal-oriented decision problems through the structure of their feasible routes, the relationships between distinct homotopic path classes, and the organization of trajectories that lead to a target. These notions of structural and topological variation offer a foundation for interpreting why some tasks are more challenging for learning agents than others. In the chapters that follow, a range of applied environments are considered—thermodynamic controls, nautical navigation under uncertainty, and chemical laboratory procedures—and in each case, many of the observed learning

behaviours can be related back to differences in the underlying goal-oriented structure of the tasks. The concepts introduced here therefore act as a reference point for understanding how task organization influences agent performance across diverse domains and how differently these structures can appear.

Chapter 4

Maximizing Thermal Efficiency

In this chapter, we examine the model heat engine system, expanding on the original learning results with a deeper analysis informed by the complexity framework developed in Chapter 3. Specifically, we examine how the trajectory structure and observability of the system affect learning dynamics, relating our findings to the concepts of path organization and sample complexity introduced previously. The original learning results on this system used evolutionary learning to accurately approximate the optimal policy in the POMDP setting [2], which were then expanded upon to also include RL to approximate the optimal policy in the MDP setting [1]. Much of this chapter is adapted from the original work that has been previously published [1], first authored by C. Beeler. The material has been revised for coherence and consistency within the overall structure of this thesis.

To approximate the optimal policy, a neural network takes as input the current observation of the engine and chooses one of a set of basic thermodynamic processes to transform the state of the system, which produces a new observation. This simple and well-known system was chosen for pedagogical purposes. Using an easy-to-understand system allows us to focus on how RL methods can be applied to the physics problems

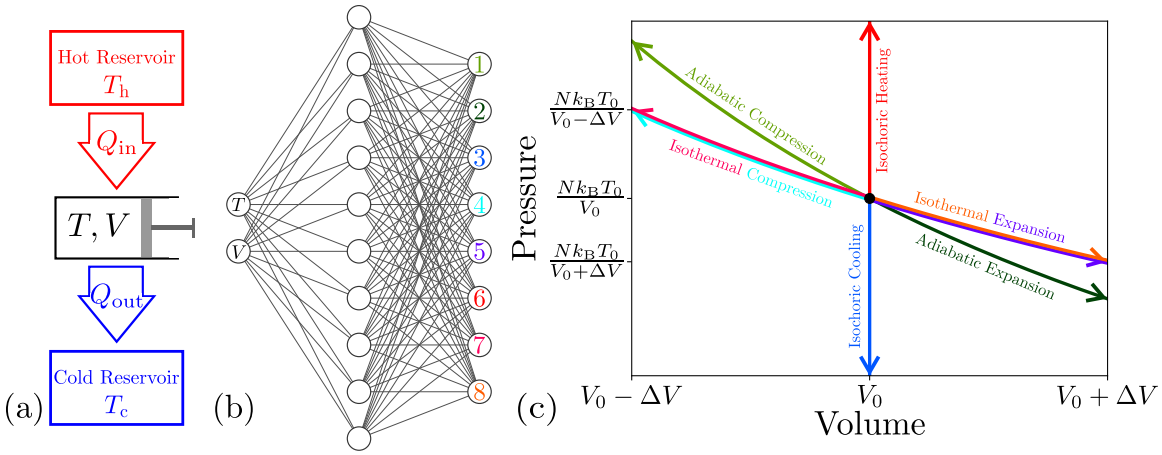


Figure 4.1: (a) Model heat engine and (b) the neural network that controls it. Note that the diagram of the neural network has two additional input nodes for the gradient-based RL method. (c) A summary of the actions in P-V space available to the network; see Table 4.1.

and what makes them more difficult to learn. We generate a set of trajectories of fixed length using a set of networks whose parameters are initially randomly chosen and optimized using two different methods. In the first method (gradient-free), we retain and mutate only those networks whose trajectories show the greatest thermal efficiency. Repeating this evolutionary process many times results in networks whose trajectories reproduce the maximally efficient Carnot, Stirling, or Otto cycles, depending upon which basic thermodynamic processes are allowed. This evolutionary procedure can also learn previously unknown thermodynamic cycles if new processes are allowed. In the second method (gradient-based), we update the network parameters using gradient-based RL. In this study, we explore how the framing of the problem affects the types of solutions acquired by these two methods. We also compare the advantages and disadvantages of each in finding a solution to this problem.

4.1 Partially Observable Model Heat Engine

In Figure 4.1(a) we show a model heat engine, a device able to transform thermal energy into work [64,65]. The engine consists of a working substance, which we assume to be a monatomic ideal gas, housed within a frictionless container of variable volume V, whose minimum and maximum values are $V_{\min}$ and $V_{\max}$, respectively. The working substance may be connected to a hot or cold reservoir held at temperature, $T_{\rm h} = 500$ K and $T_{\rm c} = 300$ K, respectively. For the gradient-free method, the instantaneous observation s_t of the system at time t is then specified by the volume-temperature vector $s_t = (V, T)$, with the pressure of the system fixed by the ideal-gas equation $PV = Nk_{\rm B}T$ [66]. Further on, we will discuss the reasons this formulation of the problem cannot be used for standard gradient-based RL methods.

The elementary actions available to the network correspond to the basic thermodynamic processes shown in Table 4.1, summarized graphically in Figure 4.1(c). These processes include reversible compression and expansion (variable volume), along isotherms (constant temperature) or adiabats (no heat transfer), and reversible temperature changes along isochores (constant volume). Implicitly this means infinitely many heat baths spaced between $T_{\rm h}$ and $T_{\rm c}$ are available, a condition—which we prove in Appendix 8.3—does not undermine Carnot's Theorem of maximum efficiency. All compression and expansion processes are performed using a fixed change in volume. If an isothermal process at $T_{\rm h}$ (or $T_{\rm c}$) is selected when the system is not at the correct temperature, then isochoric heating (or cooling) is performed first to reach the necessary temperature for the isothermal process to occur. Note that the isochoric processes are not used in the Carnot cycle, but are required to make approximations of it when using fixed changes in volume because the system must reach specific volumes in order to be adiabatically heated from $T_{\rm c}$ to $T_{\rm h}$ and cooled from $T_{\rm h}$

to T_c.

Table 4.1: All possible actions that can be taken on our model heat engine and their corresponding ΔW and ΔQ equations.

Action	ΔW	ΔQ
Adiabatic Compression	$-\frac{3}{2}Nk_{\mathrm{B}}T_{\mathrm{i}}\left(\left(\frac{V_{\mathrm{i}}}{V_{\mathrm{f}}}\right)^{\frac{2}{3}}-1\right)$	0
Adiabatic Expansion	$-\frac{3}{2}Nk_{\mathrm{B}}T_{\mathrm{i}}\left(\left(\frac{V_{\mathrm{i}}}{V_{\mathrm{f}}}\right)^{\frac{2}{3}}-1\right)$	0
Isothermal Compression at T_{h} $(T=T_{\mathrm{h}})$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$
Isothermal Expansion at T_{h} $(T=T_{\mathrm{h}})$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$
Isothermal Compression at T_{h} $(T\neq T_{\mathrm{h}})$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$ $+\frac{3}{2}Nk_{\mathrm{B}}\left(T_{\mathrm{h}}-T_{\mathrm{i}}\right)$
Isothermal Expansion at T_{h} $(T\neq T_{\mathrm{h}})$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{h}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$ $+\frac{3}{2}Nk_{\mathrm{B}}\left(T_{\mathrm{h}}-T_{\mathrm{i}}\right)$
Isothermal Compression at T_{c} $(T=T_{\mathrm{c}})$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$
Isothermal Expansion at T_{c} $(T=T_{\mathrm{c}})$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$
Isothermal Compression at T_{c} $(T\neq T_{\mathrm{c}})$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$ $+\frac{3}{2}Nk_{\mathrm{B}}\left(T_{\mathrm{c}}-T_{\mathrm{i}}\right)$
Isothermal Expansion at T_{c} $(T\neq T_{\mathrm{c}})$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$	$Nk_{\mathrm{B}}T_{\mathrm{c}}\log\left(\frac{V_{\mathrm{f}}}{V_{\mathrm{i}}}\right)$ $+\frac{3}{2}Nk_{\mathrm{B}}\left(T_{\mathrm{c}}-T_{\mathrm{i}}\right)$
Isochoric Heating	0	$\frac{3}{2}Nk_{\mathrm{B}}\left(T_{\mathrm{h}}-T_{\mathrm{i}}\right)$
Isochoric Cooling	0	$\frac{3}{2}Nk_{\mathrm{B}}\left(T_{\mathrm{c}}-T_{\mathrm{i}}\right)$

Upon undertaking any action $s_t \to s_{t+1}$, we record the resulting changes of work, $\Delta W_{s_t s_{t+1}}$, and heat input from the hot reservoir,

$$\Delta Q^{\mathrm{in}}_{s_t s_{t+1}} = \Delta Q_{s_t s_{t+1}} H(\Delta Q_{s_t s_{t+1}} \delta_{T_{\mathrm{f}},T_{\mathrm{h}}}); \tag{4.1}$$

these are listed in Table 4.1. Here $H(\cdot)$ is the Heaviside function, equal to 1 for positive values of $\Delta Q_{s_t s_{t+1}}$ and 0 otherwise, and T_{f} is the temperature of the system

following the move. $\delta_{\alpha,\beta}$ is the Kronecker delta (1 if $\alpha = \beta$ and 0 otherwise). We define the thermodynamic efficiency of a K-step trajectory as

$$\eta_K \equiv \frac{\sum_{t=0}^{K-1} \Delta W_{s_t s_{t+1}}}{\sum_{t=0}^{K-1} \Delta Q^{\text{in}}_{s_t s_{t+1}}}. \tag{4.2}$$

The thermal efficiency, a trajectory-based quantity, is used as a means of ranking trajectories, and the networks that generate them, during our evolutionary learning procedure. The neural network selects processes deterministically based on observations, therefore once a trajectory produces a single cycle, that cycle will repeat until the maximum number of trajectory steps have occurred. For this reason, the maximum value $\eta = \max_K \eta_K$ for all K points along a long trajectory is sufficient to identify efficient thermodynamic cycles. We could terminate a trajectory after it produces a single cycle, however we chose this way for consistency with the gradient-based RL method used where we do require multiple cycles. The phase plots (volume vs pressure) and thermal efficiency as a function of the number of cycles performed for the Carnot, Stirling, and Otto cycles are shown in Figure 4.2.

4.1.1 Formalization as a Generalized GOMDP

The model heat engine can be understood within the generalized GOMDP framework developed in Chapter 3. The state space $\mathcal{S}$ consists of the thermodynamic coordinates (V, T) (or (V, T, W^*, Q^*) in the MDP formulation), forming a compact subset of $\mathbb{R}^2$ (or $\mathbb{R}^4$). The goal-states $\mathcal{G}$ correspond to states where a complete thermodynamic cycle has been executed—that is, states (V_0, T_0) that return the system to its initial configuration after a sequence of actions. The reward structure satisfies the generalized GOMDP conditions: intermediate actions incur non-positive rewards (representing the thermodynamic cost of each process), while the completion of an

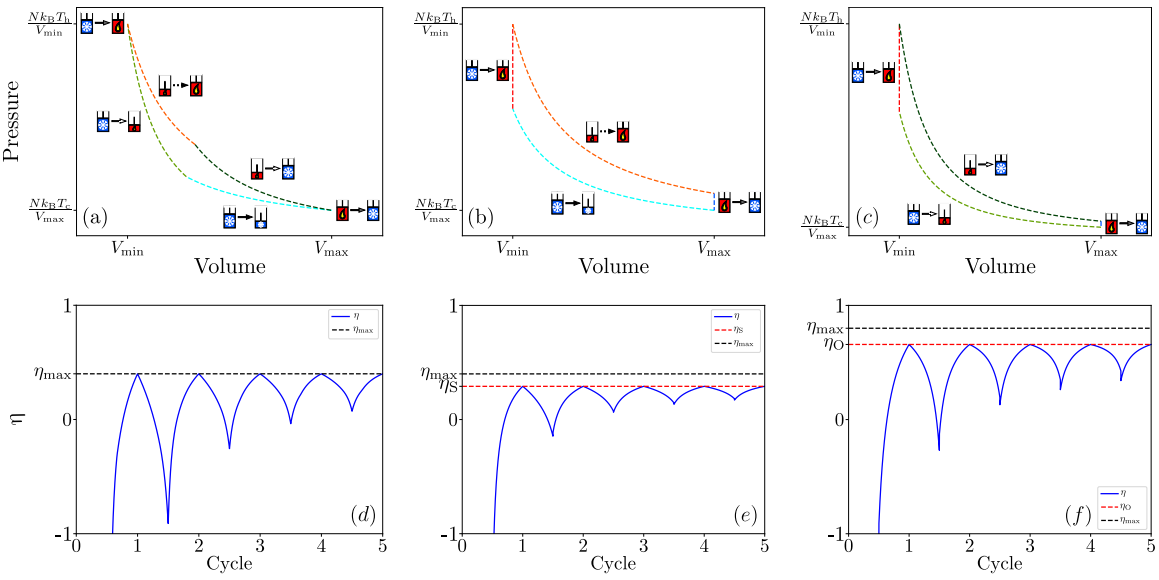


Figure 4.2: The phase plots of a heat engine as it performs the (a) Carnot, (b) Stirling, and (c) Otto cycles with each action being taken labelled with its graphical representation and a different colour. The thermal efficiency of a heat engine as it performs the (d) Carnot, (e) Stirling, and (f) Otto cycles several times with $\eta_{\max}$, η_{S}, and η_{O} for reference.

efficient cycle yields positive reward through the accumulated work output.

The phase space topology of this problem is particularly instructive. Different thermodynamic cycles—Carnot, Stirling, Otto, and others—represent distinct classes of closed trajectories in the (P, V) or (V, T) phase space. From the perspective of path homotopy (Section 3.3), these cycles can be viewed as representatives of different homotopy classes of closed paths, where continuous deformations of one cycle type into another are constrained by the available thermodynamic processes. For instance, the Carnot cycle requires specific adiabatic transitions that are not part of the Stirling cycle, meaning that small perturbations to a Stirling-following policy cannot continuously deform it into a Carnot-following policy without fundamentally changing the action selection at certain states.

This topological perspective helps explain the learning dynamics observed in subsequent sections: an agent that has converged to a suboptimal cycle type (such as

Stirling) faces a structural barrier to discovering the optimal cycle (Carnot), as the transition requires a non-local change in policy space rather than gradual refinement.

4.2 Evolutionary Learning Dynamics

For the evolutionary learning dynamics, we start with a population of 100 networks, with the internal parameters θ of each initialized in the random fashion described above, named generation 1. This population produces thermodynamic trajectories ω of K steps. Even the best-performing members of this population produce efficiencies much lower than the Carnot efficiency, which is the most efficient trajectory possible given the set of allowed thermodynamic processes [65]. For the chosen parameters, we have the

$$\eta_{\max} = 1 - \frac{T_{\mathrm{c}}}{T_{\mathrm{h}}} = 0.4, \tag{4.3}$$

as the Carnot efficiency.

The next step of evolutionary learning dynamics is done by keeping the 25 first generation networks whose trajectories have the largest η, discarding the rest. We create 75 new networks by drawing uniformly from the set of 25, each time "mutating" all weights w and biases b: for each weight or bias we draw a random number δ from a Gaussian distribution with zero mean and unit variance, and update the weight or bias as $w \to w + \epsilon\delta$ or $b \to b + \epsilon\delta$, where $\epsilon = 0.05$ is an evolutionary learning rate.

The new population of the 25 best generation-1 networks and their 75 mutant offspring constitute generation 2. We simulate those 100 networks for K steps as before. Continuing this alternation of evolutionary dynamics (retaining and mutating the best networks of the current generation) and physical dynamics (using the new generation of networks to generate a set of trajectories) gives rise to networks able to propagate increasingly efficient trajectories. After about 100 generations, we obtain

networks whose efficiencies are equal to that of the Carnot cycle (to within four decimal places). A quantitative inspection of the trajectories corresponding to these values of η show that they indeed form Carnot cycles.

4.3 Gradient-Based RL

As an alternative to the gradient-free evolutionary learning, we now consider a method which includes gradient information. We note however, that while the observation $s_t = (V, T)$ contains sufficient information to correctly determine the required thermodynamic process at any step t, as shown above, it is not sufficient for determining thermal efficiency. For example, suppose the Carnot cycle requires K steps with $s_0 = (V_{\max}, T_{\mathrm{c}})$. K steps later we arrive back at the same observation, $s_K = s_0$, achieving a thermal efficiency of $\eta_{\max}$. Now if we perform the reverse Carnot cycle, we would begin and end at the same observations as before, however this time achieving a thermal efficiency of $-\eta_{\max}$, meaning that the relation of the current observation representation to thermal efficiency is one-to-many. Unlike gradient-free algorithms, traditional gradient-based RL algorithms operate using the rewards associated with specific steps of a trajectory, instead of assigning a score to an entire trajectory. For this reason, these algorithms cannot practically be used on the specific simulated heat engine above because it is not an MDP. Alternatively, a learning algorithm could be adapted for for the partially observable setting used above, however the focus here is not on the algorithm, but the environment itself.

4.3.1 MDP Model Heat Engine

In this new formulation of the problem, which will be referred to as the MDP heat engine problem, we provide the agent with non-negative work and heat budgets,

defined as $W_K^* = W_0^* + \sum_{t=0}^{K-1} \Delta W_{s_t s_{t+1}}$ and $Q_K^* = Q_0^* - \sum_{t=0}^{K-1} \Delta Q^{\text{in}}_{s_t s_{t+1}}$ respectively, where W_0^* and Q_0^* are the initial budget values. For a thermodynamic process to be performed at time t, $W_{t-1}^* \geq -\Delta W_{s_{t-1} s_t}$ and $Q_{t-1}^* \geq Q^{\text{in}}_{s_{t-1} s_t}$ are required to ensure W_t^* and Q_t^* are non-negative. These budgets represent the amount of work the agent can put into the system, where any work produced during a trajectory is added to the budget, and the amount of heat that can be transferred from the hot thermal reservoir to the heat engine. We define the observation for this new heat engine as the state $s_t = (V, T, W_t^*, Q_t^*)$, however we still consider a cycle as a trajectory that returns the same values of V and T. With this new representation, the states at the beginning and end of a given singular cycle are now unique, unlike the previous observations. This means that a given observation contains all the required information about the state of the simulated heat engine and therefore the previous trajectory is not required. Full Markovity is not necessarily required to apply gradient-based reinforcement learning, however it does ensure the mapping from state–action pairs to rewards can be learned by the agent. As W_0^* and Q_0^* are fixed, the relation of state to thermal efficiency is now many-to-one, solving the issue of mapping an state to a single thermal efficiency.

As the states for the start of each repeated cycle are unique, if a trajectory produces one cycle, it does not guarantee it will produce many cycles. This poses a greater challenge as each subsequent cycle must now be learned individually. This could be solved by re-initializing the system to the initial state after a cycle is performed, however this would force a solution for single cycles where we would like to find a more general solution for the system. Thermal efficiency could be assigned to the final state-action pair of a trajectory to update a given policy using gradients in order to produce the most thermally efficient cycle, similar to what was done in the previous case, however this does not encourage trajectories containing more than a single cycle as thermal efficiency does not change when a cycle is repeated exactly.

To avoid explicitly penalizing a trajectory for stopping after a single cycle, we instead use $\Delta W = W_K^* - W_0^*$ at the end of a trajectory. As $\Delta Q = Q_0^* - Q_K^*$ has a fixed finite maximum, using Equation 4.2, maximizing ΔW maximizes thermal efficiency while also encouraging trajectories with many cycles, satisfying our desired requirements. To help clarify this, consider the following example. Suppose some cycle on this heat engine produces 0.2 units of work and consumes 0.5 units of heat. This would give a thermal efficiency of 0.4. Now if we performed this cycle 10 times we would produce 2.0 units of work and consume 5.0 units of heat. However this would still give a thermal efficiency of 0.4. If we use thermal efficiency as the reward, these two cases are equivalent, however if we use ΔW as the reward, the second case is preferred.

4.3.2 Proximal Policy Optimization

With our newly defined environment, we turn to PPO [43] as the gradient-based RL algorithm to find a policy, π_θ, that produces the desired trajectories. PPO is a commonly used gradient-based RL algorithm which has had good performance on many problems [67–71]. The true policy, π_θ^*, requires certain features from the system in order to produce these desired trajectories. While the state may not explicitly contain these features, we assume we can extract them using a neural network approximation of π_θ^*. PPO aims to maximize the policy gradient objective function,

$$L_t^{\mathrm{PG}}(\theta) = \hat{\mathbb{E}}\left[\log \pi_\theta(a_t|s_t)\, \hat{A}_t\right], \tag{4.4}$$

where the expectation is taken with respect to the actions a_t and $\hat{A}_t = \hat{A}_\theta(s_t|a_t)$ is the estimate of the advantage function defined as the difference between the Q-function $\mathcal{Q}_{\pi_\theta}(s_t, a_t)$ and the value function $V_{\pi_\theta}(s_t)$. The Q-function is the expected discounted

future reward if the process a_t is performed starting at s_t and π_θ is followed for the remainder of the trajectory starting at s_{t+1}, defined as

$$\mathcal{Q}_{\pi_\theta}(s_t, a_t) = r_t + \gamma \max_a \mathcal{Q}_{\pi_\theta}(s_{t+1}, a), \tag{4.5}$$

where $0 \leq \gamma \leq 1$ is the discount factor. The value function is the expected discounted future reward if π_θ is followed for the remainder of the trajectory starting at s_t, defined as

$$V_{\pi_\theta}(s_t) = \max_a \mathcal{Q}_{\pi_\theta}(s_t, a). \tag{4.6}$$

Our estimate of future reward is rarely perfect, therefore it is discounted relative to the time scale of the reward. For example, when deciding on where to eat, we only care about the immediate reward we get, so we heavily discount our estimate of future reward in this case, however when investing in stocks, we care about the long-term reward, so our estimate of future reward should be minimally discounted. $\mathcal{Q}_{\pi_\theta}$ and V_{π_θ} are not known analytically but it is assumed they require the same features as π_θ and therefore we have our same neural network approximation of π_θ output the value at s_t, i.e. the output of the neural network is $(\pi_\theta(s_t), V_{\pi_\theta}(s_t))$. However, it is more efficient to instead maximize the trust region policy optimization [72] objective function,

$$\begin{aligned} L_t^{\text{CPI}}(\theta) &= \hat{\mathbb{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \\ &= \hat{\mathbb{E}}\left[r_{\theta,\theta_{\text{old}}}(a_t|s_t)\,\hat{A}_t\right], \end{aligned} \tag{4.7}$$

where $\pi_{\theta_{\text{old}}}$ is the set of parameters of the previous policy. The trivial solution to this is to make extreme modifications to π_θ, therefore PPO clips the probability ratio

$r_{\theta,\theta_{\text{old}}}$ and takes the minimum,

$$\begin{aligned} L_t^{\text{CLIP}}(\theta) &= \hat{\mathbb{E}}\left[\min\left(r_\theta(a_t|s_t)\hat{A}_t, r_\theta^{\text{CLIP}}(a_t|s_t)\hat{A}_t\right)\right] \\ r_{\theta,\theta_{\text{old}}}^{\text{CLIP}}(a_t|s_t) &= \text{clip}\left(r_{\theta,\theta_{\text{old}}}(a_t|s_t), 1-\epsilon, 1+\epsilon\right), \end{aligned} \tag{4.8}$$

where ϵ is a hyperparameter and the clip function is defined as

$$\text{clip}(x,a,b) = \min(\max(x,a),b) = \max(\min(x,b),a). \tag{4.9}$$

While this objective function is designed to optimize the policy, it is not designed to optimize the value function, therefore we use a mean squared error cost function,

$$L_t^{\text{VF}}(\theta) = \left(V_{\pi_\theta}(s_t) - V_t^{\text{targ}}\right)^2, \tag{4.10}$$

where V_t^{targ} is the target value. Combing this cost function with our objective function and adding in a so called "entropy" term S to encourage exploration, we have

$$L_t^{\text{PPO}}(\theta) = \hat{\mathbb{E}}\left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)\right], \tag{4.11}$$

for our overall PPO objective function, where c_1 and c_2 are hyper-parameters and $S[\pi_\theta]$ is defined by

$$S[\pi_\theta](s_t) = \sum_a \pi_\theta(a_t|s_t)\log\left(\pi_\theta(a_t|s_t)\right). \tag{4.12}$$

Note that we have the negative of our cost term, so maximizing L_t^{PPO} simultaneously maximizes our objective function and minimizes our cost function. To update the parameters of our neural network, the gradient of L_t^{PPO} is estimated with respect to θ using backpropagation and we make changes to θ in the ascending direction of this

gradient.

4.3.3 Results and Discussion

Using the identical network architecture as with the gradient-free method, we randomly initialize a policy and allow it to produce trajectories in the newly-defined thermodynamic system. As the policy interacts with this system, we collect tuples of the form (s_t, a_t, r_t, s_{t+1}) which are the experiences used to update the policy with PPO. We require all four of these because the reward r_t specifically arises from how the action a_t maps the state s_t to the next state s_{t+1}. r_t and s_{t+1} are also required to update our approximation of the value function and calculate $\hat{A}_t$. To perform these updates, we used the following hyper-parameters: discount factor of 0.99, $\epsilon = 0.2$, $c_1 = 0.5$, $c_2 = 0.01$, a batch size of 128, a total number of training steps of 2×10^6, and a learning rate of 2.5×10^{-4} which determines the step size when updating the parameters θ.

Unlike the gradient-free approach previously discussed, PPO is unable to achieve an near-optimal result. However the agent was able to produce a trajectory that not only follows the Stirling cycle as shown in Figure 4.3(a), and does so more than once. When looking at the learning dynamics shown in Figure 4.3(b), we can see that the policy that produces the most thermally efficient trajectory occurs at $\sim 2.5 \times 10^5$ training steps, and then thermal efficiency steadily decreases in all subsequent policies. This is likely due to the Stirling cycle being a local maximum (in terms of ΔW). When a policy is at a local maximum for long enough, the value function is optimized on the trajectories found at this local maximum to the point that it becomes useless when evaluated on any trajectories found outside of the local maximum. Due to the deterministic nature of these problems, only a single trajectory is seen at this local maximum. Eventually, the entropy term used for stochastic exploration forces the

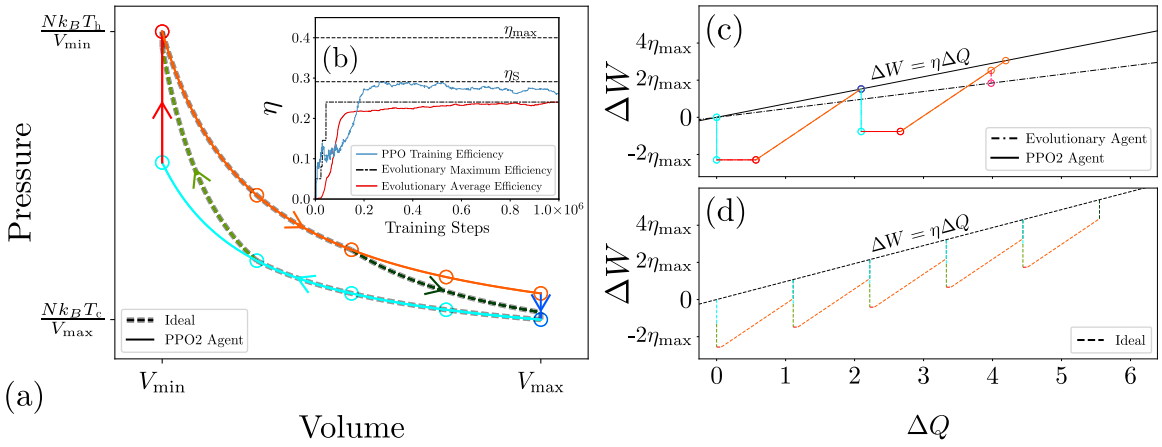


Figure 4.3: We apply the PPO reinforcement learning process to our Markovian simulated heat engine; panel (a) shows the most thermally efficient trajectory produced in this process overlaid with the ideal solution, the discretized Carnot cycle. (b) The learning dynamics of thermal efficiency as a function of training steps for both the PPO and evolutionary agents as they learn to maximize ΔW. (c) ΔW as a function of ΔQ for the trajectory produced by the best performing PPO and evolutionary agents and (d) the ideal solution (discretized Carnot cycle), where $\Delta Q = 1$ maps to the amount of energy used during a single analytic Carnot cycle.

policy to stray from this locally optimal trajectory, causing the policy to produce a trajectory unseen by the agent recently. As the value function is no longer accurate at this state, the agent is unable to bring the policy back to this local maximum causing performance to diminish. The Stirling cycle is a local maximum because it produces more work than the Carnot cycle, i.e. achieves a higher reward per cycle, however as seen when comparing Figures 4.3(c) and (d), the Stirling cycle consumes a greater amount of heat than the Carnot cycle. This makes it more viable in the short term, however, due to the upper bound constraint placed on ΔQ, the Carnot cycle is still the ideal solution when tasked with maximizing ΔW.

With the maximum amount of heat allowed, the only way to increase ΔW beyond what is produced by two Stirling cycles, a trajectory would need to instead produce either three Carnot cycles and one Stirling cycle or five Carnot cycles. The differences between these trajectories and the one the agent produces are not trivial. Making

any slight modifications to the found trajectory decreases ΔW. It is only when an entire Stirling cycle is replaced with multiple Carnot cycles in a trajectory that there would be any improvement. Additionally, the reward signal used to update a policy comes from the end of a trajectory, whereas due to the repetitive nature of the partially observable problem, we were able to select the maximal score along the entire trajectory. Now the policy is required to produce trajectories that not only follow optimal paths but also end in optimal positions, increasing the difficulty of the fully observable problem even further.

Remark (Topological Interpretation of the Stirling Trap)**.** *From the perspective of the generalized GOMDP framework (Section 4.1.1), the Stirling cycle represents a local optimum within its homotopy class of trajectories. The agent's inability to escape this local maximum can be understood as a fundamental topological barrier: transforming a Stirling-following policy into a Carnot-following policy requires not a gradual refinement, but a discrete "jump" between homotopy classes. This is because the two cycle types traverse fundamentally different regions of the phase space—the Carnot cycle requires adiabatic transitions that are absent from the Stirling cycle. The sample complexity required to discover this transition is related to the probability of exploratory actions generating trajectories in the Carnot homotopy class, which is low when the policy has already converged to the Stirling class.*

4.4 Evolutionary Learning on MDP Engine

If ΔW is used as the scoring metric, the evolutionary learning method from before can also be applied to this variant of the simulated heat engine. Doing so yields trajectories with *lower* thermal efficiencies and less work produced compared with the gradient-based method as shown in Figures 4.3(b) and (c). Similar to the gradient-

based method, the gradient-free method starts it's trajectory by producing a Stirling cycle, however it is unable to complete the second cycle. While the desired trajectory is identical to that of the partially observable problem, this task is a more challenging one because the cycle must be learned over and over again as the heat budget is consumed. This shows that the evolutionary learning method is not necessarily superior to the gradient-based method in all ways, but rather each approach is suited for different tasks.

The gradient-based method was able to outperform the evolutionary learning method on the fully observable problem, however, it was the evolutionary learning method that was able to find the maximally efficient trajectory when applied to the partially observable problem. The evolutionary learning method was able to be applied to both tasks, however, because of the nature of gradient-based RL, we could only apply it to the fully observable problem. The fact the gradient-based method finds this local maximum while the gradient-free does not supports the idea that gradient-based RL can propagate the reward assigned to specific state-action pairs in order to produce at least a partially optimal solution. Evolutionary learning methods cannot straightforwardly make use of these specific assignments and this is likely the reason why it is outperformed by the gradient-based learning method on the fully observable problem. Although the evolutionary learning method seems to be the go-to method, being able to define a problem the way we did for the partially observable one is not always possible. It would be possible to make the fully observable problem easier for the gradient-based learning method, but not without providing such a high level of feedback that the problem becomes pointless to solve. For example, one could reward the gradient-based learning agent as it performs each of the correct steps in the most thermally efficient trajectory, however, this approach would only work in the case an optimal solution was known beforehand. Novelty search techniques might

allow this gradient-based learning method to find the optimal trajectory, however, this would increase the computational cost of this method even further. Given that gradient-based methods are already more computationally expensive than gradient-free methods, and that the gradient-free method has already achieved the optimal result, we have chosen not to explore these options.

As mentioned before, the most notable difference between the partially observable and fully observable systems set-up is representation of the state and cycles. In the partially observable case, observations were 2-dimensional, with cycles consisting of those repetitions in those same dimensions. This forces an implicit bias for policies that only these dimensions are needed for following trajectories. In the fully observable case, the observations were the entire 4-dimensional state, whereas cycles only consist of repetitions in two of the dimensions but not the other two, increasing the number of observable trajectories.

4.5 Conclusions

Motivated by the correspondence between games and physical trajectories, we have shown that neural network-based evolutionary learning can optimize the efficiency of trajectories of classical thermodynamics. Given a set of physical processes and a path-extensive measure of efficiency, networks evolve to learn the maximally-efficient Carnot cycle, reproducing classic results of physics that were originally derived by application of physical insight. Given new processes, the evolutionary framework identifies solutions that when interrogated provide physical insight into the problem at hand. We have also shown that with slight modifications, gradient-based RL can optimize the efficiency of trajectories of classical thermodynamics, although not quite as effectively.

The experiments in this chapter illustrate how the organization of a decision problem's underlying structure shapes learning behaviour, particularly when the agent's observations reveal only part of that structure. In the fully observable setting, the agent can exploit the true arrangement of feasible trajectories and state transitions, much like the explicit path relationships examined in Chapter 3. When the same system is viewed through limited or noisy observations, however, the agent effectively navigates an obscured structural landscape, in which the underlying routes to high-performing behaviour are harder to distinguish. The differences between the MDP and POMDP cases highlight how access to structural information—and the clarity with which goal-directed trajectories can be inferred—affects the dynamics of learning. This relationship between observable organization and agent performance forms a conceptual bridge to the next chapters, where uncertainty and partial information play an even more prominent role. The concept of uncertainty in observations and its relationship to learning difficulty is analyzed further in Chapter 5.

Chapter 5

Nautical Navigation

This chapter focuses on a nautical navigation-themed generalized GOMDP [3], where the navigational space consists of a varying number of holes—represented as islands, composed of inaccessible states—creating maps of varying topological complexity. The problem consists of two components, an underlying MDP and a partially observable addition. The aim is to leverage the underlying MDP to increase performance on the POMDP. Much of this chapter is adapted from the original work that has been previously published [3], first authored by C. Beeler. The material has been revised for coherence and consistency within the overall structure of this thesis.

5.1 Introduction

Uncertainty creates a major obstacle in solving control problems. The goal of these problems is to construct a policy that is expected to produce optimal trajectories. In some cases, uncertainty causes small deviations from the optimal trajectory, which are nevertheless still acceptable solutions. For example, if a driver is uncertain of exactly *which* road they are on, they might deviate from the optimal route to their

destination; however, they can still arrive via a less optimal route. In other cases, uncertainty can lead to highly undesired results. With the previous example, if a driver is instead uncertain of *where* they are on the road, this can result in a collision, which we refer to as a catastrophic failure. Even if these deviations are symmetric in nature, catastrophic failure could be the most likely result.

These distinctions between benign and catastrophic uncertainty are central to the sequential decision problems considered in this thesis. In the fully observable setting, such problems take the form of MDPs [54], which have been widely analyzed in both DP [14,73–75] and RL [6] (both traditional [49,76,77] and deep [7,78,79]). While the majority of MDP results are simulated, there are real-world applications. The Airborne Collision Avoidance System X [5]—as discussed in Section 2.4.6—uses methods of solving POMDPs with DP to aid actual operating aircraft to avoid collisions in real-time, using a distribution of estimates for the state of the surrounding aircraft. We will study problems like this through the formalism of POMDPs [55], which were defined in Section 2.4, and will use to present a modified version of Bellman's DP equations, Equation 5.3. While POMDPs are also well studied in DP [80–84], it is only more recently that they have been studied in RL [85–88].

In an MDP, the state of the system is known, however, in a POMDP it must be estimated, leading to some amount of uncertainty. Much of the difficulty in solving a POMDP stems from estimating the state of the system before choosing an action. This is where the majority of research in this area focuses. Controlled sensing problems are a special type of POMDP where some of the actions reduce uncertainty for a cost, rather than modifying the state of the system. Some work has been done in this area [55,89–93], however, it is still largely unexplored.

Separate from the question of the partial observation of the current state is knowledge of the environment itself, i.e. the space of all possible states and how the available

actions cause transitions between them. Depending on how much of the system's information is available to the agent, different approaches are possible to optimize agent behaviour. DP methods require full knowledge of the environment and thus amount strictly to optimization, without "learning" per se. At the other end of the spectrum, RL methods assume little or no access to information about the system; they involve learning from experience to deduce which actions have the most desirable effects. In this work, we consider POMDPs whose underlying MDP is fully known to the agent. The MDP setting allows for analytic solutions via DP, and we propose a method to adapt such solutions to the related POMDP where the agent must contend with uncertainty regarding its current state. While purely RL methods could be used instead, they would not take into account the agent's knowledge of the MDP. Our work thus fills a gap, providing POMDP solutions in a DP-grounded rather than RL-grounded approach. In particular, the settings we consider include the areas of controlled sensing and traditional POMDPs.

The systems that this problem structure applies to include, but are not limited to: navigation [5], healthcare [94], and even chemical experiments [4]. In a chemical experiment, there are many variables to consider and even slight variations in them can change the outcome of a reaction. While a chemist can record every step they have made throughout an experiment, there will always be variations in the outcome. The only way to determine this variation is to take various measurements, each with an associated cost. Hence the problem of optimally performing an experiment while managing access to various measurements is located in the combined space of traditional and controlled sensing POMDPs.

Nautical navigation has been the subject of several DP studies [95–97], however, the primary focus has been on collision avoidance and route optimization (i.e. speed and fuel consumption) rather than uncertainty and controlled sensing. Here we intro-

duce a nautical navigation environment described in detail in Section 5.2. We assume the agent has incomplete access to the information of the system, which leads to a level of uncertainty. A set of information-revealing actions (or measurements) are accessible that help reduce uncertainty at a cost. In a controlled sensing POMDP, the state transition matrix is typically independent of the chosen action, but the observation and reward functions may not be. Note that measurement actions that can be taken without the state changing, cause the transition matrix to become the identity for that action. This is equivalent to time not advancing during this step.

The methods outlined in Section 2.4.5 for solving POMDPs use DP and assume the agent has complete access to each element of the POMDP septuple. When solving a POMDP with RL, the main difference from the DP solutions is that it is assumed the state transition dynamics, state-to-observation mapping, and reward functions are not available to the agent and must be learned. However, if the agent has incomplete access to this information, it is ignored in RL algorithms. In the next section, we present a novel environment in which the agent has incomplete access to information and controlled sensing actions.

The main contributions we present in this chapter are: a novel nautical navigation environment that allows for the control of the level of information and can be generalized to many fields, a modified version of Bellman's DP equations, Equation 5.3, and policy construction for POMDPs with incomplete information, as well as POMDP solutions that combine state altering actions with controlled sensing techniques that outperform the baseline of non-adapted DP solution for the underlying MDP.

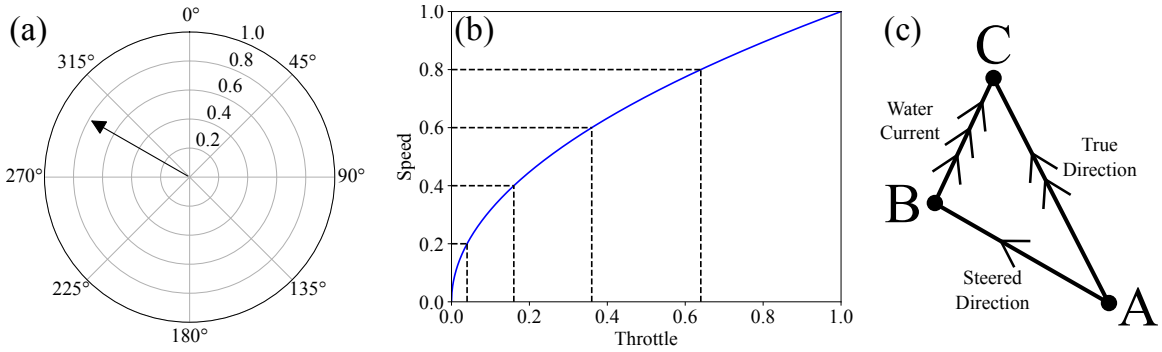


Figure 5.1: (a) A graphical representation of the non-measurement action space the agent can choose from at each step. (b) The relationship of throttle and speed through the water where the black dashed lines represent the throttle discretization used in Section 5.4. (c) A graphical representation of how the non-measurement actions map to the submarine's displacement. **A** represents the starting position of the submarine. **B** represents the estimated final position of the submarine based on the non-measurement action chosen by the agent (or velocity through the water) represented by the single arrow line. **C** represents the true final position of the submarine where the true path represented by the double arrow line is the combination of **B** and the water current represented by the triple arrow line (or velocity overland).

5.2 Nautical Navigation Environment

To explore the concept of incomplete access to information in POMDPs, we introduce a nautical navigation environment, that serves as an easy-to-understand and very generalizable system for our discussion. In this environment, the agent must navigate a submarine through a set of islands to a specified circular target region. To navigate, the agent must specify a heading and throttle setting that provides a movement vector, shown in Figure 5.1(a). Typically there is a non-linear relation between throttle and speed. In this case, speed through the water is the square root of throttle, as shown in Figure 5.1(b). An RL agent would have to learn this relationship, however, in our case, it can be included in our agent's information access setup. If the agent reaches the target region, it receives a reward (also known as a negative cost) and if the agent crashes into an island, it receives a large cost (negative reward). The trajectory

terminates when either of these cases occurs.

This system also contains water currents that cause drifts from the expected trajectory of the specified movement vector. Together, the movement vector and water current give the velocity of the submarine over land, defining how the state of the system changes. Unlike the set of island obstacles, the exact water current is assumed to be unknown by the agent, (but it can be partially observed indirectly). If the agent knows the movement vector chosen and their true position before and after an action, the average water current over that action can be obtained from the displacement between the expected and true final positions, as shown in Figure 5.1(c).

The unknown water current gives rise to a level of uncertainty in the movement of the submarine, which in turn, gives rise to a level of uncertainty in the resulting position. The agent has two measurement actions available to it, and can use them to help overcome these uncertainties:

1. **GPS**: Returns the true position of the submarine, therefore reducing positional uncertainty to zero. This allows for the calculation of the average water current between the previous and present GPS measurements. Hence, this measurement slightly reduces the water current uncertainty, although not completely.

2. **Current Profiler**: Returns the true water current for the true position of the submarine, therefore reducing the water current uncertainty to zero. Note that because the analytic water current is unknown, this measurement does not reveal any information regarding the position of the submarine. Hence, the positional uncertainty is unaffected.

Note that for these measurement actions, $P(a) = I$ and $r_{\mathrm{m}}(a) \equiv$ const. where the constant is some specific instantaneous measurement cost (negative reward) assigned for employing that measurement and $r_{\mathrm{m}}(a) \equiv 0$ for all non-measurement actions.

These costs represent both the monetary costs of using and maintaining each device and the time required to operate them.

Charts: The system the agent is navigating in is contained inside a rectangular area with periodic boundary conditions and dimensions $x_{\max}$ and $y_{\max}$. We call the pairing of this area with the set of islands a chart. Each island obstacle is represented by a 2-dimensional Gaussian function where the parameters are independently sampled uniformly to ensure convergence. We then define the land height function $f(x, y)$ as the summation over several islands. The notation for outputs of $f(x, y)$ used here are: 0 is the ocean floor, 1 is sea level, and 0.9 is the height at which the submarine operates, i.e. the agent navigating to any point (x, y) such that $f(x, y) \geq 0.9$ results in a crash. During a trajectory, the agent always has access to the charts.

Water Currents: While it is assumed the agent does not know the analytic water current, it is generated deterministically for each given land function. The water current vector $W(x, y)$ at (x, y) is perpendicular to $\nabla f(x, y)$ with magnitude bounded by $w_{\max}$ and linearly related to $-\|\nabla f(x, y)\|_2$.

Remark (GOMDP Conditions and Topological Complexity)**.** *The nautical navigation environment satisfies the conditions for a generalized GOMDP as defined in Chapter 3: (1) the goal-states $\mathcal{G}$ consist of all positions within the target region; (2) rewards are non-positive during navigation (fuel costs, measurement costs) except for reaching the target; (3) the continuous state space $\mathcal{S}$ with island obstacles creates non-trivial topology.*

Furthermore, the relationship between island count and topological complexity can be made precise. Each island creates a hole in the navigation space, and the number of distinct homotopy classes of paths between a starting position and target grows with the number of such holes. In the simplest case, N non-overlapping islands in a navigation region can create up to $O(N!)$ distinct homotopy classes of paths (depending on the

relative positions of start, goal, and obstacles), though in practice the number of viable path classes is typically closer to $O(N)$ for well-separated obstacles. This scaling explains the empirical results presented later: as island density increases, both the topological complexity and the sample complexity of distinguishing among path classes increase, leading to decreased success rates and increased crash rates.

5.3 Finding an Optimal Policy

With a navigation environment defined, we can now use it as an example of how to develop a policy construction method. As the agent does not have complete knowledge of the system, Bellman's DP algorithms presented throughout Chapter 2 cannot be used directly. In this system, if there are no water currents, or if the agent knows the water currents exactly, the problem becomes an MDP, and Bellman's equation is applicable. As we assume the agent does *not* know the water current, we turn to the former to be the base model for constructing a solution. In the next section, we present how to form this base.

5.3.1 Value Function

During a single trajectory, $f(x, y)$ and $W(x, y)$ do not change, therefore for simplicity, we refer to the submarine position (x, y) as the state of the system. The velocity of the submarine need not be included as we assume the time scale of acceleration and changing directions is insignificant relative to the time between actions. In the first step in constructing our solution, we assume the system contains no water current, i.e. $W(x, y) \equiv 0$. With this assumption, for any given chart we can generate a value function using Bellman's equation, where $\|M\|_2 \leq 1$. To encourage faster routes, we introduce a fuel cost defined as $r_{\text{f}}(a) = -0.01\|M\|_2$ for all non-measurement actions.

We define the positional reward function as $r_{\mathrm{p}}(x, y) = -100$ for any (x, y) such that $f(x, y) \geq 0.9$, $r_{\mathrm{p}}(x, y) = 1$ for any resulting submarine positions (x, y) inside the specified target region such that $f(x, y) < 0.9$, and $r_{\mathrm{p}}(x, y) = 0$ otherwise. This gives us the reward function $r(x, y, a) = r_{\mathrm{p}}(x, y) + r_{\mathrm{f}}(a) + r_{\mathrm{m}}(a)$, where $a \in \mathcal{A}$ consists of a non-measurement component a_M and a component measurement action. Note that the trajectory terminates when $r_{\mathrm{p}}(x, y) \neq 0$.

With the water current and reward function formally defined, we can now define the movement of the submarine at any given time. For a specified movement vector M such that $\|M\|_2 \leq 1$, the movement of the submarine is given by $d(x, y, W, M, t) = (x, y) + t(M + W(d(x, y, M, t)))$. The non-measurement action corresponding to M is then defined as $a_M(x, y, W) = d(x, y, W, M, t')$, where

$$t' = \sup\{t \in [0, 1] | r_{\mathrm{p}}(d(x, y, W, M, t)) > -100\}. \tag{5.1}$$

Note that $t' = 1$ only occurs if the agent does not crash into an island during the action.

For any chart, with or without water currents, we have $V_0 \equiv 0$ and $V_1(x, y) = \max_{a \in \mathcal{A}} r(x, y, a)$. Examples of a chart without and with water currents are shown in Figures 5.2(a) and 5.2(b) with V_1 for each case shown in Figures 5.2(c) and 5.2(d) respectively. If the water current is known, Bellman's equation for our system becomes

$$V_n(x, y) = \max_{a \in \mathcal{A}} r(x, y, a) + \gamma V_{n-1}(a_M(x, y, W)), \tag{5.2}$$

where $W \equiv 0$ for Figure 5.2(e) and $W = W(x, y)$ for Figure 5.2(f). In this system, 5.2 results in a converged value function V after finite n. The converged value functions for the examples above are shown in Figures 5.2(e) and 5.2(f) respectively, with three regions of notable difference between the two circled. As we assume the agent does

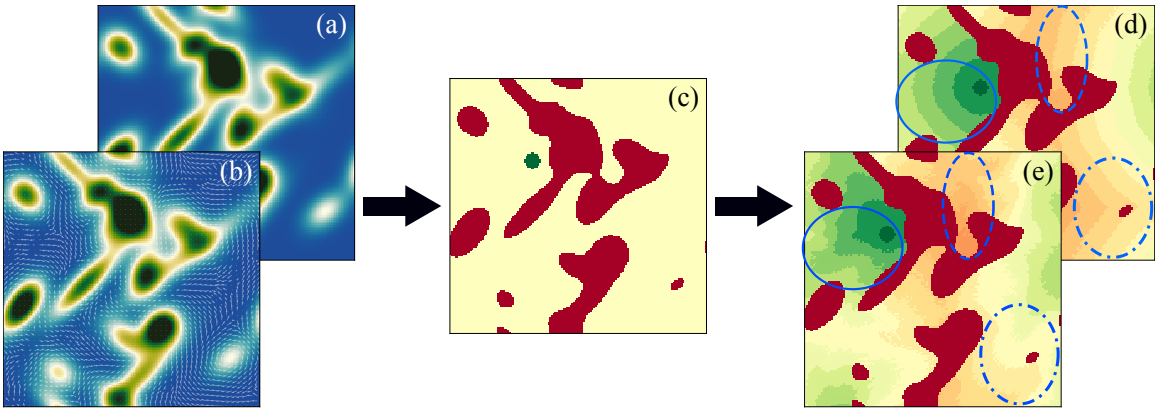


Figure 5.2: A graphical representation of the process of generating a value function without and with water currents. (a) Shows a chart without water currents and (b) shows the same chart with water currents. (c) Shows V_1 with a specified circular target region for the charts with and without water currents. (d) Shows the optimized value function for the chart without water currents and (e) shows the optimized value function for the chart with water currents. Regions of notable difference between (d) & (e) are highlighted with blue ellipsis, where each line style corresponds to a specific region.

not know $W(x, y)$, we continue with the value functions of the type in Figure 5.2(e) for the next section.

5.3.2 Policy Construction

With the water current unknown, the goal is to construct a policy in a similar manner to the value iteration algorithm for POMDPs in Equation 2.47. Doing so requires using the expected states and uncertainty to determine the agent's belief state. At the initial step of each trajectory, the agent knows the true initial submarine position and water current for that specific position, therefore uncertainty in both is zero. This is the **first case** of four considered. As the water current changes when the submarine moves away from this position, uncertainty in the water current grows during any non-measurement action taken, leading to the growth of uncertainty in the position shown in Figure 5.3(a). With the expected trajectory based on the known position,

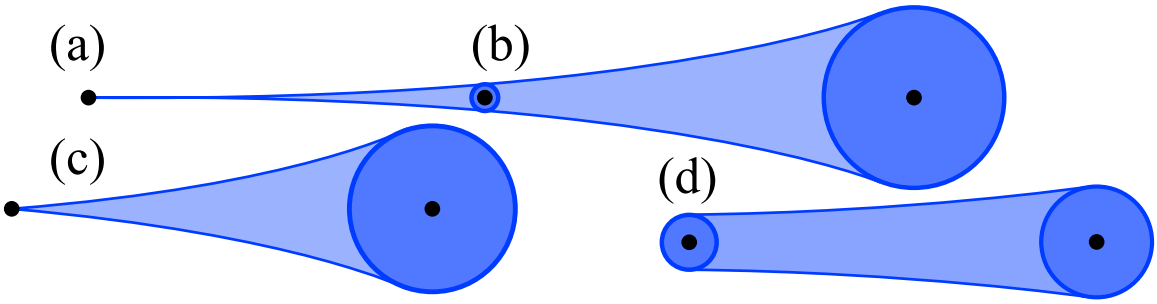


Figure 5.3: A graphical representation of how the positional and water current uncertainties evolve throughout an action where the black dots represent the expected positions and the blue shaded regions represent the uncertainty. The initial conditions are (a) the position and water current are both known, (b) the position and water current are both unknown, (c) the position is known and the water current is unknown, and (d) the position is unknown and the water current is known.

starting water current, and action taken, this gives us a distribution of trajectories that may occur. Therefore each possible non-measurement action can be assigned an expected value and instantaneous reward based on these distributions. Then the policy chooses: select the non-measurement action with the highest expected value (i.e. lowest expected total cost) based on the distribution of trajectories.

During all subsequent steps of the trajectory, the agent has an expected position and water current, however, it also has uncertainty in both these estimates. This is the **second case**. As before, uncertainty in position increases due to uncertainty in the water current growing over time. However, uncertainty in position also increases due to the initial non-zero uncertainty in the water current. This combination leads to the growth of uncertainty in the position shown in Figure 5.3(b), where the initial positional uncertainty is now non-zero. As before, this gives us a distribution of trajectories that may occur. Hence each possible non-measurement action can be assigned an expected value and instantaneous reward. The uncertainty growth rate is the rate the agent's uncertainty of the water current increases over each action. The agent's uncertainty of position increases relative to the uncertainty of the water

current, not just the uncertainty growth rate. The maximum water current magnitude represents the true uncertainty that is present in the system, whereas the uncertainty growth rate represents the uncertainty the agent assumes is present in the system.

As mentioned before, the agent has access to two types of measurements to reduce this uncertainty; each with an associated instantaneous cost. If the lowest expected instantaneous cost of any action is greater than the cost of any of the available measurement actions, the policy chooses to take a measurement. If the expected position of the submarine is inside the target region, the policy chooses to specifically take a GPS measurement. Otherwise, the policy chooses to select the non-measurement action with the lowest expected value based on the distribution of trajectories.

If the GPS measurement is taken, the positional uncertainty goes back to zero and the water current uncertainty is slightly reduced; however, it is still non-zero. This is the **third case**. During any non-measurement action now, the positional uncertainty grows similar to the second case, with however, an initial positional uncertainty of zero, shown in Figure 5.3(c).

If the current profiler measurement is taken, the water current uncertainty goes back to zero and the positional uncertainty is unaffected, therefore still non-zero. This is the **fourth case**. During any non-measurement action now, the positional uncertainty grows similarly to the first case, with however, an initial positional uncertainty of non-zero, shown in Figure 5.3(d).

In either the third or fourth cases, the expected values and instantaneous costs must be re-determined for each non-measurement action. If the lowest expected cost is greater than the cost of the other measurement, that measurement will also be taken, bringing the agent back to the first case. Otherwise, the policy chooses to select the non-measurement action with the lowest expected value based on the new distribution of trajectories.

In this problem we are assuming the agent does not have access to all components of the POMDP tuple, therefore we must replace the hidden Markov model filter with something that incorporates the uncertainty of our system. This gives us the modified Q-function

$$\begin{aligned}\mathcal{Q}(\hat{x},\hat{y},\sigma_p,\hat{W},\sigma_w,a) = \frac{1}{16\sigma_p^2\sigma_w^2}\oiint_{\sigma_p}\oiint_{\sigma_w}\Big(r(\hat{x}+x',\hat{y}+y',a) \\ + \gamma V(a_M(\hat{x}+x',\hat{y}+y',\hat{W}+(w_x,w_y)))\Big)dw_x dw_y dx' dy',\end{aligned} \tag{5.3}$$

where $\oiint_\sigma$ is the 2D integration over the circular region of radius σ centered at the origin, $(\hat{x},\hat{y})$ is the agents estimate of their position, $\hat{W}$ is the agents estimate of the local water current, σ_w is the water current uncertainty, σ_p is the positional uncertainty. We also define

$$\begin{aligned}\mathcal{Q}(\hat{x},\hat{y},\sigma_p,\hat{W},0,a) &= \frac{1}{4\sigma_p^2}\oiint_{\sigma_p}\Big(r(\hat{x}+x',\hat{y}+y',a) \\ &\qquad + \gamma V(a_M(\hat{x}+x',\hat{y}+y',\hat{W}))\Big)dx'dy', \\ \mathcal{Q}(\hat{x},\hat{y},0,\hat{W},\sigma_w,a) &= \frac{1}{4\sigma_w^2}\oiint_{\sigma_w}\Big(r(\hat{x},\hat{y},a) \\ &\qquad + \gamma V(a_M(\hat{x},\hat{y},\hat{W}+(w_x,w_y)))\Big)dw_x dw_y, \\ \mathcal{Q}(\hat{x},\hat{y},0,\hat{W},0,a) &= r(\hat{x},\hat{y},a(\hat{x},\hat{y},\hat{W})) + \gamma V(a_M(\hat{x},\hat{y},\hat{W})).\end{aligned} \tag{5.4}$$

Our policy is then constructed by choosing the non-measurement action that maximizes the Q-function. If the expected cost of that action is greater than the cost of the measurement actions, then a measurement action is taken instead (with the GPS taking precedence over the current profiler).

5.4 Computational Set-up

For computational purposes, we discretize the action space to 96 non-measurement actions for the agent (6 throttle settings and 16 heading directions), state-space to a resolution of 152×152 for the value function, and integrals in 5.3. Note that only the input to the value function is discretized and the actual state-space remains continuous. As the relationship between speed through the water and throttle is included in the agent's incomplete access to information, the discrete actions available are chosen such that the non-measurement action choices are linear with respect to speed through the water for simplicity, as shown in Figure 5.1(b), inclusive of 0 and 1.

For the chart generation, we have $x_{\max} = y_{\max} = 10$ for all charts and a varying number of islands $0 \leq N \leq 20$. We consider $1 \leq N \leq 5$ charts of low island density, $8 \leq N \leq 12$ charts of medium island density, and $16 \leq N \leq 20$ charts of high island density. 100 charts of low density density, 150 charts of medium density, and 250 charts of high density will be used with 10 different initial states each. The maximum water current magnitude and the linear rates at which the uncertainty used in a policy grows (uncertainty growth rate) will be parameters of experimentation, each varying from 0 to 1. The estimates in water currents are bounded by $\left\|\hat{W}(x,y)\right\|_2 \leq w_{\max}$. The GPS measurement has a cost of 0.45 and the current profiler measurement has a cost of 0.1 (i.e. a reward of -0.45 and -0.1 respectively).

5.5 Results

Based on preliminary tests, it is possible that a trajectory can be cyclic and these (potentially) infinite trajectories are typically the only ones that lasted more than 25

steps. For this reason, we limit all trajectories to 25 steps. We consider the following three types of outcomes: a policy that reaches the target within 25 steps is considered successful, a policy that crashes within 25 steps is a failure, and a policy that neither is successful nor a failure.

For each chart, a value function is generated using the method described in Section 5.3.1. For each initial state, a policy is constructed several times using the method described in Section 5.3.2, where the uncertainty growth rate is varied. An uncertainty growth rate of zero is equivalent to using Bellman's equation and assuming there does not exist any water current.

Figures 5.4(a)-(c) show the success rate as a function of uncertainty growth rate and maximum water current for policies constructed for low, medium, and high island densities, respectively. When the maximum water current is zero, the uncertainty growth rate does not affect the agent's behaviour due to the upper bound on the water current estimates. Without any water current, the problem is equivalent to the MDP problem initially used to generate the value functions. Hence, the agent succeeds in 100% of the charts for all three island density sets, which is expected as the agent has the true value functions for the problem, however, this is no longer true once the maximum water current is non-zero.

For an uncertainty growth rate of zero, the agent performs quite well at extremely low maximum water currents and lower island densities. However, even for low (non-zero) maximum water currents, the agent's performance begins to decline for charts with higher island densities, succeeding in less than 90% of charts. As the maximum water current increases the agent's performance steadily decreases to the point it succeeds in 0% of all charts. This drop to a 0% success rate is most notable in the charts with higher island densities.

Excluding a few outliers for the more extreme maximum water currents, even the

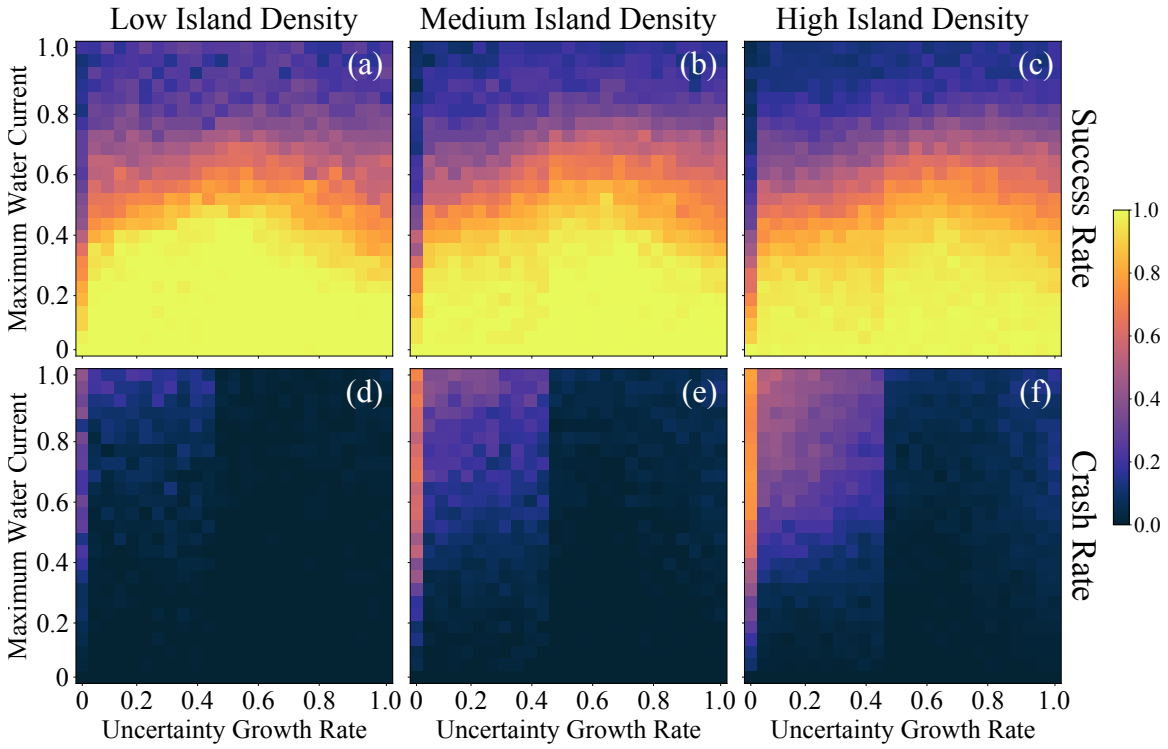


Figure 5.4: Policy statistics constructed over 500 unique charts for various uncertainty growth rates and maximum water currents. (a)-(c) The agent's success rate, where success means the agent navigated the submarine into the target area for low, medium, and high island densities, respectively. (d)-(f) The agent's crash rate for low, medium, and high island densities respectively. Note that the agent's success and crash rates do not include the cases where the agent's trajectory lasts more than 20 steps. In all cases, a maximum water current of 0 is equivalent to no water current existing and an uncertainty growth rate of zero is equivalent to using standard DP for MDPs.

smallest tested non-zero uncertainty growth rate outperforms the zero case in charts of all island densities. For maximum water currents less than 0.45, 0.4, and 0.35, there exists at least one tested uncertainty growth rate that gives the agent a success rate of 100% for all charts of low, medium, and high island densities, respectively. At those maximum water currents, when using a non-zero uncertainty growth rate the agent is able to get an increase in success rate of up to 58%, 67%, and 63% for all charts of low, medium, and high island densities.

While the specific non-zero value for the uncertainty growth rate does not make

much difference in the agent's success rates at lower maximum water currents, it matters significantly for larger maxima. For the larger maximum water currents, the agent's success rate increases on average as the uncertainty growth rate increases (approximately 0.7). The agent's success rate begins to decline on average once the uncertainty growth rate is increased beyond this point. At these high uncertainty growth rates, any target remotely close to an island appears too risky to reach, i.e. the expected cost due to crashing is greater than the expected reward of succeeding.

The trends discussed here all tend to break for the largest maximum water currents as the agent's success rate stays close to 0%. For maximum water currents near 1.0, the displacement caused by the water current can be as large as the distance the agent can possibly cover in a single action. This can make it impossible for the agent to overcome the water current and reach the target in most cases, regardless of the uncertainty growth rate or method used.

Figures 5.4(d)-(f) instead show the crash rate as a function of uncertainty growth rate and maximum water current for policies constructed for low, medium, and high island densities, respectively. In the cases the agent's success rate is near 100% the crash rate must be near 0%, however lower success rates do not imply higher crash rates. For an uncertainty growth rate of zero, the agent's crash rate increases at a similar rate to the decrease in success rate as the maximum water current is increased, reaching 80% in some cases.

When the maximum water current is increased, we see a similar trend for the lower non-zero uncertainty rates as before, where the crash rate increases simultaneously as the success rate decreases. The increase in crash rate is much less significant compared to the zero case though, even as the success rate goes to 0%. Therefore, even when the agent does not succeed, it is much better at managing to avoid islands. For the largest of the maximum water currents, even the smallest non-zero uncertainty

growth rate decreases the crash rate by over 25%.

Regardless of the success rate, we see the crash rate drop to (or near) 0% for larger uncertainty growth rates. The larger uncertainty growth rates use the extremes of water current estimates, therefore the agent uses most actions avoiding islands, rather than reaching the target. The crash rate slightly increases again for the charts with higher island densities at larger uncertainty growth rates and maximum water currents. In these cases, the uncertainty of each action is so large that the agent estimates they will all end in crashing. The "safest" action in these cases is then to do nothing, in which case the water current causes the agent to drift into an island, resulting in a crash.

In every case, the agent uses slightly less than one measurement per action on average. If the agent were to use both the GPS and current profiler measurements for every action, it would result in an average measurement cost of 0.55 per action. Our agent's measurement costs fall into three regimes based on the uncertainty growth rate: approximately 0.2, 0.3, and 0.45 for uncertainty growth rates less than 0.25, between 0.25 and 0.45, and greater than 0.45, respectively. For small uncertainty growth rates, we have a large reduction, and for large uncertainty growth rates we have a minor, but non-zero, reduction in measurement costs. This tells us to choose the smallest uncertainty growth rate that results in the largest success rate for any given maximum water current, thus minimizing measurement costs without sacrificing navigational safety. An example of a successful policy trajectory in a high island density chart with a non-zero water current magnitude is shown in Figure 5.5.

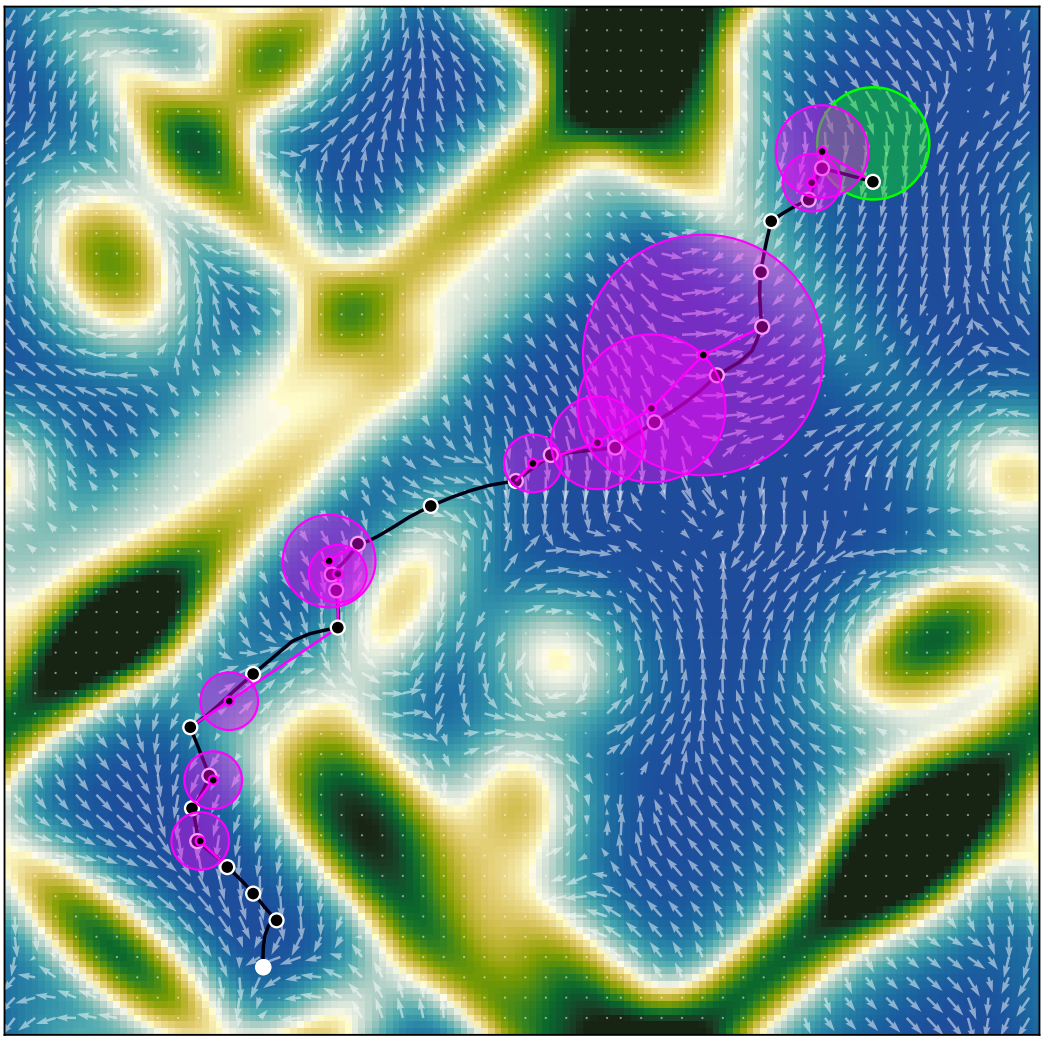


Figure 5.5: An example trajectory of a successful policy on a high island density chart. The white circle represents the agent's starting position. The black lines represent the agent's true trajectory, where each white outlined circle indicates the points when the agent was required to select an action. The magenta lines represent the agent's estimated trajectory, where the transparent magenta shading represents the magnitude of uncertainty and each magenta outlined circle represents when the agent was required to select an action. When a white-outlined circle is connected to a magenta one, this represents when the agent takes an action and does not use the GPS measurement. When a magenta-outlined circle is connected to a white one, or when no magenta-outlined circle is present for that step, this represents when the agent takes an action and then uses the GPS measurement. The transparent green circle represents the target region.

5.6 Conclusions & Future Work

Motivated by the real-world applicability of POMDPs and systems with uncertainty, we have shown that incomplete access to information can be leveraged with DP methods to construct navigational policies that both maintain safety and reduce total measurement cost. The navigation environment we introduced serves as a relevant introduction to the problems of interest in the combined area of traditional and controlled sensing POMDPs. The methods provided allow the construction of value functions through DP that contain the basic information of the system of interest. We show that without any additional constraints (uncertainty growth rate of zero), the policies produced using these value functions perform very poorly. However, when uncertainty methods are included, the success rate on average is doubled and the crash rate is brought to (or nearly to) zero.

While the method shown here has been quite successful, it is not perfect. The success of using a fixed uncertainty growth rate makes the assumption that the maximum water current is known. We would like to include an adaptive uncertainty growth rate in future versions of this algorithm. This adaptive method could be a neural network-based learned mapping from charts to optimal uncertainty growth rate for that trajectory or a constantly updating value based on calculated average water currents between GPS measurements. For further comparison of our method's performance, we would like to develop a deep RL-based policy as well, however the complexity of a deep RL solution on this type of problem can be quite large [98].

This nautical navigation problem further demonstrates how an agent's behaviour is shaped by the structure of the paths available to it and the degree to which that structure is revealed through its observations. Here, feasible trajectories are influenced not only by the spatial arrangement of obstacles, but also by the directional

dynamics imposed by water currents, the coupling of control inputs to vessel motion, and the nonholonomic constraints inherent in nautical navigation. These features create a richly organized landscape of potential routes to the goal—one that becomes significantly more difficult to reason about when observations provide only partial or noisy glimpses of the underlying state. In this setting, the agent must infer the structure of the goal-directed paths indirectly, reconstructing both feasible headings and longer-term strategy from incomplete information. The resulting differences in performance and policy quality highlight how uncertainty reshapes the agent's internal representation of the task's path structure, paralleling the distinctions observed between fully observable and partially observable cases in earlier chapters. This interplay between structural organization and limited information provides a transition to the environment of interest in the next chapter, where complex procedural branching introduces yet another form of structural challenge for learning agents.

Chapter 6

Learning Chemistry

In material design, the goal is to determine a pathway of chemical and physical manipulation that can be performed on some starting materials or substances in order to transform them into a desired target material. The aim of this research is to demonstrate the potential of RL for generalized GOMDPs in automated labs. Our experiments show that when given an objective (such as a target material) and a set of initial materials, RL can learn general pathways to achieve that objective. We postulate that well-trained RL chemist-agents could help reduce experimentation time and cost in these and related fields by learning to complete tasks that are repetitive, labour intensive and/or require a high degree of precision. With increased simulation complexity, well trained RL chemist-agents could potentially discover new materials and/or reaction pathways in this system as well. To support this, we share the **ChemGymRL** environment that allows scientists and researchers to simulate chemical laboratories for the development of RL agents. Much of this chapter is adapted from the original work that has been previously published [4], first authored by C. Beeler. The material has been revised for coherence and consistency within the overall structure of this thesis.

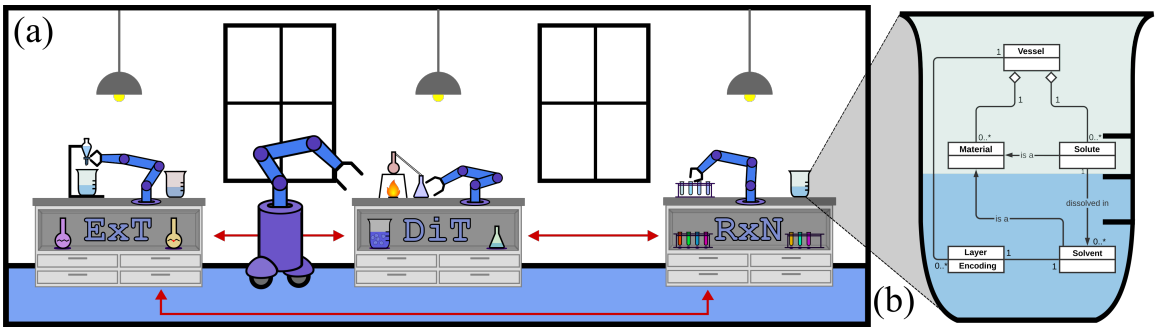


Figure 6.1: (a) The ChemGymRL simulation. Individual agents operate at each bench, working towards their own goals. The benches pictured are extraction (`ExT`), distillation (`DiT`), and reaction (`RxN`). The user determines which materials the bench agents have access to and what vessels they start with. Vessels can move between benches; the output of one bench becomes an input of another, just as in a real chemistry lab. Successfully making a material requires the skilled operation of each individual bench as well as using them as a collective. (b) Materials within a laboratory environment are stored and transported between benches within a vessel. Benches can act on these vessels by combining them, adding materials to them, allowing for a reaction to occur, observing them (thereby producing a measurement), etc.

ChemGymRL is a collection of interconnected environments (chemistry benches) that enable the training of RL agents for discovery and optimization of chemical synthesis. These environments are each a virtual variant of a chemistry "bench", an experiment or process that would otherwise be performed in real-world chemistry labs. As shown in Figure 6.1, the ChemGymRL environment includes reaction, distillation, and extraction benches on which RL agents learn to perform actions and satisfy objectives. Each bench could be a standalone environment (as the majority of RL environments are), this would counter the purpose of their inter-connectivity. While their inter-connectivity is not important for training a specific agent on a single bench, sharing an overarching framework for the benches ensures a compatibility between them, allowing the results of one experiment to be easily used as input to another bench's task. This makes inter-connectivity important if one wishes to perform a multi-task experiment, requiring several benches to be used in differing orders.

From an outside perspective, an individual bench being operated by a separate entity makes it equivalent to a process that performs a series of operations that transform the system from one state to another, generating a trajectory. Navigating these trajectories in chemical space, only to be rewarded upon confirmation of producing a desired chemical, makes each bench analogous to a generalized GOMDP (although likely a partially observable one in most cases). Unfeasible energy barriers in this space create navigational "holes", much like the islands in Chapter 5, that make the space more complex. Each of these benches are independent from the others, therefore the trajectories *within* a bench are separated as well, meaning each bench is its own homotopy class of trajectories from the view of the laboratory. While the benches are more similar to the goal-directed problems in Chapters 4 & 5, the laboratory is more about selecting homotopy classes to enter rather than specific path optimization, much like the separated-path MDPs in Chapter 3.

The need for a simulated chemistry environment for designing, developing, and evaluating artificial intelligence algorithms is motivated by the recent growth in research on topics, such as automated chemistry and self-driving laboratories [99–103], laboratory robots [104–111] and digital chemistry for materials and drug discovery [112–120]. Given RL's appropriateness for sequential decision making, and its ability to learn via online interactions with a physical or simulated environment without a supervised training signal, we see it as having a great potential within digital chemistry and self-driving laboratories.

Within this context, recent work has demonstrated some successful applications of RL [121, 122] or methods inspired by parts of RL [123] to automated chemistry.

Nonetheless, it remains an understudied area of research. Our work aims to partially address this problem by sharing an easy to use, extensible, open source, simulated chemical laboratory. This serves to simplify the design and development of

application specific RL agents.

Although RL agents could be trained online in physical laboratories, this approach has many limitations, particularly in early stages of the research before mature policies exist. Training agents in a robotic laboratory in real-time would be costly (in both time and supplies) and restrictive (due to potential safety hazards). Our simulated ChemGymRL environment remedies this by allowing the early exploration phase to occur digitally, speeding up the process and reducing the waste of chemical materials. It provides a mechanism to design, develop, evaluate and refine RL for chemistry applications and research, which cannot safely be achieved in a physical setting. We would also like to specifically highlight ChemGymRL as a unique testbed for RL research. Since ChemGymRL is open source and highly customizable, it provides a training environment to accelerate both chemical and RL research in several directions in addition to providing a useful training environment with real-world applications.

The software is developed according to the `Gymnasium`[1] standard, which facilitates easy experimentation and exploration with novel and off-the-shelf RL algorithms. When users download it, they gain access to a standard Gymnasium compatible environment that simulates chemical reactions using rate law differential equations, the mixing/settling of soluble and non-soluble solutions for solute extractions, the distillation of solutions, and a digital format for storing the state of the vessels used. In addition to this article, further detailed information about this software package, documentation and tutorials, including code and videos can be found at https://www.chemgymrl.com/.

In our experimental results, we illustrate how to setup and use each bench with two distinct classes of reactions, along with how they can be extended to new re-

[1]The Gymnasium API https://github.com/Farama-Foundation/Gymnasium is the continuing development name of the original OpenAI Gym library for RL environments.

action types. We evaluate the capabilities of a wide cross-section of off-the-shelf RL algorithms for goal-based policy learning in ChemGymRL, and compare these against hand-designed heuristic baseline agents. In our analysis, we find that only one off-the-shelf RL algorithm, PPO, is able to consistently outperform these heuristics on each bench. This suggests that the heuristics are a challenging baseline to compare to but that they are also far from optimal. Thus there is space for an optimization approach such as RL to achieve optimal behaviour on each bench. Near the end of the paper, we discuss some of the challenges, limitations and potential improvements in RL algorithms required to learn better, more sample efficient policies for discovery and optimization of material design pathways.

The remainder of this chapter is organized as follows. The next section describes the ChemGymRL environment, including the three primary benches: Reaction, Extraction and Distillation. Section 6.2 provides an overview of RL and Section 6.3 contains a case study of the Wurtz Reaction and its use in each bench. Our experimental setup involves training off-the-shelf RL algorithms on each of the benches. The RL algorithms and hyper-parameters are discussed in Section 6.4 and the specific laboratory settings and objectives used in our experiments are described in Section 6.5. The results of the RL experiments are presented in Section 6.6 and the limitations of the simulations are discussed in Section 6.7 followed by our general conclusions and some ideas for future directions.

6.1 ChemGymRL

6.1.1 The Laboratory

ChemGymRL is designed in a modular fashion so that new simulated benches can be added or modified with minimal difficulty or changes to the source code. The environment can be thought of as a virtual chemistry laboratory consisting of different stations or benches where a variety of tasks can be completed, represented in Figure 6.1(a). The main 2 components of the laboratory are vessels and benches.

Benches recreate a simplified version of a task in a material design pipeline and *vessels* contain materials and track the hidden internal state of their contents, as shown in Figure 6.1(b). A bench must be able to receive a set of initial experimental supplies, possibly including vessels, and return the results of the intended experiment, also including modified vessels. A vessel is used to transfer the chemical state contained in one bench to another, allowing for continuation of experiments.

We provide some core elements of a basic chemistry lab which enable the simulation of essential materials experiments. Critically, each bench and the gym as a whole is extensible. Therefore there is no limit on the complexity and precision simulations can be implemented. In the following sections we describe each of these benches in turn and demonstrate an example workflow.

These benches each have three crucial components required for operating them. The *observation space* is the possible set of observations which the agent (or human user) can use to learn the status of the bench and take appropriate actions. These observations are usually only a partial representation of the internal state of the system. The *action space* for a bench, is a set of actions the user can take on that bench. These actions are methods of modifying the state of the system or observation.

Lastly, the *reward function* is a measure of success based on the states the system has been in and the actions that have been taken. Generally in RL, the goal is to maximize its expected outputs, known as *rewards*, over time. These rewards are usually discounted over time. An *episode* refers to a single play of a bench from starting materials until the agent stops. The total, cumulative expected reward over an entire episode is called the *return*.

Remark (Chemistry Benches as Generalized GOMDPs)**.** *Each chemistry bench can be understood as a generalized GOMDP in the sense of Chapter 3:*

- ***Reaction Bench*** *(`RxN`): The goal-states $\mathcal{G}$ consist of vessel compositions where the target material yield exceeds a threshold. Rewards are zero (non-positive) during the reaction, with positive reward only upon achieving the target composition.*

- ***Extraction Bench*** *(`ExT`): The goal-states correspond to achieving a target solute purity. Intermediate actions (mixing, pouring, waiting) incur zero reward, with positive reward for purity improvement at episode end.*

- ***Distillation Bench*** *(`DiT`): The goal-states correspond to achieving target material purity. Energy expenditure during heating/cooling represents non-positive intermediate rewards.*

Moreover, the state spaces of these benches contain topological structure: energy barriers between chemical states create "holes" analogous to obstacles in spatial navigation, where certain state transitions are infeasible or require specific reaction pathways. The homotopy classes of synthesis routes correspond to distinct reaction sequences that cannot be continuously deformed into one another.

6.1.2 Reaction Bench (RxN)

The sole purpose of the **reaction bench** (RxN) is to allow the agent to transform available reactants into various products via a chemical reaction. The agent has the ability to control temperature and pressure of the vessel as well as the amounts of reactants added. The mechanics of this bench are quite simple in comparison to real-life, which enables low computational cost for RL training. Reactions are modelled by solving a system of differential equations which define the rates of changes in concentration (see Appendix 8.5).

The goal of the agent operating on this bench is to modify the reaction parameters, in order to increase, or decrease, the yield of certain desired, or undesired, materials. The key to the agent's success in this bench is learning how best to allow certain reactions to occur such that the yield of the desired material is maximized and the yield of the undesired material is minimized. Therefore the reward in this bench is zero at all steps except the final step, at which point it is the difference in the number of mols of the desired material and undesired material(s) produced.

Observation Space: In this bench, the agent is able to observe a UV-Vis absorption spectra of the materials present in the vessel as shown in Figure 6.2(a), the normalized temperature, volume, pressure, and available materials for the system.

Action Space: The agent can increase or decrease the temperature and volume of the vessel, as well as add any fraction of the remaining reactants available to it. In this bench, the actions returned by an agent are a continuous valued vector of size $n + 2$, where n is the number of reactants. These actions are also shown in Figure 6.2(b).

A main feature of ChemGymRL is its modularity. If one wanted to make the results of RxN more accurate and generalizable, they could replace the current system

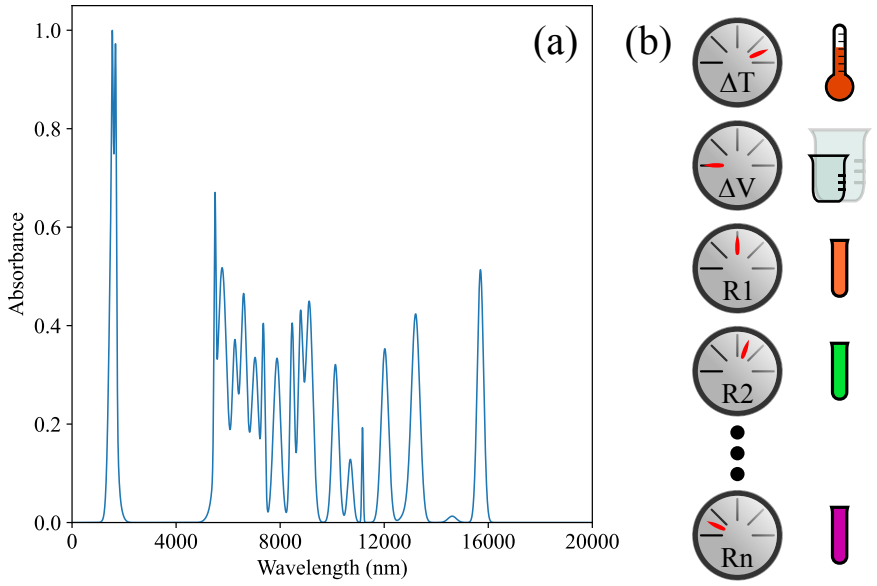


Figure 6.2: A visualization of the reaction bench (`RxN`) observation and action space. (a) An example of a UV-Vis spectra that would been seen in an observation and (b) The icons representing each action in `RxN`.

of differential equations with a molecular dynamics simulation without needing to change how the agent interacts with the bench or how the bench interacts with the rest of ChemGymRL. In its current state, this bench takes approximately 0.73ms to initialize and 0.87ms to perform an action.

6.1.3 Extraction Bench (`ExT`)

Chemical reactions commonly result in a mixture of desired and undesired products. Extraction is a method to separate them. The extraction bench (`ExT`) aims to isolate and extract certain dissolved materials from an input vessel containing multiple materials through the use of various insoluble solvents. This is done by means of transferring materials between a number of vessels and utilizing specifically selected solvents to separate materials from each other.

A simple extraction experiment example is extracting salt from an oil solvent using water. Suppose we have a vessel containing sodium chloride dissolved in hexane.

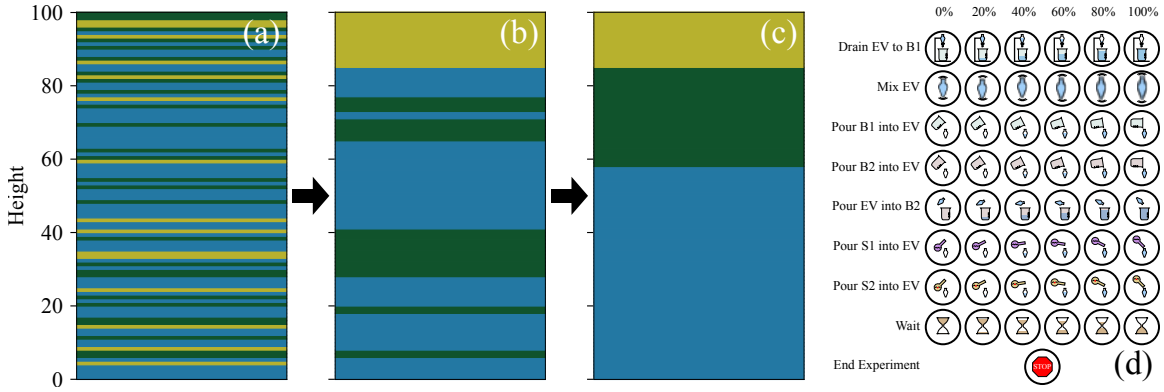


Figure 6.3: Typical observations seen in extraction bench (`ExT`) for a vessel containing air, hexane, and water. (a) The vessel in a fully mixed state. Each material is uniformly distributed throughout the vessel with little to no distinct layers formed. (b) The vessel in a partially mixed state. The air has formed a single layer at the top of the vessel and some distinct water and hexane layers have formed, however they are still mixed with each other. (c) The vessel in a fully settled state. Three distinct layers have formed in order of increasing density: water, hexane, and then air. (d) The icons representing each action and their multiplier values available in `ExT`. The extraction vessel (EV) is the primary vessel used, B1/B2 are the auxiliary vessels used in the experiment, and S1/S2 are the solvents available.

Water is added to a vessel and the vessel is shaken to mix the two solvents. When the contents of the vessel settle, the water and hexane will have separated into two different layers. Sodium chloride is an ionic compound, therefore there is a distinct separation of charges when dissolved. Due to hexane being a non-polar solvent and water being a polar solvent, a large portion of the dissolved sodium chloride is pulled from the hexane into the water. Since water has a higher density than hexane, it is found at the bottom of the vessel and can be easily drained away, bringing the dissolved sodium chloride with it.

Observation Space: For a visual representation of the solvent layers in the vessel for the agent, as seen in Figure 6.3(a)-(c), each pixel is sampled from an evolving distribution of solvent in the vessel. The exact details of this process are outlined in Section 8.5.2. This representation makes this bench a POMDP as the true state of the solutes distribution through the solvents is not shown.

Action Space: In this environment, the agent has 8 actions it can take to manipulate the vessels and their contents. In contrast to `RxN`, the actions *conceptually* consist of two discrete components, as follows: (1) a value that determines *which* of the processes are performed; these are mutually exclusive action types (e.g. "mix", "pour", etc.), and (2) a second value that determines the *magnitude* of that process occurs (e.g. "how much", "how long").

The mutually exclusive actions are: (1) mix the vessel or let it settle (i.e. wait), (2) add an amount of solvent to the vessel, (3) drain contents of the vessel into an auxiliary vessel *bottom first*, (4) pour contents of the vessel into a second auxiliary vessel, (5) pour contents of the first auxiliary vessel into the second, (6) pour contents of first auxiliary vessel into the second, (7) pour contents back into the original vessel, (8) end the experiment.

The multiplier for each action corresponds to either the duration (mix, wait), the amount to pour, or the amount to drain, with 5 discrete non-zero values each. These actions are depicted in Figure 6.3(d).

Note that, for practical implementation purposes, the two-part action described above is flattened into a single discrete value to reduce redundancy in the action space.

The **goal of the agent** in this bench is to use these processes in order to maximize the purity of a desired *solute* relative to other *solutes* in the vessel. This means the agent must isolate the desired solute in one vessel, while separating any other solutes into the other vessels. Note that the solute's relative purity *is not* affected by the presence of solvents, only the presence of other solutes. Therefore the reward in this bench is zero at all steps except the final step, at which point it is the difference in the relative purity of the desired solute at the first and final steps.

As with `RxN`, the realism of `ExT` could be improved by replacing the separation

equations with a physics-based simulation without needing to change how the agent interacts with the bench or how the bench interacts with the rest of ChemGymRL. In its current state, this bench takes approximately 0.87ms to initialize and 0.47ms to perform an action.

6.1.4 Distillation Bench (`DiT`)

Similar to the `ExT`, the distillation bench (`DiT`) aims to isolate certain materials from a provided vessel containing multiple materials (albeit with a different process). This is done by means of transferring materials between a number of vessels and heating/cooling the vessel to separate materials from each other.

A simple distillation example is extracting a solute dissolved in a single solvent. Suppose we have a vessel containing sodium chloride dissolved in water. If we heat the vessel to 100°C, the water will begin to boil. With any added heat, more water will evaporate and be collected in an auxiliary vessel, leaving the dissolved sodium chloride behind to precipitate out as solid sodium chloride in the original vessel.

Observation Space: For a visual representation for the agent, we use the same approach described for `ExT`. For the precipitation of any solutes, we define a precipitation reaction and use the same approach described for `RxN`.

Action Space: The agent has the ability to heat the vessel or let it cool down and pour the contents of any of the vessels (original and auxiliaries) into one another. When the agent heats/cools the vessel, the temperature of the vessel and its materials are altered by

$$\Delta T = \frac{Q}{C}, \tag{6.1}$$

where Q is the amount of heat added and C is the total heat capacity of the contents of the vessel. However, if the temperature of the vessel is at the boiling point of one

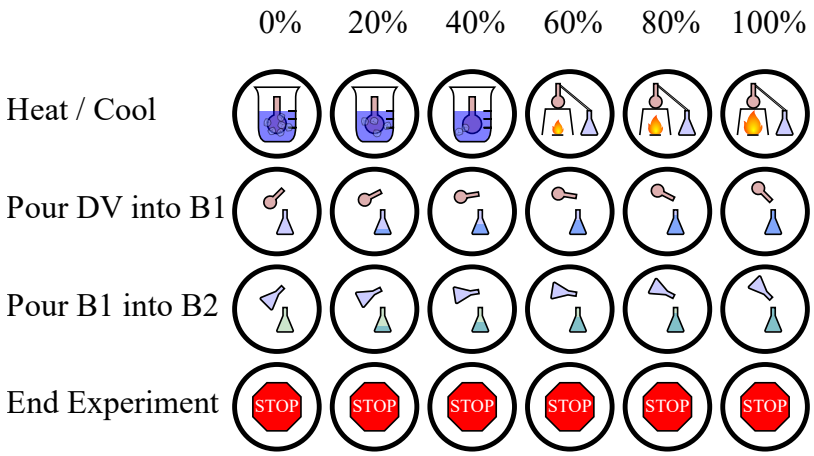


Figure 6.4: The icons representing each action and their multiplier values available in `DiT`. The distillation vessel (DV) is the primary vessel and B1/B2 are the auxiliary vessels in the experiment.

of its materials, the temperature no longer increases. Instead, any heat added is used to vaporize that material according to

$$\Delta n_l = \frac{Q}{\Delta H_v}, \tag{6.2}$$

where n_l is the number of mols of the material in the liquid phase and H_v is the enthalpy of vaporization for that material.

Similar to the `ExT` bench, these processes are mutually exclusive and each have a magnitude (temperature change, amount to pour). Thus, the same kind of (action, multiplier) definition is used for `DiT` bench. The actions can be one of the following four choices: (1) heat/cool by some amount, (2) pour from the distillation vessel into an auxiliary vessel, (3) pour from an one auxiliary vessel into another, or (4) end the experiment. Actions (1-3) each can have one of 10 multiplier values specifying magnitude. These actions are depicted in Figure 6.4. Just as in `ExT`, the actions are returned by the agent are flattened into a single discrete value to reduce redundancy in the action space.

Reaction	R_1	R_2	R_1-R_2
1	1-chlorohexane	1-chlorohexane	dodecane
2	1-chlorohexane	2-chlorohexane	5-methylundecane
3	1-chlorohexane	3-chlorohexane	4-ethyldecane
4	2-chlorohexane	2-chlorohexane	5,6-dimethyldecane
5	2-chlorohexane	3-chlorohexane	4-ethyl-5-methylnonane
6	3-chlorohexane	3-chlorohexane	4,5-diethyloctane

Table 6.1: All possible Wurtz reactions involving chlorohexanes. Symmetrically equivalent entries have been removed from the table as R_1-R_2 = R_2-R_1 and 6, 5, 4-chlorohexane is equivalent to 1, 2, 3-chlorohexane, respectively.

The goal of the agent in this bench is to use these processes to maximize the absolute purity of a desired material in the vessel. This means the agent must isolate the desired *material* in one vessel, while separating any other *materials* into other vessels. Note that unlike `ExT`, the material's absolute purity is affected by the presence of all materials. Therefore the reward in this bench is zero at all steps except the final step, at which point it is the difference in the absolute purity of the desired material at the first and final steps. In its current state, this bench takes approximately 0.87ms to initialize and 0.86ms to perform an action.

6.1.5 Characterization Bench

In general, it is impossible to determine the exact contents of a vessel just by looking at it. Techniques exist to help characterize the contents of a vessel, however each comes with a cost. The primary cost is the monetary cost to acquire/maintain/run the instrument used. In some cases, the sample of the vessel contents being measured is destroyed during the measurement, thus incurring a different type of cost. While these costs are not implemented in this version of ChemGymRL, they are important to consider when expanding the system.

The characterization bench is the primary method to obtain insight as to what

the vessel contains. The purpose of the characterization bench is not to manipulate the input vessel, but to subject it to analysis techniques that observe the state of the vessel, possibly including the materials inside it and their relative quantities. This does not mean that the contents of the input vessel cannot be modified by the characterization bench. This allows an agent or user to observe vessels, determine their contents, and allocate the vessel to the necessary bench for further experimentation.

The characterization bench is the only bench that is not "operated". A vessel is provided to the bench along with a characterization method and the results of said method on that vessel are returned. Currently, the characterization bench consists of a UV-Vis spectrometer that returns the UV-Vis absorption spectrum of the provided vessel. Each material in ChemGymRL has a set of UV-Vis absorption peaks defined and the UV-Vis spectrum for a vessel is the combination of the peaks for all materials present, weighted proportionally by their concentrations. In future versions of ChemGymRL we will expand the characterization bench to include other forms of partial observation.

6.2 Reinforcement Learning

Recall from Section 6.2 that RL [6] is one possible solution to an MDP. Also recall from Section 2.1.3 that infinite horizon MDPs are represented as a tuple $(\mathcal{S}, \mathcal{A}, R, T, \gamma)$ where $s \in \mathcal{S} \subseteq \mathbb{R}^n$ denotes the state space, $a \in \mathcal{A} \subseteq \mathbb{R}^m$ denotes the action space, $r \in R \subseteq \mathbb{R}$ denotes the reward function and $T = P(s_{t+1}|s_t, a_t)$ denotes the transition dynamics (or model) that provides the probability of state s_{t+1} at the next time step given that the agent is in state s_t and performs action a_t. The objective for an RL agent is to learn a policy $\pi(a|s)$ that maximizes the discounted sum of expected rewards provided by the equation $J_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s]$, where $\gamma \in [0, 1)$ is

the discount factor.

In the field of model-free RL on which we focus here, a major distinction between solution algorithms is between *value optimization* approaches and *direct policy optimization* approaches. A popular example of value optimization is Q-learning [7, 26], where a *state-action value* function $Q(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, is learned iteratively using the Bellman optimality operator $\mathcal{B}^* Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim T(s'|s,a)}[\max_{a'} Q(s', a')]$. Here s and s' denote the current and next state respectively, a and a' denote the current and next action, respectively. After this process converges, an exact or approximate scheme of maximization is used to extract the greedy policy from the Q-function. These methods are often restricted to environments with discrete actions, although many generalizations exist to other formulations [124–126].

Direct policy optimization [127] approaches are ones which iteratively improve the target policy directly. They may do this as the only optimization approach, or they may do it in combination with value-function optimization. Actor-critic [128–130] methods are a currently popular approach for doing just that. In actor-critic methods, the algorithm alternates between estimating a value function Q^π (the "critic") of a current policy via a partial policy evaluation routine using the Bellman operator on an initially random, stochastic policy π. The current policy π (the "actor") is then improved by biasing it towards selecting actions that maximize the estimate maintained by the current Q-values and the value function for this improved policy is then re-estimated again. This family of methods can easily apply to both discrete and continuous action space environments, thus may be used on any bench in chemistry gym environment.

6.3 Case Study

As a simple example, we outline how a particular chemical production process uses each of the benches.

6.3.1 Wurtz Reaction

Wurtz reactions are commonly used for the formation of certain hydrocarbons. These reactions are of the form:

$$2\text{R-Cl} + 2\text{Na} \xrightarrow{\text{diethyl ether}} \text{R-R} + 2\text{NaCl}. \tag{6.3}$$

Here we consider the case of hexane (C_6H_{14}) for R, where one hydrogen atoms is replaced with chlorine, giving us 1-, 2-, and 3-chlorohexane as reactants with sodium. Note that we may have 2R-Cl and R-R be replaced with R_1-Cl, R_2-Cl, and R_1-R_2 in this reaction format. Table 6.1 shows the possible outcomes of this reaction. Note that it is impossible to produce just 5-methylundecane, 4-ethyldecane, or 4-ethyl-5-methylnonane. If the desired reaction is

$$\text{R}_1\text{-Cl} + \text{R}_2\text{-Cl} + 2\text{Na} \xrightarrow{\text{diethyl ether}} \text{R}_1\text{-R}_2 + 2\text{NaCl}, \tag{6.4}$$

then we will unavoidably also have

$$\begin{aligned} 2\text{R}_1\text{-Cl} + 2\text{Na} &\xrightarrow{\text{diethyl ether}} \text{R}_1\text{-R}_1 + 2\text{NaCl} \\ 2\text{R}_2\text{-Cl} + 2\text{Na} &\xrightarrow{\text{diethyl ether}} \text{R}_2\text{-R}_2 + 2\text{NaCl}, \end{aligned} \tag{6.5}$$

occurring simultaneously.

Wurtz can be an interesting and challenging reaction because the yield varies

greatly between each product, making it difficult to train an agent which can optimally make each of them.

6.3.2 Workflow

Suppose that we have the previously listed chlorohexanes, sodium, diethyl ether, and water available to us with the goal to produce dodecane. Using `RxN` we can add diethyl ether, 1-chlorohexane, and sodium to a vessel. With time, this will produce a vessel containing dodecane and sodium chloride dissolved in diethyl ether. The UV-Vis spectrometer in the `RxN` can be used to measure the progression of the reaction.

The vessel can then be brought to the `ExT` to separate dodecane from sodium chloride. Dodecane is non-polar, so if we add water to the vessel and mix, most of the sodium chloride will be extracted into the water while most of the dodecane will be left in the diethyl ether. We can then drain the water out of the vessel while keeping the diethyl ether. While it's impossible to get all of the sodium chloride out with this method, we can repeat this process to increase the purity of dodecane.

The vessel can then be brought to the `DiT` to separate the dodecane from the diethyl ether. Diethyl ether has a much lower boiling point than dodecane so it will boil first. Heating the vessel enough will cause all of the diethyl ether to vaporize, leaving the dodecane in the vessel with trace amounts of sodium chloride.

Alternatively, because dodecane has a much lower boiling point than sodium chloride, we can skip the `ExT` and bring the vessel to `DiT` right after `RxN`. As before, heating the vessel enough will cause all of the diethyl ether to vaporize, condensing into an auxiliary vessel. We can then put the collect diethyl ether elsewhere such that the auxiliary vessel collected the vaporized materials is now empty. If the vessel is heated up even further now, the dodecane will be vaporized and collected into the empty auxiliary vessel, leaving the sodium chloride behind. Our auxiliary vessel now contains

pure dodecane, concluding the experiment.

While this example workflow uses the benches in a specific order, more complicated experiments may use them in a completely different order or even use each bench multiple times. Given specific goals, below we will outline how RL can be used to emulate this behaviour for various cases.

6.4 RL Details

In our discrete benches, PPO [43], Advantageous Actor-Critic (A2C) [130] and Deep Q-Network (DQN) [7] were used. In our continuous benches, Soft Actor-Critic (SAC) [131] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [132] were used instead of DQN.

Unless otherwise specified, all RL agents were trained for 100K time steps across 10 environments in parallel (for a total of 1M time steps). Training was done by repeatedly gathering 256 time steps of experience (in each environment), then updating our policy and/or Q-function with this new experience. Since PPO and A2C are on-policy, their policies were only updated with the 2560 steps of new experiences. In contrast, a replay buffer of size 1M was maintained and sampled when training with DQN, SAC, and TD3. For the first 30K steps of DQN training, a linear exploration schedule beginning at 1.0 and ending at 0.01 was used. Exploration remained at 0.01 afterwards. All of these RL algorithms were performed using the Stable Baselines 3 implementations [133].

6.5 Laboratory Setup

6.5.1 Reaction Bench Methodology

For the reaction bench (`RxN`), we consider two chemical processes. In both processes, each episode begins with a vessel containing 4 mols of diethyl ether, and operates for 20 steps. We chose 20 steps because it's long enough that the agent can explore the space to find the optimal behaviour but short enough that the reward acquired at the end of the episode can be propagated back through the trajectory. In the first process, the agent has access to 1.0 mol each of 1, 2, 3-chlorohexane, and sodium, where the system dynamics are defined by the Wurtz reaction outlined above. Each episode, a target material is specified to the agent via length 7 one-hot vector where the first 6 indices represent the 6 Wurtz reaction products in Table 6.1 and the last represents NaCl. After the 20 steps have elapsed, the agent receives a reward equal to the molar amount of the target material produced.

In the second experiment, we introduce a new set of reaction dynamics given by

$$\begin{aligned} A + B + C &\rightarrow E \\ A + D &\rightarrow F \\ B + D &\rightarrow G \\ C + D &\rightarrow H \\ F + G + H &\rightarrow I \end{aligned} \tag{6.6}$$

where the agent has access to 1.0 mol of A, B, C and 3.0 mol of D. We introduce this second reaction explore different mechanics required in the optimal solution. This reaction includes undesired and intermediate products, adding difficulty to the

problem. Each episode, a target material is specified to the agent via length 5 one-hot vector with indices representing E, F, G, H, and I. If the target is E, the agent receives a reward equal to the molar amount of E produced after the 20 steps have elapsed. Otherwise, the agent receives a reward equal to the difference in molar amounts between the target material and E after the 20 steps have elapsed. Here, E is an undesired material. The reaction $A + B + C \rightarrow E$ occurs quickly relative to the others, adding difficulty to the reaction when E is not the target.

6.5.2 Extraction Bench Methodology

For the extraction bench (`ExT`), we consider a layer-separation process where the agent operates for up to 50 steps. We chose a larger number of steps in this experiment because the optimal solution is more complicated than the previous bench. Similar to the Wurtz reaction, the target material is specified via length 7 one-hot vector. Each episode begins with a vessel containing 4 mols of diethyl ether, 1 mol of dissolved sodium chloride, and 1 mol of one of the 6 Wurtz reaction products in Table 6.1. The Wurtz reaction product contained in the vessel is the same as the target material, unless the target material is sodium chloride, in which case dodecane is added since sodium chloride is already present. After the episode has ended, the agent receives a reward equal to the change in solute purity of the target material weighted by the molar amount of that target material, where the change in solute purity is relative to the start of the experiment. If the target material is present in multiple vessels, a weighted average of the solute purity across each vessel is used.

As an example, consider when the target material is dodecane. In this experiment, the 1 mol of dissolved sodium chloride becomes 1 mol each of Na^+ and Cl^-, so the initial solute purity of dodecane is 1/3. Suppose we end the experiment with 0.7 mols of dodecane with 0.2 mols each of Na^+ and Cl^- in one vessel, and the

remaining molar amounts in a second vessel. Dodecane has a solute purity of 7/11 and 3/19 in each vessel respectively. The final solute purity of dodecane would be $0.7 \times 7/11 + 0.3 \times 3/19 \approx 0.493$. Thus the agent would receive a reward of 0.159.

6.5.3 Distillation Bench Methodology

For the distillation bench (`DiT`), we consider a similar experimental set-up to the `ExT` one. Each episode begins with a vessel containing 4 mols of diethyl ether, 1 mol of the dissolved target material, and possibly 1 mol of another material. If the target material is sodium chloride, the additional material is dodecane, otherwise the additional material is sodium chloride. After the episode has ended, the agent receives a reward calculated similarly to the `ExT`, except using absolute purity rather than solute purity.

6.6 RL Results

6.6.1 Reaction Bench

Since reaction bench (`RxN`) has a continuous action space, we trained SAC and TD3 in addition to A2C and PPO. For the first experiment, we are looking at the Wurtz reaction dynamics. Given that we know the system dynamics in this case, we have also devised a heuristic agent for the experiment, which we expect to be optimal. Since the Wurtz reaction is a single step process, the optimal behaviour is quite simple. For target R_1-R_2, exclusively add R_1 and R_2 at step 1, and increase the temperature to speed up the reaction in order to produce as much of the target before the experiment ends. Thus the heuristic was designed to follow this behaviour. This heuristic agent achieves an average return of approximately 0.62. Using the performance of this

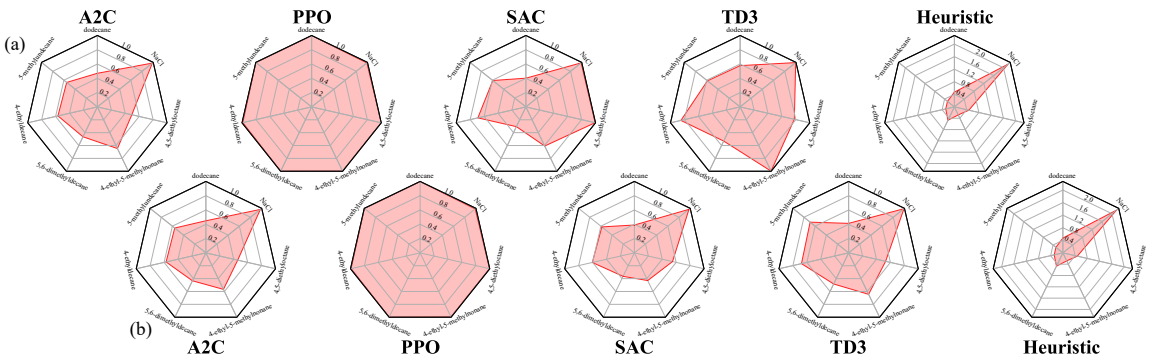


Figure 6.5: Radar graphs detailing the average return of each policy with respect to each target material in Wurtz `RxN`. Panel (a) uses the best policy produced from 10 runs, whereas panel (b) averages across the 10 runs (still using the best policy of each run). Returns of each RL algorithm are relative to the heuristic policy and clipped into the range $[0, \infty)$. Note that we show the unnormalized return values for the heuristic policy so the different scales between targets can be seen. For the RL agents, a return of 0 means the agent produces no target material, a return of 1 means the agent produced as much target material as the heuristic. Here, the PPO agents consistently outperform the A2C, SAC, and TD3 agents for all 7 target materials. Target materials with high returns across each algorithm (such as sodium chloride) appear to be easier tasks to learn, where target materials with less consistent returns across each algorithm (such as 5,6-dimethyldecane) appear to be more difficult tasks to learn.

heuristic as a reference, the best and mean relative performances of the agents trained with each algorithm is shown in Figure 6.5. Each algorithm can consistently give rise to agents that produce sodium chloride when requested. Since this is a by-product of all reactions in our set-up, it is the easiest product to create. While the other products are not hard to produce either, they require specific reactants, and in order to maximize the yield, they require the absence of other reactants. The PPO agents are able to match the heuristic agent for all targets, while some SAC and TD3 agents are able to come close on a few targets. A2C only comes close to the heuristic on producing sodium chloride.

The average return as a function of training steps for each algorithm is shown in Figure 6.6. On average, the agents trained with each algorithm are able to achieve a

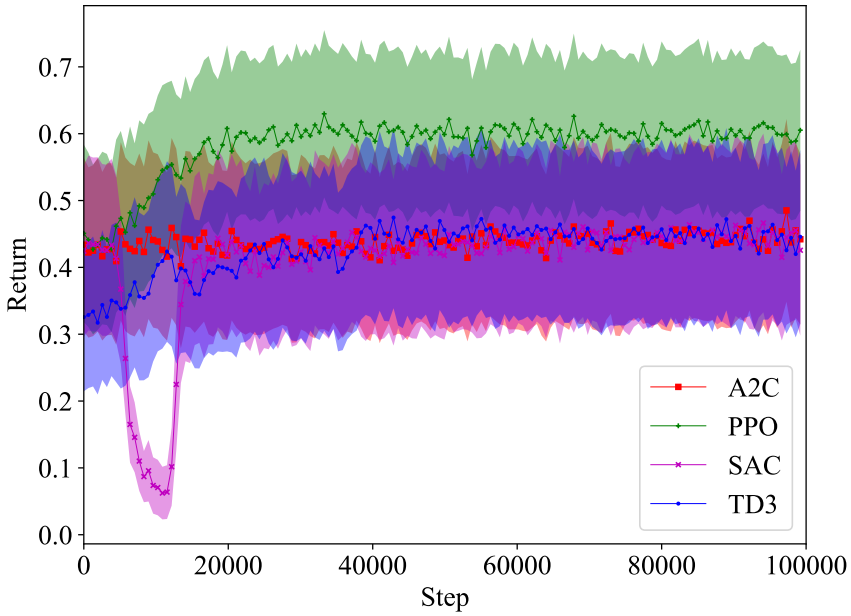


Figure 6.6: Wurtz `RxN`, average return with $\sigma/5$ shaded, 10 runs for each algorithm with 10 environments in parallel per run, 1M (100K sequential steps x 10 environments) total steps per run, averages are over 3200 returns. The performance of each algorithm converges before 300K total steps, with only PPO converging on an optimal policy. Despite training for an additional 700K total steps, A2C, SAC, and TD3 were not able to escape the local maxima they converged to.

return of at least 0.4. This is expected as even an agent taking random actions can achieve an average return of approximately 0.44. The agents trained with A2C, SAC, and TD3 do not perform much better than a random agent in most cases, however the ones trained with PPO significantly outperform it. While on average, A2C, SAC, and TD3 have similar performance, we saw in Figure 6.5 that the best performing SAC and TD3 agents outperformed the best A2C agents.

The second `RxN` experiment uses reaction dynamics more complicated than the Wurtz reaction. In the Wurtz reaction, the agent need only add the required reactants for the desired product all together. In this new reaction, this is still true for some desired products, however not all of them. Similarly to the previous experiment, we also devised a heuristic agent for this experiment, which achieves an average return

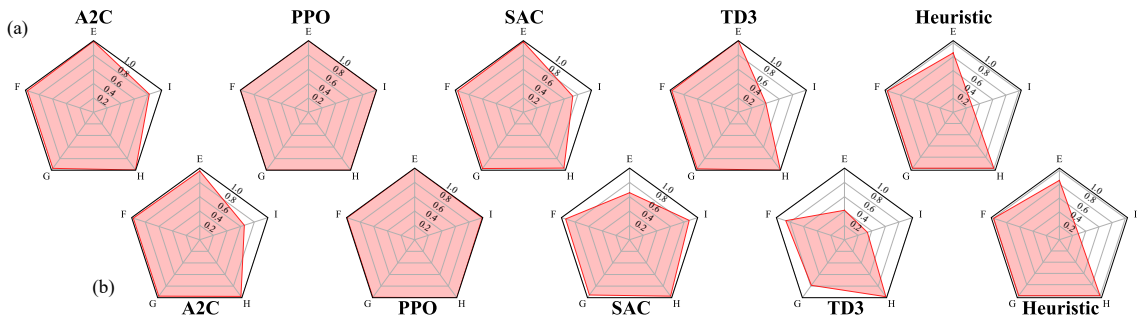


Figure 6.7: Radar Graph detailing the average return of each policy with respect to each target material in Fictitious `RxN`. Panel a) uses the best policy produced from 10 runs, whereas panel b) averages across the 10 runs (still using the best policy of each run). Returns of each RL algorithm are relative to the heuristic policy and clipped into the range $[0, \infty)$. Note that we show the unnormalized return values for the heuristic policy so the different scales between targets can be seen. Again, the PPO agents consistently outperform the A2C, SAC, and TD3 agents for all 5 target materials, however it is not as significant of a gap as in Wurtz `RxN`. Target materials with high returns across each algorithm (such as F, G, and H) appear to be easier tasks to learn, where target materials with less consistent return across each algorithm (such as E and I) appear to be more difficult tasks to learn.

of approximately 0.83. For target materials E, F, G, and H, the required reaction is a single step process like before. Therefore the optimal behaviour is to exclusively add the required reactants at step 1, and increase the temperature to speed up the reaction in order to produce as much of the target before the experiment ends. For target material I, the required reaction is a two step process, allowing for variation in how to material is produced. While all four reactants are required to produce I, adding them all at once would also produce E, wasting needed materials. Hence the optimal behaviour is not necessarily producing all intermediate products simultaneously. As any two of the three intermediates can be safely produced simultaneously, the heuristic policy is designed to add only the reactants required for two intermediates products (we arbitrarily choose F and G). Given the limited number possibilities it is easily determined by brute force that step 6 is the optimal step for the heuristic policy to add the reactants required to create the third intermediate products.

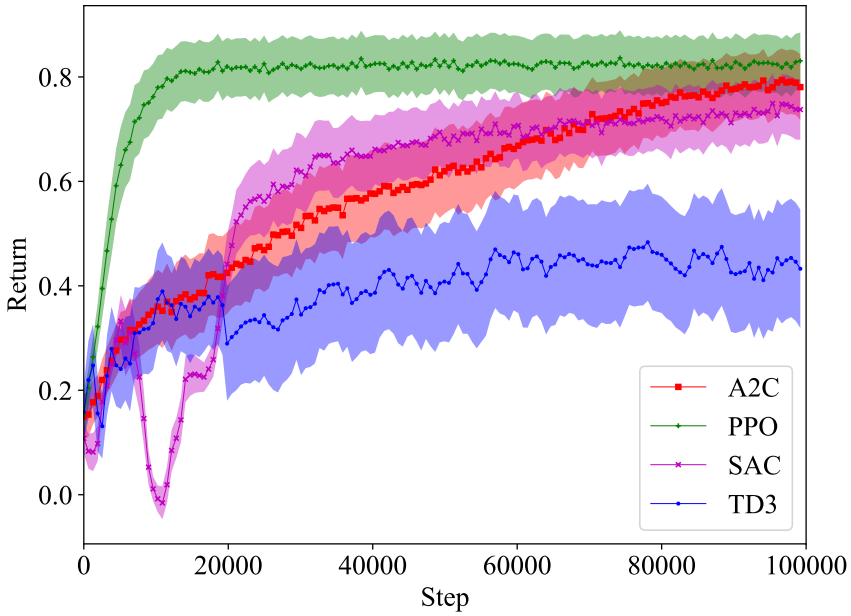


Figure 6.8: Fictitious `RxN`, average return with $\sigma/5$ shaded, 10 runs for each algorithm with 10 environments in parallel per run, 1M (100K sequential steps x 10 environments) total steps per run, averages are over 3200 returns. PPO quickly converges to an optimal policy, like in Wurtz `RxN`. Unlike in Wurtz `RxN`, the other algorithms take much longer to converge. While they still converge to sub-optimal performances, the gap between optimal performance is less severe.

Using the performance of the heuristic agent as reference again, the best and mean relative performances of the agents trained with each algorithm are shown in Figure 6.7. Once again, PPO consistently produces agents that can match the performance of the heuristic agent. The best performing policies produced by A2C, SAC, and TD3 are able to nearly match the heuristic agent for all desired products excluding I. This is not unexpected as producing I requires producing intermediate products at different times during the reaction. On average, the policies produced by SAC and TD3 however, are unable to match the heuristic agent when asked to produce E. This is also not unexpected, given that producing E is penalized for all other desired products.

Unlike PPO, the other algorithms used appear to be less reliable at producing

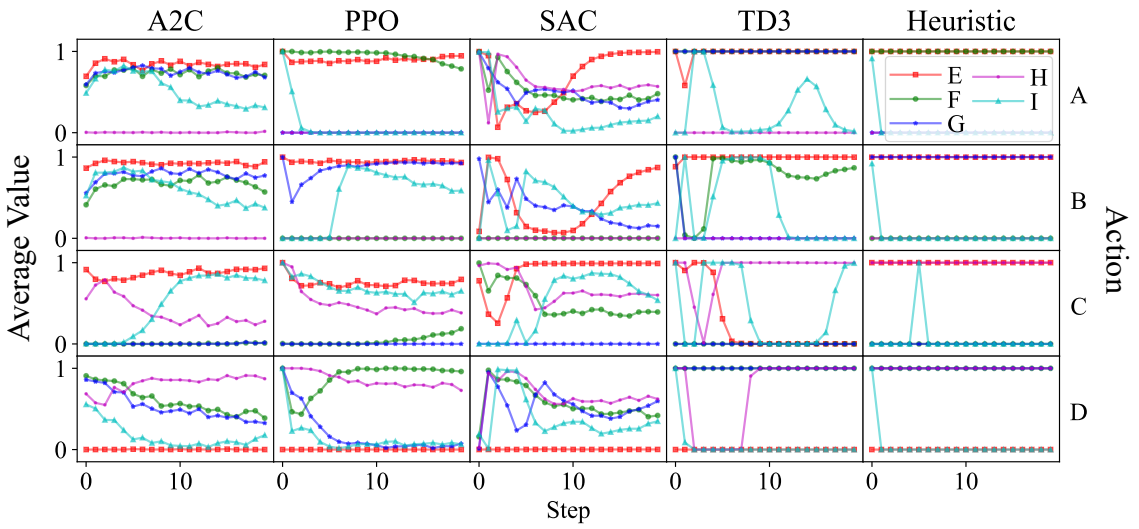


Figure 6.9: Fictitious `RxN`, average value of each action at every step for the best performing policies for each algorithm. The five curves in each box represents the sequence of actions for the five different target materials. Comparing the same curve across a single column outlines how a single policy acts for a single target material. Comparing different curves within a single box outlines how a single policy acts differently between different target materials. Comparing the same curve across a single row outlines how different policies act for the same target material. For actions corresponding to adding material, the curves represent how quickly those materials are added. The well performing policies are the ones that add only the required reactants (such as A2C and SAC), while the best performing policies are the ones that add them according to the right schedule (such as PPO).

these best performing agents. This could be due to PPO learning these policies much faster than the other algorithms, as seen in Figure 6.8. Since PPO converges to optimal behaviour so quickly, there's very little room for variation in the policy. The other algorithms however are slowly converging to non-optimal behaviours, leaving much more room for variation in the policies (and returns) that they converge to.

For the best performing agents produced by each algorithm, the average action values for each target are shown in Figure 6.9. Looking at the heuristic policy, a constant action can be used for each target product, excluding I. When the target is I, the desired action must change after several steps have passed, meaning the agent

cannot just rely on what the specified target is. Note that if all of a material has been added by step t, then it does not matter what value is specified for adding that material at step $t + 1$.

The best performing agent for each algorithm were all able to produce E when requested and Figure 6.9 shows that they each have learned to add A, B, C, and not D. It can also be seen that all four algorithms learned to add two of A, B, or C in addition to D, then add the third one several steps later when I is the target product, mimicking the behaviour of the heuristic policy. Note that even though the heuristic waits to add C, waiting to add A or B instead would be equally optimal. While each algorithm does this, PPO and A2C do so better than the others. PPO is also the only one that succeeds in both of these cases, showing that an RL agent can learn the required behaviour in this system.

6.6.2 Extraction Bench

With the results seen in the `RxN` tests, we now move onto the extraction bench (`ExT`) experiment. Regardless of the target material in our Wurtz extraction experiment, the optimal behaviour is quite similar so we will not focus on the different cases as before. Since the `ExT` uses discrete actions, we replace SAC and TD3 with DQN. We also use what we call PPO-XL which is PPO trained with more environments in parallel. PPO-XL was not implemented for the `RxN` tests as regular PPO was able to achieve optimal results as it is, thus not justifying the increased computational cost associated with PPO-XL.

Unlike the reaction bench, we do not have an analytic solution for this bench, therefore we have devised a heuristic policy for this experiment based on what an undergraduate chemist would learn. These lessons involve adding a solvent of opposite polarity to the existing solution (i.e. adding a non-polar solvent to a vessel containing

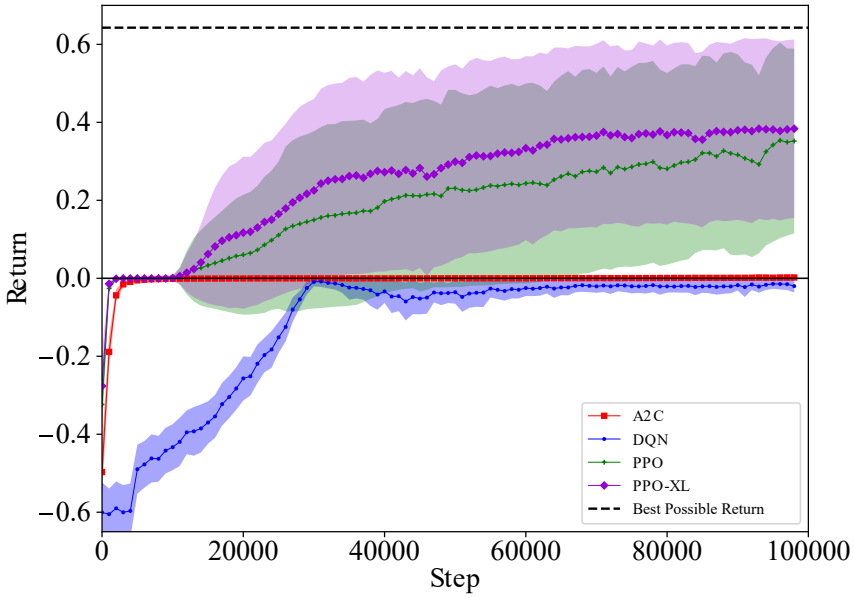


Figure 6.10: Wurtz `ExT`, average return with σ shaded, 30 runs for each algorithm with 1M total steps per run (2M for PPO-XL). For each run, returns are averaged over 1000 steps (only using terminated episodes). The mean and standard deviation are then calculated across the 30 runs (σ is calculated from 30 points). The PPO and PPO-XL agents consistently acquire positive returns, even approaching the theoretical maximum in some cases. The A2C agents learn policies which perform equivalently to ending the experiment immediately and are unable to escape those local maxima. The DQN agents acquire negative return, which is a worse performance than not running the experiment.

a polar solvent solution or vice-versa), mixing everything, letting it settle until distinct layers are formed, and separating the two solvents into separate vessels. The vessel containing the solvent with a similar polarity to the target material is kept while the other vessel is discarded. Thus, our heuristic policy is designed mimic this behaviour. However, as the dynamics are more complex we do not necessarily expect it to be optimal.

As seen in Figure 6.10, the agents trained with A2C do not achieve a return above zero, while the agents trained with DQN ended up achieving a negative return. Not only do both PPO and PPO-XL produce agents that achieve significantly more reward than the other algorithms, they are able to outperform the heuristic policy

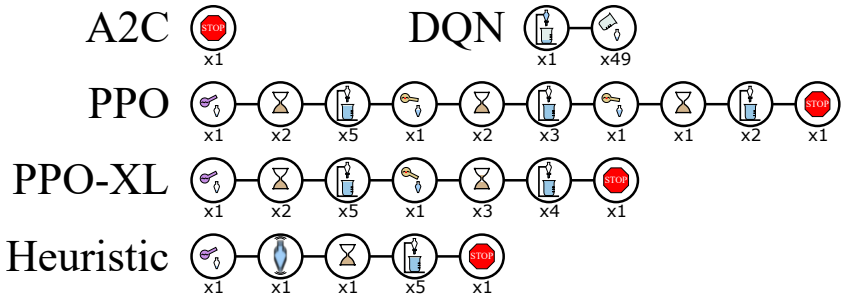


Figure 6.11: Wurtz `ExT`, the sequence of actions with the highest return when dodecane is the target material seen during the rollout of the best performing policy learned by each algorithm. Each picture represents an action and average value described by Figure 6.3(d). The number beneath the image represents how many times that action was repeated. While it is more difficult to interpret these policies than with `RxN`, similarities can be seen between the PPO, PPO-XL, and heuristic policies, explaining their high performances. The A2C policy uses a similar action set, however in a different order, outlining the precision required by the agent. The DQN policy use many actions that either undo previous actions or do nothing in that specific state.

as well. On average, the best performing agent trained with PPO-XL manages to achieve a return of approximately 0.1 higher than the heuristic (see Figure 6.10), resulting in roughly a 10% higher solute purity. While there is a large variance in the final performance of the agents trained with PPO and PPO-XL, they consistently outperform the agents trained with the other algorithms.

As shown in Figure 6.11, the action sequences of the policies learned from A2C, DQN, and PPO are quite different. The action sequences of the policies learned by PPO and PPO-XL are much more similar, as expected. The first half of these sequences are comparable to the heuristic, however the agents in both cases have learned a second component to the trajectory to achieve that extra return. Interestingly, both PPO and PPO-XL agents have learned to end the experiment when they achieve the desired results, whereas the A2C and DQN agents do not. PPO once again shows that an RL agent can learn the required behaviour in this system.

6.6.3 Distillation Bench

Lastly, we now move onto the final experiment, distillation bench (`DiT`). Similar to `ExT`, the desired target material in the Wurtz distillation experiment does not have much effect on the optimal behaviour so we will not focus on the different target cases. Instead we will focus on the different cases of when salt is and is not present with another material in the initial distillation vessel. Note that a single agent operates on both of these cases, not two agents trained independently on each case.

As before, we have devised a heuristic policy and as with the `RxN` experiments, we expect it to be optimal once again. In distillation, the optimal behaviour is to heat the vessel until everything with a boiling point lower than the target material has boiled off, discarding the boiled off contents, then continuing to heat the vessel until just the target material has boiled off, condensing in a separate vessel. Our heuristic policy is designed to mimic this behaviour.

In Figure 6.12 we can see that on average, the algorithms (excluding A2C) converge faster than in the other experiments, however, there is much less variation in return compared to before. For the case when no salt is present in the distillation vessel, the best-performing agents trained with each algorithm learn a very similar policy to the heuristic one, as seen in Figure 6.13. They heat the vessel until the solvent has boiled away, then end the experiment. For the case when salt and additional material are present, the best-performing agents trained with PPO and PPO-XL modify their actions similar to the heuristic policy, achieving the optimal return in both cases. The best-performing agent trained with A2C modifies their actions in a similar fashion, however, it does so in a way that also achieves a much lower return. The best-performing agent trained with DQN makes much more significant changes to their policy, however, it still achieves a return closer to optimal than A2C. This

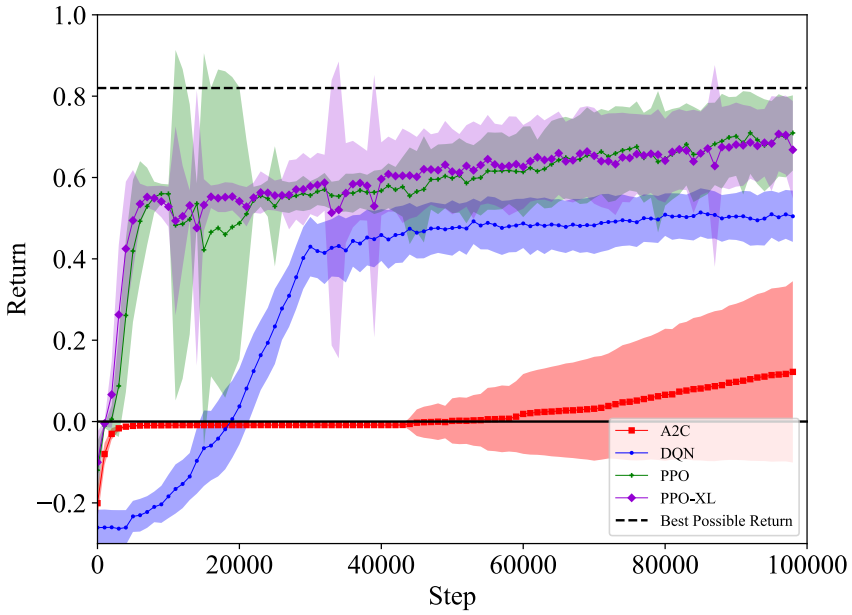


Figure 6.12: Wurtz `DiT`, average return with σ shaded, 30 runs for each algorithm with 1M total steps per run (2M for PPO-XL). For each run, returns are averaged over 1000 steps (only using terminated episodes). The mean and standard deviation are then calculated across the 30 runs (σ is calculated from 30 points). The DQN, PPO, and PPO-XL agents consistently acquire positive returns whereas the A2C agents only get positive returns on average. While DQN and PPO acquire similar returns on average, the variance with PPO is much higher, meaning the best performing PPO policy outperforms the best DQN policy. The PPO-XL policies outperform the other algorithms both on average and in the best case scenarios.

shows that the expected behaviour in our final bench can also be learned by an RL agent.

In this chapter, we choose to use DQN as a representative of the Q-learning family of algorithms since it is a very standard benchmark in RL, and the central themes that were initially introduced in DQN by Mnih et al. [7] led to an explosion of different RL techniques that emerged subsequently. DQN is known to overestimate the values for multiple states and Double DQN (DDQN) [134] was introduced to decouple action selection and action evaluation. However, DDQN may not be necessarily better than DQN in all settings. It has been shown that DDQN suffers from an underestimation

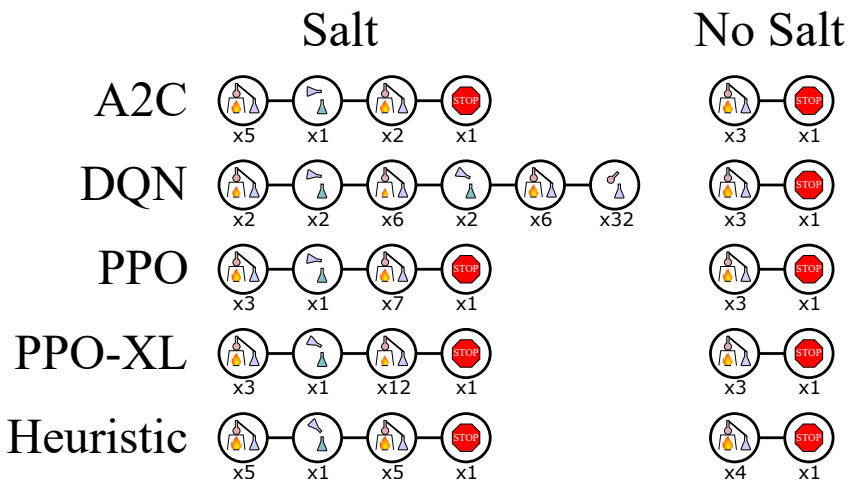


Figure 6.13: Wurtz `DiT`, the sequences of actions with the highest return produced by the best performing policy learned with each algorithm for two cases: when salt is and is not initially present in the distillation vessel with another material. Each picture represents an action and average value described by Figure 6.4. The number beneath the image represents how many times that action was repeated. The PPO, PPO-XL, and heuristic policies are nearly identical in both cases, with only minor differences. When no salt is present, the A2C and DQN policies are similar to the others, however when salt is present they continue to behave as if it is not.

bias that is equally bad for performance [135]. Duelling DQN [136] was suggested as another improvement over DQN, where the Q-values are split into two parts, the value function and the advantage function. Though Duelling DQN shows better performances as compared to DQN, it is known to take vastly more training time and require larger networks as compared to DQN [137]. Similarly, several enhancements to experience replay techniques, like the prioritized experience replay [138] and the hindsight experience replay [139], have been proposed, that improve performances but take a longer training time as compared to DQN. In future work, we would like to try some recent variants of DQN and elaborately study the performance improvements obtained as a function of computational cost when these methods are used in ChemGymRL.

6.7 Limitations

ChemGymRL has limitations; any reaction or material that one wishes to model must be predefined with all properties specified by the user. Additionally, the solvent dynamics are modeled using simple approximations and while they suffice for these introductory tests, they would not for real-world chemistry.

As previously mentioned, the ChemGymRL framework was designed in a modular fashion for the ease of improvement. The differential equations used to model the reactions could be replaced with a molecular dynamics simulation. This would allow `RxN` to operate with on a more generalizable rule-set. Without having to manually define the possible reactions, the `RxN` could be used to discover new, more efficient reaction pathways by an RL agent. Currently, the reward metric used in `RxN` is the molar amount of desired material produced by the agent. If this metric was changed to reflect a certain desired property for the produced material, then the `RxN` could be used for drug discovery. Making similar improvements to `ExT` and `DiT`, the RL agent could then learn to purify these new discoveries.

6.8 Future Work

In the future, the lab manager environment will be formatted in a way that allows an RL agent to operate in it. Using pre-trained agents for the individual benches, the lab manager agent would decide which vessel to give to which bench while also specifying the desired goal to each bench, in order to achieve the lab manager's own goal. The lab manager agent would make proper use of the agentless characterization bench introduced here as well, which will have characterization methods with associated costs. In addition to this, the implementation of new benches will be ex-

plored, allowing more complicated experiments to be conducted and new insights into the benefits and challenges of the integration of RL into automated chemistry and self-driving labs.

6.9 Conclusions

We have introduced and outlined the ChemGymRL interactive framework for RL in chemistry. We have included three benches that RL agents can operate and learn in. We also include a characterization bench for making observations and presented directions for improvement. To show these benches are operational, we have successfully, and reproducibly, trained at least one RL agent on each of them. Included in this framework is a vessel state format compatible with each bench, therefore allowing the outputs of one bench to be the input to another.

In the Wurtz `RxN` experiment, A2C, SAC, and TD3 were not able to show better performances than an agent taking random actions, where PPO was able to achieve optimal returns on all targets. In the second `RxN` experiment, A2C, SAC, and TD3 were able to show performances that achieves optimal returns for one of the two difficult tasks, whereas PPO was able to achieve optimal returns on both.

In the Wurtz `ExT` experiment, A2C and DQN were not able to produce agents that perform better than doing nothing, whereas PPO was able to achieve higher returns than the devised heuristics. In the Wurtz `DiT` experiment each algorithm was able to produce an agent that performs better than doing nothing and much better than an agent taking random actions.

This chemistry laboratory environment illustrates yet another way in which the structure of a decision problem shapes the behaviour of learning agents—here through the branching and compositional nature of the available trajectories. Each bench

presents its own set of feasible actions and local paths, but the transition from one bench to the next multiplies these possibilities, creating a combinatorial expansion in the number of distinct goal-directed routes that an agent might follow. As a result, the structure of the overall task is not merely a spatial or dynamical arrangement, but a layered sequence of interdependent subroutines whose branching patterns define the landscape of feasible solutions. When observations are incomplete or delayed, the agent must infer both its current position within this procedural structure and the consequences of each branch for the downstream path. The observed variation in learning performance across tasks therefore reflects how easily the agent can piece together this multi-stage trajectory structure from limited experience. In contrast to the continuous, dynamically constrained paths of the navigation and control settings in earlier chapters, the laboratory domain highlights how procedural branching and combinatorial growth introduce a different—but structurally analogous—form of complexity in goal-directed decision making.

Chapter 7

Conclusions

With the rise in popularity of optimal control problems and widespread applicability of their solutions, this thesis focuses on understanding them through the lens of RL. Evolving beyond video games and simple robotics, applications of RL have extended to critical real-world domains including self-driving cars, healthcare, and conversational systems. These applications highlight the increasing importance of understanding not only how RL algorithms perform, but also why they succeed or fail in particular settings.

Our investigation has focused on how structural properties of MDPs and POMDPs shape the learning challenges encountered by RL algorithms, and how mathematical simplifications can shed light on their behaviour. By analyzing what makes certain problems more difficult to learn than others, and by employing simpler mathematical frameworks to improve interpretability, this research contributes to clarifying the complexity underlying modern solutions. Although navigation-style problems were used as the primary vehicle for presenting results, the insights developed can be generalized across domains.

This work has analyzed and discussed MDPs and their various properties through

the lenses of learning theory, topology, and observability, with a special focus on generalized GOMDPs for their widely applicable use-cases and explainable nature. Three separate specific sets of generalized GOMDPs were then considered, solved with either DP or RL, and the insights from before applied to understand what makes those examples difficult to learn. Each of these problems has its own set of challenges and approaches to solving them, giving a more expansive view into the learning process.

When it comes to SL, PAC-learning is a standard way to measure complexity, however these do not naturally extend to other types of learning, such as RL. The sizes of the state and action space have been the primary focus when creating PAC bounds for RL, with some extensions to state connectivity related properties such as diameter and mixing time. In addition to these, this thesis discusses how less straight-forward properties have connections to complexity, utilizing concepts from sample and topological complexity to quantify the difficulty of RL tasks. The analysis shows how the distribution of possible trajectories and the structure of the state space directly affect the probability of error and learning efficiency.

Using topological complexity and path homotopy classes as justifications, the sample complexity of separated-path MDPs is analyzed to provide a discrete abstraction of continuous generalized GOMDPs. This abstraction illustrates how the sample complexity scales with the number of relevant path homotopy classes, mirroring the discrete partition of the navigational trajectory space inherently produced by its topological complexity.

The Carnot heat engine environments served as the first framework to use these tools for understanding. Comparing the solutions for the fully and partially observable environments, the discussion covers how partial observability affects the learning process for both reinforcement and evolutionary methods. With a navigation space

containing no topological obstructions, the various solutions had a natural steady progression through the trajectory space. The difference in these environments lies solely in the observability with no functional differences.

This is not true for the nautical navigation environments though, where the navigational space contains navigational discontinuities—obstructions that prevent paths from being continuously deformed into one another, also known as topological holes—and the partially observable environment contains unseen perturbations from the fully observable one. This set-up allows for a semi-SL style approach in the method. The approach leverages the information contained in the fully observable environment to approximate a solution in the partially observable one, supporting the idea that information considered useless in the PAC-learning framework may indeed have an impact on sample complexity. The results also show that as the number of navigational discontinuities in the navigational space increases (i.e., higher island density), there is a decrease in success rate and increase in crash rate.

Both of these environments differ from the chemistry environment as they are single task systems. The ChemGymRL framework shows that by breaking down an environment into separated tasks, a set-up similar to the separated-path MDPs is achieved. By splitting up tasks based on homotopy classes (benches), separated learning is enabled, which removes the potential dependency and correlation compared to if the environments are sampled together.

Across the diverse systems studied in this thesis, the behaviour of learning agents has consistently reflected the structure of the goal-directed trajectories available to them. Whether navigating environments with distinct homotopy classes of paths, interacting with physical processes whose dynamics constrain the feasible routes to high-performing behaviour, or operating within laboratory settings where each decision branches into a growing tree of procedural possibilities, agent performance

has been shaped by how clearly the underlying trajectory structure can be inferred from experience. In fully observable settings, the relationships among these paths are explicit, allowing agents to exploit the organization of the task directly. Under uncertainty or partial observability, the same structural relationships remain, but the agent must reconstruct them from incomplete or noisy information, often leading to degraded or qualitatively different behaviour. Taken together, the results demonstrate that differences in learning outcomes across domains arise not from the surface details of each environment, but from how their path structures—continuous, uncertain, or combinatorially branching—interact with the information available to the learner. This perspective provides a unifying way to understand the challenges and successes of reinforcement learning across a wide range of decision-making settings.

An important avenue for future theoretical work involves formalizing the mapping from a continuous GOMDP to a discrete separated-path MDP. Specifically, rigorous criteria must be established to produce a finite, functionally representative subset of path homotopy classes (e.g. of simple paths) before applying separated-path sample bounds.

As the field of AI constantly evolves, growing rapidly, the complexity of the solutions produced grows with it, which in turn allows for more complex problems to be solved. AI tools are used more and more everyday in the general population, and as with all useful tools, they are becoming integral. For this reason, it's crucial that our understanding of these problems and their solutions grows at an equivalent rate. Knowing how difficult a problem is, in a comparative sense, helps us estimate our ability to solve it. Assuming some fixed number of samples n, this work does not necessarily tell us the absolute quality of our current solution; however, it indicates that compared to an easier problem, a harder problem's solution would have a larger failure probability δ for the same error margin ϵ—meaning a lower probability of

achieving high accuracy. Now is the time to open the black box of modern AI and bring its complex inner workings into the light for us all to learn from.

Chapter 8

Appendix

8.1 Algorithm for Computing Topological Complexity

Here we present an original algorithm developed for this work that computes an approximation for the topological complexity of a navigational space. It is worth noting that while this algorithm was designed specifically for estimating the topological complexity in GOMDPs, there exist established algorithms for estimating the related Lyusternik-Schnirelmann category, and adapting those approaches for topological complexity could be an avenue for future improvement. First we approximate a topological space as a graph using Algorithm 8.1. For each pair of vertices (s, t), we compute all the shortest paths between them using the Breadth-First Search algorithm in Algorithm 8.2 and check if each set of shortest paths are "homotopic" to each other (within the limit of discretization) using Algorithm 8.3. This checks if there are any discontinuities between paths. For each pair of neighbour s' to s and t' to t, we check that the shortest paths from s' to t and s to t' are "close" to the paths between s and t using Algorithm 8.4. This checks if there are any disconti-

nuities between neighbours. If these checks succeed, then our topological space as a topological complexity of 1. If not, we generate all connected bi-partitions of the graph and repeat these checks for the individual partitions. If this succeeds for some partition, then the topological complexity is 2. If not, we repeat again, increasing the number of partitions each time using Algorithm 8.5. The topological complexity is the smallest number of partitions required to pass these checks. This last check is approximately finding the open cover such that each set in the cover has a topological complexity of 1.

Algorithm 8.1 Homology-Preserving Grid Approximation via Cubical Complex

Require: Topological space $\mathcal{X} \subset \mathbb{R}^2$ (given as membership oracle χ or sample set), grid step $h > 0$

Ensure: Grid-like cell complex $\mathcal{K} = (V, E, F)$; 1-skeleton $G = (V, E)$ preserves holes (first homology)

$V \leftarrow \emptyset$, $E \leftarrow \emptyset$, $F \leftarrow \emptyset$

Define axis-aligned grid $\mathcal{G} = \{C_{i,j}\}$ with cell size $h \times h$ ▷ Foreground uses 4-adjacency; background uses 8-adjacency

for all cells $C_{i,j} \in \mathcal{G}$ **do**

  if $C_{i,j} \cap \mathcal{X} \neq \emptyset$ **(or** $\exists x \in C_{i,j}$ with $\chi(x) = 1$**) then**

    add vertex $v_{i,j}$ to V

  end if

end for

for all occupied cells $C_{i,j}$ **do**

  for all 4-neighbors $C_{p,q} \in \{C_{i\pm1,j}, C_{i,j\pm1}\}$ **do**

    if $v_{i,j} \in V$ **and** $v_{p,q} \in V$ **then**

      add edge $(v_{i,j}, v_{p,q})$ to E

    end if

  end for

end for

for all index blocks (i, j) **do**

  if $v_{i,j}, v_{i+1,j}, v_{i,j+1}, v_{i+1,j+1} \in V$ **then**

    add square face $f_{i,j}$ to F with boundary

      $\partial f_{i,j} = \{(v_{i,j}, v_{i+1,j}), (v_{i+1,j}, v_{i+1,j+1}), (v_{i+1,j+1}, v_{i,j+1}), (v_{i,j+1}, v_{i,j})\} \subseteq E$

  end if

end for

return $\mathcal{K} = (V, E, F)$ and its 1-skeleton $G = (V, E)$

Algorithm 8.2 Breadth-First Search for All Shortest Paths

Require: Graph $G = (V, E)$, source vertex $s \in V$, target vertex $t \in V$
Ensure: Set of all shortest paths $\Pi(s, t)$

for all $v \in V$ **do** $d(v) \leftarrow \infty$, parents$(v) \leftarrow \emptyset$
$d(s) \leftarrow 0$
Initialize queue $Q \leftarrow [s]$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Dequeue(Q)
    for all neighbours v of u in G **do**
        if $d(v) = \infty$ **then**
            $d(v) \leftarrow d(u) + 1$
            parents$(v) \leftarrow \{u\}$
            Enqueue v into Q
        else if $d(v) = d(u) + 1$ **then**
            parents$(v) \leftarrow$ parents$(v) \cup \{u\}$
        end if
    end for
end while
if $d(t) = \infty$ **then**
    return "No path exists"
else
    function Backtrack(u)
        if $u = s$ **then**
            return $\{\, [s] \,\}$
        else
            $\Pi \leftarrow \emptyset$
            for all $p \in$ parents(u) **do**
                for all $\pi \in$ Backtrack(p) **do**
                    Add $(\pi \,\|\, u)$ to Π ▷ $\pi \,\|\, u$ is path π extended with u
                end for
            end for
            return Π
        end if
    end function
    return Backtrack(t)
end if

Algorithm 8.3 Detection of Discontinuities (Homotopy Classes) Among Shortest Paths

Require: Graph $G = (V, E)$, set of shortest paths $\Pi(s, t)$ between s and t
Ensure: Partition of $\Pi(s, t)$ into homotopy classes $\{\Pi_1, \Pi_2, \dots\}$

function AreHomotopic(π_1, π_2)
    if endpoints of π_1 and π_2 differ **then**
        return false
    end if
    Initialize queue $Q \leftarrow \{\pi_1\}$, visited $\leftarrow \emptyset$
    while $Q \neq \emptyset$ **do**
        $\rho \leftarrow$ Dequeue(Q)
        if $\rho = \pi_2$ **then**
            return true
        end if
        for all local modifications ρ' of ρ **do**
            if $\rho' \notin$ visited **then**
                Enqueue(Q, ρ')
            end if
        end for
        visited $\leftarrow$ visited $\cup\{\rho\}$
    end while
    return false
end function
Initialize all paths unassigned
$\mathcal{C} \leftarrow \emptyset$
for all $\pi \in \Pi(s, t)$ **do**
    if π unassigned **then**
        $C \leftarrow \{\pi\}$
        for all $\pi' \in \Pi(s, t)$ with π' unassigned **do**
            if AreHomotopic(π, π') **then**
                $C \leftarrow C \cup \{\pi'\}$
            end if
        end for
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{C\}$
    end if
end for
if $|\mathcal{C}| > 1$ **then**
    Report "Discontinuities detected"
else
    Report "No discontinuities"
end if
return $\mathcal{C}$

Algorithm 8.4 Neighbour Similarity Check for All Shortest Paths

Require: Graph $G = (V, E)$, source s, target t, base shortest-path set $\Pi(s, t)$ (all homotopic)

Ensure: **true** iff all shortest paths from s' to t (for $s' \sim s$) and from s to t' (for $t' \sim t$) are similar to $\Pi(s, t)$

Compute $d_s(\cdot)$ by BFS from s
Compute $d^t(\cdot)$ by BFS on G reversed from t ▷ $d^t(v) = \mathrm{dist}(v, t)$
$D \leftarrow d_s(t)$
function DropHead(π)
    return π without its first vertex
end function
function DropTail(π)
    return π without its last vertex
end function
function Prepend(u, π)
    return path with u added at the front of π
end function
function Append(π, u)
    return path with u added at the end of π
end function
function AllShortest(a, b)
    return the set of all shortest $a \to b$ paths (Algorithm 8.2)
end function
function SimilarFromSPrime(s')
    $\Delta \leftarrow d^t(s') - d^t(s)$ ▷ negative = closer, positive = further
    $\Sigma \leftarrow$ AllShortest(s', t)
    for all $\sigma \in \Sigma$ **do**
        if $\Delta = 1$ **then** ▷ s' further from t than s
            $C \leftarrow \{\text{DropHead}(\sigma)\}$
        else if $\Delta = -1$ **then** ▷ s' closer to t than s
            $C \leftarrow \{\text{Prepend}(s, \sigma)\}$
        else ▷ $\Delta = 0$
            $C \leftarrow \{\text{DropHead}(\sigma),\ \text{Prepend}(s, \sigma)\}$
        end if
        if $C \cap \Pi(s, t) = \emptyset$ **then**
            return false
        end if
    end for
    return true
end function

```
36: function SimilarToTPrime(t′)
37:     Δ ← d_s(t′) − d_s(t)                    ▷ negative = closer, positive = further
38:     Σ ← AllShortest(s, t′)
39:     for all σ ∈ Σ do
40:         if Δ = 1 then                          ▷ t′ further from s than t
41:             C ← {DropTail(σ)}
42:         else if Δ = −1 then                    ▷ t′ closer to s than t
43:             C ← {Append(σ, t)}
44:         else                                   ▷ Δ = 0
45:             C ← {DropTail(σ), Append(σ, t)}
46:         end if
47:         if C ∩ Π(s, t) = ∅ then
48:             return false
49:         end if
50:     end for
51:     return true
52: end function
53: for all s′ ∈ V : (s, s′) ∈ E do
54:     if SimilarFromSPrime(s′) = false then
55:         return false
56:     end if
57: end for
58: for all t′ ∈ V : (t, t′) ∈ E do
59:     if SimilarToTPrime(t′) = false then
60:         return false
61:     end if
62: end for
63: return true
```

Algorithm 8.5 Partition-Based Homotopy and Similarity Check

Require: Graph $G = (V, E)$
Ensure: Minimum number of partitions $k_{\min}$ such that all shortest-path checks succeed

$k \leftarrow 1$ ▷ start with the whole graph
while true **do**
  Generate all partitions of V into k connected components: $\mathcal{P}_k = \{P_1, \ldots, P_k\}$
  success $\leftarrow$ true
  for all components $P_i \in \mathcal{P}_k$ **do**
    Let G_i be the subgraph induced by P_i
    for all vertex pairs $(s, t) \in V(G_i)$ **do**
      Compute all shortest paths $\Pi(s, t)$ in G_i (Algorithm 8.2)
      if no paths exists **or** paths in $\Pi(s, t)$ are not homotopic (Algorithm 8.3) **or** neighbor similarity check fails (Algorithm 8.4) **then**
        success $\leftarrow$ false
        break out of vertex-pair loop
      end if
    end for
    if *success* = false **then break** out of component loop
    end if
  end for
  if *success* = true **then**
    return $k_{\min} \leftarrow k$
  end if
  $k \leftarrow k + 1$
  if $k > |V|$ **then** ▷ cannot partition further
    return $k_{\min} \leftarrow \infty$
  end if
end while

8.2 Key Lemmas from Azar et al.

The sample complexity bounds derived in Chapter 3 rely on the following lemmas from Azar et al. [35]. We restate them here for convenience with updated notation.

Lemma (Lemma 8 from Azar et al. [35]). *Let p_M be the (constant) transition probability for all states in $\mathcal{Y}^1$ and let T_{sa} be the number of times state-action pair (s, a) has been observed. Define $\alpha = 2(1 - \gamma p_M)^2 \epsilon / \gamma^2$. Then for every RL algorithm, there exists an MDP $M \in \mathcal{M}$ and constants $c_1' > 0$ and $c_2' > 0$ such that*

$$Pr(|Q^*(s,a) - Q_{T_{sa}}(s,a)| > \epsilon) > \frac{1}{c_2'} \exp\left(-\frac{c_1' \alpha^2 T_{sa}}{p_M(1-p_M)}\right).$$

With the specific choice $c_1' = 1$ and $c_2' = 6$, this gives Equation 3.7 in Chapter 3 (with p_{sa} generalizing p_M).

Lemma (Lemma 11 from Azar et al. [35]). *Assume that for every RL algorithm and every state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, we have*

$$Pr(|Q^*(s,a) - Q_{T_{sa}}(s,a)| > \epsilon \mid T_{sa} = t_{sa}) > \delta',$$

whenever $t_{sa} < \xi(\epsilon, \delta')$ for some function ξ. Then for any (ϵ, δ, T)-correct algorithm, if $T < N \cdot \xi(\epsilon, \delta/(12N))$, the algorithm cannot be (ϵ, δ, T)-correct on all MDPs in $\mathcal{M}$.

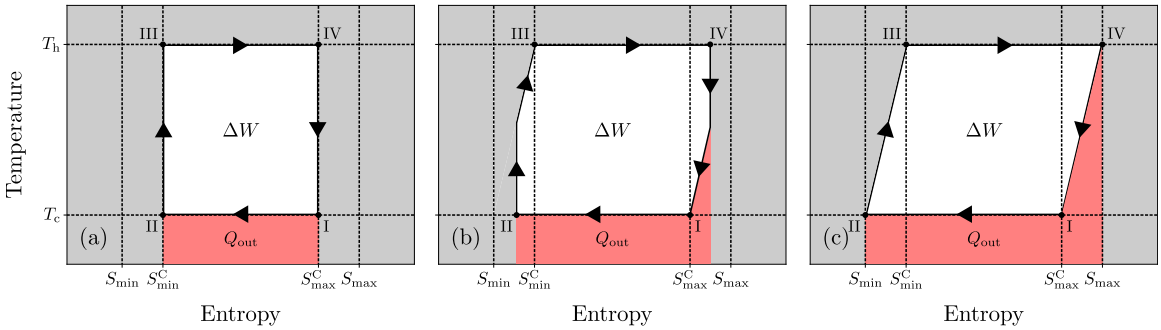


Figure 8.1: T-S diagram for the (a) Carnot, (b) discretized Carnot, and (c) Stirling cycles. The red (dark gray) area (labelled Q_{out}) represents the heat removed from the system, the white area (labelled ΔW) represents the work produced by the system, and the red (dark gray) and white areas together represent the heat added to the system.

8.3 Proof of Carnot's theorem on infinite heat baths

Using T-S diagrams, shown in Figure 8.1, the thermal efficiency of a cycle is determined by $\eta = \Delta W/(\Delta W + Q_{\text{out}})$ where ΔW is the work produced by the system and Q_{out} is the heat removed from the system. Using Figure 8.1(a), the thermal efficiency of the Carnot cycle is given by Equations 4.3, which is independent of the minimum and maximum entropies reach by the Carnot cycle, denoted by $S^{\text{C}}_{\text{min}}$ and $S^{\text{C}}_{\text{max}}$ respectively. For cycles that operate between $S^{\text{C}}_{\text{min}}$ and $S^{\text{C}}_{\text{max}}$, since T_{c} is the minimum temperature the system can reach, η is maximized by maximizing ΔW and minimizing Q_{out}, which results in the Carnot cycle. Now consider cycles that operate outside of the entropy range $S^{\text{C}}_{\text{min}}$ to $S^{\text{C}}_{\text{max}}$, such as the ones shown in Figures 8.1(b) and (c). It is impossible for the system to simultaneously be at T_{h} and an entropy less than $S^{\text{C}}_{\text{min}}$, therefore the ratio of work produced and heat removed by the system between points II and III will be less than that of the Carnot cycle. Similarly, it is impossible for the system to simultaneously be at T_{c} and entropy greater than $S^{\text{C}}_{\text{max}}$,

therefore the ratio of work produced and heat removed by the system between points IV and I will be less than the Carnot cycle. Putting all these together, any cycle operating within $S^{\mathrm{C}}_{\min}$ and $S^{\mathrm{C}}_{\max}$ is at most efficient as the Carnot cycle, and any cycle operating outside of $S^{\mathrm{C}}_{\min}$ and $S^{\mathrm{C}}_{\max}$ is less efficient, therefore Carnot's theorem holds for infinitely many heat baths.

8.4 Nautical Navigation

8.4.1 Value Function and Policy Construction

In Chapter 5, Algorithm 8.6 is used to generate the value function for a chart without water current and Algorithm 8.7 is used to generate a policy with uncertainty incorporated.

Algorithm 8.6 The algorithm used to generate the value function for a chart without water current.

Require: $\mathcal{A}$ contains only movement actions.
Ensure: $V(x,y) = V(x + kx_{\max}, y + ly_{\max})\ \forall\ k,l \in \mathbb{Z}$ and $\forall\ (x,y)$.

$V_0(x,y) \leftarrow 0\ \forall\ (x,y)$
$V_1(x,y) \leftarrow \min_{a\in\mathcal{A}} c(x,y,a)\ \forall\ (x,y)$
$n \leftarrow 1$
while $\|V_n - V_{n-1}\|_\infty > 0$ **do**
    $n \leftarrow n+1$
    $V_n(\cdot) \leftarrow \min_{a\in\mathcal{A}} c(\cdot,a) + \gamma V_{n-1}(a_M(\cdot,W))$
end while
$V(x,y) \leftarrow V_n(x,y)$

8.4.2 Chart Generation Method

The system the agent is navigating in is contained inside a rectangular area with periodic boundary conditions and dimensions $x_{\max}$ and $y_{\max}$. Each of these island

Algorithm 8.7 Algorithm used to generate a policy with uncertainty incorporated.

Initialize position estimate $(\hat{x}, \hat{y}) \leftarrow (x, y)$.
Initialize water current estimate $\hat{W} \leftarrow W(x, y)$.
Initial uncertainty in position and water current as zero.
while $-1 < V(x, y) < 100$ **do**
    Predict minimal value $V_{\min}$ over all movement actions using state estimate and uncertainties.
    if $V_{\min}$ is greater than the cost to measure both position and water current **then**
        $(\hat{x}, \hat{y}) \leftarrow (x, y)$.
        Positional uncertainty $\leftarrow 0$.
        $\hat{W} \leftarrow W(x, y)$.
        Water current uncertainty $\leftarrow 0$.
    else if $V_{\min}$ is greater than the cost to measure position or $V(\hat{x}, \hat{y}) = -1$ **then**
        $(\hat{x}, \hat{y}) \leftarrow (x, y)$.
        Positional uncertainty $\leftarrow 0$.
        Update $\hat{W}$ using positions.
        Water current uncertainty decreases.
    else if $V_{\min}$ is greater than the cost to measure water current **then**
        $\hat{W} \leftarrow W(x, y)$.
        Water current uncertainty $\leftarrow 0$.
    end if
    Predict value for each movement action using estimates and uncertainties.
    Choose movement action with minimal value.
    Increase water current uncertainty by one unit.
    Increase positional uncertainty relative to water current uncertainty.
    Update $(\hat{x}, \hat{y})$ using movement action and $\hat{W}$.
    Update (x, y) using movement action and $W(x, y)$.
end while

obstacles is represented by a 2-dimensional Gaussian function, defined as

$$g(x,y) = Ae^{-\left(a(x-x_0)^2+2b(x-x_0)(y-y_0)+c(y-y_0)^2\right)}, \tag{8.1}$$

where $x_0 \in [0, x_{\max})$, $y_0 \in [0, y_{\max})$, $A \in [1,2]$, $a, c \in [1, \infty)$ are independently sampled uniformly from their respective ranges and $b \in (-\sqrt{ac}, \sqrt{ac})$ is also sampled uniformly, (however it is dependent on a and c). We then define the land function as

$$f(x,y) = \sum_{i=1}^{N} \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} g_i(x + jx_{\max}, y + ky_{\max}), \tag{8.2}$$

where N is the number of islands and the parameters for each g_i are sampled as described above and independently from each other island. While true periodic boundary conditions require the infinite sums, the bounds $-1 \leq j, k \leq 1$ are sufficient for our purposes.

8.4.3 Water Current Generation Method

For each island $g_i(x,y)$, the water current $W(x,y)$ vector at position (x,y) has direction given by

$$\begin{aligned} w(x,y) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \sum_{i=1}^{N} \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \Big((-1)^{m_i(x,y)} \times \\ \nabla g_i(x + jx_{\max}, y + ky_{\max}) \Big), \end{aligned} \tag{8.3}$$

where each m_i only returns the discrete values 0 and 1 and is chosen to maximize $\|w(x,y)\|_2$. The water current $W(x,y)$ function is then defined as

$$W(x,y) = \frac{w_{\max} - \|w(x,y)\|_2}{2w_{\max}\|w(x,y)\|_2} w(x,y) \tag{8.4}$$

if $f(x,y) < 0.9$ and $0 < \|w(x,y)\|_2 \leq w_{\max}$, $W(x,y) = 0$ if $f(x,y) \geq 0.9$ or $\|w(x,y)\|_2 > w_{\max}$, and

$$W(x,y) = \frac{w_{\max}}{\|w(x,y)\|_2} w(x,y) \tag{8.5}$$

if $f(x,y) < 0.9$ and $w(x,y) = 0$, where $w_{\max} \geq 0$. In other words, the water current vector at (x,y) is perpendicular to $\nabla f(x,y)$ with magnitude bounded and related to $-\|\nabla f(x,y)\|_2$.

8.5 ChemGymRL

8.5.1 Reactions

As an example, consider the reaction $X + Y \rightarrow Z$. Its system of differential equations is defined as

$$\begin{aligned} \frac{\partial[X]}{\partial t} &= -k[X][Y] \\ \frac{\partial[Y]}{\partial t} &= -k[X][Y] \\ \frac{\partial[Z]}{\partial t} &= k[X][Y], \end{aligned} \tag{8.6}$$

where $[X]$ denotes the concentration of material X and k is the rate constant, defined by the Arrhenius equation

$$k = Ae^{-\frac{E_a}{RT}}, \tag{8.7}$$

where A is the pre-exponential factor, E_a is the activation energy, R is the ideal gas constant, and T is the temperature. The possible reactions that can occur in this bench are determined by selecting a family of reactions from a directory of supported, available reactions. New reactions can be easily added to this bench by following the provided template. This reaction template includes parameters for the pre-exponential factors, and the activation energies, the stoichiometric coefficients of

each material for each reaction.

Chemical reactions of this form can be considered as special cases of the initial value problem:

$$\frac{d\vec{y}}{dt} = \vec{f}(t, \vec{y}), \tag{8.8}$$

where $\frac{\partial \vec{f}}{\partial t} = \vec{0}$. Note, $\vec{y}$ are your concentrations and $\vec{f}(t, \vec{y})$ are your rates. The RK45 (Runge-Kutta-Fehlberg) method [140] was used to solve these ODE equations and obtain new chemical concentrations as time passes.

8.5.2 Extractions

In the layer separation equations we present here, we consider the solvents settling as moving forward in time and mixing the contents of a vessel as moving backwards through time. This `ExT` uses Gaussian distributions to represent the solvent layers. For solvents L_1, L_2, ..., L_n, the center of solvent L_i, or mean of the Gaussian, is given by

$$\mu_{L_i} = (t - t_{\text{mix}}) \sum_{j=1, j\neq i}^{n} (D_{L_j} - D_{L_i}), \tag{8.9}$$

where $t_{\text{mix}} \leq t$ is the time value assigned to a fully mixed solution and D_{L_i} is the density of L_i. The spread of solvent L_i, or the variance of the Gaussian, is given by

$$\sigma_{L_i}^2 = \frac{1}{\sqrt{2\pi}} e^{-t}. \tag{8.10}$$

While these solvents separate, the solutes are being dispersed based on their relative polarities and amounts of the solvents. In the `ExT`, the relative solute amount at time

t in solvent L is defined by

$$S_{L,t} = \begin{cases} S_L^*(t) + \frac{t-t_{\text{mix}}}{t'-t_{\text{mix}}}(S_{L,t'} - S_L^*(t')) & t < t' \\ S_L^*(t) + S_{L,t'} - S_L^*(t') & t > t' \\ S_{L,t'} & t = t' \end{cases}, \tag{8.11}$$

with $t' = t - \Delta t$ and

$$S_L^*(t) = \left([S]\frac{[L]}{\sum_{l\in\mathcal{L}}[l]}\right) e^{30(t_{\text{mix}}-t)} + \frac{1 - \frac{|P_S - P_L|}{\sum_{l\in\mathcal{L}}|P_S - P_l|}}{1 - \frac{\sum_{l\in\mathcal{L}}[l]|P_S - P_l|}{(\sum_{l\in\mathcal{L}}[l])(\sum_{l\in\mathcal{L}}|P_S - P_l|)}} \left(1 - e^{30(t_{\text{mix}}-t)}\right), \tag{8.12}$$

where $[X]$ is the total concentration of X in the vessel, $\mathcal{L}$ is the set of all solvents in the vessel, and P_X is the polarity of X.

Bibliography

[1] Chris Beeler, Uladzimir Yahorau, Rory Coles, Kyle Mills, Stephen Whitelam, and Isaac Tamblyn. Optimizing thermodynamic trajectories using evolutionary and gradient-based reinforcement learning. *Physical Review E*, 104(6):064128, 2021.

[2] Chris Beeler. *Perpetually playing physics*. University of Ontario Institute of Technology (Canada), 2019.

[3] Chris Beeler, Xinkai Li, Colin Bellinger, Mark Crowley, Maia Fraser, and Isaac Tamblyn. Dynamic programming with incomplete information to overcome navigational uncertainty in POMDPs. *Proceedings of the Canadian Conference on Artificial Intelligence*, 2024. https://caiac.pubpub.org/pub/qdmqsaj7.

[4] Chris Beeler, Sriram Ganapathi Subramanian, Kyle Sprague, Mark Baula, Nouha Chatti, Amanuel Dawit, Xinkai Li, Nicholas Paquin, Mitchell Shahen, Zihan Yang, et al. Chemgymrl: A customizable interactive framework for reinforcement learning for digital chemistry. *Digital Discovery*, 3(4):742–758, 2024.

[5] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[6] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, 2018.

[7] Volodymyr Mnih, 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*, 518(7540):529–533, 2015.

[8] Erika Puiutta and Eric MSP Veith. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, pages 77–95. Springer, 2020.

[9] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[10] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[11] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

[12] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[13] Andrey Markov. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *Dynam Probabilist Syst*, 1:552, 1971.

[14] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[15] Richard Bellman. An introduction to the theory of dynamic programming. Technical report, RAND CORP SANTA MONICA CA, 1953.

[16] Ronald A Howard. Dynamic programming and markov processes. *MIT Press google schola*, 2:39–47, 1960.

[17] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527–535, 1952.

[18] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[19] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[20] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29, 2016.

[21] Ian Osband, Benjamin Van Roy, Daniel J Russo, and Zheng Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019.

[22] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[23] Edward L. Thorndike. *Animal Intelligence: Experimental Studies*. Macmillan, 1911.

[24] B.F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. Appleton-Century, 1938.

[25] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[26] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[27] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[28] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[29] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[30] Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981*, pages 762–770. Springer, 2005.

[31] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[32] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 49:209–232, 2002.

[33] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[34] Sham Machandranath Kakade. *On the sample complexity of reinforcement learning*. University of London, University College London (United Kingdom), 2003.

[35] Mohammad Gheshlaghi Azar, Rémi Munos, and Bert Kappen. On the sample complexity of reinforcement learning with a generative model. *arXiv preprint arXiv:1206.6461*, 2012.

[36] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *International conference on machine learning*, pages 263–272. PMLR, 2017.

[37] Christoph Dann and Emma Brunskill. Sample complexity of episodic fixed-horizon reinforcement learning. *Advances in Neural Information Processing Systems*, 28, 2015.

[38] Nan Jiang and Alekh Agarwal. Open problem: The dependence of sample complexity lower bounds on planning horizon. In *Conference On Learning Theory*, pages 3395–3398. PMLR, 2018.

[39] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

[40] Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Provably efficient reinforcement learning with linear function approximation. In *Conference on learning theory*, pages 2137–2143. PMLR, 2020.

[41] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[42] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *International Conference on Learning Representations*, 2018.

[43] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[44] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[45] Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid reward architecture for reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[46] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[47] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287, 1999.

[48] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[49] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[50] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[51] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.

[52] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[53] Maryanne Garry, Way Ming Chan, Jeffrey Foster, and Linda A Henkel. Large language models (llms) and the institutionalization of misinformation. *Trends in cognitive sciences*, 2024.

[54] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[55] Vikram Krishnamurthy. *Partially observed Markov decision processes*. Cambridge university press, 2016.

[56] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[57] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

[58] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[59] Andrey Kolobov, Daniel Weld, et al. A theory of goal-oriented mdps with dead ends. *arXiv preprint arXiv:1210.4875*, 2012.

[60] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[61] Sven Koenig and Reid G Simmons. Complexity analysis of real-time reinforcement learning. In *AAAI*, volume 93, pages 99–105, 1993.

[62] Farber. Topological complexity of motion planning. *Discrete & Computational Geometry*, 29:211–221, 2003.

[63] Daniel C Cohen and Lucile Vandembroucq. Topological complexity of the klein bottle. *Journal of applied and computational topology*, 1(2):199–213, 2017.

[64] Herbert B Callen. *Thermodynamics and an Introduction to Thermostatistics*. John wiley & sons, 1991.

[65] Sadi Carnot. Reflections on the motive power of fire, and on machines fitted to develop that power. *Paris: Bachelier*, 108, 1824.

[66] Martin Stuart Silberberg. *Principles of general chemistry*. McGraw-Hill Higher Education New York, 2007.

[67] Eivind Bøhn, Erlend M Coates, Signe Moe, and Tor Ame Johansen. Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 523–533. IEEE, 2019.

[68] Luckeciano Carvalho Melo and Marcos Ricardo Omena Albuquerque Máximo. Learning humanoid robot running skills through proximal policy optimization. In *2019 Latin american robotics symposium (LARS), 2019 Brazilian symposium on robotics (SBR) and 2019 workshop on robotics in education (WRE)*, pages 37–42. IEEE, 2019.

[69] Haoran Wei, Xuanzhang Liu, Lena Mashayekhy, and Keith Decker. Mixed-autonomy traffic control with proximal policy optimization. In *2019 IEEE Vehicular Networking Conference (VNC)*, pages 1–8. IEEE, 2019.

[70] Le Zhang, Yanshuo Zhang, Xin Zhao, and Zexiao Zou. Image captioning via proximal policy optimization. *Image and Vision Computing*, 108:104126, 2021.

[71] Leanne De Vree and Raffaella Carloni. Deep reinforcement learning for physics-based musculoskeletal simulations of healthy subjects and transfemoral prostheses' users during normal walking. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29:607–618, 2021.

[72] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[73] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial intelligence*, 121(1-2):49–107, 2000.

[74] Zahra Zamani, Scott Sanner, and Cheng Fang. Symbolic dynamic programming for continuous state and action mdps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 2012.

[75] Bram Bakker, Zoran Zivkovic, and Ben Krose. Hierarchical dynamic programming for robot path planning. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2756–2761. IEEE, 2005.

[76] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(11), 2009.

[77] Tianhao Wu, Yunchang Yang, Simon Du, and Liwei Wang. On reinforcement learning with adversarial corruption and its application to block mdp. In *International Conference on Machine Learning*, pages 11296–11306. PMLR, 2021.

[78] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[79] Elise van der Pol, Daniel Worrall, Herke van Hoof, Frans Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[80] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for dec-pomdps. In *IJCAI*, pages 2009–2015, 2007.

[81] Konstantinos G Papakonstantinou and Masanobu Shinozuka. Planning structural inspection and maintenance policies via dynamic programming and markov processes. part ii: Pomdp implementation. *Reliability Engineering & System Safety*, 130:214–224, 2014.

[82] Scott Sanner and Kristian Kersting. Symbolic dynamic programming for first-order pomdps. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[83] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

[84] Donghwan Lee, Niao He, and Jianghai Hu. Dynamic programming for pomdp with jointly discrete and continuous state-spaces. In *2019 American Control Conference (ACC)*, pages 1250–1255. IEEE, 2019.

[85] Gautam Singh, Skand Peri, Junghyun Kim, Hyunseok Kim, and Sungjin Ahn. Structured world belief for reinforcement learning in pomdp. In *International Conference on Machine Learning*, pages 9744–9755. PMLR, 2021.

[86] Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Reinforcement learning of pomdps using spectral methods. In *Conference on Learning Theory*, pages 193–256. PMLR, 2016.

[87] Sushmita Bhattacharya, Sahil Badyal, Thomas Wheeler, Stephanie Gil, and Dimitri Bertsekas. Reinforcement learning for pomdp: Partitioned rollout and policy iteration with application to autonomous sequential repair problems. *IEEE Robotics and Automation Letters*, 5(3):3967–3974, 2020.

[88] Denis Steckelmacher, Diederik M Roijers, Anna Harutyunyan, Peter Vrancx, Hélène Plisnier, and Ann Nowé. Reinforcement learning in pomdps with memoryless options and option-observation initiation sets. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[89] Vikram Krishnamurthy. Convex stochastic dominance in bayesian localization, filtering, and controlled sensing pomdps. *IEEE Transactions on Information Theory*, 66(5):3187–3201, 2019.

[90] Emrah Akyol, Urbashi Mitra, and Ashutosh Nayyar. Controlled sensing and event based communication for remote estimation. In *2014 52nd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 545–549. IEEE, 2014.

[91] Daphney-Stavroula Zois and Urbashi Mitra. Controlled sensing: a myopic fisher information sensor selection algorithm. In *2014 IEEE Global Communications Conference*, pages 3401–3406. IEEE, 2014.

[92] Colin Bellinger, Rory Coles, Mark Crowley, and Isaac Tamblyn. Active measure reinforcement learning for observation cost minimization. In *Canadian Conference on Artificial Intelligence*, Lecture Notes in Computer Science (LNCS), page 12. Springer, Springer, 2021.

[93] HyunJi Nam, Scott Fleming, and Emma Brunskill. Reinforcement learning with state observation costs in action-contingent noiselessly observable markov decision processes. *Advances in Neural Information Processing Systems*, 34, 2021.

[94] Ashish K Jha, David C Chan, Abigail B Ridgway, Calvin Franz, and David W Bates. Improving safety and eliminating redundant tests: cutting costs in us hospitals. *Health affairs*, 28(5):1475–1484, 2009.

[95] R Zaccone, E Ottaviani, M Figari, and M Altosole. Ship voyage optimization for safe and energy-efficient navigation: A dynamic programming approach. *Ocean engineering*, 153:215–224, 2018.

[96] Xiongfei Geng, Yongcai Wang, Ping Wang, and Baochen Zhang. Motion plan of maritime autonomous surface ships by dynamic programming for collision avoidance and speed optimization. *Sensors*, 19(2):434, 2019.

[97] Ailong Fan, Zheng Wang, Liu Yang, Junteng Wang, and Nikola Vladimir. Multi-stage decision-making method for ship speed optimisation considering inland navigational environment. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 235(2):372–382, 2021.

[98] Davide Corsi, Davide Camponogara, and Alessandro Farinelli. Aquatic navigation: A challenging benchmark for deep reinforcement learning. *arXiv preprint arXiv:2405.20534*, 2024.

[99] J Sebastián Manzano, Wenduan Hou, Sergey S Zalesskiy, Przemyslaw Frei, Hsin Wang, Philip J Kitson, and Leroy Cronin. An autonomous portable platform for universal chemical synthesis. *Nature Chemistry*, 14(11):1311–1318, 2022.

[100] Benjamin P MacLeod, Fraser GL Parlane, Connor C Rupnow, Kevan E Dettelbach, Michael S Elliott, Thomas D Morrissey, Ted H Haley, Oleksii Proskurin, Michael B Rooney, Nina Taherimakhsousi, et al. A self-driving laboratory

advances the pareto front for material properties. *Nature communications*, 13(1):995, 2022.

[101] Benjamin P MacLeod, Fraser GL Parlane, Amanda K Brown, Jason E Hein, and Curtis P Berlinguette. Flexible automation for self-driving laboratories. In *Accelerated Materials Discovery*, pages 105–122. De Gruyter, 2022.

[102] Martin Seifrid, Robert Pollice, Andrés Aguilar-Granda, Zamyla Morgan Chan, Kazuhiro Hotta, Cher Tian Ser, Jenya Vestfrid, Tony C Wu, and Alán Aspuru-Guzik. Autonomous chemical experiments: Challenges and perspectives on establishing a self-driving lab. *Accounts of Chemical Research*, 55(17):2454–2466, 2022.

[103] Martha M Flores-Leonar, Luis M Mejía-Mendoza, Andrés Aguilar-Granda, Benjamin Sanchez-Lengeling, Hermann Tribukait, Carlos Amador-Bedolla, and Alán Aspuru-Guzik. Materials acceleration platforms: On the way to autonomous experimentation. *Current Opinion in Green and Sustainable Chemistry*, 25:100370, 2020.

[104] Yibin Jiang, Daniel Salley, Abhishek Sharma, Graham Keenan, Margaret Mullin, and Leroy Cronin. An artificial intelligence enabled chemical synthesis robot for exploration and optimization of nanomaterials. *Science Advances*, 8(40):eabo2626, 2022.

[105] Shan She, Nicola L Bell, Dazhong Zheng, Jennifer S Mathieson, Maria D Castro, De-Liang Long, Jesko Koehnke, and Leroy Cronin. Robotic synthesis of peptides containing metal-oxide-based amino acids. *Chem*, 8(10):2734–2748, 2022.

[106] Dario Caramelli, Jarosław M Granda, S Hessam M Mehr, Dario Cambié, Alon B Henson, and Leroy Cronin. Discovering new chemistry with an autonomous

robotic platform driven by a reactivity-seeking neural network. *ACS Central Science*, 7(11):1821–1830, 2021.

[107] Hatem Fakhruldeen, Gabriella Pizzuto, Jakub Glowacki, and Andrew Ian Cooper. Archemist: Autonomous robotic chemistry system architecture. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6013–6019. IEEE, 2022.

[108] Michael B Rooney, Benjamin P MacLeod, Ryan Oldford, Zachary J Thompson, Kolby L White, Justin Tungjunyatham, Brian J Stankiewicz, and Curtis P Berlinguette. A self-driving laboratory designed to accelerate the discovery of adhesive materials. *Digital Discovery*, 1(4):382–389, 2022.

[109] Riley J Hickman, Pauric Bannigan, Zeqing Bao, Alán Aspuru-Guzik, and Christine Allen. Self-driving laboratories: A paradigm shift in nanomedicine development. *Matter*, 6(4):1071–1081, 2023.

[110] Jeffrey A Bennett and Milad Abolhasani. Autonomous chemical science and engineering enabled by self-driving laboratories. *Current Opinion in Chemical Engineering*, 36:100831, 2022.

[111] Luzian Porwol, Daniel J Kowalski, Alon Henson, De-Liang Long, Nicola L Bell, and Leroy Cronin. An autonomous chemical robot discovers the rules of inorganic coordination chemistry without prior knowledge. *Angewandte Chemie*, 132(28):11352–11357, 2020.

[112] S Hessam M. Mehr, Dario Caramelli, and Leroy Cronin. Digitizing chemical discovery with a bayesian explorer for interpreting reactivity data. *Proceedings of the National Academy of Sciences*, 120(17):e2220045120, 2023.

[113] Andrius Bubliauskas, Daniel J Blair, Henry Powell-Davies, Philip J Kitson, Martin D Burke, and Leroy Cronin. Digitizing chemical synthesis in 3d printed reactionware. *Angewandte Chemie*, 134(24):e202116108, 2022.

[114] Gabriella Pizzuto, Jacopo De Berardinis, Louis Longley, Hatem Fakhruldeen, and Andrew I Cooper. Solis: Autonomous solubility screening using deep neural networks. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2022.

[115] Edward O Pyzer-Knapp, Linjiang Chen, Graeme M Day, and Andrew I Cooper. Accelerating computational discovery of porous solids through improved navigation of energy-structure-function maps. *Science Advances*, 7(33):eabi4763, 2021.

[116] Xiaobo Li, Phillip M Maffettone, Yu Che, Tao Liu, Linjiang Chen, and Andrew I Cooper. Combining machine learning and high-throughput experimentation to discover photocatalytically active organic molecules. *Chemical Science*, 12(32):10742–10754, 2021.

[117] Mathilde Fievez, Nina Taherimakhsousi, Benjamin P MacLeod, Edward P Booker, Muriel Matheron, Matthieu Manceau, Stéphane Cros, Solenn Berson, and Curtis P Berlinguette. A machine vision tool for facilitating the optimization of large-area perovskite photovoltaics. In *2022 IEEE 49th Photovoltaics Specialists Conference (PVSC)*, pages 1072–1072. IEEE, 2022.

[118] Naruki Yoshikawa, Kourosh Darvish, Animesh Garg, and Alan Aspuru-Guzik. Digital pipette: Open hardware for liquid transfer in self-driving laboratories. *ChemRxiv preprint*, 2023.

[119] Hitarth Choubisa, Jehad Abed, Douglas Mendoza, Hidetoshi Matsumura, Masahiko Sugimura, Zhenpeng Yao, Ziyun Wang, Brandon R Sutherland, Alán Aspuru-Guzik, and Edward H Sargent. Accelerated chemical space search using a quantum-inspired cluster expansion approach. *Matter*, 6(2):605–625, 2023.

[120] Loïc M Roch, Florian Häse, Christoph Kreisbeck, Teresa Tamayo-Mendoza, Lars PE Yunker, Jason E Hein, and Alán Aspuru-Guzik. Chemos: An orchestration software to democratize autonomous discovery. *PLoS One*, 15(4):e0229862, 2020.

[121] Amanda A Volk, Robert W Epps, Daniel T Yonemoto, Benjamin S Masters, Felix N Castellano, Kristofer G Reyes, and Milad Abolhasani. Alphaflow: autonomous discovery and optimization of multi-step chemistry using a self-driven fluidic lab guided by reinforcement learning. *Nature Communications*, 14(1):1403, 2023.

[122] Sai Krishna Gottipati, Boris Sattarov, Sufeng Niu, Yashaswi Pathak, Haoran Wei, Shengchao Liu, Simon Blackburn, Karam Thomas, Connor Coley, Jian Tang, et al. Learning to navigate the synthetically accessible chemical space using reinforcement learning. In *International Conference on Machine Learning*, pages 3668–3679. PMLR, 2020.

[123] Zhenpeng Zhou, Xiaocheng Li, and Richard N Zare. Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, 3(12):1337–1344, 2017.

[124] Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks

learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

[125] Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier. Caql: Continuous action q-learning. *arXiv preprint arXiv:1909.12397*, 2019.

[126] Chris Gaskett, David Wettergreen, and Alexander Zelinsky. Q-learning in continuous state and action spaces. In *Australasian joint conference on artificial intelligence*, pages 417–428. Springer, 1999.

[127] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[128] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[129] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[130] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[131] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[132] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[133] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1):12348–12355, 2021.

[134] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[135] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. *arXiv preprint arXiv:2002.06487*, 2020.

[136] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[137] Zhengwei Zhu, Can Hu, Chenyang Zhu, Yanping Zhu, and Yu Sheng. An improved dueling deep double-q network based on prioritized experience replay for path planning of unmanned surface vehicles. *Journal of Marine Science and Engineering*, 9(11):1267, 2021.

[138] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *4th International Conference on Learning Representations*, 2015.

[139] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[140] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving ordinary differential equations. 1, Nonstiff problems*. Springer-Vlg, 1993.