Learning Structured Models for Active Planning

Beyond the Markov Paradigm Towards Adaptable Abstractions

Von der Fakultät 5: Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von
Robert Lieck
aus Berlin

Hauptberichter:	Prof. Dr. Marc Toussaint
Mitberichter:	Prof. Dr. Klaus Obermayer

Tag der mündlichen Prüfung: 27. April 2018

Institut für Parallele und Verteilte Systeme
der Universität Stuttgart

2018

Zusammenfassung

Eine wesentliche Eigenschaft menschlicher Intelligenz ist die Fähigkeit aus Erfahrung zu lernen und komplexe und abstrakte Pläne zu ersinnen und zu verfolgen. Im Fall menschlicher Intelligenz sind lernen und planen untrennbar verknüpft, in der Forschung zu künstlicher Intelligenz besteht jedoch eine anhaltende Kluft zwischen diesen Bereichen. Es existieren einerseits klassische Planungsansätze, die komplexe Pläne generieren können und in abstrakten Räumen agieren. Die verwendeten Modelle sind jedoch zumeist von menschlichen Experten spezifiziert und eignen sich kaum dazu von Daten gelernt zu werden. Andererseits gibt es hochentwickelte Methoden im Bereich des maschinellen Lernens, die die Umgebung auf der Basis beobachteter Daten präzise vorhersagen und simulieren können. Planungsmethoden, die diese Modellen verwenden, sind jedoch darauf beschränkt einfache, schrittweise Pläne zu erstellen.

In dieser Arbeit bin ich bestrebt diese Kluft zu überbrücken. Zunächst identifiziere ich die Hindernisse auf beiden Seiten. Auf der Seite datenbasierter Modelle ist dies die Fixiertheit auf zustandsbasierter Vorhersage und Simulation (die so genannte Markov-Eigenschaft) sowie der Mangel an Strukturen, die variable Abstraktion ermöglichen. Auf der Planungsseite ist dies die Anforderung, dass Modelle in hoch spezialisierten formalen Sprachen beschrieben werden müssen (im Fall von klassischen Planungsmethoden) und die Unfähigkeit mit Modellen auf einer strukturellen Ebene zu interagieren (im Fall von zustandsbasierten Planungsmethoden).

Kapitel 1 führt das grundsätzliche Problem ein und gibt einen Überblick über die Arbeit, während Kapitel 2 den notwendigen Hintergrund zu Entscheidungsprozessen, sowie Lern- und Planungsmethoden etabliert. Kapitel 3 widmet sich dem Lernen von nicht-Markovschen Modellen, die eine ausdrucksstarke und flexible Struktur, basierend auf binären, zeitlich ausgedehnten Merkmalen, aufweisen. Um das damit verbundene Lernproblem zu lösen wird das *PULSE*-System eingeführt, das einen systematischen Zugang zum Durchsuchen der mit nicht-Markovscher Modellierung einhergehenden Merkmalsräume ermöglicht. Das Leistungsvermögen der mit *PULSE* gelernten merkmalsbasierten Modelle wird anhand zweier Anwendungen demonstriert: (1) Modellierung nicht-Markovscher Umgebungen im Bereich des Verstärkungslernens und (2) Modellierung mono-

phoner Melodien.

In Kapitel 4 stelle ich die Methode des *aktiven Planens* vor, die es einem Planungsalgorithmus ermöglicht in vielfältiger Weise während des Planens mit einem Modell zu interagieren. Das Potential der Methode wird demonstriert, indem sie zur Verbesserung einer dem Stand der Forschung entsprechenden Planungsmethode, der Monte-Carlo-Baumsuche, verwendet wird.

Abschließend beschreibe ich in Kapitel 5 wie die Methode des aktiven Planens in Verbindung mit merkmalsbasierten Modellen zum Planen in abstrakten Merkmalsräumen genutzt werden kann. Dies demonstriert wie sich planen in abstrakten Räumen mit Modellen, die auf Basis von Daten gelernt wurden, realisieren lässt.

Summary

An essential facet of human intelligence is the ability to learn from experience and to conceive and pursue complex and abstract plans. While in humans learning and planning are inseparably linked, in artificial intelligence research there is a persistent gap between the two areas. On the one hand, there are classical approaches to planning that generate complex plans and operate in abstract spaces but the employed models are usually specified by human domain experts and are hardly amenable to being learned from data. On the other hand, there are sophisticated machine learning methods that are able to accurately predict and simulate the environment based on observed data but the planning methods that make use of these models are restricted to producing simple step-by-step plans.

In this thesis I attempt to bridge this gap. I start by identifying the obstacles on both sides. On the side of data-based models this is the fixation on state-based prediction and simulation (the so-called Markov property) as well as a lack of structure for providing adaptable abstractions. On the side of planning this is the requirement to define models in terms of highly specialized formal languages (for the classical planners) and the inability to interact with a model on the structural level (for state-based planners).

Chapter 1 introduces the general problem and gives an overview of the thesis while Chapter 2 establishes the required background on decision processes, learning, and planning approaches. Chapter 3 is devoted to learning non-Markov models that exhibit an expressive and flexible structure based on binary temporally extended features. To solve the associated learning problem the *PULSE* framework is introduced, which provides a systematic approach to searching through the complex feature spaces that occur in non-Markov modeling. The capability of the feature-based models learned with *PULSE* is demonstrated in two applications: (1) modeling non-Markov environments in reinforcement learning and (2) modeling monophonic melodies.

In Chapter 4 I present the *active planning* approach that allows a planner to interact in various ways with a model during planning. The potential of active planning is demonstrated by employing it for enhancing the performance of *Monte-Carlo tree search*, a state-of-the-art planning method.

Finally, in Chapter 5 I describe how active planning can be used in conjunction with binary feature-based models to perform abstract planning in feature space. This demonstrates how planning in abstract spaces can be achieved with models that are learned from data.

Acknowledgments

As with all larger projects it is hardly possible to mention all the people and all the favorable circumstances that have contributed to its realization.

Throughout the whole time I was surrounded by numerous wonderful people, friends and family, without whom this work would not have been possible. You are the soil on which the determination and persistence, the creativity and passion grows that allowed me to pursue my work.

I consider myself lucky, personally and professionally, to having had the chance of working with Marc Toussaint as a supervisor. Marc's profound technical expertise and his openness to novel views and unconventional approaches was a reliable source of inspiration and advice and has made this work possible in the first place.

Many thanks to Oliver Brock who made it possible to decouple the delocalized nature of scientific life from the localized one of personal life by providing office space at TU Berlin.

I would also like to thank my colleagues and friends from the MLR group in Stuttgart/Berlin and the RBO group in Berlin who are not only responsible for many enjoyable hours but also contributed to this work in the form of numerous discussions and excellent feedback.

This work was supported by the German Academic Scholarship Foundation.

Publications and Statement of Contribution

[A] Robert Lieck and Marc Toussaint (2015). “Discovering Temporally Extended Features for Reinforcement Learning in Domains with Delayed Causalities”. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. Presses universitaires de Louvain, p. 183

Contributions: I am the sole first author of this publication. Marc Toussaint (MT) gave scientific advice and contributed to writing the paper. This paper is the shorter conference version of [B].

[B] Robert Lieck and Marc Toussaint (2016). “Temporally Extended Features in Model-Based Reinforcement Learning with Partial Observability”. In: *Neurocomputing* 192, pp. 49–60

Contributions: I am the sole first author of this publication. MT gave scientific advice and contributed to writing the paper. This paper is the extended journal version of [A].

[C] Johannes Kulick, Robert Lieck, and Marc Toussaint (2016). “Cross-Entropy as a Criterion for Robust Interactive Learning of Latent Properties”. In: *NIPS Workshop on the Future of Interactive Learning Machines*

Contributions: Johannes Kulick (JK) implemented, performed and evaluated all experiments and contributed most to writing the paper. JK and I conceived the approach together and contributed equally to developing the theory. MT gave scientific advice.

[D] Robert Lieck, Vien Ngo, and Marc Toussaint (2017). “Exploiting Variance Information in Monte-Carlo Tree Search”. In: *ICAPS Workshop on Heuristics and Search for Domain-Independent Planning*

Contributions: I conceived the approach, implemented, performed and evaluated all experiments, and wrote the paper. Vien Ngo contributed the proofs. MT gave scientific advice.

[E] Robert Lieck and Marc Toussaint (2017). "Active Tree Search". In: *ICAPS Workshop on Planning, Search, and Optimization*

Contributions: I am the sole first author of this publication. I conceived the approach, developed the theory, implemented, performed and evaluated all experiments, and wrote the paper. MT gave scientific advice.

[F] Jonas Langhabel, Robert Lieck, Marc Toussaint, and Martin Rohrmeier (2017). "Feature Discovery for Sequential Prediction of Monophonic Music". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*. International Society for Music Information Retrieval Conference

Contributions: Jonas Langhabel implemented, performed and evaluated all experiments, and contributed to writing the paper. I gave scientific advice and contributed most to writing the paper. MT was available for scientific advice. Martin Rohrmeier gave musicological advice and contributed to writing the introduction.

Contents

Chapter 1

Introduction

Learning from experience and choosing actions accordingly is a fundamental skill of humans and animals. The ability to learn from past experience allows us to adapt to the environment, better accomplish desired tasks, and have an overall higher chance of survival. One way to adapt behavior is by acquiring stimulus-response patterns, such as in the well-known example from classical conditioning (Pavlov, 2003) where dogs learn to associate a neutral stimulus (the sound of a metronome or buzzer) with food and start to salivate in response to that stimulus, even in the absence of food. Stimulus-response patterns allow for rapid reaction in specific situations and are indispensable to most activities in the real world. However, acting based on stimulus-response patterns is not usually considered a striking indication of intelligence. Rather it is regarded as a virtue to *think* before acting. Planning can be defined as the "art and practice" of doing so (Patrik Haslum).

Planning, as compared to following stimulus-response patterns, is a rather tedious process that involves deliberate cognitive efforts with the goal of finding appropriate actions. Humans are able to conceive and carry out complex plans with many interdependent components. Similar capabilities can be observed in non-human primates when they stack multiple boxes upon each other to reach a banana (Köhler, 1917) or in birds when they use a series of tools to get hold of a nut (Auersperg, Kacelnik, and von Bayern, 2013). Acquiring stimulus-response patterns and conceiving a plan both rely on experience, but in very different ways. The information required for learning a stimulus-response pattern is extracted directly from the experience, which is why there is no sense in punishing your dog hours after it has swiped the sausage from the table – it will not make the connection. For planning, on the other hand, the experience is stored in some kind of mental representation or model that can be used during planning to figure out the consequences of a hypothetical action. If you ask your neighbor, for instance, to keep the music down after midnight she might consider your night's rest when planning the next party – or decide to stimulate your eardrums all the more in response.

Learning and planning are two main pillars in *artificial intelligence* (AI) research. Planning methods have a long history in classical AI where environment models are traditionally specified by human domain experts. Many classical planning approaches are designed to exploit the rich structural information provided by these models. Learning methods have taken a data-oriented turn with the advent of faster computers that enable learning of probabilistic models from interaction with the environment. These models are able to accurately predict and simulate the environment's dynamics in situations where a human specified model is hardly available. Both methods, classical planning approaches and data-based models, are indispensable to modern AI. However, they are largely incompatible because the learned models do not provide the structural information that classical planning methods are designed to leverage. In this thesis I attempt to bring these two worlds a little closer together by exploring how models with a richer structure can be learned from data and how this structure can be exploited to enhance planning.

1.1 Learning and Planning in Artificial Intelligence

A key ambition of AI research is to create artificial agents – such as robots or intelligent software – that act autonomously, similar to humans. Questions dealing with how to learn from experience (more technically: data) and how to choose actions that accomplish a certain task are, thus, central to the field. Just as AI itself comprises many disciplines these questions have also been addressed from different sides. As of today, there is no clear separation between the different subfields and rapidly growing areas such as robotics and reinforcement learning have begun to bridge more and more gaps by combining and making use of a multitude of methods. Nevertheless, the seams are still discernible and each field brings its own tradition of problems and methods.

1.1.1 Symbolic AI versus Control Engineering

One major dimension along which different approaches can be roughly arranged is spanned by the poles of classical symbolic AI on the one hand and control engineering on the other. Traditionally, these two fields are concerned with very different problems, even though both comprise planning as a central component. This becomes clear in the following examples.

Example 1 (Delivering Packages – Classical Symbolic AI). *A company specialized in package delivery has a fleet of trucks that serve a designated area and each day* *a varying number of packages has to be delivered to designated locations. A crucial task is to decide which packages should be loaded into which truck and to determine the best route for each truck given the packages it has to deliver. All packages have to be delivered and the cost (e.g. in terms of time or fuel) should be kept minimal.*

This problem can be solved by, first, formulating a model that takes all relevant details into account: It comprises a map of the operating area, all the trucks and drivers, the different packages and their destination, and also information about traveling distances and related costs. Ignoring highly improbable events (such as a route being entirely blocked without possible bypasses) this is a deterministic, discrete environment with well-defined start and goal state. The corresponding planning problem can be solved by formulating it in a formal language for planning problems and handing it to a standard planning algorithm that will return a detailed list of instructions (possibly leaving open certain choices that do not affect the result) that lead to the goal.

Example 2 (Flying a Drone – Control Engineering)**.** *A toy manufacturer wants to launch a new quad-copter onto the market. The prototype is ready and the engineers say that the motors deliver by far enough thrust to keep everything in the air. The only problem is that it is almost impossible to fly the little machine because it is extremely unstable, especially in windy conditions.*

This problem can be solved by adding sensors to the drone that measure position and orientation and defining a detailed model that describes how motor controls influence position and orientation. To operate the quad-copter the four motors are not directly controlled. Instead, a desired location is specified, an optimizer determines what motion is required to reach that location, and the control signals for the motors are chosen based on the description provided by the model such that the quad-copter closely follows this path.

Both of these examples involve some kind of planning. For delivering packages the instructions concerning which package to load into which truck, what destinations to target in which order, and which routes to take have to be determined. For flying the quad-copter the exact motor control signals have to be determined such that it actually follows the designated path. Beyond this commonality, however, the two cases have little in common, neither with respect to the characteristics of the problem nor concerning the employed solution techniques.

1.1.2 Different Models – Different Planners

In the field of classical planning and scheduling that originates from symbolic AI there is a wealth of formal languages, such as STRIPS (Fikes and Nilsson, 1971), PDDL (McDermott et al., 1998), and RDDL (Sanner, 2010), to specify highly structured and abstract descriptions of different kinds of planning problems. At the same time, the planning algorithms are designed to read in these descriptions and exploit the provided information to perform analyses of the problem structure and derive relaxations or temporal abstractions (Ghallab, Nau, and Traverso, 2016; S. J. Russell and Norvig, 2003). In control engineering, on the other hand, the models are typically formulated in terms of difference or differential equations that describe the detailed development of the system's state including stochasticity and external noise. Based on this description, approaches like optimal control (Bertsekas et al., 1995; Zhou, Doyle, Glover, et al., 1996) or model predictive control (Camacho and Alba, 2013) can be used to derive feed-back controllers or predict the system's dynamics and optimize the control signals.

The main difference between these two approaches is that the *high-level white-box* models in classical planning and scheduling provide a large amount of structural information about how the problem is composed and how the different components interact that goes far beyond only describing the dynamics of the system. At the same time, the description is on a very abstract level and ignores many details of the physical world in which the plans effectively have to be carried out. In contrast, the models in control engineering are microscopic *low-level* models that aim at precisely predicting the environment's dynamic. Often these models are *black-box* models that provide little information about the problem structure beyond predicting or simulating the next state of the system.[1]

The major advantage of the high-level white-box models is that the structural information is openly available and can be exploited by the planner. The advantage of the low-level black-box models is their accuracy on the detailed level and, as discussed in the next section, their amenability to learning.

1.1.3 Learning from Experience

So far the described scenarios did not involve any learning. From the control engineering side learning can be integrated relatively easily. As the employed models are supposed to describe the actual dynamics of the system it is straight-forward to come up with a criterion of what is a good or a bad model based on whether it accurately describes the observed dynamics. This criterion can be used to evaluate and optimize candidate models

[1]Some of these low-level models may contain structural information about the system's state, though, like factorizations or decompositions that may be used to speed up computations.

based on data that were collected from the real system. The corresponding branch of control engineering is called *system identification* (Ljung, 1998). From the side of classical planning, on the other hand, incorporating learning is relatively hard. The reason for this is that the models as well as the planning algorithms make extensive use of structural information that is not directly observable. Learning, therefore, primarily takes place on the side of the human domain experts that have to come up with an appropriate structure for describing the problem in one of the formal languages. Only once the structure is fixed it is possible to adjust remaining parameters based on observed data.

1.2 The Best of Two Worlds

Obviously, it is desirable to strive for the best of both worlds. On the one hand, we would like to employ models that provide highly structured and abstract information about the environment and use planning methods that leverage this information for planning. On the other hand, we want to learn models from data that capture the most detailed dynamics of the system.

There are various efforts from both sides to incorporate more learning into structured models and, vice versa, more structure into learned models. One indicator on the side of the classical planning community is that the International Planning Competition, held by the International Conference on Automated Planning and Scheduling, in 2018 for the first time features a *data-based track* where only sample trajectories are provided as the problem description.[2] On the other side there are fields like hierarchical reinforcement learning (Barto and Mahadevan, 2003) that attempt to decompose a decision process into multiple levels of abstraction. These abstractions mitigate the problems associated with a pure low-level description but they have a limited expressive capacity, are difficult to learn, and do not interact with planners on the structural level. There are also hybrid models being employed that provide a combined symbolic-continuous description (Ghallab, Nau, and Traverso, 2016; Toussaint and Lopes, 2017). However, the fact that they are appropriately termed *hybrid* approaches is indicative for the persistent separation of the two worlds. So what exactly prevents us from fully merging the two? In my opinion it is the contradiction between temporal abstraction and the so-called Markov property.

1.2.1 The Markov Bottleneck

A statistical process exhibits the *Markov property* if future states are statistically independent of past states conditioned on the current state. The Markov property lies at

[2] *Learning tracks* where example domains are specified in PDDL exist since 2008.

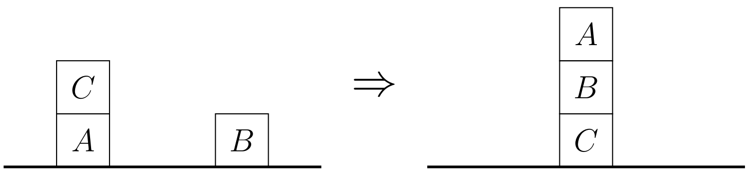


Figure 1.1: Sussman Anomaly: The task is to get from the initial state (left hand side) to the goal state (right hand side) by moving one block at a time.

the basis of all models that describe a dynamical system in a step-by-step manner, such that at any time the system is in a well-defined state and its future development can be predicted solely based on its current state. If we believe in the established physical theories any physical system adheres to the Markov property and there is always such a thing as the "true" state of the environment, at least in principle. The most appealing property of a Markov environment is that if we know its state we do not need to remember anything from the past. This is such a huge advantage in practice that even in situations where the true state is not known, such as in *partially observable Markov decision processes* (POMDPs), the Markov property is reestablished on a more abstract level by working with *belief states* that incorporate all possibilities of what the true state of the environment could be.

A major problem with the Markov property is that it fundamentally contradicts temporal abstraction. Temporal abstraction in planning consists in ignoring certain aspects, such as the ordering of specific actions or their precise timing, as long as this does not affect the actual goal. Planning may hugely benefit from temporal abstraction as a multitude of different possibilities that would otherwise need to be treated separately can now be grouped into one. Temporal abstraction is incompatible with a Markov model of the environment because in a temporal abstraction the environment's state is not generally well-defined at all times, which renders the application of a Markov model impossible.

Why Do We Need Temporal Abstraction?

A well-known illustrative example in classical planning is the Sussman anomaly (Sussman, 1975), which is illustrated in Figure 1.1. In the initial state C is on A, and A and B are on the table. The goal is to reach a state where A is on B, which in turn is on C. The Sussman anomaly became famous because the problem cannot easily be decomposed into sub-goals: A planner that first tries to achieve the sub-goal "A is on B" in the most direct way and then the sub-goal "B is on C" runs into a dead end. The same happens if it tries to reach the two sub-goals in the opposite order. A Markov planner that explores all possibilities by simulating the effect of actions step-by-step will eventually find the solution but it may waste time by trying out dead ends before that. A *partial-order*

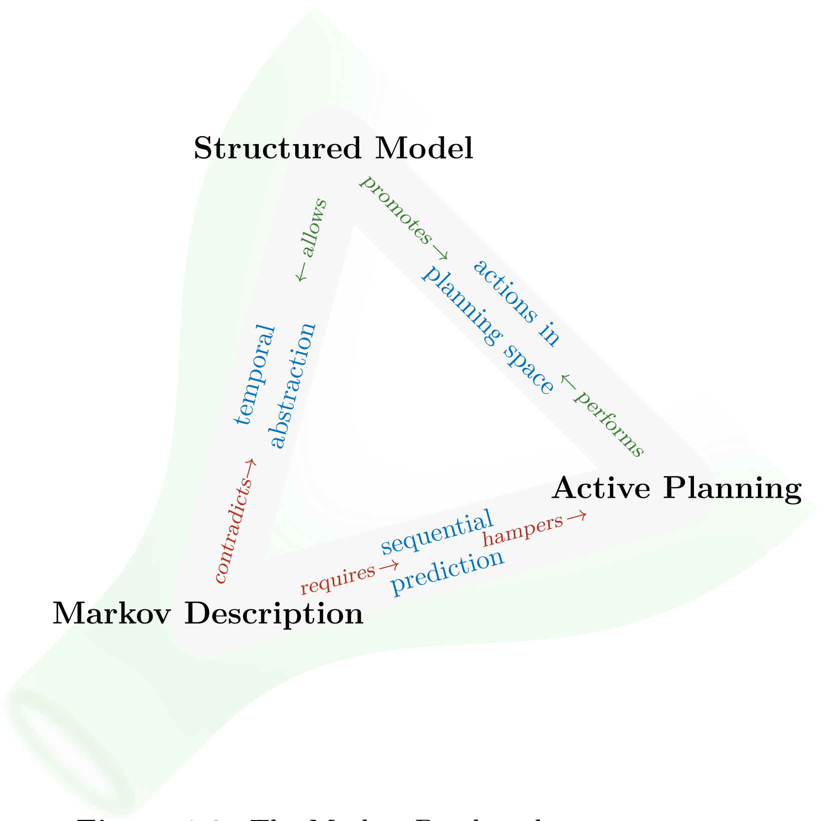


Figure 1.2: The Markov Bottleneck.

planner, in contrast, will first identify the three actions that are required to reach the goal while ignoring the order in which they need to be carried out. It will then detect that there is only one possible order in which the actions can successfully be performed and return the corresponding action sequence. In situations where multiple orderings are feasible the partial-order planner specifies only a partial order of actions (hence its name). It should be clear that in situations where such a temporal dependency structure occurs on a larger time scale the temporal abstraction performed by a partial-order planner provides a huge advantage compared to the detailed simulations performed by a Markov planner.

Planning as Active Learning

A crucial feature of planners that exploit structural information is that they perform planning actions in a different space than that of the environment actions. In the above example, the actions performed by a Markov planner correspond to moving single blocks, just as actions in the environment do, while the actions performed by a partial-order planner occur in the abstract space of possible orderings of the three necessary environment actions. The planner thus interacts with the model in a much more sophisticated way to extract the provided structural information. I will refer to this feature of a planner to actively seek to learn about properties of the environment as *active planning*. Providing an active planner with an unstructured Markov model limits its planning capabilities as its planning actions are restricted to querying information about single transitions in the environment. This supporting and inhibiting relation between a structured model, an active planner, and a Markov description of the environment is depicted in Figure 1.2.

1.2.2 Beyond the Markov Paradigm

An inevitable consequence of these considerations, it seems to me, is that we have to free ourselves from the Markov paradigm. We have to stop pretending there was some "true" state of the environment that we need to use as a basis for learning our models and planning actions. It is true that in the end we will need to build a system that has a well-defined state at any time. It is also true that all attempts to go beyond the Markov principle can be rendered futile by resorting to more and more abstract notions of what a "state" is. However, all this does not help with arriving at representations that facilitate learning expressive models of the environment that can be used by powerful planning methods to successfully act in a complex world. Overcoming the Markov paradigm means much more than just going for POMDPs, recurrent neural networks, or non-Markov time series prediction. We have to embrace the new freedom that is gained by dropping this burden and rethink what is desirable and what is possible – this is the endeavor of this thesis and what it is devoted to.

Learning Structured Models

The existing description languages for structured models are designed to be used by humans to express their structural knowledge such that it can be processed by a computer. The first major challenge in going beyond Markov models consists in finding a generic model structure that is simple enough to be learned from data but at the same time flexible and expressive enough to capture the relevant aspects of existing approaches for abstraction. At the same time, it has to be accessible to a planner for making active use of its structural information. The relevant questions to ask therefore are:

How should a structured model that allows for active planning look like?

How can we learn it?

The answer proposed in Chapter 3 consists in a learning framework for discovering non-Markov features that capture relevant domain properties and allow for adaptable abstractions.

Active Planning

The second challenge, which is closely intertwined with the first one, is to develop an approach to planning that supports interaction with this kind of structured model. Such a planner cannot build on existing frameworks that make use of human-defined structural information because the kind of structure that lends itself to learning is likely to be of a substantially different kind. At the same time, the existing approaches for planning with Markov models are of little avail as they do not incorporate the notion of actively interacting with a model on the structural level. The relevant question to ask, therefore, is:

How does a planner that actively queries information from the model work?

Developing the concept of active planning and demonstrating its advantages using simple Markov models is the aim of Chapter 4.

Planning in Feature Space

The eventual goal of this thesis is to forge an initial link between models that can be learned from data and approaches for planning in abstract spaces. A basis on both sides is established in Chapter 3 and 4, respectively, so that the final question to be addressed is:

How can we perform active planning in the abstract feature space of a structured model?

In Chapter 5 I describe how this connection can be made. The presented approach combines some of the most relevant properties of abstract planning methods with flexible and expressive models that can be learned from data.

These results constitute, I hope, an initial step towards a novel kind of planning algorithms that unify the structural expressiveness of classical planning methods with the descriptive power of modern probabilistic models learned from data.

Chapter 2

Background

2.1 Interactive Processes

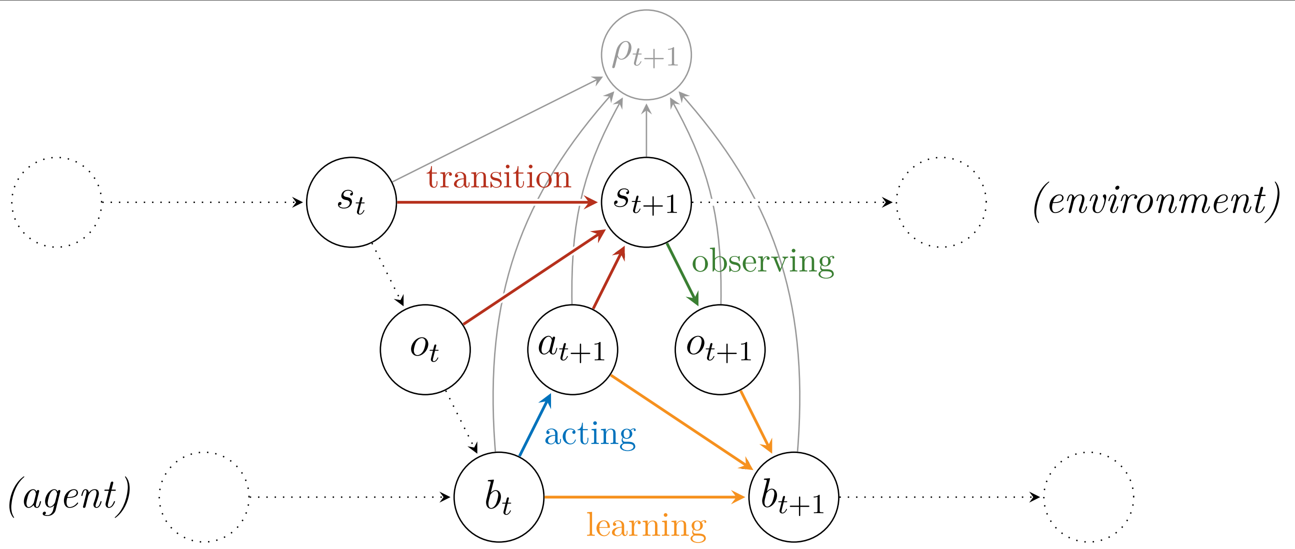
The basic scenario of an agent that interacts with an environment occurs in many contexts and, depending on the application, the focus lies on different aspects. All these cases share the overarching structure of an *interactive process* (IP) that I will present in Section 2.1.1. I will then describe how this general formulation can be reduced to the different special cases that are divided into three groups: Decision processes (Section 2.1.2) focus on how the agent's actions influence the dynamics of a known environment; the area of active learning (Section 2.1.3) considers how actions influence what information is gathered and how the agent learns from its observations; and the area of reinforcement learning (Section 2.1.4) is concerned with the process of acting in an initially unknown environment while simultaneously learning about how actions affect the environment.

2.1.1 Overarching Structure

An IP, as visualized in Figure 2.1, is a stochastic process consisting of an environment with state $s \in \mathcal{S}$; an agent with internal state or *belief* $b \in \mathcal{B}$; actions $a \in \mathcal{A}$ the agent can take in the environment; and observations $o \in \mathcal{O}$ it receives in return. The dynamics of an IP are determined by the observation function $p_{\mathcal{O}}(o_{t+1}|s_{t+1})\colon \mathcal{S} \rightsquigarrow \mathcal{O}$[3] that describes what the agent observes given the current state of the environment, the transition function $p_{\mathcal{S}}(s_{t+1}|s_t, o_t, a_{t+1})\colon \mathcal{S} \times \mathcal{O} \times \mathcal{A} \rightsquigarrow \mathcal{S}$ that describes how the environment's state changes under the influence of an action, the policy $\pi(a_{t+1}|b_t)\colon \mathcal{B} \rightsquigarrow \mathcal{A}$ that describes how the agent chooses its actions based on its internal state, and the internal state update $u(b_{t+1}|b_t, a_{t+1}, o_{t+1})\colon \mathcal{B} \times \mathcal{A} \times \mathcal{O} \rightsquigarrow \mathcal{B}$ that describes how the agent's internal state

[3]The notation $\mathcal{X} \rightsquigarrow \mathcal{Y}$ describes a stochastic mapping, that is, $p(y \in \mathcal{Y} \,|\, x \in \mathcal{X}) \in [0,1]$ and $\sum_{y \in \mathcal{Y}} p(y \,|\, x) = 1$.

Interactive Process



variable	description	distributed as
$s \in \mathcal{S}$	state of the environment	$p_{\mathcal{S}}(s_{t+1} \mid s_t, o_t, a_{t+1})\colon \mathcal{S} \times \mathcal{O} \times \mathcal{A} \rightsquigarrow \mathcal{S}$
$b \in \mathcal{B}$	internal state (belief) of the agent	$u(b_{t+1} \mid b_t, a_{t+1}, o_{t+1})\colon \mathcal{B} \times \mathcal{A} \times \mathcal{O} \rightsquigarrow \mathcal{B}$
$a \in \mathcal{A}$	actions taken by the agent	$\pi(a_{t+1} \mid b_t)\colon \mathcal{B} \rightsquigarrow \mathcal{A}$
$o \in \mathcal{O}$	observations made by the agent	$p_{\mathcal{O}}(o_{t+1} \mid s_{t+1})\colon \mathcal{S} \rightsquigarrow \mathcal{O}$
$\rho \in \mathcal{R} \subseteq \mathbb{R}$	reward received by the agent	$p_{\mathcal{R}}(\rho_{t+1} \mid s_t, s_{t+1}, b_t, b_{t+1}, a_{t+1}, o_{t+1})\colon \mathcal{S}^2 \times \mathcal{B}^2 \times \mathcal{A} \times \mathcal{O} \rightsquigarrow \mathcal{R}$

Figure 2.1: Graphical model of an interactive process and overview of the involved variables and probability distributions.

changes under the influence of a new observation. The agent's internal state represents its knowledge about the environment and may comprise a probability distribution over the environment's state, relevant parameters, properties, or possible models for the observation and transition function. At each time step the agent receives a reward signal $\rho \in \mathcal{R} \subseteq \mathbb{R}$ and the goal is to find an optimal policy π^* so that the agent chooses actions that maximizes one of the following objectives

$$\pi^* = \mathrm{argmax}_\pi \mathbb{E}\left[\!\!\left[\rho_T \mid \pi \right]\!\!\right] \qquad \textit{(expected reward after finite horizon)} \tag{2.1}$$

$$\pi^* = \mathrm{argmax}_\pi \mathbb{E}\left[\!\!\left[\sum_{t=1}^{T} \rho_t \mid \pi \right]\!\!\right] \qquad \textit{(expected cumulative reward over finite horizon)} \tag{2.2}$$

$$\pi^* = \mathrm{argmax}_\pi \mathbb{E}\left[\!\!\left[\sum_{t=1}^{\infty} \gamma^t \rho_t \mid \pi \right]\!\!\right] \qquad \textit{(expected cumulative discounted reward over infinite horizon)} \tag{2.3}$$

where $\gamma \in [0, 1]$ is a *discount factor* that may be used to assign a lower weight to rewards in the remote future.

Figure 2.2 gives an overview of the different reductions of an IP in terms of their graphical models. Most reductions have a non-interactive version, which ignores actions and rewards. These are included for completeness but not discussed in detail. In some descriptions found in the literature the internal state and/or the policy are not included in the graphical model as they are not usually part of the problem description but rather part of the problem solution. However, including them gives a more consistent picture and emphasizes the underlying common structure.

The terminology of an "agent" that is taking actions in an "environment" suggests an asymmetric structure, which is also reflected in the interpretation of the different variables. Note, however, that the graphical model of an IP is perfectly symmetric so that the "environment" could just as well be another agent's internal state, in which case the IP describes the process of communication between two agents. In the single-agent setting, the symmetric structure reveals that an observation acts on the internal state of the agent just as the agent's actions act on the state of the environment. Furthermore, the subjective nature of the observation process is reflected by the fact that the way an observation is processed depends on the internal state and the actions of the observer.

2.1.2 Decision Processes

Decision processes focus on the effect the agent has on the environment. The reward is assumed to depend only on the environment's state and the agent's actions. In the infinite horizon scenario, which I will be mostly concerned with in this work, the objective is to

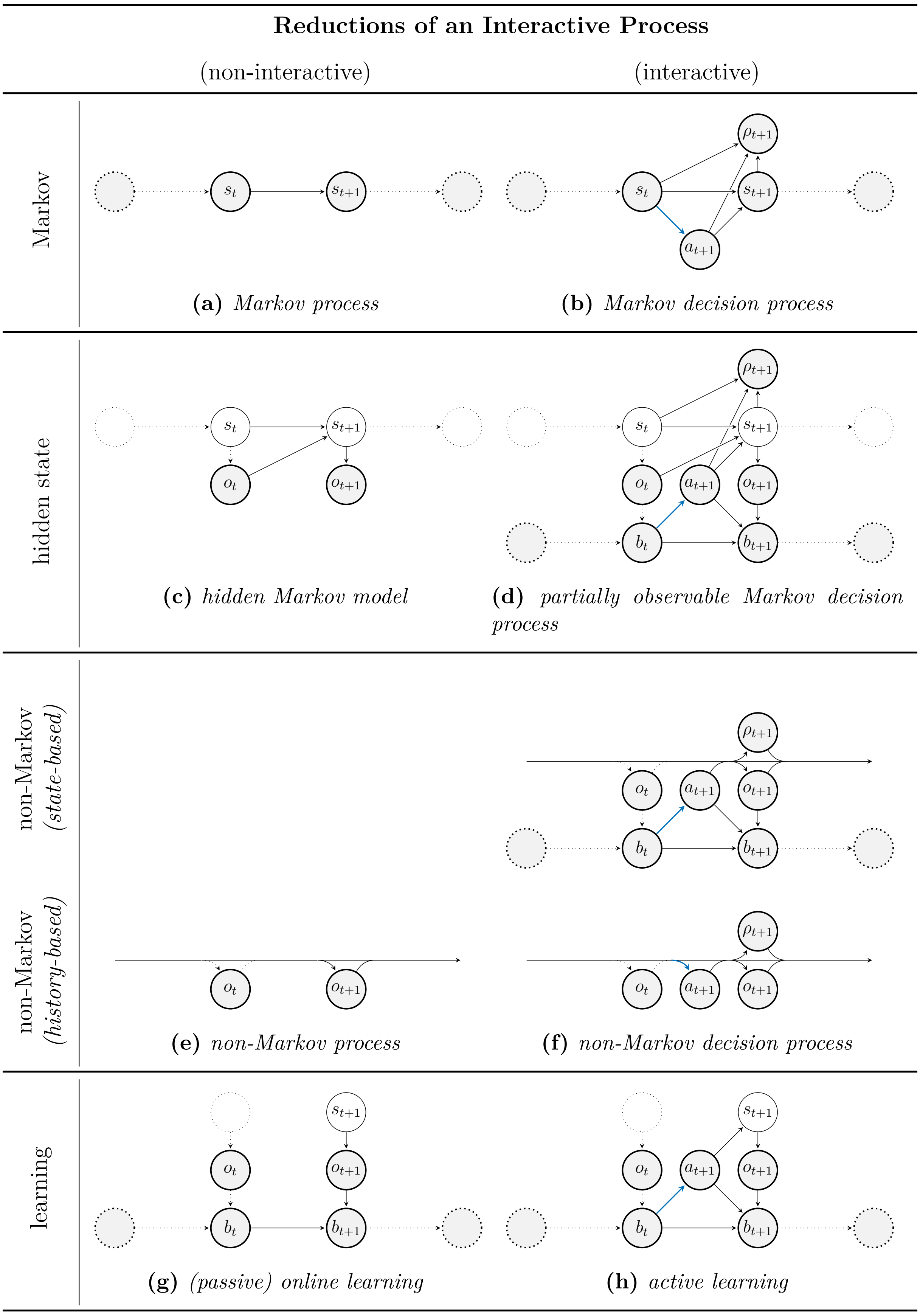

Reductions of an Interactive Process
(non-interactive)
(interactive)
Markov
hidden state
non-Markov (state-based)
non-Markov (history-based)
learning
s_t
s_{t+1}
ρ_{t+1}
a_{t+1}
o_t
o_{t+1}
b_t
b_{t+1}
(a) *Markov process*
(b) *Markov decision process*
(c) *hidden Markov model*
(d) *partially observable Markov decision process*
(e) *non-Markov process*
(f) *non-Markov decision process*
(g) *(passive) online learning*
(h) *active learning*

Figure 2.2: Graphical models of the interactive and non-interactive reductions of a general interactive process. Observed variables are shown in light-gray, latent/hidden variables in white. The colored edges indicate the policy π, which is optimized based on the objective. Junctions/forks of arcs in the non-Markov case signify that every incoming arc is connected to every outgoing arc.

maximize the expectation of the *discounted return*

$$\begin{aligned} R_t &= \rho_{t+1} + \gamma \rho_{t+2} + \gamma^2 \rho_{t+3} + \dots \\ &= \sum_{\tau=0}^{\infty} \gamma^\tau \rho_{t+\tau+1} \\ &= \rho_{t+1} + \gamma\, R_{t+1} \ , \end{aligned} \tag{2.4}$$

which corresponds to Eq. (2.3). As the expectation $\mathbb{E}[\![R_t]\!]$ only depends on the expected reward

$$r_{(s_t, a_{t+1}, s_{t+1})} = \mathbb{E}[\![\rho_{t+1} | s_t, a_{t+1}, s_{t+1}]\!] \ , \tag{2.5}$$

all relevant quantities can be stated in terms of r instead of the full reward distribution. The expectation of R conditioned on state s or state-action pair (s, a) is called the *state value* V or *state-action value* (or just *action value*, for short) Q, respectively

$$\begin{aligned} V_s^\pi &= \mathbb{E}\Big[\!\!\Big[R \,|\, s\Big]\!\!\Big] \\ &= \sum_a \pi_{(a\,|\,s)} \underbrace{\sum_{s'} p_{(s'\,|\,s,a)} \Big(r_{(s,a,s')} + \gamma\, V_{s'}^\pi \Big)}_{Q_{(s,a)}^\pi} \end{aligned} \tag{2.6}$$

$$\begin{aligned} Q_{(s,a)}^\pi &= \mathbb{E}\Big[\!\!\Big[R \,|\, s, a\Big]\!\!\Big] \\ &= \sum_{s'} p_{(s'\,|\,s,a)} \Big(r_{(s,a,s')} + \gamma \underbrace{\sum_{a'} \pi_{(a'\,|\,s')}\, Q_{(s',a')}^\pi}_{V_{s'}^\pi} \Big) \ , \end{aligned} \tag{2.7}$$

where a super-script π may be used to indicate the dependence on the policy if not clear from the context. Different decision processes differ in terms of what simplifying relations between variables they assume and what variables are assumed to be fundamentally inaccessible.

Markov Decision Process

The *Markov decision process* (MDP) (Figure 2.2b) with its non-interactive counterpart, the *Markov process* or *Markov chain* (Figure 2.2a), results from an IP by assuming that the agent has perfect knowledge about the world, that is, it directly observes the environment's state. As a result the agent's internal state is identical to the environment's state and no separate observation variables are needed. MDPs are used to describe decision problems in different disciplines, from AI and robotics to economics. A common approach to solving more complex scenarios is to reformulate them as an abstract MDP.

Partially Observable Markov Decision Process

Partially observable Markov decision processes (POMDPs) (Figure 2.2d) and their non-interactive counter parts, *hidden Markov models* (HMMs) (Figure 2.2c), do not make simplifying assumptions about the variables and their structure. However, the transition and observation function, $p_\mathcal{S}$ and $p_\mathcal{O}$, and the environment's state space $\mathcal{S}$ are assumed to be known. This means that the agent still cannot directly observe the environment's state but if it *knew* the current state s_t it *could* predict the next state s_{t+1} and observation o_{t+1}, given its action a_{t+1}. The learning aspect on the side of the agent is thus reduced to inferring the environment's state s, while learning about $p_\mathcal{S}$ and $p_\mathcal{O}$ is omitted. As a result the agent's internal state space can be much smaller as it only needs to represent its belief over the environment's state. The dynamics of the agent's internal state corresponds to an MDP in belief state, which can, in principle, be used to compute policies via *belief-state planning* (Kaelbling, Littman, and Cassandra, 1998). In general, however, optimal policies in POMDPs are hard to compute (Lusena, Goldsmith, and Mundhenk, 2001; Madani, Hanks, and Condon, 1999).

Non-Markov Decision Process

The *non-Markov decision process* (non-MDP) (Figure 2.2f) and its non-interactive counter part, the non-Markov process (Figure 2.2e), result from an IP if there is no information whatsoever available about the environment's state. As opposed to the POMDP setting, where the state space and the transition and observation function are known, in a non-MDP the state variable is to be marginalized out, which on the symbolic level accounts for the total lack of information. When marginalizing out a state variable s_i in the graphical model we have to add new connections between all neighboring variables that s_i was connecting via a head-to-tail or tail-to-tail path (but not those connected via a head-to-head path) in order to retain the same set of (conditional) independence relations (see e.g. Bishop, 2007). Before marginalizing out the environment's state, the observation o_{t+1} was independent of past actions and observations given the state s_{t+1}

$$o_{t+1} \perp\!\!\!\perp a_{t+1}, o_t, a_t, \ldots \,|\, s_{t+1} \tag{2.8}$$

$$p_\mathcal{O}(o_{t+1}|s_{t+1}) : \mathcal{S} \rightsquigarrow \mathcal{O} \ , \tag{2.9}$$

so that the observation function $p_\mathcal{O}$ was a function of just the state variable. In contrast, after marginalizing out the state variables, $p_\mathcal{O}$ is a function of all past actions and observations

$$o_{t+1} \not\perp\!\!\!\perp a_{t+1}, o_t, a_t, \ldots \,|\, \emptyset \tag{2.10}$$

$$p_{\mathcal{O}}(o_{t+1}|a_{t+1}, o_t, a_t, \ldots) : (\mathcal{A} \times \mathcal{O})^* \rightsquigarrow \mathcal{O} \; . \tag{2.11}$$

In a non-MDP the Markov assumption cannot any longer be preserved, which makes it particularly relevant for this work. Concerning the internal state of the agent, the Markov property is preserved *conditional* on the observations received from the environment. Adopting this view when solving a non-MDPs corresponds to the *state-based* approach (upper part of Figure 2.2f) as it is, for instance, taken when working with recurrent neural networks or *finite state machines* (FSMs).

As an alternative, it is possible to also marginalize out the agent's internal state, which means that the actions, just as the observations, now depend on the entire history of past actions and observations. This corresponds to the *history-based* approach to solving non-MDPs (bottom part of Figure 2.2f). As a major concern in this thesis is to go beyond Markov representations I will adopt the history-based approach throughout this work.

2.1.3 Active Learning

In an *active learning* process the environment is assumed to be *state-less* or *memory-less*, that is, the environment produces independent and identically distributed observations in response to the agent's actions.[4] The graphical model of an active learning process and its non-interactive version, which corresponds to (passive) online learning, are depicted in Figure 2.2h and Figure 2.2g, respectively. While the graphical model in Figure 2.2h highlights the similarity of an active learning process and a general IP, the environment's state s only serves as a dummy variable for the process that generates observations. Depending in the concrete problem it is common to make more specific assumption about how observations are being generated by the environment and how the agent learns from them.

The goal in active learning is to choose actions optimally for learning, either about latent properties of the environment that are involved in generating observations or about the observation function $p_{\mathcal{O}}$ itself. The models depicted in Figure 2.3 capture the general case where the variable f represents the unknown observation function and the variable θ represents latent properties of the environment that determine f. Depending on the specific setting other models may be more convenient (see Chaloner and Verdinelli, 1995; Settles, 2010, for a broader overview of the field).

[4]Sometimes the terms *queries* and *labels* are used instead of actions and observations.

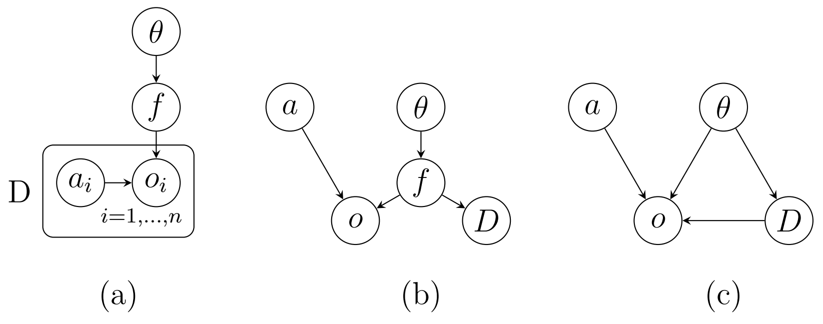


Figure 2.3: Active Learning: After performing action $a \in \mathcal{A}$ the agent receives an observation $o \in \mathcal{O}$ distributed as $p_{\mathcal{O}}(o \,|\, a; f)$, where f determines the distribution. In active learning the parameters f are unknown and treated as a random variable with prior distribution $p(f \,|\, \theta)$, where θ represents latent properties of the environment. The data $D \in (\mathcal{A} \times \mathcal{O})^*$ correspond to the set of known action-observation pairs. **(a):** The first n steps of the chain in Figure 2.2h are written in plate notation ignoring the agents internal state. **(b):** The same model with D collapsed into a single random variable and the next action a and observation o kept separately. **(c):** For inferring the latent properties θ the observation function determined by f was marginalized out.

Concerning the agent's internal state b it is possible to adopt two different views similar to the history-based and state-based approach in non-MDPs. From a theoretical point of view, especially when taking a Bayesian approach to learning, it may be convenient to ignore the internal state b by marginalizing it out, which corresponds to the history-based approach in non-MDPs. As a consequence, actions have to be chosen based on the entire history of past actions and observations, which is represented by the data D in Figure 2.3. Alternatively, the agent's internal state may represent a sufficient statistic of the data that is updated after each new action-observation pair. This approach is more practical when working in an iterative setting and corresponds to the state-based approach in non-MDPs. In some situations it may be convenient to switch between the two views. I will, therefore, use a unified notation where $\boldsymbol{\xi}$ represents the agent's knowledge and is updated as

$$\boldsymbol{\xi}[D,a,o] \xleftarrow{a,o} \boldsymbol{\xi}[D] \qquad \text{(state-based/sufficient statistics)} \tag{2.12}$$

$$\text{or} \qquad p(\boldsymbol{\xi} \,|\, D,a,o) \xleftarrow{a,o} p(\boldsymbol{\xi} \,|\, D) \qquad \text{(history-based/Bayesian)} \tag{2.13}$$

respectively. In this notation $\boldsymbol{\xi}[D]$ corresponds to a (non-random) variable that represents a sufficient static of D while $p(\boldsymbol{\xi} \,|\, D)$ corresponds to the belief distribution over the quantity of interest for which $\boldsymbol{\xi}[D]$ is the sufficient statistic. A typical example for the equivalence of these two views are conjugate priors in the Bayesian framework where the update from prior to posterior belief can be expressed as a parameter update of the corresponding distribution.

Active learning will become relevant in Chapter 4 where I formulate the task of planning as an active learning problem, which provides a sound theoretical basis for describing the interaction of an active planner with a structured model.

Reward Signal

The reward in an active learning process quantifies the learning progress by assessing the uncertainty associated with the agent's knowledge $\boldsymbol{\xi}$. Let $\mathcal{O}(\boldsymbol{\xi})$ be an objective that quantifies uncertainty then the expected reward $r_{(D,a)}$ for an action a based on the available data D is

$$\begin{aligned} r_{(D,a)} &= \mathbb{E}\Big[\!\!\Big[-\mathcal{O}\Big(\boldsymbol{\xi}[D,a,o]\Big)\Big]\!\!\Big]_{o\,|\,a,D} && (2.14) \\ &= -\int p(o\,|\,a,D)\ \mathcal{O}\Big(\boldsymbol{\xi}[D,a,o]\Big)\ do\ . && (2.15) \end{aligned}$$

That is, the reward corresponds to the negative uncertainty after taking action a, and the expectation has to be taken over possible observations o conditioned on a and the available data D. I will assume the reward to be defined in terms of $\mathcal{O}$ as in Eq. (2.15) and proceed with discussing uncertainty measures. Common choices for $\mathcal{O}$ are the entropy or the variance of the agent's belief $p(\boldsymbol{\xi}\,|\,D)$

$$\begin{aligned} \mathrm{H}(\boldsymbol{\xi}\,|\,D) &= -\int p(\boldsymbol{\xi}\,|\,D)\log p(\boldsymbol{\xi}\,|\,D)\ d\boldsymbol{\xi} && (2.16) \\ \mathrm{Var}(\boldsymbol{\xi}\,|\,D) &= \left[\int \boldsymbol{\xi}^2\ p(\boldsymbol{\xi}\,|\,D)\ d\boldsymbol{\xi}\right] - \left[\int \boldsymbol{\xi}\ p(\boldsymbol{\xi}\,|\,D)\ d\boldsymbol{\xi}\right]^2 . && (2.17) \end{aligned}$$

A low entropy of the belief means that its information content is high (the entropy is the expected information contained in observing the true value of $\boldsymbol{\xi}$) while a low variance means that the belief is highly condensed (the variance is the expected squared deviation of $\boldsymbol{\xi}$ from its mean). If the belief is Gaussian the entropy and the variance are linked as

$$\begin{aligned} p(\boldsymbol{\xi}\,|\,D) &= \mathcal{N}(\mu,\sigma^2) && (2.18) \\ \mathrm{Var}(\boldsymbol{\xi}\,|\,D) &= \sigma^2 && (2.19) \\ \mathrm{H}(\boldsymbol{\xi}\,|\,D) &= \frac{1}{2}\log(2\pi e\sigma^2)\ . && (2.20) \end{aligned}$$

Note that the normal distribution is the maximum entropy distribution for fixed variance σ^2, which means that the variance defines an upper bound for the entropy given by Eq. (2.20). Minimizing the variance is thus a stronger objective in the sense that it implies a minimal entropy.

2.1.4 Reinforcement Learning

The area of *reinforcement learning* (RL) is not so much characterized by a reduction of the IP structure but on the contrary by embracing its full complexity. RL is concerned with finding approaches to solve the complex decision problems that arise if an agent simultaneously interacts with an environment and learns about its dynamics. Most areas of RL are not directly relevant to this work (see Richard S. Sutton and Barto, 1998, for a standard reference). In the following, I discuss three aspects that *are* of particular interest, namely the distinction between model-based and model-free approaches, the exploration/exploitation dilemma and, most notably, approaches from hierarchical RL.

Model-Based – Model-Free

The distinction between *model-based* and *model-free* approaches in RL roughly corresponds to that of planned actions versus stimulus-response patterns. In a model-based approach the task of choosing actions is decomposed into first learning a model that captures the dynamics of the environment and then using this model to choose actions optimally. In contrast, model-free approaches attempt to directly learn the action values or the policy. While in this work I take a model-based approach to decision problems the area of model-free RL provides a wide range of methods and the two approaches are not mutually exclusive. One example from the field of hierarchical RL are options, which can be understood as model-free building blocks that can be used as part of a model-based approach on a higher level of abstraction.

Exploration – Exploitation

A problem that is particular to RL is the so-called *exploration/exploitation dilemma*: On the one hand, the agent has to explore the environment in order to learn about its dynamics; on the other hand, it has to exploit what it already knows in order to obtain as much reward as possible. When only maximizing reward the agent risks to miss out on rewards that it has not yet discovered. Likewise, pure exploration does not result in an optimal behavior. Instead, the agent has to balance exploration and exploitation.

A theoretically optimal solution to the exploration/exploitation dilemma is provided by the *Bayesian reinforcement learning* (Bayesian RL) framework (Vlassis et al., 2012) where the agent's belief over possible models is treated as part of the overall state. This allows to optimally trade off actions that lead to known rewards with actions that offer the chance to discover new and possibly higher rewards. While an exact solution to the Bayesian RL problem is usually intractable there are heuristics that perform near-optimal. Most of these approximate approaches follow the principle of *optimism in the* *face of uncertainty*, which roughly corresponds to adding intrinsic reward to options that are not well explored but bear the potential for high reward. There are two branches of methods that pursue a slightly different objective. One branch attempts to perform near-optimally with respect to the optimal solution from Bayesian RL, such as the Bayesian exploration bonus (Kolter and Ng, 2009). The other branch, comprising methods like R_{max} (Brafman and Tennenholtz, 2003) or KWIK (L. Li et al., 2011), attempts to learn a model that is *probably approximately correct* (PAC), that is, they try to ensure that the learned model is close to the true one. An extensively studied special case of the exploration/exploitation dilemma that corresponds to a discrete state-less environment is the multi-armed bandit problem (Berry and Fristedt, 1985), which is also relevant in active learning and *Monte-Carlo tree search* (MCTS).

Hierarchical Reinforcement Learning

Methods from hierarchical reinforcement learning (Barto and Mahadevan, 2003) decompose a decision problem into multiple levels of abstraction. An abstraction on a higher level corresponds to a serial decomposition of the decision problem on the next-lower level. The actions on an abstract level are thus *macro actions* that each perform a series of actions on the next-lower level. Macro actions describe enduring activities of a varying temporal extent. To account for their varying extent the decision problems on the abstract levels have to be described in the semi-MDP framework. The hierarchical serial decomposition introduced by the use of macro actions is a special kind of temporal abstraction that preserves the relevant aspects of the Markov property (in Section 2.2.2 I discuss other forms of decomposition and abstraction). They can, therefore, easily be integrated with standard planning algorithms, which are designed for (strict) Markov models of the environment, such as MCTS (de Waard, 2016). The most widely used formalization of the concept of macro actions are *options* (Richard S. Sutton, Precup, and Singh, 1999). In the following I describe options as well as several other approaches from hierarchical RL.

Options: For an MDP with state space $\mathcal{S}$ and action space $\mathcal{A}$ an *option* (Richard S. Sutton, Precup, and Singh, 1999) is defined as a triplet $\langle \mathcal{I}, \pi, \beta \rangle$, where $\mathcal{I} \subseteq \mathcal{S}$ is the initiation set, $\pi : \mathcal{S} \rightsquigarrow \mathcal{A}$ is the option's policy, and $\beta : \mathcal{S} \to [0, 1]$ is the termination function. An option is available in all states of the initiation set $\mathcal{I}$; when the option is chosen, the agent follows policy π; and after every step the option terminates with a probability $\beta(s)$, where s is the current state. Since an option follows its own policy after being called, options can be interpreted as encapsulating a model-free component that can be reused on the next abstraction level by a model-based or model-free approach. A common approach to defining options is by identifying characteristic regions in the state space (Mannor et al., 2004) or sub-goals that need to be reached (Menache, Mannor, and Shimkin, 2002; Şimşek and Barto, 2004). Separate options are then defined and trained for each region or sub-goal. Options were shown to speed up learning and improve skill transfer to related tasks (Taylor and Stone, 2009).

MAXQ: The *MAXQ decomposition of the value function* (MAXQ, Thomas G. Dietterich, 1998) uses the "call stack" K of all currently active macro actions to augment the state s of the original MDP. Policies are defined with respect to this augmented state $[s, K]$, which results in a decomposition of the value function. Based on this decomposition it is possible to learn hierarchical policies from sample trajectories (T. G. Dietterich, 2000).

HAM: In *hierarchies of abstract machines* (Parr and S. Russell, 1997) the original MDP is decomposed into sub-tasks that are handled by abstract machines (stochastic finite state machine). Each abstract machine corresponds to a macro action. A machine may execute four different kinds of actions: **`Action`** executes an action in the original MDP. **`Call`** pauses the currently running machine and calls a new one. **`Choice`** induces a state transition of the machine's internal state. **`Stop`** terminates the execution of the currently running machine and returns control to the next higher level. Parr and S. Russell (1997) show that the original MDP is transformed to a semi-MDP that can be solved by only considering *choice* states of the hierarchy of abstract machines.

HEXQ: HEXQ (Hengst, 2002) uses a factored state representation and introduces a separate control level for each state variable. Levels are ordered based on how frequently the corresponding variable changes. Beginning with the lowest level (highest frequency) the state space is divided into regions where the level variable changes in a predictable way and none of the higher-level variables change value. These regions form the abstract states of the next higher level. Abstract actions are formed by policies that traverse and exit a region on the lower level thereby inducing a change of variable on the higher level.

Hierarchical RL is the most relevant attempt to learning models that exhibit a richer temporal structure. The central concept underlying hierarchical RL is the semi-Markov abstraction that allows for temporally extended macro actions. In Chapter 5 I show how semi-Markov abstraction may be realized in a more flexible manner using feature-based models, relieving the need to predefine macro actions.

2.2 Planning

Planning problems occur in a wide range of disciplines and depending on the specific problems there exists a variety of approaches to solve them. I do not attempt to give a comprehensive overview of all the different approaches. Rather, I will selectively focus on distinct aspects and methods that are relevant to this thesis. The main concern in this work is to bridge the gap between learned models that accurately simulate and predict the environment, on the one hand, and planning methods that make use of structured information and abstractions, on the other hand. It is, therefore, important to highlight the differences between these two worlds, which is done in Section 2.2.1 and 2.2.2. The active planning approach developed Chapter 4 forms the basis to bridge this gap from the planning side. Two aspects become particularly relevant in that respect, namely action selection in active learning problems (Section 2.2.3) and Monte-Carlo tree search (Section 2.2.4).

For a broader overview of classical planning methods see the book by Ghallab, Nau, and Traverso (2016). The book by Richard S. Sutton and Barto (1998) provides a complementary overview of RL methods for planning, including dynamic programming and Monte-Carlo approaches.

2.2.1 Temporal Abstraction versus State-Based Simulation

An abstraction is a description that ignores certain details of the underlying problem. In contrast, a state-based simulation describes the dynamics of a system by specifying the complete state in every time step. Abstraction and simulation are not necessarily mutually exclusive. Generally, abstractions that retain the concept of the system's state are amenable to simulation. A state in this context is defined via the Markov property, that is, a state description has to be self-sufficient in the sense that it contains all information that is needed to predict the next state. In principle, for instance, it is possible to describe a system on different levels of abstraction by defining abstract states and actions on each level. It is then possible to perform a simulation of the system dynamics on each of these abstract levels. The methods from hierarchical RL rely on this approach: If a high-level macro action invokes a low-level macro action, the system's state on the higher level freezes until the low-level action terminates. The outcome and the duration of the high-level action is determined by the dynamics on the lower level. This kind of *durative actions* represent a relatively mild form of temporal abstraction where only the temporal extent of an action is left unspecified. Durative actions formally require to switch to a semi-Markov description but they retain the state concept and do, therefore, not contradict state-based simulation.

However, some abstractions are incompatible with simulation. This is the case if the system's state is only partly specified. The abstract action of "setting the table", for instance, may be defined as "putting forks, knives and plates on the table" while ignoring the order in which the single items are put. As a consequence, during the time the action is being carried out the system's state is not fully specified: Do the forks already lie on the table or have I started with the knives? If I intend to simulate this abstract action I am forced to either use it as a monolithic block on an abstract level or switch to a lower level and specify all the details. Leaving open the ordering of some actions is a typical example for temporal abstraction. Classical planning approaches make extensive use of abstractions in a much more flexible manner than it is possible with state-based simulations. A partial-order planner, for instance, does not need to maintain different levels of abstraction in order handle partially ordered action sequences. This becomes relevant if there are additional ordering requirements. The "setting the table" action may, for instance, additionally include laying the table cloth, which has to take place *before* putting forks, knives and plates. More general forms of temporal abstraction that include partial orders and interval constraints can be formulated as *simple temporal networks*, and different forms of temporal logic can be employed to specify temporal conditions (Ghallab, Nau, and Traverso, 2016).

If we want to bride the gap between learned models that accurately simulate and predict the environment and temporal abstraction as they are used in classical planning, we inevitably have to abandon the Markov property. That is, our models have to be able to predict the system's state if sufficient information is available but they also have to provide useful information about the environment if the state is not fully specified.

2.2.2 Decomposition and Abstraction

Abstraction and decomposition are related in that any decomposition, in principle, implies the possibility for abstraction by ignoring some of the components and retaining others. Decomposability has been discussed in such diverse fields as cognitive science, artificial intelligence and robotics, political and social science, and philosophy of science (DeLanda, 2006; Höfer, 2017; Schierwagen, 2012; Simon, 1996). The unifying idea is to describe a system as an assemblage of components and interactions between these components. A system is decomposable if the relevant properties of the overall system can be recovered from its components. In general, whether a problem is decomposable or not is a gradual property and a given problem lies somewhere within this spectrum of decomposability (Höfer, 2017; Schierwagen, 2012). In the following, I will discuss some approaches commonly used for decomposing decision processes, which facilitates planning as the respective components can be solved independently.

Parallel Decomposition

In a parallel decomposition the overall decision process is decomposed into a set of sub-processes that run in parallel and have independent state and action spaces (Boutilier, Dean, and Hanks, 1999). Each of the sub-processes can be solved independently and a solution to the overall process is assembled from the corresponding sub-solutions. The state and action space of the overall decision process corresponds to the product space of the state and action spaces of the sub-processes.

Serial Decomposition

In a serial decomposition the overall decision process is decomposed into a sequence of sub-problems that can be solved one after the other (Boutilier, Dean, and Hanks, 1999). Methods for detecting sub-goals in the state space (Maron and Pérez, 1998; McGovern and Barto, 2001; Menache, Mannor, and Shimkin, 2002; Şimşek and Barto, 2004) can be used to define a serial decomposition. Similarly, one can attempt to describe an overall policy as a series of macro actions by identifying reoccurring policy fragments (Pickett and Barto, 2002; Thrun and Schwartz, 1995).

Hierarchical Decomposition

A hierarchical decomposition consists of a decomposition into sub-systems that are themselves recursively decomposed into sub-systems (Simon, 1996). Approaches in hierarchical RL involve a hierarchy either in terms of macro actions that call other macro actions or in terms of a factored state representation where lower-level state factors change more frequently than higher-level state factors. Common to all hierarchical decompositions is a tree structure where a change on the higher level (e.g. termination of a macro action) requires a change on a lower level (e.g. termination of the called macro action) but not vice versa.

2.2.3 Action Selection in Active Learning

The problem of choosing optimal actions in an active learning problem is equivalent to that of choosing optimal actions in a POMDP, and exactly solving the associated planning problem is intractable in all but the most restricted cases. In many situations actions, therefore, have to be chosen in a myopic and greedy way maximizing only for short-term rewards. A one-step greedy policy for a given objective $\mathcal{O}$ chooses actions as

$$a^* = \operatorname{argmin}_a \mathbb{E}\Big[\!\!\Big[\mathcal{O}\Big(\boldsymbol{\xi}[D, a, o]\Big)\Big]\!\!\Big]_{o\,|\,a,D} . \tag{2.21}$$

Greedy action selection does not guarantee to optimize the objective $\mathcal{O}$. Some objectives exhibit the property that optimizing them greedily guarantees bounded regret with respect to choosing optimal actions. This is, for instance, the case for submodular functions (Golovin and Krause, 2011; Nemhauser, Wolsey, and Fisher, 1978). Otherwise, we are in a similar situation as for the exploration/exploitation dilemma in RL with the advantage that in active learning the environment does not change. A well-studied active learning problem for which the planning problem is tractable and multiple policies with bounded regret guarantees exist is the multi-armed bandit problem (Berry and Fristedt, 1985). The UCB1 policy (Auer, Cesa-Bianchi, and Fischer, 2002) for multi-armed bandits is widely used, among others in global optimization and MCTS.

Another heuristic that has proved to be beneficial in practice (even though I am not aware of any proofs of bounded optimality) is to maximize for a *change* in the quantity of interest $\boldsymbol{\xi}$ instead of minimizing its uncertainty. This idea of *expected model change* can be applied if $\boldsymbol{\xi}$ represents the parameters of a model (Settles, 2010) as well as to the agent's belief $p(\boldsymbol{\xi} \mid D)$ (Kulick, Lieck, and Toussaint, 2016). The rationale behind maximizing the expected change of the quantity of interest is that we expect uncertainty to eventually become small as we collect more data but we want to avoid it becoming small prematurely. Maximizing the expected change attempts to ensure *robust* reduction of uncertainty in the case of myopic one-step greedy action selection.

2.2.4 Monte-Carlo Tree Search

Monte-Carlo tree search (MCTS) has become a standard planning method and has been successfully applied in various domains, ranging from computer Go to large-scale POMDPs (Browne et al., 2012; Silver, Huang, et al., 2016). Some of the most appealing properties of MCTS are that it is easy to implement, does not require a full probabilistic model of the environment but only the ability to simulate transitions, is suited for large-scale environments, and provides theoretical convergence guarantees.

The core idea in MCTS is to build up a local search tree by performing simulated *rollouts* from the current state. There exists a wide variety of MCTS algorithms that differ in a number of aspects. Most of them follow the generic scheme shown in Algorithm 1. In this scheme each rollout proceeds in four steps as depicted in Figure 2.4. Some recent suggestions deviate slightly from this scheme (Feldman and Domshlak, 2014b; Keller and Helmert, 2013). The generic scheme in Algorithm 1 contains a number of open parameters that define a specific MCTS implementation.

`BestAction(`n_0`)` determines how to choose the action that is eventually recommended.

It is common to recommend either the action with maximum empirical mean return

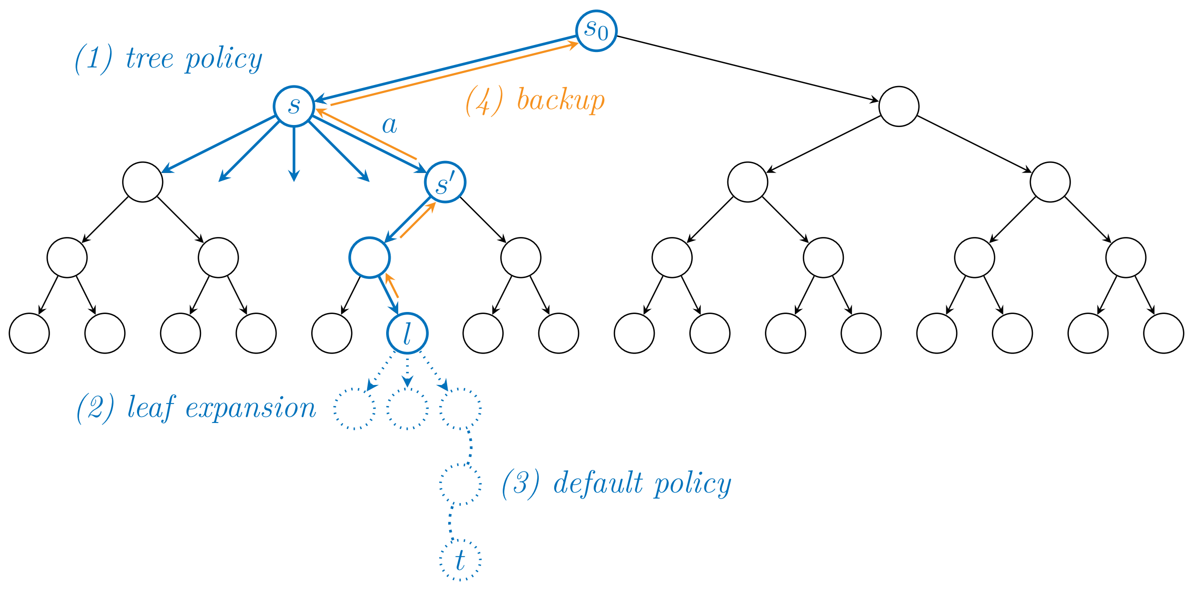


Figure 2.4: A rollout in MCTS proceeds in four steps: (1) Transitions within the tree are sampled using the tree policy. (2) When reaching a leaf node l it is expanded. (3) From the expanded leaf node transitions are sampled using the default policy until reaching a terminal state t. (4) From the expanded leaf node a backup is performed back to the root node.

Algorithm 1 Generic MCTS algorithm for finite-horizon non-discounted environments. Notation: ( ) is a tuple; ⟨ ⟩ is a list, + appends an element to the list, $|\,l\,|$ is the length of a list, and l_i is the i^{th} element in a list.

```
Input: $n_0$, $s_0$, $M$          → root node, root state, model
Output: $a^*$                            → optimal action

1:  function MCTS($n_0, s_0, M$)
2:      while time permits do
3:          $(\rho, s) \leftarrow$ FollowTreePolicy($n_0, s_0$)
4:          $R \leftarrow$FollowDefaultPolicy($s$)
5:          Update($\rho, R$)
6:      end while
7:      return BestAction($n_0$)          → open parameter
8:  end function

9:  function FollowTreePolicy($n, s$)
10:     $\rho \leftarrow \langle\rangle$
11:     repeat
12:         $a \leftarrow$ TreePolicy($n$)          → open parameter
13:         $(s', r) \leftarrow M(a, s)$
14:         $\rho \leftarrow \rho + \langle(n, s, a, s', r)\rangle$
15:         $n \leftarrow$ FindNode($n, a, s'$)    → open parameter
16:         $s \leftarrow s'$
17:     until $n$ is new leaf node
18:     return $(\rho, s)$
19: end function

20: function FollowDefaultPolicy($s$)
21:     $R \leftarrow 0$
22:     repeat
23:         $a \leftarrow$ DefaultPolicy($s$)    → open parameter
24:         $(s', r) \leftarrow M(a, s)$
25:         $R \leftarrow R + r$
26:         $s \leftarrow s'$
27:     until $s$ is terminal state
28:     return $R$
29: end function

30: function Update($\rho, R$)
31:     for $i$ in $|\rho|, \ldots, 1$ do
32:         $(n, s, a, s', r) \leftarrow \rho_i$
33:         Backup($n, s, a, s', r, R$)    → open parameter
34:         $R \leftarrow r + R$
35:     end for
36: end function
```

or the one that was most sampled during planning (see Browne et al., 2012, for other alternatives).

`FindNode(`n, s, a, s'`)` selects a child node or creates a new leaf node if the child does not exist. The most common implementations build up a tree structure but it is also possible to construct directed acyclic graphs (Saffidine, Cazenave, and Méhat, 2012) and there exist other trial-based heuristic search methods similar to MCTS (Keller and Helmert, 2013), such as real-time dynamic programming (Barto, Bradtke, and Singh, 1995; McMahan, Likhachev, and Gordon, 2005; Sanner et al., 2009), that do not construct a tree.

`DefaultPolicy(`s`)` is a heuristic policy for initializing the return value of new leaf nodes. In the simplest case the uniform policy is used. For large state spaces with scarce reward signal the returns produced by the uniform policy may not contain enough information about the return of a near-optimal policy. In that case more informed heuristics may be used such a recently demonstrated for the game of Go (Silver, Huang, et al., 2016).

`TreePolicy(`n`)` The tree policy is a core component of an MCTS algorithm. It selects actions for rollouts within the tree and is faced with the exploration/exploitation dilemma: On the one hand, it has to focus on high-return branches to find the optimal policy (exploitation); on the other hand, it has to sample sub-optimal branches to reduce the estimation errors (exploration). MCTS became popular mainly due to the use of UCB1 (Auer, Cesa-Bianchi, and Fischer, 2002) as tree policy, which chooses actions as

$$a^* = \operatorname{argmax}_a B_{(s,a)} \qquad \text{with} \tag{2.22}$$

$$B_{(s,a)} = \widehat{R}_{(s,a)} + 2\, C_p \sqrt{\frac{2\,\log n_s}{n_{(s,a)}}} \tag{2.23}$$

where $\widehat{R}_{(s,a)}$ is the mean return of action a in state s; n_s is the number of visits to state s; $n_{(s,a)}$ is the number of times action a was taken in state s; the returns are assumed to be in $[\,0,1\,]$; and the constant $C_p > 0$ controls exploration. $B_{(s,a)}$ can be interpreted as an optimistic upper bound of the expected return $\widehat{R}_{(s,a)}$. UCB1 was developed in the context of *multi-armed bandit* problems (Berry and Fristedt, 1985) and applying it in MCTS amounts to describing action selection as a sequence of separate multi-armed bandits. As opposed to the classical multi-armed bandit setting in MCTS the return distributions are not stationary because they depend on the tree policy at later steps. Kocsis and Szepesvári (2006) showed that the convergence guarantees of UCB1 transfer from multi-armed bandits to MCTS so that the probability of choosing a sub-optimal action at the root node converges to zero at a polynomial rate as the number of trials grows to infinity. More recently, the UCB-V policy Audibert, Munos, and Szepesvári (2009) was shown to provide guarantees similar to that of UCB1 for multi-armed bandits. UCB-V takes the empirical variance of the return into account and chooses actions as

$$a^* = \operatorname{argmax}_a B_{(s,a)} \qquad \text{with} \tag{2.24}$$

$$B_{(s,a)} = \widehat{R}_{(s,a)} + \sqrt{\frac{2\,\widetilde{R}_{(s,a)}\zeta \log n_s}{n_{(s,a)}} + 3cb\frac{\zeta \log n_s}{n_{(s,a)}}}\ , \tag{2.25}$$

where $\widehat{R}_{(s,a)}$, n_s, $n_{(s,a)}$ as above; $\widetilde{R}_{(s,a)}$ is the empirical variance of the return; rewards are assumed to be in $[0, b]$; and the constants $c, \zeta > 0$ control the algorithm's behavior. In (Lieck, Ngo, and Toussaint, 2017) we showed that UCB-V provides the same guarantees as UCB1 in the MCTS setting. The variance backups needed to make use of variance-based policies such as UCB-V in conjunction with dynamic programming backups are derived in Section 4.2.2, Eqs. (4.34) and (4.43).

`Backup(`n, s, a, s', r, R`)` The `Backup` procedure is another core component that distinguishes different MCTS algorithms. It is responsible for updating node n given the transition $(s, a) \to (s', r)$ and the return R of the corresponding trial. It has to maintain the data needed for evaluating the tree policy. In the simplest case of Monte-Carlo (MC) backups the `Backup` procedure maintains visit counts n_s, action counts $n_{(s,a)}$, and an estimate of the expected return $\widehat{R}_{(s,a)}$ by accumulating the average of R. In the more recently suggested dynamic programming (DP) backups (Keller and Helmert, 2013) the `Backup` procedure additionally maintains a transition model and an estimate of the expected immediate reward that are then used to calculate $\widehat{R}_{(s,a)}$ while the return samples R are ignored. MC and DP backups have significantly different characteristics that are subject of ongoing research (Feldman and Domshlak, 2014a; Lieck, Ngo, and Toussaint, 2017). Recently, temporal difference learning and function approximation have also been proposed as backup methods (Guez et al., 2014; Silver, Richard S Sutton, and Müller, 2012). It has also been suggested to use different backup methods depending on the empirical variance of returns (Bnaya et al., 2015).

Abstraction in MCTS

Abstraction in MCTS occurs in two different ways. For one thing, it is possible to perform explicit abstraction in state and action space, which may be static or adaptable. The approach of *progressive widening* (Chaslot et al., 2007; Couëtoux et al., 2011; Wang, Audibert, and Munos, 2009), for instance, performs an adaptable state and action abstraction in each node, which allows to apply MCTS in continuous state and action spaces.

For another thing, MCTS effectively implements a form of adaptable abstraction in trajectory space, which can be considered a form of temporal abstraction. Even though the MCTS algorithm is not usually described from this point of view, it is a major reason for its success. This form of temporal abstraction in trajectory space occurs because the tree policy chooses among the actions locally in each node based on information that was compiled from the entire subtree of the respective actions. For taking a decision the tree policy thus ignores details in the future and agglomerates all trajectories with a common prefix up to that point. A rollout can be interpreted as a series of refinements that traverses this hierarchy of abstractions from the highest level (at the root node) to the lowest level (at the leaf nodes).

Relevance for this Work

MCTS is relevant to this work because it is used as a test bed for the active planning approach in Chapter 4. The reason for this choice was twofold. First, MCTS is a state-of-the-art planning method that represents a good starting point for further developments. Second, and more importantly, even though one of the most appealing properties of MCTS is that it can be used with state-based simulation, MCTS is not restricted to a state-based approach to planning. The view of MCTS as implementing an adaptable abstraction in trajectory space is a first indication of this capacity. A second indication is the fact that each node does not only represent a specific state of the environment but is also associated with a distinct history represented by the path from the root node. This allows MCTS to be used for planning in POMDPs (Silver and Veness, 2010) and means that it is open to non-Markov models of the environment that employ a history-based approach. In Chapter 5 I develop *abstract search trees* that extend classical search trees and serve as the representation for planning in feature space.

2.3 Learning a Model

In this section I touch upon two points with respect to learning a model that are relevant to this work. First, a topic that has been known for a long time in machine learning but that is gaining an increasing amount of attention recently is the idea of *learning to learn* or *meta-learning*, which I discuss in Section 2.3.1. Second, in Section 2.3.2 I review common approaches for selecting features, which can be seen as a kind of meta-learning. These aspects are relevant to this work insofar as the *PULSE* framework, developed in Section 3.2, can essentially be considered a meta-learning framework for feature discovery.

2.3.1 Learning to Learn

As artificial systems display super-human performance in an increasing number of benchmark tasks the discussion of what differentiates human intelligence from artificial intelligence focuses on more complex aspects, one of them being the ability of learning to learn (Lake et al., 2016). While methods for model selection and hyper-parameter learning are part of the standard portfolio of machine learning (see e.g. Hastie, Tibshirani, and Friedman, 2008), more recently there are also increasing efforts being made to improve domain specific optimization algorithms (Andrychowicz et al., 2016; K. Li and Malik, 2016) by learning how to *better* optimize problems from the given domain.

The knowledge that is acquired by a meta learner can be considered prior knowledge with respect to the base task. In general this prior knowledge captures particularities of the problem domain that help solving a specific task more efficiently or using less training data. Instead of using a meta-learning method this prior knowledge can also be included by domain experts, if available. A challenge in this respect is how to formalize the knowledge a human expert has such that it can be exploited by the learning algorithm. One intention in developing the *PULSE* framework was to facilitate the inclusion of implicit prior knowledge. In Section 3.2.1, I discuss the role of prior knowledge in more detail.

Another problem that is closely connected to the meta-learning topic is that of *overfitting*. The overfitting problem arises due to the fact that the objective used by a naive learning algorithm – predicting the training data – is not the objective that actually matters, which consists in predicting *unknown* data. In general, including prior knowledge about the domain results in the learner being biased in a way that increases the training error but decreases the prediction error by reducing the variance of predictions that were learned from different training data. This is known as *bias-variance decomposition* (see e.g. Bishop, 2007; Hastie, Tibshirani, and Friedman, 2008).

The overfitting problem poses a dilemma because it is fundamentally impossible to use the actual objective of predicting unknown data *directly*: As soon as we make use of data for training they cease being unknown to the learner. We can, however, use it *indirectly* by assessing the learner's predictive performance on a separate validation data set after being trained on the training data. If a learning algorithm has hyper-parameters that shape its behavior the validation data can be used to optimize its predictive performance. Of course, the same dilemma arises on the meta level where the validation data have become the training data, which results in a hierarchy of learners with arbitrarily many levels. It has been argued that two levels of learning are a minimum requirement for human-like intelligence (Nikolić, 2015).

Hierarchy of Learning

I will now briefly describe how an arbitrarily deep hierarchy of learners can be defined recursively by encapsulating lower levels within a single learner. Assume the task is to learn a probability distribution over the space $\mathcal{X}$ based on a sample $D \in \mathcal{X}^*$. A *model* M_θ defines a parameterized predictor $p_{M_\theta}(x;\theta)$, where θ are the model parameters. Let $\mathcal{L}_\lambda : D, M_\theta \mapsto \theta$ be a learning method with hyper-parameters λ that is used to optimize the model parameters based on data D. Then the tuple $M_\lambda = (\mathcal{L}_\lambda, D, M_\theta)$ of the learning method $\mathcal{L}_\lambda$, the data set D, and the model M_θ again defines a model

$$p_{(M_\theta,\mathcal{L}_\lambda,D)}(x;\lambda) := p_{M_\theta}\Big(x;\mathcal{L}_\lambda(M_\theta,D)\Big)\ , \tag{2.26}$$

which has the learner's hyper-parameters as model parameters. Using this encapsulation we can define a hierarchy of learners by specifying a basis model M_θ, a series of learning methods $\{\mathcal{L}^{(0)}_{\lambda^{(0)}},\dots,\mathcal{L}^{(N)}_{\lambda^{(N)}}\}$, and the hyper-parameters $\lambda^{(N)}$ of the learning method on the highest level. The hierarchy of learners is then defined recursively as

$$p_{M_{\lambda^{(N)}}}(x;\lambda^{(N)}) = p_{M_\theta}(x;\theta) \tag{2.27}$$

$$\textbf{basis:}\begin{cases} \theta = \mathcal{L}^{(0)}_{\lambda^{(0)}}(D^{(0)},M_\theta) & \text{(parameters)}\\ \mathcal{L}^{(0)}_{\lambda^{(0)}} : D^{(0)},M_\theta \mapsto \theta & \text{(learner)}\\ M_{\lambda^{(0)}} := (\mathcal{L}^{(0)}_{\lambda^{(0)}},D^{(0)},M_\theta) & \text{(model)}\end{cases}$$

$$\textbf{recursion:}\begin{cases} \lambda^{(n)} = \mathcal{L}^{(n+1)}_{\lambda^{(n+1)}}(D^{(n+1)},M_{\lambda^{(n)}}) & \text{(parameters)}\\ \mathcal{L}^{(n)}_{\lambda^{(n)}} : D^{(n)},M_{\lambda^{(n-1)}} \mapsto \lambda^{(n-1)} & \text{(learner)}\\ M_{\lambda^{(n)}} := (\mathcal{L}^{(n)}_{\lambda^{(n)}},D^{(n)},M_{\lambda^{(n-1)}}) & \text{(model)}\ .\end{cases}$$

Due to the encapsulation, data on a lower level are a subset of the data on a higher level, $D^{(n-1)} \subseteq D^{(n)}$. Hyper-parameter learning corresponds to one step of recursion. An example would be to train, on the first level, a predictive model M_θ using a learning method $\mathcal{L}^{(0)}_{\lambda^{(0)}}$ that maximizes the posterior estimate of the model parameters

$$\begin{aligned}\mathcal{L}^{(0)}_{\lambda^{(0)}}(D^{(0)}, M_\theta) &:= \operatorname{argmax}_{\theta'} p(\theta' \mid D^{(0)}; \lambda^{(0)}) && (2.28)\\ &= \operatorname{argmax}_{\theta'} \frac{1}{|D^{(0)}|} \sum_{(x,y)\in D^{(0)}} \log \frac{p_{M_\theta}(x;\theta')\, p(\theta';\lambda^{(0)})}{\int_{\theta''} p_{M_\theta}(x;\theta')\, p(\theta';\lambda^{(0)})\, d\theta''}\ , && (2.29)\end{aligned}$$

where the hyper-parameters $\lambda^{(0)}$ determine the prior over model parameters. On the second level, one can use some validation method, such as cross-validation, to optimize the hyper-parameters for generalization

$$\begin{aligned}\mathcal{L}^{(1)}_{\lambda^{(1)}}(D^{(1)}, M_{\lambda^{(0)}}) &:= \operatorname{argmax}_{\bar\lambda^{(0)}} \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|D_i|} \sum_{(x,y)\in D_i} \log p_{M_{\lambda^{(0)}}}(x; \bar\lambda^{(0)}) && (2.30)\\ &= \operatorname{argmax}_{\bar\lambda^{(0)}} \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|D_i|} \sum_{(x,y)\in D_i} \log p_{M_\theta}\Big(x; \mathcal{L}^{(0)}_{\lambda^{(0)}}(\bar D_i, M_\theta)\Big) && (2.31)\\ &= \operatorname{argmax}_{\bar\lambda^{(0)}} \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|D_i|} \sum_{(x,y)\in D_i} \log \Big[\ldots\\ &\quad \ldots p_{M_\theta}\Big(x; \operatorname{argmax}_{\theta'} \frac{1}{|\bar D_i|} \sum_{(x,y)\in \bar D_i} \log \frac{p_{M_\theta}(x;\theta')\, p(\theta';\lambda^{(0)})}{\int_{\theta''} p_{M_\theta}(x;\theta')\, p(\theta';\lambda^{(0)})\, d\theta''}\Big)\Big]\ , && (2.32)\end{aligned}$$

where $\lambda^{(1)}$ determines an N-fold partition of the data $D^{(1)}$; D_i is the i^{th} part that is used as validation set; and $\bar D_i = D^{(1)} \setminus D_i$ is the remaining data with D_i being held out that is used for optimizing θ on the first level. A strict encapsulation makes learning on higher levels challenging as it corresponds to black-box global optimization. By unrolling the recursion as in Eq. (2.32) the encapsulation becomes transparent so that in principle gradients and other information can be used (see e.g. Y. Bengio, 2000).

2.3.2 Feature Discovery

Many machine learning methods do not operate directly on the raw input data but instead change the representation by first applying a set of feature functions to the input data before feeding it to the actual learning algorithm. Features can be arbitrary mappings from the original input domain to a feature space. If the features are chosen appropriately, for instance, by making use of prior knowledge about the problem structure, the learning problem may become substantially easier to solve – general linear regression, for instance, assumes the target function to be linear in feature space. An appropriate feature set is thus a crucial prerequisite in many applications. In the following I will discuss some common approaches for finding features.[5]

Forward Selection In *forward selection*, the final feature set is successively built up by including features from a pool of candidates. Features are included greedily based on how much they improve the model quality (Hastie, Tibshirani, and Friedman, 2008). In situations where the overall model improvement is too expensive to assess, for instance because it involves expensive optimization of the feature weights, it is possible to estimate the overall improvement by optimizing only for the new candidate feature (see e.g. S. Della Pietra, V. Della Pietra, and J. Lafferty, 1997). Being a greedy approach, the resulting feature set is not in general guaranteed to be optimal. As for greedy policies in active learning or RL for some classes of objective functions the resulting set is still guaranteed to be near-optimal (Golovin and Krause, 2011; Krause and Guestrin, 2007; Nemhauser, Wolsey, and Fisher, 1978). In general, forward selection requires evaluating the improvement of all candidate features and is thus only applicable to finite feature sets of a moderate size.

Feature Expansion In situations where candidate features cannot be evaluated exhaustively, for instance, because there are infinitely many of them, *feature expansion* is an approach to perform forward selection based on a finite subset of features. The idea of feature expansion is to construct a set of candidate features by expanding existing features that were already included in the set. A common approach is to define a set of binary basis features and construct candidate features as conjunctions of an existing feature and a basis feature. This technique has been successfully applied to learn (conditional) random field models for text (S. Della Pietra, V. Della Pietra, and J. Lafferty, 1997; A. McCallum, 2002) and for linear approximations of the value or transition function in reinforcement learning (Geramifard et al., 2011; Ure et al., 2012).

Feature expansion is an essential part of the approach presented in Section 3.3 for learning feature-based models based on the *Pulse* framework. The *Pulse*-based approach goes beyond simple feature expansion in two respects. First, extension of the feature set is complemented by a backward selection step that reduces the size of the feature set. The interplay of both operations creates a fundamentally different behavior and allows to search through the feature space in novel ways. Second, features are suggested and included in groups that adhere to certain criteria, which

[5] I will not discuss the closely related tasks of learning a kernel function or a metric as they are not directly relevant to this work.

allows to derive guarantees concerning the expressiveness of the resulting feature sets.

Backward Selection In *backward selection*, a feature set is constructed by starting with the full set containing all possible features and successively removing features. Analogous to forward selection the feature that least contributes to the model quality is dropped (Hastie, Tibshirani, and Friedman, 2008). Backward selection initially requires the full feature set to be processed and is thus not applicable for infinite or very large feature sets.

Shrinkage Methods and Regularization An approach to backward selection that is not restricted to stepwise removal of single features is to frame feature selection as a continuous optimization problem. This can either be done by adding a regularization term that penalizes feature weights, such as an L_1-regularization, or by formulating the corresponding constrained optimization problem. For quadratic objectives there exist efficient methods to compute the L_1-path of the feature weights as the regularization strength is varied (Efron et al., 2004; Hastie, Tibshirani, and Friedman, 2008). The approach for learning feature-based model using *PULSE*, presented in Section 3.3, makes use of L_1-regularization to shrink the feature set.

Chapter 3

Learning Structured Models

The aim of this chapter is to develop an approach for learning models that capture complex structural information about the environment and allow for planning algorithms to actively access this information.

The purpose of a model is to represent knowledge in a condensed and accessible form. What *accessible* means depends a lot on the specific use of the model, that is, what knowledge should be accessed and how it should be accessed. In the case of planning the goal is to find the action with the highest value. Ironically, the methods that represent this knowledge in the most direct way are called *model-free* approaches. This highlights an aspect that is inherent in the concept of a *model* beyond the realm of machine learning: A model is not something to simply look up knowledge; a model is something to be used as a cognitive tool in the process of arriving at the desired knowledge. That is, a model is something to work and interact with. On a metaphoric level, a model-free approach provides a dictionary where the answers to a fixed set of specific questions are listed and can be looked up quickly whereas a model-based approach rather corresponds to providing a textbook that has to be read and understood but afterwards allows to apply the contained knowledge in a flexible manner to arrive at the desired answers. Unfortunately, many models in machine learning are quite similar to look-up tables for predictions and do not fulfill this more elaborate idea of what constitutes a model. A main concern of this thesis is to improve models for planing in this respect and show how they may be used accordingly. Therefore, in order to answer the question how a suitable model for planning looks like we have to understand the characteristics of the planning process so that we can provide a model that serves as an appropriate tool.

In Chapter 4 I develop the idea of planning as an active learning process. The two main implications that follow from this approach with respect to designing a model are the following: First, a model should provide a versatile interface for querying the required knowledge. Just as a good textbook does not simply contain a list of isolated facts a good model should allow to query the same information in different ways, with respect to different contexts, and put it in relation to other pieces of information. Second, while being versatile, the interface should, at the same time, adhere to certain structural standards. Ideally, it should be possible for a planner to use a new model without first 'studying the manual', that is, without a large overhead for adapting its routines to the new model.

In this chapter I will address three questions: (1) What are the characteristics of a model that provides a versatile interface while adhering to structural standards (Section 3.1)? (2) How can the structure of such a model be learned from data (Section 3.2)? (3) How can both be brought together to produce a tangible model that can be applied to concrete problems (Section 3.3)?

3.1 Model Characteristics

In this section I will discuss what characteristics a structured model for active planning should exhibit. The first crucial feature is the ability to represent adaptable abstractions, as discussed in Section 3.1.1. Second, in order for the planner to be able to access these abstractions the model must provide an interface that adheres to certain structural standards and constraints, as discussed in Section 3.1.2.

3.1.1 Adaptable Abstraction

An indispensable means to tackle the planning problem is *abstraction*, that is, the ability to ignore certain details in order to get the "big picture". Abstraction can be seen as a compressed representation of the environment that retains the most relevant information. What is relevant in a given environment and what not depends, for one thing, on the specific task that is to be solved: Different abstractions may be required to solve different tasks in the same environment. For another thing, what is relevant depends on details of the planning process, that is, *how* a solution is going to be achieved: A planner with access to very limited computational resources, for instance, is forced to use a higher level of abstraction as it can not handle the complexity of a low-level description. As a consequence, if a model is supposed to be used by different planners to solve a variety of tasks we cannot expect a single abstraction or even a single hierarchy of abstractions to fulfill all the different needs. Instead, our model has to provide a multitude of abstractions at different levels of granularity and along different dimensions such that each planner can actively pick an appropriate abstraction that suits its specific characteristics and those of the task it is trying to solve.

Dimensions of Abstraction

In a decision process there are three canonical dimensions for abstraction: the state space, the action space, and time. A generic way to perform abstraction in state or action space are *aggregation* methods that group multiple states or actions into one. Boutilier, Dean, and Hanks (1999) differentiate between uniform/non-uniform, exact/approximate, and fixed/adaptable aggregation methods. Progressive widening in MCTS, for instance, is a form of non-uniform, approximate, adaptable abstraction. Abstraction along the temporal dimension is more involved as it usually also implies some kind of state and action abstraction since both change over time. A prevailing idea for temporal abstraction are *macro actions* (Section 2.1.4). Macro actions comprise abstraction along all three dimensions (states, actions, time), which highlights the fact that the dimensions of abstraction are not predefined and will in general be embedded in state-action-time space, that is, in the space of trajectories. This is an important aspect because it implies that even in a Markov environment appropriate abstractions will generally break the Markov property.

In addition to state-action abstraction and macro actions the decompositions and abstractions discussed in Section 2.2.2 are relevant, that is, serial decomposition, parallel decomposition, and hierarchical decomposition.

Modes of Adaptation

When we look at existing approaches to abstraction and group them based on two properties, namely

1. based on whether they are *static* or *adaptable* and
2. based on whether they are *generic* or *problem-specific*,

they mainly fall into one of two categories. On the one hand, there are abstractions that are *static* and *problem-specific*, such as a predefined set of options or a hierarchical decomposition of the state space as in HEXQ. On the other hand, there are abstractions that are *adaptable* and *generic*, such as the unclustering of states and actions in progressive widening. To optimally support active planning, however, we need abstractions that are *adaptable* and at the same time *problem-specific*. There are two reasons why this combination is rarely found in models that are learned from data. For one thing, extracting the structural information needed to represent an adaptable abstraction for a specific problem is a difficult learning task. This is, for instance, reflected in the various attempts to automatically extract the information required for defining options for a specific problem. For another thing, planning methods that are designed to work with learned models typically assume a state-based Markov representation and are not designed to make use of adaptable abstractions even if this was supported by the model. This results in a deadlock where there is little incentive for change, neither on the modeling side nor on the planning side. To overcome this situation we need to identify some minimum common ground that serves as a target to work towards from both sides.

There are two modes of adaptation that are relevant for active planning independently of the specific environment and task. At the same time, their realization will depend on the specific problem:

- The *level of detail* of an abstraction has to vary as planning proceeds to promote a general branch-and-bound[6] strategy: At an early stage a low level of detail permits to ignore large parts of the search space that appear to be sub-optimal while at a later stage a high level of detail is required to converge to the exact optimum. In classical planning this procedure of successively increasing the level of detail of a plan is called *refinement* (Ghallab, Nau, and Traverso, 2016). To allow for refinement during planning the corresponding structures have to be learned as part of the model.

- An abstraction has to be *context-specific* in order to adapt to the specific task and the prevailing preconditions. Distinctions that are crucial for one task may be completely irrelevant for another and some aspects are completely determined by the given preconditions. In either case, the model should not force the planner to deal with aspects that are better ignored. The model thus has to adapt to the given situation and context. These adaptations are somewhat different from those for specifying a level of detail but they also need to be accessible for explicit adaptation by the planner as this allows to simulate certain situations and preconditions during planning.

It has to be possible for planner to adapt these properties using some kind of interface to interact with the model. A possible structure for this interface is suggested in the next section.

3.1.2 Structural Constraints

In the last section I discussed relevant characteristics of an adaptable abstraction for planning:

- The abstraction has to be adaptable to specific situations.

[6]I am also subsuming “soft” branch-and-bound strategies such as UCB1 for multi-armed bandits under this term.

- The level of detail has to be adaptable.
- These adaptations have to be applied to state-action abstraction, macro actions, as well as parallel, serial, and hierarchical decompositions.

We need to identify a general model structure that allows to implement these properties. To adapt the different properties of the abstraction the model has to provide a number of parameters that can be changed by the planner. While in principle this parameter space might have an arbitrary structure, the role of abstraction in planning suggests certain properties to be particularly beneficial. Abstraction is about ignoring certain aspects during planning, which suggests parameters to be bounded by zero (completely ignoring an aspect) and one (completely taking an aspect into account). Furthermore, on the representational level *ignoring an aspect to a certain extent* amounts to not ignoring it at all since the planner has to represent it even if it is assigned a lower relevance. This suggests in particular a set of discrete binary parameters. The parameters do not need to be independent, for instance, the level of detail might be adjustable by a series of coupled binary parameters. Of course, these could be represented by a single integer but this would hide the important aspect that from a representational point of view distinct aspects can only be either *on* or *off*. As a consequence, it seems that feature-based models that allow to flexibly decide which feature to ignore and which ones to include are a particularly favorable choice for structured models.

Feature-Based Models

Working with features of the original states and actions offers great flexibility. It allows representing adaptable abstractions along arbitrary dimensions in state-action space. As discussed above, useful abstractions will more generally be embedded in the trajectory space, which means that the employed features should be features of entire trajectories and their domain should extend in time. This allows the representation of temporal aspects, such as whether a specific sub-goal has been reached, which is needed for macro actions and serial decompositions. Features describing independent aspects of the state and action space can be used to describe parallel decompositions. Finally, hierarchical abstractions can be described by subsets of dependent features, where the values of higher-level features completely determines those of lower-level features.

An important question is how the option of *ignoring a feature* is realized. Ignoring a feature must not break the model, that is, the model has to provide meaningful predictions even if an arbitrary number of features is ignored. Using a linear combination of features is convenient insofar as ignoring a feature can be realized by temporally assigning it zero

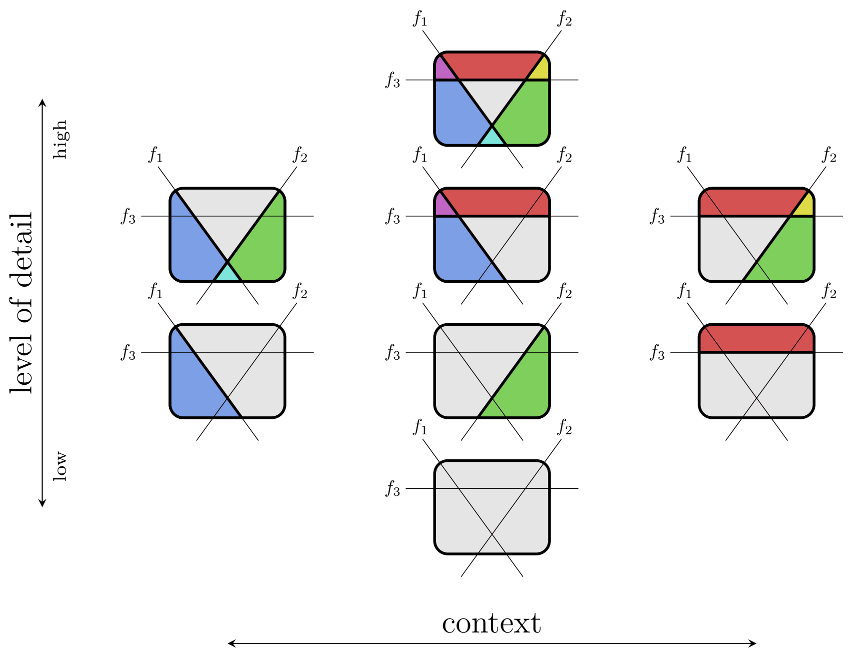


Figure 3.1: Factorized Abstraction: The possibility to arbitrary combine features allows for an exponential number of distinctions that can be made. The abstraction may be adapted to a specific context by choosing which features to include. The level of detail may be increased by adding more features.

weight, which is also semantically consistent with considering the respective information to be irrelevant.

Factorization: The Blessing of Dimensionality

The *curse of dimensionality* refers to the fact that the number of possible assignments for a set of variables grows exponentially with the number of variables. If the variables describe the solution space of a problem exhaustive search quickly becomes intractable as the number of variables (the dimensionality of the problem) grows. If, on the other hand, the task is to find a compact yet powerful and expressive representation this effect turns into a blessing. This is exploited in factorized representations where the number of factors only grows logarithmically with the number of distinctions to be represented. Figure 3.1 illustrates the expressiveness of a factorized representation with three features. Parallel, serial, and hierarchical decompositions can all be considered special cases of factorization. In a parallel decomposition factorization is performed in state-action space while in serial decomposition factorization is performed in time. A hierarchical decomposition corresponds to a constrained factorization where some assignments are prohibited, namely those that break the tree structure. As planning is one of the problems that severely suffer from the curse of dimensionality it seems natural to rely on the blessing of dimensionality for solving it. A factorized representation allows for the expressiveness and flexibility needed for an adaptable abstraction in planning. Feature-based models from the exponential family realize a factorized structure based on a linear combination of features

$$p(\mathbf{x}) \propto \exp \sum_{f \in \mathcal{F}} \theta_f f(\mathbf{x}) = \prod_{f \in \mathcal{F}} e^{\theta_f f(\mathbf{x})} \,. \tag{3.1}$$

This combination seems particularly favorable as it meets all requirements identified so far and I shall mostly work with models of this kind throughout the thesis.

3.2 The *PULSE* Framework

In Section 3.1 I specified certain characteristics that a structured model for active planning should exhibit, among others structural constraints that allow a planning algorithm to interact with the model. Given these structural constraints the question is how a concrete model structure can be found. As the model structure is supposed to represent characteristics of the environment that are used for abstraction it has to be learned from data. In this section, I will present a general framework for designing domain-specific algorithms that learn the model structure.

3.2.1 General Idea and Problem Statement

The challenge of solving a machine learning problem emerges from the interplay of two major components. On the one hand, there is an abundant number of machine learning methods that each have their own characteristics, strengths, and pitfalls. On the other hand, each problem domain comes with its own particularities and requirements. The challenge consists in finding a good match between a machine learning approach and the specific problem that is to be solved. This requires a profound understanding of both sides – of the technical details as well as of the assumptions that can be made for the given problem – to arrive at a solution that is practically feasible and produces results that meet the requirements. In that respect, being able to implement domain-specific prior knowledge is the crucial aspect.

Prior Knowledge

Prior knowledge can be introduced in the form of *hard assumptions*, by ignoring certain options that are thought to be strictly impossible, or in the form of *soft biases*, by giving

	hard assumptions	*soft biases*
make problem tractable	Reduce the space of possible options so that it may be searched more easily.	Provide heuristic during search so that good results are found more quickly.
improve generalization	Ignore options that are known to be wrong even if the data support them.	Counterbalance missing information and noise by weighing up empirical evidence and expected results.

Table 3.1: Contributions of prior knowledge to practical feasibility and generalization properties.

more weight to certain options as long as there is no contradicting evidence.[7] Prior knowledge is required to make a machine learning problem tractable (cf. the *no free lunch theorems*, D. H. Wolpert and W. G. Macready, 1997; David H. Wolpert, William G. Macready, et al., 1995) and improves the generalization properties in case of ambiguous or scarce data. The contribution of hard assumptions and soft biases to both aspects is summarized in Table 3.1.

In most cases, solving a machine learning problem is a hermeneutic process (cf. Gadamer, 2008) where both the understanding of the machine learning methods as well as the understanding of the problem domain is deepened. During this process prior knowledge may enter the final solution in different ways. On the one hand, it may be *explicitly* formulated in terms of precise objectives or criteria, which enter on the abstract *computational level*[8] by specifying what the algorithm is supposed to compute (for instance the solution to an optimization problem). On the other hand, prior knowledge may be *implicitly* introduced by specifying certain procedures, which enter on the *algorithmic level*. In the latter case the decisions are guided by experience, intuition and knowledge that is not necessarily reflected. The creator of an algorithm may neither be aware of her prior knowledge nor would she be able to explicitly formulate it on the computational level. Specifying this kind of implicit prior knowledge in terms of an objective function is therefore not usually possible.

Still, the inclusion of implicit prior knowledge is an important and inevitable part of solving a machine learning problem. It is therefore desirable to provide methods for conceptualizing and formalizing this process to the largest extent possible. With the *PULSE* framework I hope to contribute to this endeavor by formalizing the process of defining a good model structure.

[7]While a hard assumption can be considered the limiting case of a soft bias there is a crucial difference: As long as an option is not strictly excluded it has to be represented in some way, which requires additional resources.

[8]In the sense of Marr's three levels of analysis.

Learning the Model Structure

A major challenge in learning the model structure is that it should be chosen according to the same objective that is explicitly specified on the computational level. However, this objective can usually only be used to evaluate a given structure but not to directly deduce it. As a result finding the best model structure amounts to a random search through structure space unless implicit prior knowledge is used as a heuristic for guiding this search.

PULSE stands for ***P**eriodical **U**ncovering of **L**ocal **S**tructure **E**xtensions.* The idea of the *PULSE* framework is to add an additional level of adaptation for the model structure on top of optimizing the model parameters. To this end, the algorithm designer specifies an operation that suggests modifications to the model structure and is used to guide the search through structure space. In this way, the implicit prior knowledge that would otherwise be used to come up with a good model structure in a tedious process of trial and error is made explicitly available to the *PULSE* algorithm and can thereby by integrated in the automated learning process.

3.2.2 Definitions and Assumptions

In this section I formally state a number of definitions and assumptions underlying the *PULSE* framework.

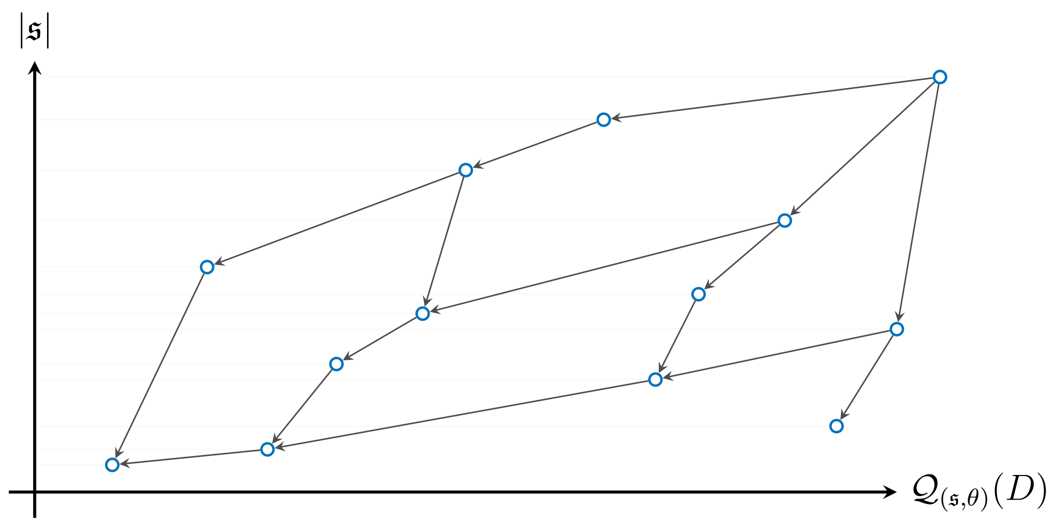
Definition 1 (Model Structure and Parameters)**.** *Let $\mathfrak{S}$ be the space of possible model structures. For each structure $\mathfrak{s} \in \mathfrak{S}$, let $\Theta_\mathfrak{s}$ be the space of possible model parameters. A model $M_{(\mathfrak{s},\theta)}$ is uniquely defined by its structure $\mathfrak{s} \in \mathfrak{S}$ and its parameters $\theta \in \Theta_\mathfrak{s}$.*

Everything being said transfers to parameter-free methods, such as Gaussian processes or nearest neighbor methods, with the only difference that some assumptions become superfluous.

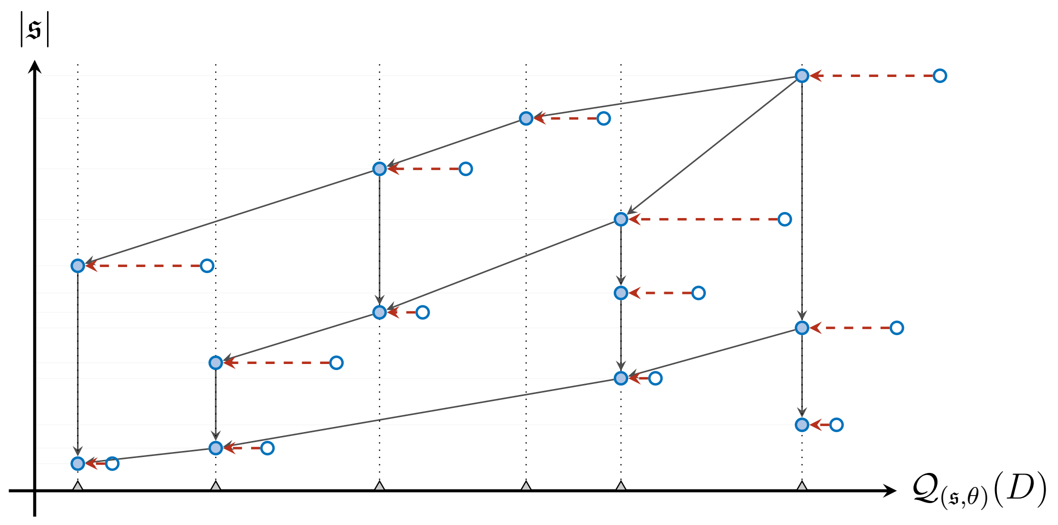
Definition 2 (Quality)**.** *The* model quality *$\mathcal{Q}_{(\mathfrak{s},\theta)}(D)$ is a property of model $M_{(\mathfrak{s},\theta)}$, that is of the pair of structure $\mathfrak{s}$ and parameters θ, for a specific set of data D. Model qualities are totally ordered and bounded and we write*

$$\mathcal{Q}_{(\mathfrak{s},\theta)}(D) < \mathcal{Q}_{(\mathfrak{s}',\theta')}(D) \tag{3.2}$$

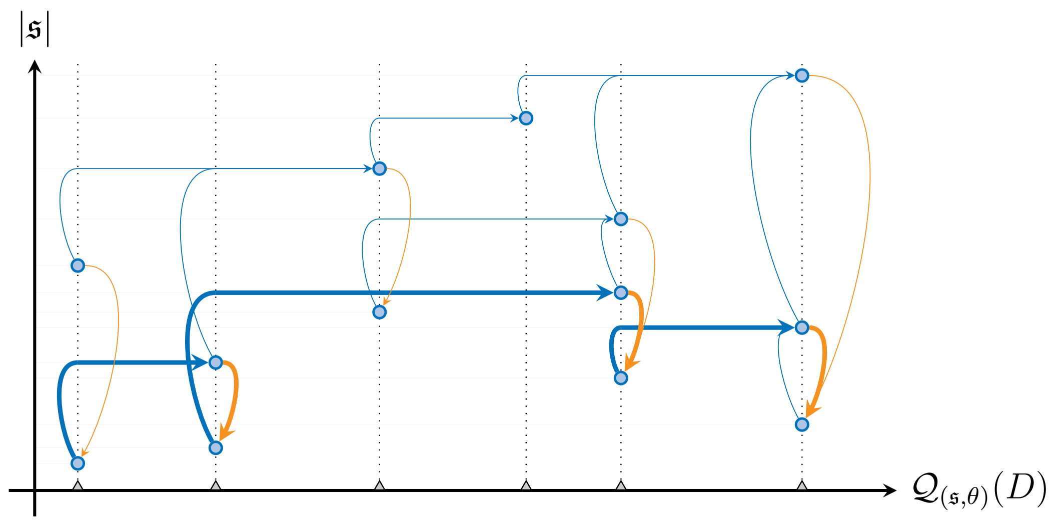
to denote that, on data D, the quality of model $M_{(\mathfrak{s}',\theta')}$ is higher than that of model $M_{(\mathfrak{s},\theta)}$.



(a) Reducibility and Quality



(b) Penalizing Complexity



(c) Learning with *PULSE* using N^+ (blue) and N^-_D (orange).

Figure 3.2: Visualization of model structures with complexity along the vertical axis and model quality along the horizontal axis. Empty circles indicate the optimal quality with respect to the original non-sparse objective, filled circles indicate the quality after penalizing complexity. Black arrows indicate reducibility.

If model parameters are chosen by optimizing an objective function, the value of the objective function corresponds to the model quality. For reasons of simplicity, I will assume that the parameter space is continuous and the model quality is a continuous mapping to the real numbers, which simplifies the presentation but is not essential for the results.

> **Corollary 1** (Optimal Quality)**.** *As the model quality is totally ordered and bounded a structure $\mathfrak{s}$ has a unique* optimal quality
>
> $$\mathcal{Q}^*_{\mathfrak{s}}(D) := \max_{\theta} \mathcal{Q}_{(\mathfrak{s},\theta)}(D) \tag{3.3}$$
>
> *for data D. We call parameters θ^** optimal *if they achieve an optimal quality, that is, $\mathcal{Q}_{(\mathfrak{s},\theta^*)}(D) = \mathcal{Q}^*_{\mathfrak{s}}(D)$.*

> **Definition 3** (Reducibility)**.** Reducibility *is a partial order of structures. We write $\mathfrak{s}' \succ \mathfrak{s}$ to indicate that $\mathfrak{s}'$ is* reducible *to $\mathfrak{s}$ ($\mathfrak{s}$ is a* reduction *of $\mathfrak{s}'$) and, likewise, if $\mathfrak{s} \prec \mathfrak{s}'$ we say that $\mathfrak{s}$ is* extendable *to $\mathfrak{s}'$ ($\mathfrak{s}'$ is an* extension *of $\mathfrak{s}$). $\mathfrak{s} \preceq \mathfrak{s}'$ and $\mathfrak{s}' \succeq \mathfrak{s}$ include the possibility that $\mathfrak{s}$ and $\mathfrak{s}'$ are the same structure.*

It is convenient to assume an additional property of structures

> **Definition 4** (Complexity)**.** *The* complexity $|\mathfrak{s}|$ *of a structure $\mathfrak{s}$ is a total order of structures that respects the partial order of reducibility. That is, if $\mathfrak{s}'$ is reducible to $\mathfrak{s}$ it has a higher complexity*
>
> $$\mathfrak{s}' \succ \mathfrak{s} \Rightarrow |\mathfrak{s}'| > |\mathfrak{s}| \; . \tag{3.4}$$

While the existence of complexity as a total order of structures is not formally relevant in what follows it allows to formulate the intuition that making a model more complex by extending its structure leads to higher computational costs for processing its structure and optimizing its parameters. As the goal of the *PULSE* framework is to define a practically feasible method for finding the optimal model structure computational costs play an essential role.

> **Assumption 1** (Reducibility and Optimal Quality)**.** *If structure $\mathfrak{s}'$ is reducible to structure $\mathfrak{s}$, $\mathfrak{s}'$ has an optimal quality at least as high as the one of $\mathfrak{s}$ (on the same data D), that is*
>
> $$\mathfrak{s}' \succ \mathfrak{s} \Rightarrow \mathcal{Q}^*_{\mathfrak{s}'} \geq \mathcal{Q}^*_{\mathfrak{s}} \; . \tag{3.5}$$

The converse is not necessarily true, that is the optimal quality of $\mathfrak{s}'$ may be higher than that of $\mathfrak{s}$ without $\mathfrak{s}'$ being reducible to $\mathfrak{s}$. The relation between reducibility and quality is visualized in Figure 3.2a.

The definitions and assumptions so far are very general and there are many methods that fall into this scope. The following two assumptions are much more particular and essential to the character of the *PULSE* framework.

Assumption 2 (Equivalent Parameters)**.** *Let structure $\mathfrak{s}'$ be reducible to $\mathfrak{s}$. Then for any set of parameters $\theta \in \Theta_{\mathfrak{s}}$ there exist equivalent parameters $\theta' \in \Theta_{\mathfrak{s}'}$ such the corresponding models $M_{(\mathfrak{s}',\theta')}$ and $M_{(\mathfrak{s},\theta)}$ have the same quality. That is*

$$\mathfrak{s}' \succ \mathfrak{s} \Rightarrow \forall_{\theta\in\Theta_{\mathfrak{s}}} \exists_{\theta'\in\Theta_{\mathfrak{s}'}} \mathcal{Q}_{(\mathfrak{s}',\theta')}(D) = \mathcal{Q}_{(\mathfrak{s},\theta)}(D) \ . \tag{3.6}$$

We write $\theta \overset{\mathfrak{s}\prec\mathfrak{s}'}{\equiv} \theta'$ ($\theta' \overset{\mathfrak{s}'\succ\mathfrak{s}}{\equiv} \theta$) to denote that θ and θ' are equivalent for an extension of $\mathfrak{s}$ to $\mathfrak{s}'$ (a reduction of $\mathfrak{s}'$ to $\mathfrak{s}$).

From Assumption 2 follows

Corollary 2 (Quality-Invariant Extension)**.** *If $\mathfrak{s}$ is extendable to $\mathfrak{s}'$ ($\mathfrak{s} \prec \mathfrak{s}'$) for any set of parameters $\theta \in \mathfrak{S}_{\mathfrak{s}}$ an extension to $\mathfrak{s}'$ can be performed without changing the model quality by choosing the corresponding equivalent parameters for $\mathfrak{s}'$.*

Corollary 3 (Quality-Invariant Reduction)**.** *If $\mathfrak{s}'$ is reducible to $\mathfrak{s}$ ($\mathfrak{s}' \succ \mathfrak{s}$) there exists a set of parameters $\theta' \in \mathfrak{S}_{\mathfrak{s}'}$ such that a reduction to $\mathfrak{s}$ can be performed without changing the model quality by choosing the corresponding equivalent parameters for $\mathfrak{s}$.*

Assumption 2 states that we can effectively delay extensions and anticipate reductions on the parameter level. That is, by choosing parameters appropriately we can control whether an extension actually has an effect (Corollary 2) and, likewise, we can “simulate” the effect of a reduction on the parameter level before actually performing it on the structural level (Corollary 3). Additionally, from Assumption 1 follows that the equivalent parameters of the reduced structure are also optimal.

Corollary 4. *If θ^* are optimal parameters for $\mathfrak{s}'$ and there exist equivalent parameters $\theta \overset{\mathfrak{s}\prec\mathfrak{s}'}{\equiv} \theta^*$ for a reduction $\mathfrak{s}$ of $\mathfrak{s}'$ then θ are optimal parameters for $\mathfrak{s}$*

$$\mathfrak{s} \prec \mathfrak{s}' \wedge \theta \overset{\mathfrak{s}\prec\mathfrak{s}'}{\equiv} \theta^* \wedge \mathcal{Q}_{(\mathfrak{s}',\theta^*)} = \mathcal{Q}^*_{\mathfrak{s}'} \Rightarrow \mathcal{Q}_{(\mathfrak{s},\theta)} = \mathcal{Q}^*_{\mathfrak{s}} \ . \tag{3.7}$$

Proof. The proof follows by contradiction: The existence of parameters $\widetilde{\theta}$ such that $\mathcal{Q}_{(\mathfrak{s},\widetilde{\theta})} > \mathcal{Q}^*_{\mathfrak{s}'} = \mathcal{Q}_{(\mathfrak{s}',\theta^*)}$ contradicts Assumption 1 as the optimal quality of $\mathfrak{s}$ would be higher than that of $\mathfrak{s}'$ even though $\mathfrak{s}'$ was assumed reducible to $\mathfrak{s}$. $\square$

In parameter-free methods Assumption 2 is superfluous as we can extend or reduce the model structure without the need to optimize any parameters.

The next assumption is a property of the employed objective function measuring model quality. It is a gradual property, that is, it can be fulfilled to a varying extent and it has a significant influence on the amount of computational resources *PULSE* requires for searching through the space of model structures. It is the technical equivalent of the *principle of parsimony* or *Occam's razor*.

Assumption 3 (Sparse Objective)**.** *With a high probability, optimal parameters θ^* for a structure $\mathfrak{s}'$ have equivalent parameters $\theta \overset{\mathfrak{s}\prec\mathfrak{s}'}{\equiv} \theta^*$ in a reduction $\mathfrak{s} \prec \mathfrak{s}'$.*

As both reducibility of structures and equivalence of parameters are transitive relations Assumption 3 implies that the optimal qualities of many structures coincide, and hence the set of quality-preserving parts (see Definition 6 below) is small. If an objective is not sparse initially it may be possible to modify it in order to fulfill the sparseness assumption, for instance, by adding a regularization term that penalizes structural complexity. One such example is adding an L_1-regularization to a linear combination of features, as discussed in Section 3.3. A regularization term implements a trade-off between an original quality measure and the structural complexity by effectively shifting the optimal parameters of a structure such that they coincide with the equivalent parameters of one of its reductions. This is visualized in Figure 3.2b.

3.2.3 Searching the Structure Space

Section 3.2.2 laid the formal basis for learning with the *PULSE* framework by stating the relevant definitions and assumptions. In this section I will describing the actual learning process in more detail. The goal of the overall process is to find the structure $\mathfrak{s}$ with the best optimal quality $\mathcal{Q}^*_{\mathfrak{s}}(D)$ for a given set of training data D, along with optimal parameters $\theta^* \in \text{argmax}_\theta\, \mathcal{Q}_{(\mathfrak{s},\theta)}(D)$. This corresponds to a hierarchical discrete-continuous optimization problem where in order to evaluate the optimal quality of a structure we need to optimize its parameters. Throughout this process, it is important to keep the complexity of the evaluated structures as low as possible in order to keep the computational costs in a tractable range. The assumptions made in Section 3.2.2 make this optimization, step by step, more tractable.

Assumption 1: The relation between reducibility and optimal quality ensures that extending a structure does not impair optimal quality, while reducing a structure cannot improve optimal quality.

Algorithm 2 The *PULSE* Algorithm

```
Input: $N^+, \mathcal{Q}, D, \mathfrak{s}_0, \theta_0$
Output: $\mathfrak{s}$, $\theta$

 1: Initialize: $\mathfrak{s} \leftarrow \mathfrak{s}_0$, $\theta \leftarrow \theta_0$
 2: repeat
 3:     EXTEND$(\mathfrak{s}, \theta)$
 4:     OPTIMIZE$(\theta)$
 5:     REDUCE$(\mathfrak{s}, \theta)$
 6: until $\mathcal{Q}_{(\mathfrak{s},\theta)}(D)$ converged
 7: return $\mathfrak{s}, \theta$

 8: function EXTEND$(\mathfrak{s}, \theta)$
 9:     $\theta \leftarrow \theta' : \theta' \overset{N^+(\mathfrak{s})\succ\mathfrak{s}}{\equiv} \theta$
10:     $\mathfrak{s} \leftarrow N^+(\mathfrak{s})$
11: end function

12: function OPTIMIZE$(\theta)$
13:     $\theta \leftarrow \operatorname{argmax}_\theta \mathcal{Q}_{(\mathfrak{s},\theta)}(D)$
14: end function

15: function REDUCE$(\mathfrak{s}, \theta)$
16:     $\theta \leftarrow \theta' : \theta' \overset{N_D^-(\mathfrak{s})\prec\mathfrak{s}}{\equiv} \theta$
17:     $\mathfrak{s} \leftarrow N_D^-(\mathfrak{s})$
18: end function
```

Assumption 2: The existence of equivalent parameters provides the general option of reducing a structure without loosing quality.

Assumption 3: A sparse objective allows to actually make use of this option for the case of optimal parameters.

Intuitively, a *PULSE* learner tries to improve model quality by extending the model structure if needed, while saving computational costs by reducing the structure whenever this is possible without loosing quality. These two tasks are performed by the extension operation N^+ and the reduction operation N_D^-. A *PULSE* learner searches for the optimal model structure by repeatedly

1. extending the model structure using the N^+ operation
2. optimizing the parameters
3. reducing the model structure using the N_D^- operation.

This general routine is shown in Algorithm 2 and visualized in Figure 3.2c. Much of the details depend on the concrete definition of the N^+ and N_D^- operation. I will discuss two such examples in Section 3.4 and Section 3.5. In Section 3.2.4 I will present a general convergence analysis for learning algorithms defined within the *PULSE* framework.

Extension Operation N^+

The *extension operation* N^+ is responsible for extending structures during the learning process.

> **Definition 5** (Extension Operation N^+). *The* extension operation N^+ *is a function that maps a structure* $\mathfrak{s} \in \mathfrak{S}$ *to one of its extensions*
>
> $$N^+ : \mathfrak{S} \to \mathfrak{S} \tag{3.8}$$
>
> $$\textit{such that} \qquad N^+(\mathfrak{s}) \succeq \mathfrak{s}\ . \tag{3.9}$$

In a way, the N^+ operation can be considered the core part of a *PULSE* learner. While all other steps of designing a *PULSE* learner – coming up with a set of possible model structures, defining an objective function for assessing model quality, finding a way to optimize model parameters, possibly adding an appropriate regularization to induce sparseness – are common in most machine learning tasks, the N^+ operation is particular to a *PULSE* learner. It represents the essential component where the prior knowledge that would otherwise remain implicit and hidden in the concrete choice of a model structure is formulated explicitly in terms of an extension operation. This makes designing the N^+ operation a challenging task. There are two guiding question for designing a good N^+ operation:

1. Given that the current structure demonstrated a certain potential for producing a good model (in the sense that it cannot further be reduced without loosing model quality): What could be a promising extension that might lead to an improved model quality?

2. For different possible extensions that all bear the potential of improving model quality: Which extension yields, after optimizing its parameters, the most information about promising future extensions?

While designing the N^+ operation the issue of computational costs has to be kept in mind: The quality of an extension has to be practically assessable by optimizing its parameters. In Section 3.2.4 I will discuss more aspects that are relevant for designing the N^+ operation.

Reduction Operation N_D^-

The *reduction operation* N_D^- is responsible for reducing structures during the learning process without a loss of model quality. While the N_D^- operation also shapes the characteristics of a *PULSE* learner it is essentially predetermined by the objective function: The objective function assesses model quality and, thereby, defines sets of structures that can be reduced without a loss of model quality.

> **Definition 6** (Quality-Preserving Partition). *The* quality-preserving partition *of model structures for data D*
>
> $$\mathcal{P}_{\mathfrak{S}}(D) = \{\widetilde{\mathfrak{S}} \subseteq \mathfrak{S} \,|\, \forall_{\mathfrak{s},\mathfrak{s}' \in \widetilde{\mathfrak{S}}} \; \mathcal{Q}^*_{\mathfrak{s}}(D) = \mathcal{Q}^*_{\mathfrak{s}'}(D)\} \tag{3.10}$$
>
> *partitions model structures by their optimal quality.*

The sparseness assumption for the objective (Assumption 3) implies $\mathcal{P}_{\mathfrak{S}}(D)$ having few parts. The N^-_D operation maps each structure to a reduction within the same quality-preserving part.

> **Definition 7** (Reduction Operation N^-_D). *The* reduction operation N^-_D *is a function that maps a structure $\mathfrak{s} \in \mathfrak{S}$ to a quality-preserving reduction*
>
> $$\begin{aligned} & N^-_D : \mathfrak{S} \to \mathfrak{S} && (3.11) \\ \textit{such that} \quad & N^-_D(\mathfrak{s}) \preceq \mathfrak{s} && (3.12) \\ \textit{and} \quad & \mathcal{Q}^*_{N^-_D(\mathfrak{s})}(D) = \mathcal{Q}^*_{\mathfrak{s}}(D) \; . && (3.13) \end{aligned}$$

Generally, we have an interest in mapping to a minimal quality-preserving reduction in order to keep the computational cost low. For reasons of simplicity, I will assume that there is a unique minimal reduction for all quality-preserving parts.

> **Assumption 4** (Unique Minimal Reduction). *All quality-preserving parts $\wp \in \mathcal{P}_{\mathfrak{S}}(D)$ have a unique minimal structure* $\min(\wp)$.

There are two versions of how Assumption 4 can be employed in the N^-_D operation.

> **Assumption 4a.** N^-_D *maps all structures of a quality-preserving part to their unique minimal reduction.*

Under Assumption 4a, whenever the model quality does not strictly improve after extension by N^+ the structure is mapped back to the minimal reduction, which results in a loop and immediately terminates the search. This might be what we want, for instance, if computational resources are scarce and we want to avoid model growth. However, better theoretical convergence guarantees – and practical ones given enough computational resources – are achieved with a slightly modified version.

Assumption 4b. N_D^- *maps all structures of a quality-preserving part to their unique minimal reduction* unless *the last* N^+ *operation started in the same quality-preserving part, in which case the structure is not reduced, that is, it is mapped to itself.*[9]

Under both versions of Assumption 4 N_D^- is uniquely defined by the objective function and the given data D.

3.2.4 Convergence Properties

In this section I am concerned with an analysis of the convergence properties of a *PULSE* learner with respect to the model structure. General converge is guaranteed by a monotone improvement of the model quality.

Theorem 1 (Convergence of a *PULSE* Learner)**.** *A PULSE learner is guaranteed to converge.*

Proof. Each iteration consists of an extension, which guarantees monotonic quality improvement (Assumption 1), followed by a quality-preserving reduction (Definition 7). As the model quality is bounded a monotonic improvement implies convergence. □

The fact that a *PULSE* learner converges does neither make a statement about the quality of the result nor about the time it takes to reach it. As the *PULSE* framework is intended to be applicable to different kinds of underlying models and different problem settings, it is difficult to make more definite statements in the general case. In the following, I will discuss some aspects that are relevant for the convergence properties of a *PULSE* learner. I will also try to establish some structures and provide some guidance that is supposed to help designing a *PULSE* learner with good convergence properties. A proof of global convergence to the optimal structure for the models and problems that are relevant in this work can be found in Section 3.3.

First, no matter what kind of underlying model we are using, it is important that for a given model structure we are able to determine optimal parameters.

Condition 1 (Optimal Parameters)**.** *For a given structure* $\mathfrak{s}$ *and data* D *there is a way to efficiently determine optimal parameters* θ^* *with* $\mathcal{Q}_{(\mathfrak{s},\theta^*)}(D) = \mathcal{Q}^*_{\mathfrak{s}}(D)$*.*

This is relevant independently of whether a structure is chosen by hand or optimized using the *PULSE* framework. In that sense, this condition is not particular to *PULSE*. Next,

[9]Technically this means that N_D^- also depends on the previous structure in the learning process.

it is important to better understand the dynamics of the learning process, which can be described using the following three graphs in the space of model structures $\mathfrak{S}$.

Definition 8. *The directed* extension graph

$$G^+ = \left(\mathfrak{S},\ \left\{(\mathfrak{s}, N^+(\mathfrak{s}))\right\}\right) \tag{3.14}$$

has the set of all model structures as vertices and an arc from each structure $\mathfrak{s}$ to its extension $N^+(\mathfrak{s})$.

Definition 9. *The directed* reduction graph

$$G^-_D = \left(\mathfrak{S},\ \left\{(\mathfrak{s}, N^-_D(\mathfrak{s}))\right\}\right) \tag{3.15}$$

has the set of all model structures as vertices and an arc from each structure $\mathfrak{s}$ to its reduction $N^-_D(\mathfrak{s})$.

Definition 10. *The directed* learning graph

$$G^L_D = \left(\mathfrak{S},\ \left\{(\mathfrak{s}, N^-_D(N^+(\mathfrak{s})))\right\}\right) \tag{3.16}$$

has the set of all model structures as vertices and an arc from each structure $\mathfrak{s}$ to the reduction of its extension $N^-_D(N^+(\mathfrak{s}))$, that is, the next structure visited in the learning process.

Each structure has exactly one outgoing arc in each of the graphs G^+, G^-_D and G^L_D. Arcs in G^L_D correspond to the concatenation of an arc in G^+ and the continuing arc in G^-_D. In Figure 3.2c blue arrows correspond to arcs in G^+ and orange arrows to arcs in G^-_D (self-loops are omitted). For a given initial structure $\mathfrak{s}_0$ G^L_D defines the *learning path* taken by a *PULSE* learner.

Definition 11 (Learning Path)**.** *For an initial structure $\mathfrak{s}_0$ the* learning path *is the sequence of consecutive arcs in the learning graph G^L_D.*

There are several ways how a *PULSE* learner can converge in an undesirable way, which are summarized in Table 3.2. For one thing, the limit may be reached only after infinitely many iterations (left column in Table 3.2). Additionally, even if we have enough resources to find a solution close to the limit for that case it may still be sub-optimal: The learner may either get caught in a non-improving loop (this theoretic possibility is already precluded by Assumption 4) or the limit itself may be sub-optimal. If the limit is optimal

	infinite iterations	*finite iterations*
sub-optimal quality	• loop • sub-optimal limit	• dead end • premature stopping (4a)
optimal quality	—acceptable—	—preferred—

Table 3.2: Different causes for sub-optimal convergence.

an approximate solution may be acceptable, however, a finite number of iterations until convergence is generally preferable. In case a finite number of iterations is guaranteed, there are still two potential problems (right column in Table 3.2). For one thing, there might be dead ends where a sub-optimal structure cannot be further extended. This may be avoided by ensuring *extension to the maximal structure*.

> **Condition 2** (Extension to the Maximal Structure). *The space of model structures has a unique maximal structure $\mathfrak{s}_{max}$ that can be reached from any initial structure by repeatedly applying N^+.*

For another thing, there is a dilemma concerning an appropriate stopping criterion. If we use version 4a of Assumption 4 learning stops whenever the quality does not strictly improve. In combination with a sparse objective (Assumption 3) this avoids infinitely many iterations as there are only few quality-preserving parts and staying within one part terminates learning. However, as version 4a may lead to premature stopping when encountering a single non-improving step (which might be a necessary transition on the way to the optimal structure) we might prefer version 4b. In that case, however, there are only two ways how learning may stop: (1) We have an oracle that tells us when we have found the optimal model structure. (2) We run out of computational resources, which does not guarantee finding the optimal structure.

As the maximal structure can always be reduced to the optimal model structure, extending to the maximal structure is a sufficient condition for having found the optimal model but in practice we will usually run out of computational resources before reaching the maximal structure. In special cases, however, it may be possible to define an appropriate oracle that tells us when we have reached the optimal structure (see Section 3.3). The key insight is that we do not need to extend to the maximal structure but only to a (non-maximal) extension of the optimal structure. An additional means to improve convergence properties in practice is to implement the concept of *gradual specialization*, described in the following, which reduces the risk of running out of computational resources prematurely.

Gradual Specialization

A general aim when designing a *PULSE* learner should be to ensure a steady progress evenly distributed along the learning path (to prevent premature stopping) without requiring to process overly complex structures (to retain practical feasibility). To this end the N^+ operation has to extend a given structure in a way that provides a broad chance for quality improvement and the N^-_D operation has to select an appropriate reduction based on the data without significantly impairing the model quality. Generally, small but assured improvements are to be preferred as compared to the chance for a big leap at the risk of stagnation and eventually premature termination. A concept that I deem to be useful in that respect is that of *gradual specialization*. While it is rather meant as a guiding principle I will try to be mathematically as precise as possible. First, consider the following definition of *generalization*, which is the complement of specialization.

Definition 12 (Generalization)**.** *Structure* $\mathfrak{s}'$ generalizes *structure* $\mathfrak{s}$ *conditioned on a reference structure* $\mathfrak{s}_0$ *if for any data* D *a quality improvement of* $\mathfrak{s}$ *relative to* $\mathfrak{s}_0$ *implies a quality improvement of* $\mathfrak{s}'$ *relative to* $\mathfrak{s}_0$ *while at the same time the quality of* $\mathfrak{s}'$ *does not surpass that of* $\mathfrak{s}$

$$\Big[\mathcal{Q}^*_{\mathfrak{s}_0}(D) < \mathcal{Q}^*_{\mathfrak{s}}(D) \Rightarrow \mathcal{Q}^*_{\mathfrak{s}_0}(D) < \mathcal{Q}^*_{\mathfrak{s}'}(D)\Big] \wedge \mathcal{Q}^*_{\mathfrak{s}'}(D) < \mathcal{Q}^*_{\mathfrak{s}}(D)\ . \tag{3.17}$$

We write $\mathfrak{s} \lhd_{\mathfrak{s}_0} \mathfrak{s}'$ *to indicate that* $\mathfrak{s}'$ generalizes $\mathfrak{s}$ *(*$\mathfrak{s}$ specializes $\mathfrak{s}'$*) conditioned on* $\mathfrak{s}_0$*.*

Importantly, generalization is defined in terms of relative improvement, not absolute quality. Intuitively, a generalization abstracts from the detailed properties of a specialization such that it may cover multiple special cases at the cost of a lower quality. In a specific case the quality improvement of the generalization is not as pronounced as that of the corresponding specialization but the improvement is guaranteed for all the different special cases that are covered.

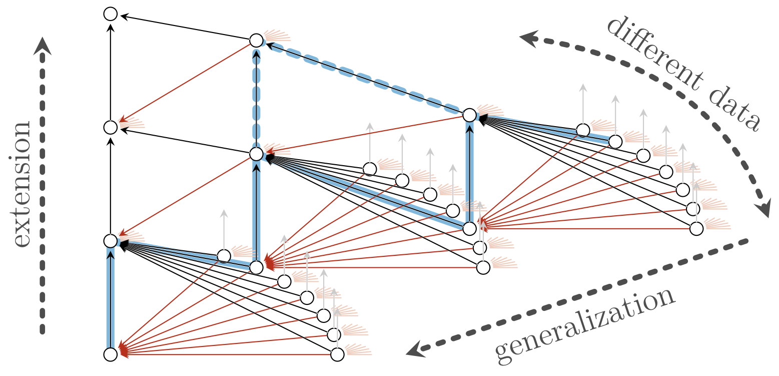
Corollary 5 (Generalization is Transitive)**.**

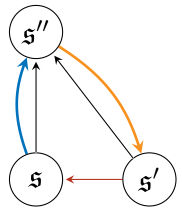
$$\mathfrak{s} \lhd_{\mathfrak{s}_0} \mathfrak{s}' \wedge \mathfrak{s}' \lhd_{\mathfrak{s}_0} \mathfrak{s}'' \Rightarrow \mathfrak{s} \lhd_{\mathfrak{s}_0} \mathfrak{s}''\ . \tag{3.18}$$

Proof. Trivial. □

Corollary 6 (Extensions are not Generalizations)**.** *If a structure* $\mathfrak{s}'$ *is an extension of a structure* $\mathfrak{s}$ *it cannot be a generalization (and vice versa)*

$$\mathfrak{s} \lhd_{\mathfrak{s}_0} \mathfrak{s}' \Rightarrow \neg\Big(\mathfrak{s} \prec \mathfrak{s}'\Big) \tag{3.19}$$





(a) The gray/light-red connections indicate continuations that were omitted to keep the figure clear. The connections with light-blue background indicate possible learning paths achievable via gradual specialization (the dotted path requires version 4b of Assumption 4).

(b) A single step of gradual specialization. The blue and orange arrow indicate the N^+ and N_D^- operation, respectively.

Figure 3.3: Schematic visualization of the concept of *gradual specialization*. Black arrows indicate extensions; red arrows indicate conditional generalizations that are not extensions. Conditional generalizations are assumed to be conditioned on the preceding structure on the learning path.

$$\mathfrak{s} \prec \mathfrak{s}' \Rightarrow \neg\Big(\mathfrak{s} \lhd_{\mathfrak{s}_0} \mathfrak{s}'\Big) \ . \tag{3.20}$$

Proof. Extensions have a higher quality than their reductions, which is incompatible with the requirement that generalizations have a lower quality than their specializations. □

The basic working principle of gradual specialization is depicted in Figure 3.3. The idea is to start with a very general structure and gradually become more specific until reaching the optimal structure while keeping approximately the same level of structural complexity. This becomes possible because generalizations and extensions are related to quality improvements in a very different way. In a single iteration (Figure 3.3b) we transition from a general structure $\mathfrak{s}$ to a specialization $\mathfrak{s}'$ by going via a common extension $\mathfrak{s}''$ of both $\mathfrak{s}'$ and $\mathfrak{s}$. The important aspect is that $\mathfrak{s}''$ may be the common extension of many different specializations of $\mathfrak{s}$ (Figure 3.3a), which allows N_D^- to pick $\mathfrak{s}'$ based on the data. This means that over multiple iterations the *PULSE* learner can work its way towards more and more specific model structures, guided by the data, without having to process complex structures on its way. Formally, the condition of gradual specialization can be formulated as follows.

Condition 3 (Gradual Specialization)**.** *Let $\mathfrak{s}$ be the current structure and $\bar{\mathfrak{s}}$ be the previous structure on the learning path. Then the N^+ operation should extend $\mathfrak{s}$ in* *such a way that for any possible data D there is a structure $\mathfrak{s}'$ among the reductions of $N^+(\mathfrak{s})$ that specializes $\mathfrak{s}$ conditional on $\bar{\mathfrak{s}}$*

$$\mathfrak{s} \rhd_{\bar{\mathfrak{s}}} \mathfrak{s}' \tag{3.21}$$

and generalizes the optimal structure $\mathfrak{s}^*$ conditional on $\mathfrak{s}$

$$\forall_{\mathfrak{s}\neq\mathfrak{s}^*} \exists_{\mathfrak{s}'\prec N^+(\mathfrak{s})} \mathfrak{s}^* \lhd_{\mathfrak{s}} \mathfrak{s}' \ . \tag{3.22}$$

N_D^- should map $N^+(\mathfrak{s})$ to $\mathfrak{s}'$.

The first sub-condition (3.21) is not required to guarantee convergence but ensures an uninterrupted chain of specializations to the optimal structure. By conditioning on the current structure in the second sub-condition (3.22) Condition 3 ensures a strict monotonic quality improvement in every iteration, which circumvents the problem of premature stopping and guarantees convergence to the optimal structure.

Non-Disruptive Sparseness

Another aspect that has not been discussed so far is the role that the level of sparseness plays in choosing a good compromise between the quality of the optimal model structure and the practical feasibility of finding it. The sparseness property of the objective (Assumption 3) has to implement a trade-off between an original quality measure and a penalty for complex model structures that are expensive to process. Choosing the right level of sparseness is a major factor for good convergence properties: If the objective is not sparse enough it allows improvements in small steps, which avoids premature stopping, but there are too few quality-preserving reductions in each quality-preserving part so that the model complexity grows extensively and learning eventually terminates due to a lack of computational resources. If, on the other hand, the objective is too sparse, relevant differences between structures are ignored by squeezing them into the same quality-preserving part and learning terminates prematurely due to a lack of quality-improving extensions. Some of the possible convergence problems mentioned above arise mainly due to an unfavorably chosen sparseness. While in the general case it seems hard to give precise instructions on how to choose the sparseness correctly, it is worth keeping in mind that it should be *non-disruptive*.

Condition 4 (Non-Disruptive Sparseness)**.** *The objective function should induce enough sparseness to save relevant computational resources but it should not interfere with a gradual improvement of model quality.*

Realizing the concept of gradual specialization crucially builds on the objective being non-disruptively sparse.

3.2.5 Different Views

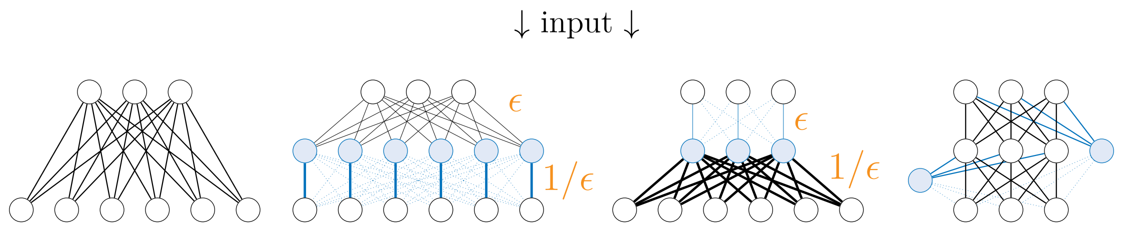
The *PULSE* framework provides a novel and versatile approach to learning model structures. In this section I discuss how the *PULSE* framework can be view in relation to some other meta-learning frameworks.

Evolutionary Algorithm View Viewing *PULSE* as a kind of evolutionary algorithm seems particularly intuitive if the model structure is defined in terms of some sort of population, like a set of components, features or factors. The N^+ operation then takes the role of the mutation operation that adds new individuals to the population, while the N^-_D operation selects individuals based on their fitness. However, in contrast to the standard procedure in evolutionary algorithms, where the fitness of each individual is determined independently, in the *PULSE* framework the fitness of the population is assessed as a whole even though eventually single individuals are selected to survive.

Markov Chain Monte-Carlo View With slight modifications the *PULSE* framework becomes similar to *Markov chain Monte-Carlo* (MCMC) methods like Metropolis-Hastings. If we include stochasticity in the N^+ operation (which is an interesting option independently of the connection to MCMC) model structures are effectively drawn from a proposal distribution while the N^-_D operation takes the role of the acceptance/rejection distribution. Exploring this relation between the *PULSE* framework and MCMC methods in more depth is an interesting research topic.

Probabilistic Programming View Probabilistic programming is becoming an increasingly popular approach for applied machine learning. Its main appeal comes from the fact that expert knowledge is formulated in terms of a stochastic generative procedure or *probabilistic program*, which is often more intuitive to specify than a mathematical probabilistic model, especially if the domain experts are not trained in formulating such models. A stochastic N^+ operation in the *PULSE* framework can be understood as a probabilistic program that generates model structures. Learning with *PULSE* then corresponds to inference in probabilistic programming. This view goes well with the MCMC view from above as MCMC is a common approach for inference with probabilistic programs.

Meta Language View For the case of binary feature-based models the N^+ operation can be understood as a meta language or a generative grammar. If features are



(a) Initial network **(b)** Duplicate output layer **(c)** Duplicate input layer **(d)** Adding single neurons

Figure 3.4: Neural networks may be manipulated by duplicating layers or adding single neurons. Newly added neurons and connections are show in blue. Solid connections have non-zero weight with scaling factors indicated in orange ($0 < \epsilon \ll 1$), dotted connections have zero weight.

interpreted as statements about the world that can be either `true` or `false` the role of the N^+ operation is to generate statements that are useful for describing the actually observed data. As opposed to classical generative grammars, however, the N^+ operation generates new statements conditional on an existing set of useful statements. It is an interesting option for further research to explore the use of existing generative grammars, for instance in natural language processing or music, for defining domain specific N^+ operations.

3.2.6 Different Models

The *PULSE* framework provides a general approach for learning model structures and is not bound to the specific kind of feature-based models developed in Section 3.3. To make use of other kinds of models with *PULSE* it is crucial to ensure that for any change of the model structure there exists a set of equivalent parameters (Assumption 2) that allows for a quality-invariant extension and reduction. An appealing property of features-based models for *PULSE* is the possibility to extend the model structure by combining existing features. In the following I discuss three alternatives that can be used in stead of feature-based models or in conjunction with feature-based models by implementing the feature functions.

Neural Networks Neural networks display state-of-the-art performance in many machine learning tasks. At the same time finding the appropriate structure of a neural network for a specific task is hard to automatized and usually involves a considerable amount of experience. *PULSE* can be used to explore the space of neural network structures by defining appropriate N^+ operations. Figure 3.4 shows some possible manipulations to the neural network structure. Given an initial network (Figure 3.4a) it is possible to add complete layers (Figure 3.4b and 3.4c) where the

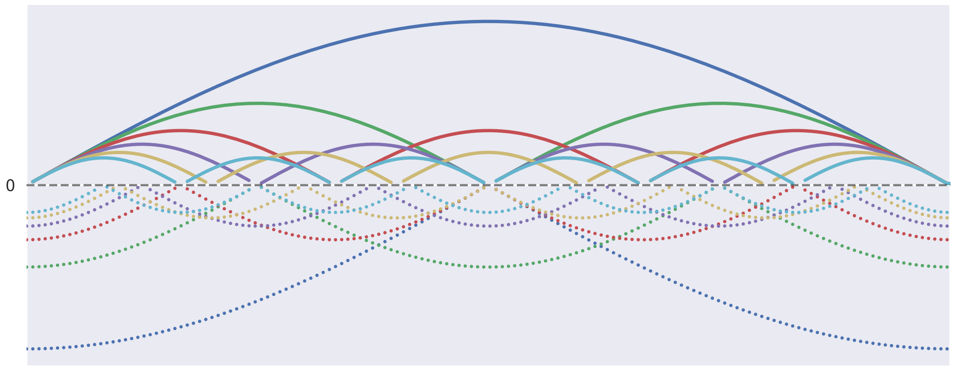


Figure 3.5: Piecewise sine (solid) and cosine (dotted) basis functions up to order 6. Different sine/cosine functions of the same order have disjoint support and are plotted in the same color.

weights are set such that the newly added layer acts in its linear regime and the overall output is not changed. It is also possible to add one or more separate neurons (Figure 3.4d) extending an existing layer or ignoring the layer structure. In that case output weights to existing neurons should have zero weight to retain the overall output while all other weights should be non-zero to avoid vanishing gradients in subsequent training. Neural networks can also be employed to implement feature functions where in addition to modifying the network structure multiple networks may be combined to form a new feature.

Function Approximation The feature-based models discussed in this work perform function approximation for the special case of probability distributions over discrete spaces. More generally, *PULSE* may be used to discover features constructed from large sets of basis functions where a single feature is the cross-product of multiple basis functions and all features are linearly combined to model the given data. It is beneficial if the basis functions are not orthogonal (they may even be linearly dependent, that is, overcomplete) in order to implement the concept of gradual specialization (Condition 3). Figure 3.5 shows an example of piecewise sine and cosine functions with finite support. This set of functions exhibits a hierarchical structure where higher-order basis functions are non-orthogonal to the corresponding lower order functions. The N^+ operation may thus be defined such that it refines an existing basis function by adding the corresponding higher-order basis functions. Structured overcomplete sets of basis function such as wavelets may also be suited for feature discovery using *PULSE*.

Stochastic Finite State Machines Stochastic FSMs can be manipulated by adding new states and setting their incoming probabilities to zero. If using a factored state

general case		**feature-based case**		**def/ass**
space of model structures	$\mathfrak{S}$	powerset of features	$\mathcal{P}(\mathfrak{F})$	1 / –
single model structure	$\mathfrak{s} \in \mathfrak{S}$	subset of features	$\mathcal{F} \subseteq \mathfrak{F}$	1 / –
parameters for a structure	$\theta \in \Theta_{\mathfrak{s}}$	feature weights	$\theta \in \Theta_{\mathcal{F}} \equiv \mathbb{R}^{\lvert\mathcal{F}\rvert}$	1 / –
model quality	$\mathcal{Q}_{(\mathfrak{s},\theta)}(D)$	data likelihood	$\prod_{x\in D} p(x;\mathcal{F},\theta)$	2 / –
reduction	$\mathfrak{s} \prec \mathfrak{s}'$	subset	$\mathcal{F} \subset \mathcal{F}'$	3 / –
complexity	$\lvert\mathfrak{s}\rvert$	size of feature set	$\lvert\mathcal{F}\rvert$	4 / –
equivalent parameters	$\theta \overset{\mathfrak{s}\prec\mathfrak{s}'}{\equiv} \theta'$	non-zero feature weights	$\theta'_f = \begin{cases} \theta_f & \text{if } f \in \mathcal{F} \\ 0 & \text{else} \end{cases}$	– / 1, 2
extension operation	$N^+(\mathfrak{s})$	add new features	$N^+(\mathcal{F}) \supseteq \mathcal{F}$	5 / –
complexity penalty		Laplace prior	$\prod_{f\in\mathcal{F}} \exp(\,-\lvert\theta_f\rvert\,)$	6 / 3
reduction operation	$N_D^-(\mathfrak{s})$	remove zero-weight features	$N^+(\mathcal{F}) := \{f \in \mathcal{F} \mid \theta^*_f \neq 0\}$	7 / 4

Table 3.3: Feature-Based Models. $\mathfrak{F}$ is the set of all possible features, θ^* denotes optimal parameters. The subset relation always defines reductions, for some features there might be additional reductions possible.

representation, new factors may be added if the FSM outputs do not depend on them. The new states and state factors are effectively included by training the FSM transition and output probabilities. When using FSMs to implement feature functions they may be combined in a factored or hierarchical manner. In a factored combination the state space of the new FSM is the product space of the initial FSMs and its output is a mapping from the initial FSMs' outputs. In a hierarchical combination a top-level FSM determines which of the low-level FSMs is used to generate the output.

3.3 Feature-Based Models

In this section I will describe how the *PULSE* framework can be applied to the task of learning a predictive feature-based model. The scenario is that we have a very large or infinite set of features $\mathfrak{F}$ and we want to find a subset of features and corresponding feature weights that maximize the likelihood of the given data D. The transfer from the general notation used in Section 3.2 to the feature-based case is listed in Table 3.3. The effective model quality, including a complexity penalty in the form of a prior distribution (or regularization) over feature weights, corresponds to the maximum posterior probability of the feature weights. I assume that features are combined linearly so that for the data likelihood removing a feature from the model is equivalent to setting its weight to zero.

Additionally, I assume that a zero weight also does not contribute to the complexity penalty. As a consequence, Assumptions 1, 2, 3, and 4 are easily shown to be fulfilled:

Assumption 1: For $\mathcal{F} \subseteq \mathcal{F}'$ the model quality is preserved by using identical weights for features f that are common in $\mathcal{F}$ and $\mathcal{F}'$ ($f \in \mathcal{F} \cup \mathcal{F}'$) and zero weight for features $f \in \mathcal{F}' \setminus \mathcal{F}$ that are not in $\mathcal{F}$. The optimal quality for $\mathcal{F}'$, where features $f \in \mathcal{F}' \setminus \mathcal{F}$ may have non-zero weight, is at least as high as the one of $\mathcal{F}$.

Assumption 2: For $\mathcal{F} \subseteq \mathcal{F}'$ the equivalent parameters $\theta \overset{\mathcal{F} \preceq \mathcal{F}'}{\equiv} \theta'$ are defined as in Table 3.3.

Assumption 3: Dropping any subset of the zero-weight features results in a quality-preserving reduction. Since a Laplace prior over feature weights (an L_1-regularization in the neg-log-likelihood) assigns high probability to zero-weight features the probability for equivalent parameters in a reduction is high.

Assumption 4: The unique minimal quality-preserving reduction corresponds to removing all zero-weight features.

For particular kinds of features it may be possible to define extensions and equivalent parameters independently of the superset relation between feature sets, such as in Eqs. (3.71) and (3.72) below. In these cases, Assumption 1 and 2 have to be shown to hold explicitly.

As discussed in Section 3.1.2 feature-based models from the exponential family have major advantages. Let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_n = \boldsymbol{\mathcal{X}}$ be a set of discrete random variables $\mathcal{X}_i \subset \mathbb{N}$ with an associated probability distribution $p(\mathbf{x})$. I write

$$p_{x'_1 \cdots x'_n} := p(x_1 = x'_1, \ldots, x_n = x'_n) \tag{3.23}$$

to denote the probability for a specific assignment of variables. Let

$$p_{\mathcal{F}}(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp \sum_{f \in \mathcal{F}} \theta_f f(\mathbf{x}) \tag{3.24}$$

$$Z(\boldsymbol{\theta}) = \sum_{\mathbf{x}'} \exp \sum_{f \in \mathcal{F}} \theta_f f(\mathbf{x}') \tag{3.25}$$

be a probability distribution defined by a feature set $\mathcal{F}$ and feature weights $\boldsymbol{\theta}$, and the negative *Kullback-Leibler divergence* (KL-divergence) between $p(\mathbf{x})$ and $p_{\mathcal{F}}(\mathbf{x}; \boldsymbol{\theta})$

$$\mathcal{Q}_{(\mathcal{F}, \boldsymbol{\theta})} = - \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{p_{\mathcal{F}}(\mathbf{x}; \boldsymbol{\theta})} \tag{3.26}$$

be our quality measure $\mathcal{Q}$. The goal is to define extension operations N^+ and reduction operations N_D^- such that the feature set grows slowly but the number of representable distributions grows quickly with an increasing number of iterations.

Let $\mathfrak{F}$ be the set of all possible binary features of $\boldsymbol{\mathcal{X}}$

$$\mathfrak{F} := \Big\{ f \,|\, f : \boldsymbol{\mathcal{X}} \to \{0,1\} \Big\} \tag{3.27}$$

consisting of all $2^{|\boldsymbol{\mathcal{X}}|}$ possible maps from $\boldsymbol{\mathcal{X}}$ to $\{0,1\}$. We can denote a single feature $f \in \mathfrak{F}$ by writing the corresponding logical expression in Iverson brackets $[\![\,\cdot\,]\!]$. I use features in both arithmetic and logical expressions where depending on the context their value has to be interpreted accordingly. As the distribution $p(\mathbf{x})$ has $|\boldsymbol{\mathcal{X}}| - 1$ degrees of freedom it is clear that we only need a small subset of all binary features to model any possible distribution.

3.3.1 Conjunctive Features

Consider the subset of conjunctive features $\mathfrak{F}_\wedge \subset \mathfrak{F}$

$$\mathfrak{F}_\wedge := \Big\{ f(\mathbf{x}) := \prod_{i=1}^{n} a_i [\![x_i = x_i']\!] \,\Big|\, \mathbf{a} \in \{0,1\}^n, \mathbf{x}' \in \boldsymbol{\mathcal{X}} \Big\} \tag{3.28}$$

that contains all $(|\mathcal{X}_1|+1)\cdots(|\mathcal{X}_n|+1)$ possible conjunctions that indicate the occurrence of specific assignments for all variables and all possible subsets of variables. Consider the extension operation

$$N_\wedge^+(\mathcal{F}) := \Big\{ f \cdot [\![x_i = x_i']\!] \,\Big|\, f \in \mathcal{F}, i \in \{1,\ldots,n\}, x_i' \in \mathcal{X}_i \Big\} \cup \mathcal{F} \cup \Big\{ [\![\texttt{true}]\!] \Big\} \tag{3.29}$$

that constructs all possible k-order conjunctions within k iterations

$$\mathcal{F}^{(0)} = \{[\![\texttt{true}]\!]\} \tag{3.30}$$
$$\mathcal{F}^{(1)} = \{[\![x_1 = 0]\!], [\![x_1 = 1]\!], \ldots, [\![x_2 = 0]\!], [\![x_2 = 1]\!] \ldots\} \cup \mathcal{F}^{(0)} \tag{3.31}$$
$$\mathcal{F}^{(2)} = \{[\![x_1 = 0 \wedge x_2 = 0]\!], [\![x_1 = 1 \wedge x_2 = 0]\!], \ldots\} \cup \mathcal{F}^{(1)} \tag{3.32}$$
$$\mathcal{F}^{(3)} = \{[\![x_1 = 0 \wedge x_2 = 0 \wedge x_3 = 0]\!], [\![x_1 = 1 \wedge x_2 = 0 \wedge x_2 = 0]\!], \ldots\} \cup \mathcal{F}^{(2)} \tag{3.33}$$
$$\vdots$$

so that $\mathcal{F}^{(n)} = \mathfrak{F}_\wedge$. As $p(\mathbf{x})$ only has $|\boldsymbol{\mathcal{X}}| - 1$ degrees of freedom we can modify N^+ to include fewer features or use N_D^- to remove some.

Default Conjunctions

We can exclude one value for each variable from the conjunctions as well as the constant feature $[\![\texttt{true}]\!]$ (which is eliminated by the normalization anyway). I will call the excluded values of the variables their *default* value $\bar{\mathbf{x}} = (\bar{x}_1 \ldots \bar{x}_n)$. A feature is then identified by a vector $\mathbf{x}' \in \boldsymbol{\mathcal{X}}$ where a default entry in $\mathbf{x}'$ indicates that the corresponding variable is ignored and all other entries specify the values tested by the conjunction. The feature set of non-default conjunctions

$$\bar{\mathfrak{F}}_\wedge := \left\{ f(\mathbf{x}) := \prod_{i=1}^{n} [\![x_i = x'_i \vee x'_i = \bar{x}_i]\!] \,\middle|\, \mathbf{x}' \in \boldsymbol{\mathcal{X}}, \mathbf{x}' \neq \bar{\mathbf{x}} \right\} \tag{3.34}$$

has a size of $|\bar{\mathfrak{F}}_\wedge| = |\mathcal{X}_1| \cdots |\mathcal{X}_n| - 1$, which matches exactly the degrees of freedom of $p(\mathbf{x})$. To simplify notation let $|\mathbf{x}'|$ be the number of non-default entries in $\mathbf{x}'$ and $\mathbf{x}' \Rightarrow \mathbf{x}''$ denote that $\mathbf{x}'$ implies $\mathbf{x}''$ in the sense that some of the non-default entries in $\mathbf{x}'$ may be default in $\mathbf{x}''$ but not the other way around. A conjunctive feature is then defined as

$$f_{\mathbf{x}'}(\mathbf{x}) = \forall_{i=1}^{n} [\![x_i = x'_i \vee x'_i = \bar{x}_i]\!] \tag{3.35}$$

$$= [\![\mathbf{x} \Rightarrow \mathbf{x}']\!] \ . \tag{3.36}$$

Let $\mathbf{x}'|_{x_i = x''}$ denote vector $\mathbf{x}'$ with x_i set to x'' and consider the N^+ operation

$$\bar{N}^+_\wedge(\mathcal{F}) := \begin{cases} \left\{ f_{\bar{\mathbf{x}}|_{x_i = x''_i}} \,\middle|\, i \in \{1, \ldots, n\}, x''_i \in \mathcal{X}_i, x''_i \neq \bar{x}_i \right\} & \text{if } \mathcal{F} = \emptyset \\ \left\{ f_{\mathbf{x}'|_{x_i = x''_i}} \,\middle|\, f_{\mathbf{x}'} \in \mathcal{F}, i \in \{1, \ldots, n\}, x''_i \in \mathcal{X}_i, x''_i \neq \bar{x}_i \right\} \cup \mathcal{F} & \text{else} \end{cases} \tag{3.37}$$

that constructs $\bar{\mathfrak{F}}_\wedge$ after n iterations. We can write down the weights for these features explicitly for a given distribution. As each feature $f_{\mathbf{x}'}$ is defined by a specific assignment $\mathbf{x}'$, intuitively, it would make sense if the weight $\theta_{\mathbf{x}'}$ was related to the log-probability $\log p_{\mathbf{x}'}$ of these values to occur. However, as default values are ignored in the conjunctions the $\cdot \Rightarrow \cdot$ relation is transitive and a feature $f_{\mathbf{x}'}$ always occurs together with all other features $f_{\mathbf{x}''}$ for which $\mathbf{x}' \Rightarrow \mathbf{x}''$ holds. This suggest an iterative construction of the feature weights where we start with low-order conjunctions and subtract their weight in higher-order conjunctions to compensate for their contribution. The feature weights then are

$$\theta_{\bar{\mathbf{x}}} = \log p_{\bar{\mathbf{x}}} = -\log Z(\boldsymbol{\theta}) \tag{3.38}$$

$$\theta_{\mathbf{x}'} = \log p_{\mathbf{x}'} - \sum_{\mathbf{x}'' \in \boldsymbol{\mathcal{X}}} [\![\mathbf{x}' \Rightarrow \mathbf{x}'' \wedge \mathbf{x}'' \neq \mathbf{x}']\!] \theta_{\mathbf{x}''} \tag{3.39}$$

$$\begin{tikzcd}
 & \theta_{\circ\circ\circ} & \\
\theta_{\circ\circ\bullet} \arrow[ur] & \theta_{\circ\bullet\circ} \arrow[u] & \theta_{\bullet\circ\circ} \arrow[ul] \\
\theta_{\circ\bullet\bullet} \arrow[u] \arrow[ur] & \theta_{\bullet\circ\bullet} \arrow[ul] \arrow[ur] & \theta_{\bullet\bullet\circ} \arrow[ul] \arrow[u] \\
 & \theta_{\bullet\bullet\bullet} \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Figure 3.6: Transitive reduction of the $\cdot \Rightarrow \cdot$ relation for the case of three binary variables. Feature weights are computed by summing along the paths to $\theta_{\circ\circ\circ}$ with alternating sign.

$$= \sum_{\mathbf{x}''\in\boldsymbol{\mathcal{X}}} [\![\mathbf{x}' \Rightarrow \mathbf{x}'']\!] (-1)^{|\mathbf{x}'|-|\mathbf{x}''|} \log p_{\mathbf{x}''} \ , \tag{3.40}$$

where the pseudo-weight $\theta_{\bar{\mathbf{x}}}$ is only needed as base case for the recursion. It corresponds to the constant feature $[\![\text{true}]\!]$ that was excluded and represents the degree of freedom eliminated by the normalization, which becomes obvious when computing $p_{\bar{\mathbf{x}}}$ via Eq. (3.24) because all conjunctions are false for that case. For the transfer from the recursive (3.39) to the closed-form equation (3.40) note that lower-order feature weights occur in higher-order weights directly via the sum in Eq. (3.39) but also indirectly via the recursion where they switch sign with each recursive step. As a consequence all occurrences of the log-probabilities cancel out except for the first one, for which the sign depends on the level of recursion where it occurs. The sum in Eq. (3.40) runs over all possible variable assignments, however, non-zero contributions only enter from the $2^{|\mathbf{x}'|}$ assignments where $\mathbf{x}' \Rightarrow \mathbf{x}''$ holds, that is, where non-default entries in $\mathbf{x}'$ are changed to the default value. The recursive structure is determined by the $\cdot \Rightarrow \cdot$ relation on the n-dimensional hypercube, which is illustrated in Figure 3.6 for the case of three binary variables.

Connection to Information Geometry

The representation of $p(\mathbf{x})$ via conjunctive features has a correspondence in information geometry (Amari, 2001a,b; Toussaint, 2004) where the feature weights are called θ-coordinates. For an illustrative direct comparison I list the feature weights for the case of three binary variables, where I assume the default values to be zero $\bar{\mathbf{x}} = (0 \dots 0) = (\circ \dots \circ)$.[10] The weights can be computed by following the paths of the $\cdot \Rightarrow \cdot$ relation depicted in Figure 3.6. To illustrate the canceling mechanism I explicitly perform the

[10]In the following I will interchangeably use the notations $\bullet \equiv 1 \equiv$ `true` and $\circ \equiv 0 \equiv$ `false` for binary numbers as well as $\blacksquare \equiv 1$, $\square \equiv -1$, and $\cdot \equiv 0$ for ternary numbers.

recursive computation and highlight terms according to where they originate from

$$\theta_{\circ\circ\circ} = \log p_{000} \tag{3.41}$$

$$\theta_{\bullet\circ\circ} = \log \frac{p_{100}}{p_{000}} \qquad \theta_{\circ\bullet\circ} = \log \frac{p_{010}}{p_{000}} \qquad \theta_{\circ\circ\bullet} = \log \frac{p_{001}}{p_{000}} \tag{3.42}$$

$$\theta_{\circ\bullet\bullet} = \log \frac{p_{011}}{p_{000}}\frac{p_{000}}{p_{010}}\frac{p_{000}}{p_{001}} \qquad \theta_{\bullet\circ\bullet} = \log \frac{p_{101}}{p_{000}}\frac{p_{000}}{p_{100}}\frac{p_{000}}{p_{001}} \qquad \theta_{\bullet\bullet\circ} = \log \frac{p_{110}}{p_{000}}\frac{p_{000}}{p_{100}}\frac{p_{000}}{p_{010}} \tag{3.43}$$

$$= \log \frac{p_{011}p_{000}}{p_{010}p_{001}} \qquad = \log \frac{p_{101}p_{000}}{p_{100}p_{001}} \qquad = \log \frac{p_{110}p_{000}}{p_{100}p_{010}} \tag{3.44}$$

$$\theta_{\bullet\bullet\bullet} = \log \frac{p_{111}}{p_{000}}\frac{p_{000}}{p_{100}}\frac{p_{000}}{p_{010}}\frac{p_{000}}{p_{001}}\frac{p_{000}}{p_{011}}\frac{p_{010}}{p_{000}}\frac{p_{001}}{p_{000}}\frac{p_{000}}{p_{101}}\frac{p_{100}}{p_{000}}\frac{p_{001}}{p_{000}}\frac{p_{000}}{p_{110}}\frac{p_{100}}{p_{000}}\frac{p_{010}}{p_{000}} \tag{3.45}$$

$$= \log \frac{p_{111}p_{100}p_{010}p_{001}}{p_{000}p_{011}p_{101}p_{110}} \,. \tag{3.46}$$

This reproduces equations (32–33) in (Amari, 2001a). In information geometry the θ-coordinates are complemented by the η-coordinates. For both the relation between the probability distribution $p_{\mathbf{x}}$ and the coordinates is governed by a $(|\boldsymbol{\mathcal{X}}|-1)\times(|\boldsymbol{\mathcal{X}}|-1)$-matrix $\mathbf{B}$ as

$$\log \frac{p_{\mathbf{x}}}{p_{\bar{\mathbf{x}}}} = \sum_{\substack{\mathbf{x}'\in\boldsymbol{\mathcal{X}}\\ \mathbf{x}'\neq\bar{\mathbf{x}}}} \Big(\mathbf{B}^{-1}\Big)_{\mathbf{x}\mathbf{x}'} \theta_{\mathbf{x}'} \tag{3.47}$$

$$p_{\mathbf{x}} = \sum_{\substack{\mathbf{x}'\in\boldsymbol{\mathcal{X}}\\ \mathbf{x}'\neq\bar{\mathbf{x}}}} \Big(\mathbf{B}\Big)_{\mathbf{x}'\mathbf{x}} \eta_{\mathbf{x}'} \tag{3.48}$$

$$\theta_{\mathbf{x}'} = \sum_{\substack{\mathbf{x}\in\boldsymbol{\mathcal{X}}\\ \mathbf{x}\neq\bar{\mathbf{x}}}} \Big(\mathbf{B}\Big)_{\mathbf{x}'\mathbf{x}} \log \frac{p_{\mathbf{x}}}{p_{\bar{\mathbf{x}}}} \tag{3.49}$$

$$\eta_{\mathbf{x}'} = \sum_{\substack{\mathbf{x}\in\boldsymbol{\mathcal{X}}\\ \mathbf{x}\neq\bar{\mathbf{x}}}} \Big(\mathbf{B}^{-1}\Big)_{\mathbf{x}\mathbf{x}'} p_{\mathbf{x}} \tag{3.50}$$

or, more compactly,

$$\log \frac{\mathbf{p}}{p_{\bar{\mathbf{x}}}} = \mathbf{B}^{-1}\boldsymbol{\theta} \tag{3.51}$$

$$\mathbf{p} = \mathbf{B}^{\top}\boldsymbol{\eta} \tag{3.52}$$

$$\boldsymbol{\theta} = \mathbf{B} \log \frac{\mathbf{p}}{p_{\bar{\mathbf{x}}}} \tag{3.53}$$

$$\boldsymbol{\eta} = \Big(\mathbf{B}^{\top}\Big)^{-1} \mathbf{p} \,. \tag{3.54}$$

The matrices $\mathbf{B}$ and $\mathbf{B}^{-1}$ can be written as the tensor product of per-variable matrices, $\mathbf{b}_i$ and $\mathbf{b}_i^{-1}$, and by establishing the connection to Eqs. (3.40) and (3.36) we gain explicit expressions for their entries

$$\Big(\mathbf{B}\Big)_{\mathbf{x}'\mathbf{x}} = \Big(\mathbf{b}_i \otimes \ldots \otimes \mathbf{b}_n\Big)_{\mathbf{x}'\mathbf{x}} \tag{3.55}$$

$$\Big(\mathbf{B}^{-1}\Big)_{\mathbf{x}\mathbf{x}'} = \Big(\mathbf{b}_i^{-1} \otimes \ldots \otimes \mathbf{b}_n^{-1}\Big)_{\mathbf{x}\mathbf{x}'} \tag{3.56}$$

$$= \prod_{i=1}^{n} (\mathbf{b}_i)_{x_i x'_i} \tag{3.57}$$

$$= \prod_{i=1}^{n} (\mathbf{b}_i^{-1})_{x_i x'_i} \tag{3.58}$$

$$= \Big[\!\Big[\mathbf{x}' \Rightarrow \mathbf{x}\Big]\!\Big] (-1)^{|\mathbf{x}'|-|\mathbf{x}|} \tag{3.59}$$

$$= \Big[\!\Big[\mathbf{x} \Rightarrow \mathbf{x}'\Big]\!\Big] \tag{3.60}$$

$$\Big(\mathbf{b}_i\Big)_{x'_i x_i} = \begin{cases} 1 & \text{if } x'_i = x_i \\ -1 & \text{else if } x_i = \bar{x}_i \\ 0 & \text{else} \end{cases} \quad (3.61) \qquad \Big(\mathbf{b}_i^{-1}\Big)_{x_i x'_i} = \begin{cases} 1 & \text{if } x_i = x'_i \lor x'_i = \bar{x}_i \\ 0 & \text{else} \end{cases} \quad (3.62)$$

$$= \begin{pmatrix} \blacksquare & \cdot & \cdot & \cdots \\ \square & \blacksquare & \cdot & \\ \square & \cdot & \blacksquare & \\ \vdots & & & \ddots \end{pmatrix}_{x'_i x_i} \quad (3.63) \qquad = \begin{pmatrix} \blacksquare & \cdot & \cdot & \cdots \\ \blacksquare & \blacksquare & \cdot & \\ \blacksquare & \cdot & \blacksquare & \\ \vdots & & & \ddots \end{pmatrix}_{x_i x'_i} , \quad (3.64)$$

where I assumed $\bar{\mathbf{x}} = (0 \ldots 0)$, otherwise the first column/row in $\mathbf{b}_i$ and $\mathbf{b}_i^{-1}$ have to be swapped with the corresponding location. In order to reproduce the $(|\boldsymbol{\mathcal{X}}|-1) \times (|\boldsymbol{\mathcal{X}}|-1)$-matrices $\mathbf{B}$ and $\mathbf{B}^{-1}$ exactly the row and column corresponding to $\bar{\mathbf{x}}$ have to be dropped after taking the tensor product. The matrices $\mathbf{B}$ and $\mathbf{B}^{-1}$ are hierarchically structured block matrices as illustrated in Figure 3.7 for the case of three variables with tree possible values.

The close connection of θ- and η-coordinates in information geometry suggests a similar connection on the feature level. From Eqs. (3.59) and (3.60) follows the definition of the corresponding features and weights as

$$g_{\mathbf{x}'}(\mathbf{x}) = \Big[\!\Big[\mathbf{x}' \Rightarrow \mathbf{x}\Big]\!\Big](-1)^{|\mathbf{x}'|-|\mathbf{x}|} \tag{3.65}$$

$$\eta_{\mathbf{x}'} = \sum_{\mathbf{x} \in \boldsymbol{\mathcal{X}}} \Big[\!\Big[\mathbf{x} \Rightarrow \mathbf{x}'\Big]\!\Big] p_{\mathbf{x}} \ . \tag{3.66}$$

For the weights $\eta_{\mathbf{x}'}$ this reestablishes the known fact that they correspond to the marginal probability of the non-default entries in $\mathbf{x}'$. To see this, note that $\mathbf{x} \Rightarrow \mathbf{x}'$ holds whenever $\mathbf{x}$ differs only in default entries so that the sum effectively runs over all possible assignments for these entries. The exclusion of $\mathbf{x} \neq \bar{\mathbf{x}}$ from the sum as in Eq. (3.50) is not necessary here as $\bar{\mathbf{x}} \Rightarrow \mathbf{x}'$ only holds for $\mathbf{x}' = \bar{\mathbf{x}}$, which corresponds to the constant feature that is excluded. The features $g_{\mathbf{x}'}$ are ternary features and can be constructed using the same $N_\land^+$ operation (3.37) as for binary conjunctive features. For the practical application two aspects have to be taken into account. First, the probability distribution with these ternary features is constructed as a direct linear combination so that when optimizing the weights $\eta_{\mathbf{x}'}$ we need additional positivity and normalization constraints. Second, one advantage of the binary conjunctive features as considered so far and defined in Eq. (3.36) is that for low-order conjunctions only the specified variables have to be inspected in order to compute their values. In contrast, for a ternary feature all variables may be relevant, which may become a practical issue as the number of variables grows large. In their definition in Eqs. (3.36) and (3.65) this is reflected by the opposite direction of the $\cdot \Rightarrow \cdot$

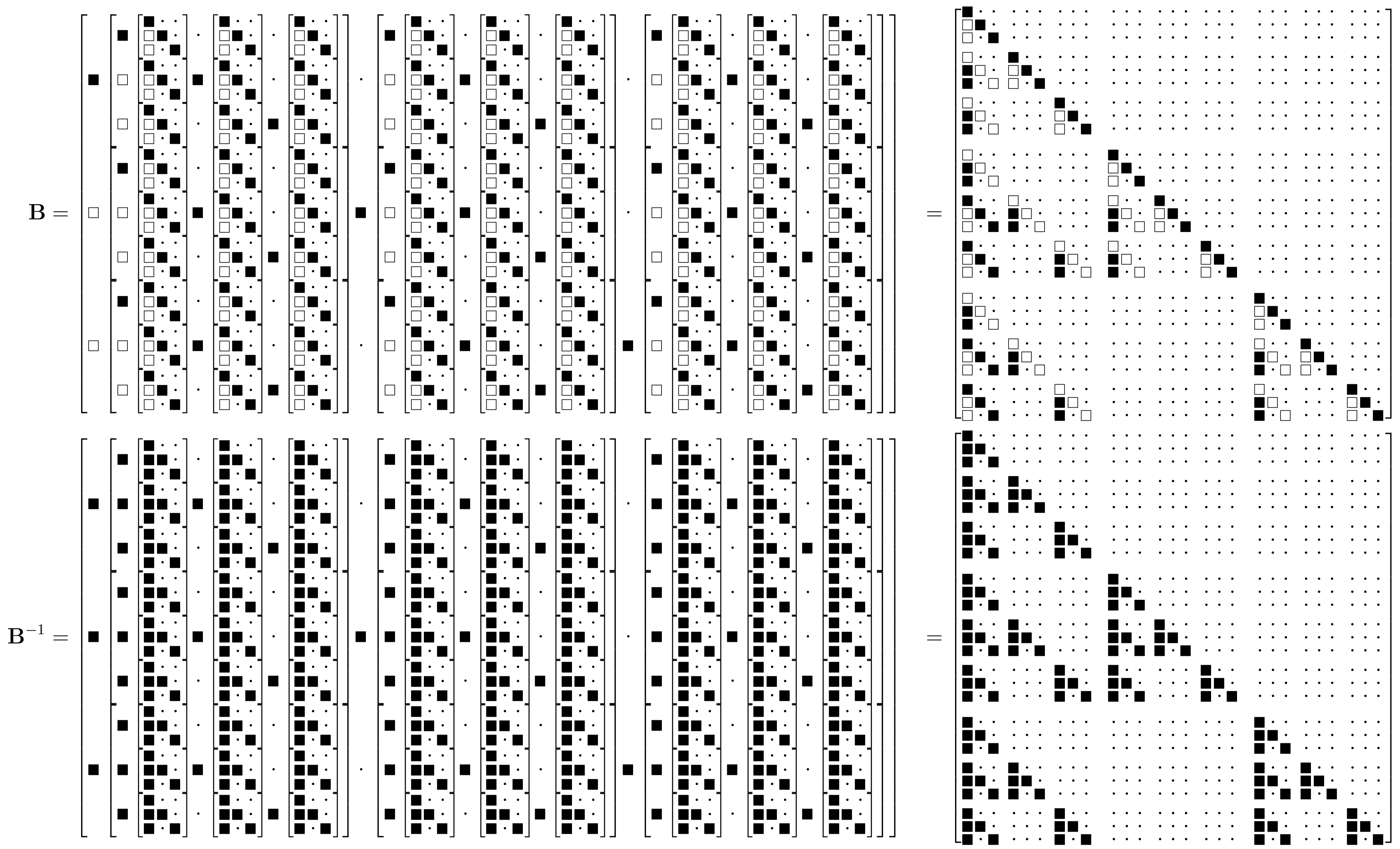


Figure 3.7: **B** and $\mathbf{B}^{-1}$ matrices for three variables that can take three values. The symbols are ■ $\equiv 1$, □ $\equiv -1$, $\cdot \equiv 0$.

relation. I did not further investigate the potential of ternary features but this might be an interesting subject for future research.

Complete Conjunctions

The set of complete conjunctive features

$$\widehat{\mathfrak{F}}_\wedge := \left\{ f(\mathbf{x}) := \prod_{i=1}^{n} [\![x_i = x'_i]\!] \,\middle|\, \mathbf{x}' \in \boldsymbol{\mathcal{X}} \right\} \tag{3.67}$$

has a size of $|\widehat{\mathfrak{F}}_\wedge| = |\mathcal{X}_1| \cdots |\mathcal{X}_n|$. In contrast to the set of default conjunctions $\bar{\mathfrak{F}}_\wedge$ (3.34) the complete conjunctive features in $\widehat{\mathfrak{F}}_\wedge$ do not ignore any variable's values. As complete conjunctions are mutually exclusive the corresponding weights are

$$\theta_{\mathbf{x}'} = \log p_{\mathbf{x}'} \ , \tag{3.68}$$

where one degree of freedom is eliminated by the normalization. There are several ways to construct $\widehat{\mathfrak{F}}_\wedge$ iteratively. We can fix an order of variables and in the k^{th} iteration incrementally add all conjunctions of the k^{th} variable x_k

$$\mathcal{F}^{(0)} = \emptyset \tag{3.69}$$

$$\mathcal{F}^{(k)} = \widehat{N}^+_{\wedge i}(\mathcal{F}^{(k-1)}) \tag{3.70}$$

$$\widehat{N}^+_{\wedge i}(\mathcal{F}^{(k-1)}) := \begin{cases} \left\{ [\![x_k = x'_k]\!] \,\middle|\, x'_k \in \mathcal{X}_k \right\} & \text{if } k = 1 \\ \left\{ f \cdot [\![x_k = x'_k]\!] \,\middle|\, f \in \mathcal{F}^{(k-1)}, x'_k \in \mathcal{X}_k \right\} & \text{else .} \end{cases} \tag{3.71}$$

$\mathcal{F}^{(k)}$ is an extension of $\mathcal{F}^{(k-1)}$ not because of being a super set (actually $\mathcal{F}^{(k)} \cap \mathcal{F}^{(k-1)} = \emptyset$ because $\widehat{N}^+_{\wedge i}$ replaces all existing features) but because $\mathcal{F}^{(k)}$ is strictly more expressive. The equivalent parameters are obtained by directly transferring the feature weights from an old feature in $\mathcal{F}^{(k-1)}$ to all of its extensions in $\mathcal{F}^{(k)}$, which are mutually exclusive and jointly cover all cases of the old feature. In some situations the problem structure may suggest a specific ordering of variables that can be used. This is, for instance, the case in time series models (see Section 3.3.3). If this is not the case, the best ordering can be determined by the N^-_D operation. To this end the N^+ operation has to add conjunctions for all missing variables

$$\widehat{N}^+_{\wedge a}(\mathcal{F}) := \begin{cases} \left\{ [\![x_i = x'_i]\!] \,\middle|\, x'_i \in \mathcal{X}_i \right\} & \text{if } \mathcal{F} = \emptyset \\ \left\{ f \cdot [\![x_i = x'_i]\!] \,\middle|\, f \in \mathcal{F}, x'_i \in \mathcal{X}_i, x_i \notin \mathcal{F} \right\} & \text{else ,} \end{cases} \tag{3.72}$$

$\mathfrak{F}$	N^+	N^-_D	$\lvert\mathcal{F}^{(1)}\rvert$	$\lvert N^+(\mathcal{F}^{(k)})\rvert$	$\lvert\mathcal{F}^{(k)}\rvert$	$\lvert\mathcal{F}^{(n)}\rvert = \lvert\mathfrak{F}\rvert$
$\mathfrak{F}_\wedge$ (3.28)	$N^+_\wedge$ (3.29)		$\lvert\mathcal{X}\rvert + 1$	$\lvert\mathcal{F}^{(k)}\rvert + \binom{n}{k+1}\lvert\mathcal{X}\rvert^{(k+1)}$		$(\lvert\mathcal{X}\rvert + 1)^n$
$\bar{\mathfrak{F}}_\wedge$ (3.34)	$\bar{N}^+_\wedge$ (3.37)		$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{F}^{(k)}\rvert + \binom{n}{k+1}(\lvert\mathcal{X}\rvert - 1)^{(k+1)}$		$\lvert\mathcal{X}\rvert^n - 1$
$\widehat{\mathfrak{F}}_\wedge$ (3.67)	$\widehat{N}^+_{\wedge i}$ (3.71)		$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{F}^{(k)}\rvert \cdot \lvert\mathcal{X}\rvert$	$\lvert\mathcal{X}\rvert^k$	$\lvert\mathcal{X}\rvert^n$
	$\widehat{N}^+_{\wedge a}$ (3.72)	$\widehat{N}^-_D$ (3.73)	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{F}^{(k)}\rvert \cdot (n-k)\lvert\mathcal{X}\rvert$	$\lvert\mathcal{X}\rvert^k$	$\lvert\mathcal{X}\rvert^n$

Table 3.4: Growth of the Feature Set

where $x_i \notin \mathcal{F}$ means that no feature in $\mathcal{F}$ depends on the variable x_i. To gain equivalent parameters we need to choose one of the newly added variables, transfer weights to the corresponding features as above, and set all other weights to zero. The N^-_D operation eliminates all features except for those containing the best newly added variable

$$\widehat{N}^-_D(\mathcal{F}) := \operatorname{argmax}_{\mathcal{F}_{x_i}} \left\{ \mathcal{Q}^*_{\mathcal{F}_{x_i}}(D) \,\middle|\, \mathcal{F}_{x_i} \subset \mathcal{F}, x_i \text{ is new} \right\} , \tag{3.73}$$

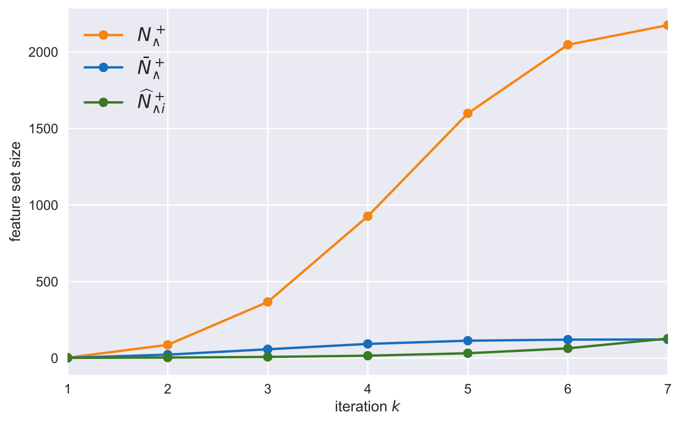
where $\mathcal{F}_{x_i} \subset \mathcal{F}$ is the subset of features that depends on variable x_i, and a variable is new if it is not contained in all features.

3.3.2 Complexity and Convergence

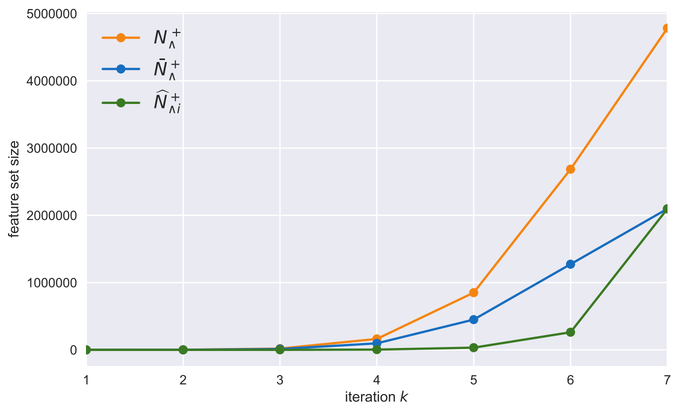
Properties of Different N^+ Operations

The correspondence between conjunctive features and θ-coordinates in information geometry provides a meaningful interpretation of the kind of distributions that can be modeled with a given feature set. In terms of information geometry, the set of k-fold conjunctive features of a set of variables captures the k^{th}-order interactions of these variables. This means that after k iterations both $N^+_\wedge$ (which generates the simple conjunctive features $\mathfrak{F}_\wedge$) and $\bar{N}^+_\wedge$ (which generates the default conjunctive features $\bar{\mathfrak{F}}_\wedge$) capture interactions up to the k^{th} order of *all* variables. Likewise, after k iterations $\widehat{N}^+_{\wedge i}$ and $\widehat{N}^+_{\wedge a}/\widehat{N}^-_D$ (which generate the complete conjunctive features $\widehat{\mathfrak{F}}_\wedge$ with predefined or data-based ordering, respectively) capture *all* interactions of the first k variables. In all cases, after n iterations all interactions between all variables are captured so that any distribution can be modeled.

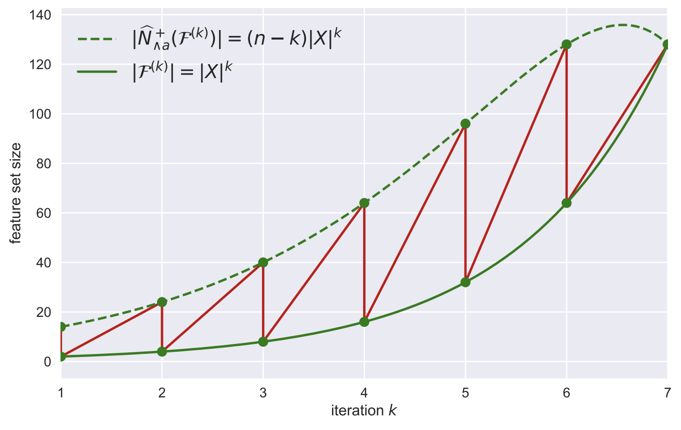
Depending on the use case the different N^+ operations have different advantages. In Table 3.4 the growth of the feature set for the different N^+ operations is listed and Figure 3.8 gives an intuition by plotting the number of features as a function of the iteration for the different N^+ operations. The decision diagram in Figure 3.9 illustrates



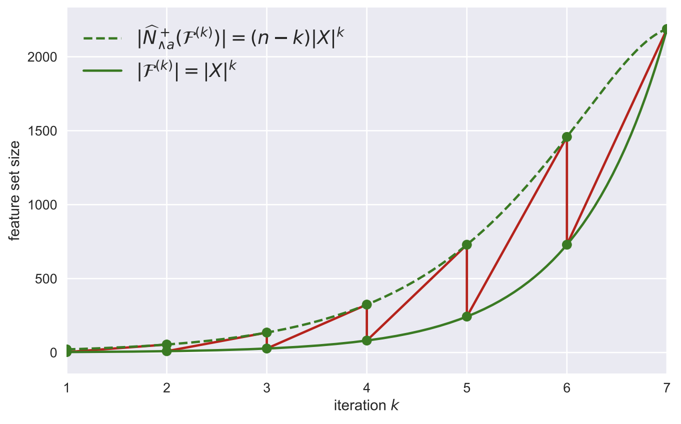
(a) $n = 7,\ |\mathcal{X}| = 2$



(b) $n = 7,\ |\mathcal{X}| = 8$



(c) $n = 7,\ |\mathcal{X}| = 2$



(d) $n = 7,\ |\mathcal{X}| = 3$

Figure 3.8: Size of the feature set for the different N^+ operations.

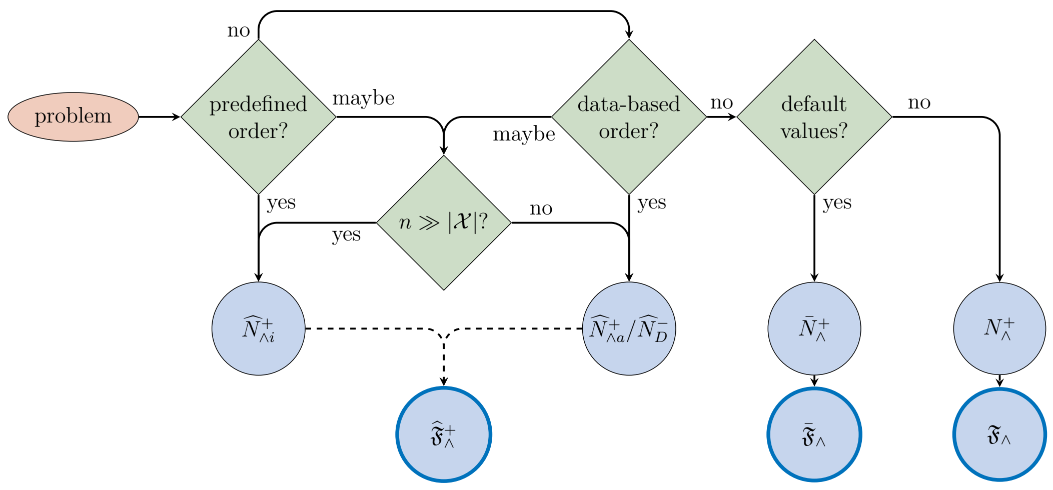


Figure 3.9: Decision diagram for choosing an N^+ operation.

the line of reasoning for choosing an appropriate N^+ operation, which is also described in the following.

First, it is a major advantage if the variables have a predefined order of relevance. In that case we can use the $\widehat{N}^+_{\wedge i}$ operation, which ensures that in the k^{th} iteration all interactions of the k most relevant variables are captured while providing exponential growth (as opposed to combinatorial growth). If the ordering is not predefined but can in each iteration be reliably inferred from the data based on the model quality, we can resort to the combination of $\widehat{N}^+_{\wedge a}$ and $\widehat{N}^-_D$ operation, which produces feature sets of the same size but requires an overhead of $(n-k-1)|\mathcal{X}|$ features in the k^{th} iteration for selecting the next variable. If the number of variables n is considerably greater than their cardinality $|\mathcal{X}|$, this overhead is more pronounced as compared to the overall growth. This may be an additional criterion in situations where it is not clear whether the order of variables should be fixed beforehand or inferred from the data.

If an ordering of variables cannot be used, the $\bar{N}^+_\wedge$ and $N^+_\wedge$ operations provide the guarantee that in the k^{th} iteration the k^{th}-order interactions of all variables are modeled at the cost of a combinatorial growth of the feature set. If possible, defining default values for the variables that are ignored in the conjunctions (but still modeled implicitly) considerably reduces the size of the feature set. This effect is less pronounced if the variables have a large cardinality $|\mathcal{X}|$.

Worst-Case Bounds

We may hope that even if our data distribution has higher-order interactions, a k^{th}-order approximation will be sufficient for our purposes. In the worst case, however, none of the lower-order approximations provides any advantage over the uniform distribution. This can be illustrated by the following distribution of the variables $x_0, \ldots, x_n \in \{-1, 1\}$

$$p(x_0, \ldots, x_n) = p(x_0 \,|\, x_1, \ldots, x_n) p(x_1, \ldots, x_n) \tag{3.74}$$

$$\text{with} \quad p(x_1, \ldots, x_n) = 2^{-n} \tag{3.75}$$

$$\text{and} \quad p(x_0 \,|\, x_1, \ldots, x_n) = \left[\!\left[x_0 = \textstyle\prod_{i=1}^n x_i \right]\!\right] . \tag{3.76}$$

The variables $x_1, \ldots, x_n$ occur with a uniform probability and marginalizing out any of them results in a uniform distribution for x_0, too. Only the highest-order interaction of all $n+1$ variables captures the fact that x_0 is completely determined by $x_1, \ldots, x_n$. Similar examples can be constructed where lower-order conjunctions need to be inspected while higher-order interactions do not provide any improvement (see, e.g., the discussion of k-cut mixed coordinates in information geometry, Amari, 2001b). In the worst case,

$p(x_1, x_2)$	$x_2 = 1$	$x_2 = 2$	$x_2 = 3$	$x_2 = 4$	$p(x_1)$
$x_1 = 1$	0.12	0.12	0.08	0.08	0.4
$x_1 = 2$	0.12	0.12	0.08	0.08	0.4
$x_1 = 3$	0.03	0.03	0.02	0.02	0.1
$x_1 = 4$	0.03	0.03	0.02	0.02	0.1
$p(x_2)$	0.3	0.3	0.2	0.2	

(a) Independent Subsets of Variables

$p(x_1, x_2)$	$x_2 = 1$	$x_2 = 2$	$x_2 = 3$	$x_2 = 4$	$x_2 = 5$	$x_2 = 6$	$x_2 = 7$	$x_2 = 8$	$x_2 = 9$
$x_1 = 1$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 2$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 3$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 4$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 5$	0.01	0.01	0.1	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 6$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 7$	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.02
$x_1 = 8$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$x_1 = 9$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

(b) Events

Figure 3.10: Special Cases

we therefore have to inspect all $|\mathcal{X}|^n - 1$ conjunctive features.

Eliminating Features

To keep the size of the feature set tractable, in practice, we strive for eliminating features that are not needed for (approximately) modeling the data distribution. This is for instance possible if the variables $\mathbf{x}$ consist of independent subsets $\mathbf{x}'$ and $\mathbf{x}''$ that may be modeled separately. This results in $(|\mathcal{X}|^{|\mathbf{x}'|} - 1) + (|\mathcal{X}|^{|\mathbf{x}''|} - 1)$ instead of $|\mathcal{X}|^{|\mathbf{x}|} - 1$ required features. Such an example is shown in Figure 3.10a, which can be modeled using 6 instead of 17 features since pairwise conjunctive features have a weight of zero (cf. Eq. (3.40)). One might expect 3 features to suffice, which is grounded on an intuitive grouping into pairs of values, that is, *disjunctions* of values. This highlights the fact that an inclusion of problem specific prior knowledge into the choice of variables and/or features – range-based feature like $[\![x_1 < 3]\!]$ in this case – may be highly beneficial. The trivial disjunction over all possible values of a variable corresponds to ignoring that variable in a conjunctive feature. Therefore, *event-like* distributions where specific variable assignments occur with particularly high probability can be modeled compactly. Figure 3.10b shows an example distribution that can be modeled using three conjunctive features with weights

$$\theta_{[x_1 = 7]} = \log \frac{0.02}{0.01} \tag{3.77}$$

$$\theta_{[x_1 = 7 \wedge x_2 = 8]} = \log \frac{0.03}{0.02} \tag{3.78}$$

$$\theta_{[x_1 = 5 \wedge x_2 = 3]} = \log \frac{0.1}{0.01} \ . \tag{3.79}$$

Note that the features have to indicate the respective variable assignments explicitly, which is a strong argument for *not* using fixed default values and using the more general $N_\wedge^+$ operation (3.29). The growing rate of the feature set, which is worse *theoretically*, results in a greater freedom for eliminating features and may therefore lead to a better empirical performance.

N_D^- with L_1-Regularization

A common way to implement a trade-off between accurately modeling the data distribution and keeping the model size tractable consists in introducing an L_1-regularization of the feature weights. When optimizing the weights for a fixed set of features, varying the strength of L_1-regularization has a continuous effect on the feature weights and the model quality. Over multiple iterations of feature expansion, however, a slight change in regularization strength may result in excluding or including a feature and may have an arbitrary large non-continuously changing impact on the final model quality by interrupting or establishing a relevant chain of feature expansions.

It is useful to distinguish the case of an infinitesimally small ϵ-regularization and that of a finite regularization strength. An ϵ-regularization excludes features only if they can be assigned zero weight without impairing the model quality. This allows eliminating any overhead of adding an overcomplete set of features (e.g. by using $N_\wedge^+$ instead of $\bar{N}_\wedge^+$). An ϵ-regularization may still interrupt a relevant expansion chain if a feature only contributes indirectly by being expanded in later iterations. However, even for pure k^{th}-order interactions the lower-order features will generally have non-zero weight as long the relevant k^{th}-order feature is not discovered, which will in most cases preserve the relevant expansion chain in the presence of an ϵ-regularization. These non-zero weights occur because projecting down a pure k^{th}-order distribution $p(\mathbf{x})$ to its $(l)^{\text{th}}$-order reduction ($l < k$) – in terms of information geometry this is the information projection or m-projection of $p(\mathbf{x})$ to the $E_l(0)$ manifold – the lower-order θ-coordinates change their value to compensate for the missing higher orders.

For the case of finite regularization strengths it is important to maintain a level of *non-disruptive sparseness* (Condition 4) that retains expansion chains to the relevant higher-order conjunctions. If this level is on the edge of practical feasibility or beyond, one can approach it by decreasing the regularization strength in steps such that only one feature at a time enters the set and each new feature bears the potential to trigger a chain of expansions. The required step size can be approximated by assuming the objective to be locally quadratic and applying techniques like *least angle regression* (Efron et al., 2004; Hastie, Tibshirani, and Friedman, 2008).

Independently of whether an ϵ-regularization or a finite regularization strength is used, features that were excluded in one iteration may become relevant in later iterations and should not be discarded once and for all. The definition of $N^+_\wedge$ (3.29) already meets this requirement and the other N^+ operations can be adapted accordingly.

3.3.3 Temporally Extended Features

In Section 3.1 I argued for the importance of features being able to capture temporally extended properties of the environment, which is why I am mainly concerned with learning non-Markov transition models. *Temporally extended features* (TEFs) are conjunctive features for non-Markov time series prediction. Let $\mathbf{y} \in \mathcal{Y}$ be a random variable that is to be predicted conditionally on an infinite set of variables $\mathbf{x} = (x_1, x_2, \ldots) \in \mathcal{X}^*$

$$p(\mathbf{y} \,|\, x_1, x_2, \ldots) : \mathcal{X}^* \rightsquigarrow \mathcal{Y} \; . \tag{3.80}$$

All being said for the unconditional case transfers to the case of conditional prediction with only minor changes. The conditional distribution is modeled as a *conditional random field* (J. D. Lafferty, Andrew McCallum, and Pereira, 2001)

$$p_{\mathcal{F}}(\mathbf{y} \,|\, \mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z(\mathbf{x}; \boldsymbol{\theta})} \exp \sum_{f \in \mathcal{F}} \theta_f \, f(\mathbf{x}, \mathbf{y}) \tag{3.81}$$

$$Z(\mathbf{x}; \boldsymbol{\theta}) = \sum_{\mathbf{y}'} \exp \sum_{f \in \mathcal{F}} \theta_f \, f(\mathbf{x}, \mathbf{y}') \; , \tag{3.82}$$

which differs from Eq. (3.24) in that the normalization runs only over $\mathbf{y}$. For specific data D the model quality is estimated by the data log-likelihood, which can be understood as a Monte-Carlo approximation of the negative KL-divergence (3.26) based on the data

$$\mathcal{Q}_{(\mathcal{F}, \boldsymbol{\theta})} = -\sum_{\mathbf{y}, \mathbf{x}} p(\mathbf{y}, \mathbf{x}) \, \log \frac{p(\mathbf{y} \,|\, \mathbf{x}) p(\mathbf{x})}{p_{\mathcal{F}}(\mathbf{y} \,|\, \mathbf{x}; \boldsymbol{\theta}) p(\mathbf{x})} \tag{3.83}$$

$$\approx \frac{1}{N} \sum_{n=1}^{N} \log p_{\mathcal{F}}(\mathbf{y}^{(n)} \,|\, \mathbf{x}^{(n)}; \boldsymbol{\theta}) + \mathit{const} \; . \tag{3.84}$$

The optimal parameters $\boldsymbol{\theta}^*$ can be found via gradient descent on the *negative logarithm of the likelihood* (neg-log-likelihood) of the data. The marginal distribution $p(\mathbf{x})$ does not need to be modeled and features that are independent of $\mathbf{y}$ are eliminated by the normalization and should not be generated by the N^+ operation in the first place. This reduced feature set corresponds to the increased number of normalization constraints in the conditional distribution. A major difference to the unconditional case is that we are now dealing with an infinite number of variables. This issue can be addressed in multiple ways.

Fixed Horizon: We may exclude all variables except for the most recent k events $x_1, \dots, x_k$ and apply $N^+_\wedge$ or $\bar{N}^+_\wedge$. This results in a *k-Markov decision process* (k-MDP) model for which in the l^{th} iteration all interactions of the k variables up to l^{th}-order are included.

Gradual Extension: The temporal structure suggest an inherent ordering of variables, which may be used to apply $\widehat{N}^+_{\wedge i}$. This produces in the k^{th} iteration a k-MDP model that includes all interactions of the first k variables of arbitrary order, that is, up to the k^{th} order.

Per-Feature Horizon: A more restrictive alternative to gradually extending the horizon for all features is to maintain a per-feature horizon and expand any k-fold conjunction with the $(k+1)^{\text{th}}$ variable. This also allows expansion to proceed at a later stage but results in a considerably smaller number of extensions as only interactions of contiguous sets of variables are considered. The corresponding N^+ operation is defined as

$$\widehat{N}^+_{\wedge b}(\mathcal{F}) := \begin{cases} \left\{ [\![y = y']\!] \,\middle|\, y' \in \mathcal{Y} \right\} & \text{if } \mathcal{F} = \emptyset \\ \left\{ f \cdot [\![x_{|f|} = x'_{|f|}]\!] \,\middle|\, f \in \mathcal{F}, x'_{|f|} \in \mathcal{X}_{|f|}, \right\} & \text{else }, \end{cases} \tag{3.85}$$

where $|f|$ is the number of variables in f so that $x_{|f|}$ is the first variable not contained in f.

Online Learning with Temporally Extended Features

Online learning poses a particular challenge because the data set is changing continuously and new data is highly correlated to previous data as the conditional variables simply shift by one time index. Usually, we cannot afford to run multiple iterations of feature expansion in every time step. A simple but very effective modification to the $\widehat{N}^+_{\wedge b}$ operation with *per-feature horizon* that we suggested in (Langhabel et al., 2017) is to carry over features from the last iteration by performing *forward* expansion instead of *backward* expansion. The basic idea is to shift existing features by one time index such that they capture the same patterns in the new data point. This provides a good basis for constructing new expansions for predicting the next data point. The corresponding N^+ operation is defined as

$$\widehat{N}^+_{\wedge f}(\mathcal{F}) := \Big\{ [\![y = y']\!] \,\Big|\, y' \in \mathcal{Y} \Big\} \cup \Big\{ [\![y = y']\!] \prod_{(x_i, x') \in f} [\![x_{i+1} = x']\!] \,\Big|\, f \in \mathcal{F} \Big\} \,, \tag{3.86}$$

where $(x_i, x') \in f$ means that $[\![x_i = x']\!]$ is a factor of f, and $y \equiv x_0$. The $\widehat{N}^+_{\wedge f}$ operation adds shifted copies of the existing features to the feature set. The time index of the existing variables in a feature are increased by one and, thereby, test for the same value at the location where they appear in the new data point. The variable y that was predicted in the last iteration now corresponds to the first conditional variable x_1. Therefore, if a feature was `true` in the previous iteration for the actually observed value of y the shifted conditions are `true` for the new data point. Additionally, prepending a test for the current variable y allows learning predictions for the newly added data point by adjusting feature weights accordingly.

Alternative Training Methods Tailored to Planning

I briefly discuss two options for learning TEFs that are specifically tailored to their application for active planning but that are not further pursued in this work.

Drop-Out to Enforce Temporal Extent As discussed in Section 3.1.1 a temporally extended non-Markov representation is important for adaptable abstractions even in Markov environments. However, when learning TEFs in a Markov environment they will not necessarily be temporally extended. We might therefore need to take additional measures to enforce the representation of temporally extended properties. In other words, our conventional objective of predicting the next event based on the entire history does not fully capture the requirements for learning an adaptable abstraction. One approach to enforce the representation of temporally extended non-Markov properties is to apply memory drop-out during training. That is, instead of providing the complete history some entries are delete so that features relying on these entries cannot be used for predicting the next event. This can be understood as simulating partial observability caused by unreliable memory or sensors. As a consequence the model has to compensate for the drop-outs by making predictions based on auxiliary information. A model capable of providing useful predictions under these conditions allows for more flexible abstractions during planning as more aspects of a plan may be left unspecified while still retaining approximate predictions that can be used to guide refinement of the plan.

Context-Specific Sparseness Another important property of an adaptable abstraction, also discussed in Section 3.1.1, is that of being context-specific, that is, the ability to adapt to the characteristics of a specific situation. For efficient planning in feature space it is beneficial if only a small number of features is needed to achieve this adaptation in any specific situation. This is partly accounted for by the L_1-regularization, however, this only aims for an overall small feature set not for a small number of active features in a given situation. It might therefore be beneficial to use an objective with an additional regularization term that penalizes the number of jointly active features, for instance,

$$\mathcal{O}(D; \mathcal{F}, \boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \Big[\underbrace{\lambda_{\bar{L}_1} \sum_{f \in \mathcal{F}} |\theta_f \, f(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})|}_{\bar{L}_1} - \underbrace{\log p_{\mathcal{F}}(\mathbf{y}^{(n)} \,|\, \mathbf{x}^{(n)}; \boldsymbol{\theta})}_{-\mathcal{Q}_{(\mathcal{F},\boldsymbol{\theta})}} \Big] + \underbrace{\lambda_{L_1} \sum_{f \in \mathcal{F}} |\theta_f|}_{L_1} \ , \tag{3.87}$$

where $-\mathcal{Q}_{(\mathcal{F},\boldsymbol{\theta})}$ corresponds to the data neg-log-likelihood, L_1 is a standard L_1-regularization term, and $\bar{L}_1$ penalizes the average number of jointly active features.

3.4 Temporally Extended Features in Model-Based Reinforcement Learning

In this section I will present an experimental evaluation of TEFs in model-based RL.[11] The aim of these experiments was to assess the capabilities of TEFs and the *PULSE* framework in several respects. First, TEFs were developed with the goal in mind to learn adaptable abstractions. As discussed in Section 3.1.1, useful abstractions will generally break the Markov property. It is, therefore, important to evaluate whether TEFs are capable of modeling non-Markov properties of an environment. At the same time, it is important to demonstrate that the promising theoretical properties of the *PULSE* framework transfer to the practical application of discovering TEFs. Another important aim was to verify whether the advantageous properties of a factorized decomposition, as discussed in Section 3.1.2, can be observed in a practical application. Finally, it is interesting to apply a method outside the realm it was designed for. In this case we used TEFs in a model-free setting for a linear approximation of the value function, which produced mixed results.

[11]The content of this section corresponds in large parts to (Lieck and Toussaint, 2016).

3.4.1 Setup

Choosing an Environment

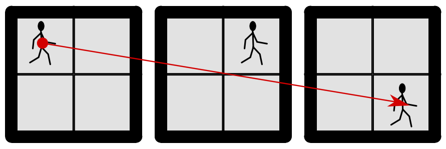
The main criterion for choosing an environment was that it has to exhibit non-Markov properties that have to be captured by the model. Additionally, we decided to employ deterministic environments to simplify the qualitative analysis of the learned models. In a deterministic environment the data are perfectly explainable, which circumvents the problem of choosing an appropriate bias-variance trade-off to reduce overfitting and, thereby, reduces variance in the results due to hyper-parameter tuning. We performed experiments in the four different maze environments shown in Figure 3.11, two mazes with delayed causalities and two classical POMDP mazes. All mazes have deterministic transitions based on the actions $\leftarrow$(`left`), $\rightarrow$(`right`), $\uparrow$(`up`), $\downarrow$(`down`), $\circ$(`stay`).

In the mazes with delayed causalities, shown in Figure 3.11a and 3.11b, the rewards and some transitions (in the 4×4 maze) depend on the location or the action a fixed number of time steps ago. Rewards are *activated* at a certain location and need to be *collected* at a different location a specific number of time steps later. Likewise, in the 4×4 maze transitions may be unblocked by opening a door. This is done by stepping into the wall at the location of the switch whereupon the door opens and remains open for another two successive time steps. The environments could alternatively be described as a k-MDP with $k = 2$ and $k = 3$, respectively, or as a POMDP with additional latent state factors for representing the various delays (also see Figure 3.11a and 3.11b).

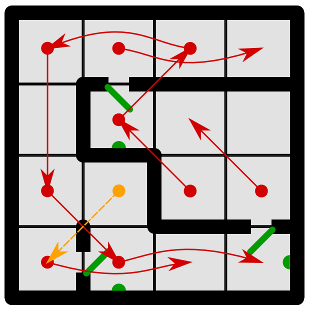
In the *Cheese Mazes* (A. K. McCallum, 1996; Shibuya and Hamagami, 2011), shown in Figure 3.11c and 3.11d, the agent only observes adjacent walls. It receives a reward of $+1$ for reaching the goal (indicated by the cheese), a reward of -1 for bumping into a wall, and a reward of -0.1 otherwise. Upon reaching the goal the agent is immediately relocated to a random location (small Cheese Maze) or the start location indicated by the mouse (large Cheese Maze). The Cheese Mazes cannot be described as a k-MDP because the agent may stay infinitely long in one of the indistinguishable locations. However, if it follows a (near) optimal policy the next action can be chosen optimally based on a history of length $k = 2$.

Choosing a Method for Comparison

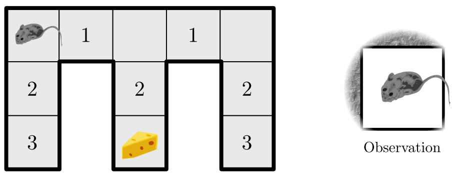
The main goal was a qualitative analysis of a model using TEFs in conjunction with conditional random fields as described in Section 3.3.3, which I will refer to as TEF+CRF. As a method for comparison we had to choose a method with as many commonalities as possible, most notably it should be feature-based and history-based. State-based ap-



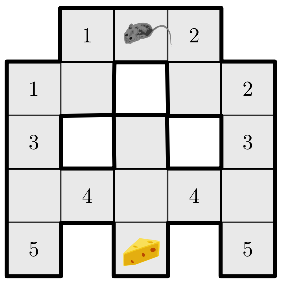
(a) 2×2 Maze (1 600 k-MDP states, 12 latent states): The maze was "unrolled" and shows a successful *activation* and *collection* of the single reward, which has a delay of $\Delta t = 2$ and is depicted by the red arrow.



(b) 4×4 Maze (4 096 000 k-MDP states; 34 012 224 latent states): The 4×4 maze is shown in compact form. Red solid arrows depict rewards with a delay of $\Delta t = 2$; the orange, dashed arrow depicts a reward with delay $\Delta t = 3$. Doors, which remain open for two successive time steps after being opened, are depicted in green with their switch being the nearby semicircle.



(c) Small Cheese Maze (∞ k-MDP states; 11 latent states): On the left, the maze is shown and locations that are indistinguishable to the agent are numbered accordingly. On the right, the current observation representing only the existence or non-existence of the adjacent walls is illustrated.



(d) Large Cheese Maze (∞ k-MDP states; 18 latent states): As for the small version, indistinguishable locations are numbered accordingly and the agent only observes adjacent walls.

Figure 3.11: Non-Markov maze environments employed for the evaluation.

proaches that strive for establishing an MDP state representation based on features (Hutter, 2009) pursue a substantially different objective. The aim in history-based approaches, as taken in our TEF+CRF model, is to find features that enable good predictions based on the history. The feature values themselves do not need to be predictable and generally do not fulfill the Markov property. Existing history-based approaches use context trees (A. K. McCallum, 1996; Nguyen, Sunehag, and Hutter, 2012; Veness et al., 2010; Willems, Shtarkov, and Tjalkens, 1995) for classifying the current history. We compared our TEF+CRF model to the *utility tree* (U-Tree, A. K. McCallum, 1996) method. U-Tree builds a decision tree based on a set of basis features that allow to represent features equivalent to our TEFs. U-Tree originally is a model-free approach that tries to predict the action value but it can be transferred to the model-based setting in a straightforward way. I will refer to the model-free and model-based version as *U-Tree (value)* and *U-Tree (model)*, respectively. U-Tree builds up a decision tree by successively choosing expansions that result in maximally distinct distributions. In U-Tree (value) the action values in each leaf node are learned via Q-iteration (see e.g. Richard S. Sutton and Barto, 1998) and the Kolmogorov-Smirnov test is used to find an expansion for which the action value distributions are maximally distinct. In U-Tree (model) the next observation and reward is modeled and we used the chi-square test to expand the tree.

Model-Free Approach

U-Tree (value) provides a model-free comparison and *PULSE* is not restricted to discovering TEFs in a model-based setting. Therefore, we chose to also employ a simple model-free method based on TEFs. To this end, we approximated the action value function as a linear combination of TEFs

$$Q_{(a,h)} = \sum_{f \in \mathcal{F}} \theta_f f(a,h) \ , \tag{3.88}$$

where h is the current history and a is the next action. This model-free approach, which I will refer to as TEF+Linear Q, was trained via L_1-regularized least-squares policy iteration (Lagoudakis and Parr, 2003). Note that while *PULSE* provides certain guarantees for discovering TEFs in the model-based setting this is not the case for TEF+Linear Q. The ternary conjunctive feature mentioned in Section 3.3.1 might be an interesting option for further investigation in that respect. Also note that a cross-comparison of model-based and model-free approaches is not easily possible because the involved learning tasks are of a different nature (Kaelbling, Littman, and Moore, 1996; Parr, L. Li, et al., 2008). Generally, in tasks with fixed reward distribution, as is the case in our experiments, model-free approaches have the advantage of a smaller prediction space.

N^+ Operation

All environments exhibit a finite horizon of two or three (in case of the 4×4 maze) that is sufficient for acting optimally. Therefore, we restricted the conjunctions to the respective horizon for each environment and used *PULSE* with the $N^+_\wedge$ operation (3.29) to discover TEFs. For a given set $\mathcal{F}$ of TEFs the resulting non-Markov model for predicting the next reward and observation is

$$p_{\mathcal{F}}(o, r \mid a, h; \boldsymbol{\theta}) = \frac{\exp \sum_{f \in \mathcal{F}} \theta_f f(o, r, a, h)}{\sum_{o', r'} \exp \sum_{f \in \mathcal{F}} \theta_f f(o', r', a, h)} \ . \tag{3.89}$$

3.4.2 Results

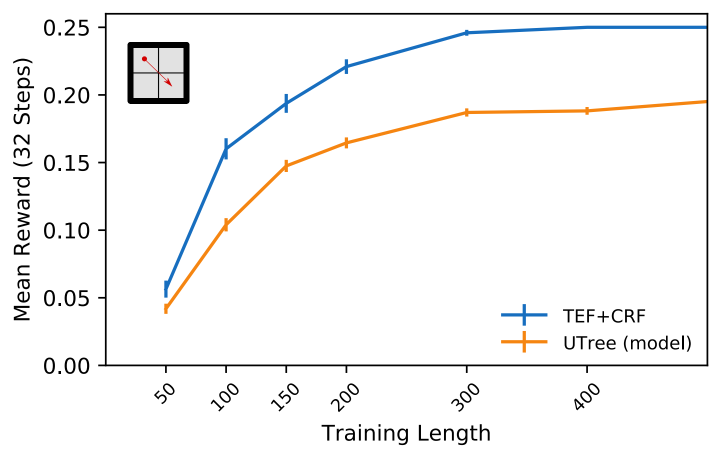
Experiments

In each of the environments we trained the models offline based on data that was collected previously using the random policy for trials of varying length. To evaluate performance we collected rewards using the optimal policy, as based on the method, for a fixed number of steps. For the model-based approaches we used MCTS for planning. The L_1-regularization in TEF+CRF and TEF+Linear Q was chosen in preliminary runs to be as large as possible without affecting the performance. Due to the environments being deterministic there was a relatively sharp point of breakdown. The results for the model-based methods are shown in Figure 3.12, where TEF+CRF clearly outperforms U-Tree (model) in all environments. Figure 3.13 shows the results for the model-free methods with the model-based data in the background (semi-transparent). The TEF+Linear Q approach is clearly inferior to U-Tree (value) in all environment except for the 2×2 maze. As discussed above, TEFs are not designed for this use case and *PULSE* does not offer any guarantees so that this result does not come as a surprise. It is interesting to observe that at some points, especially in the large Cheese Maze, the performance of U-Tree (value) is slightly superior to that of TEF+CRF. As mentioned above, this may be attributed to the smaller prediction space of a model-free method, which results in a higher effective data density.

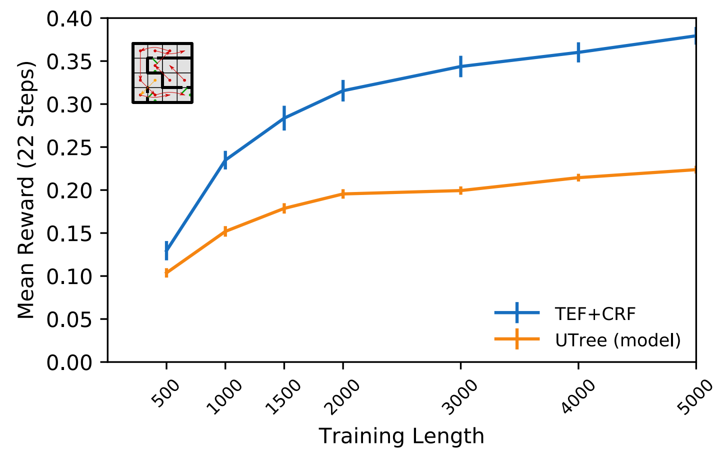
Characteristics of the Discovered Features

The TEFs discovered by *PULSE* can be listed explicitly for the 2×2 maze, which is done in Table 3.5. They are grouped according to their semantics.

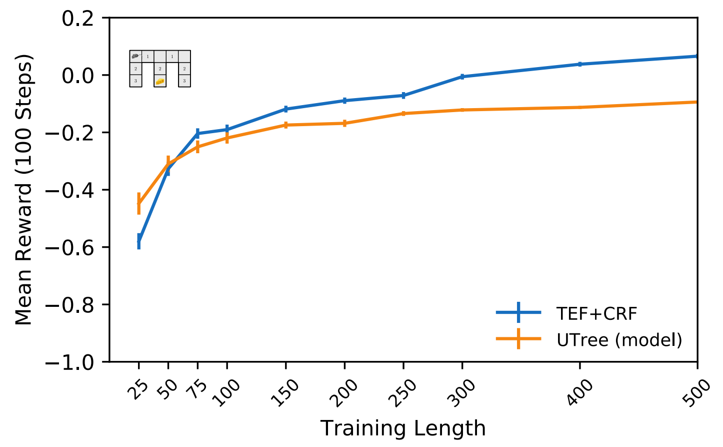
Rewards: The first two features ($\surd_0$, $\times_0$) describe the marginal probability of getting a reward. Due to the uniform policy for collecting the data the marginal distribution



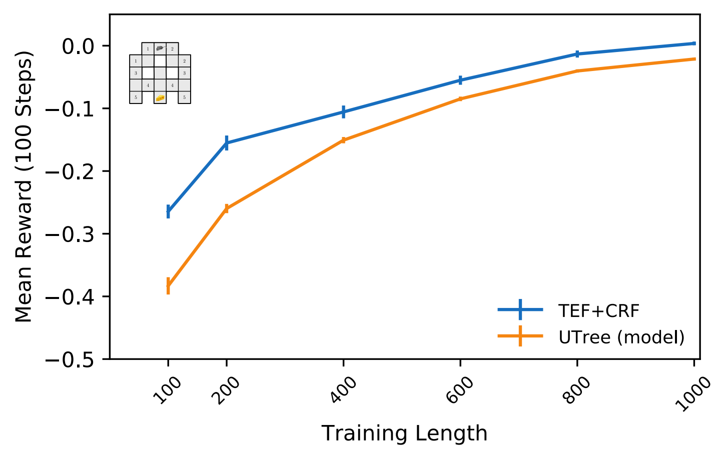
(a) 2×2 Maze



(b) 4×4 Maze

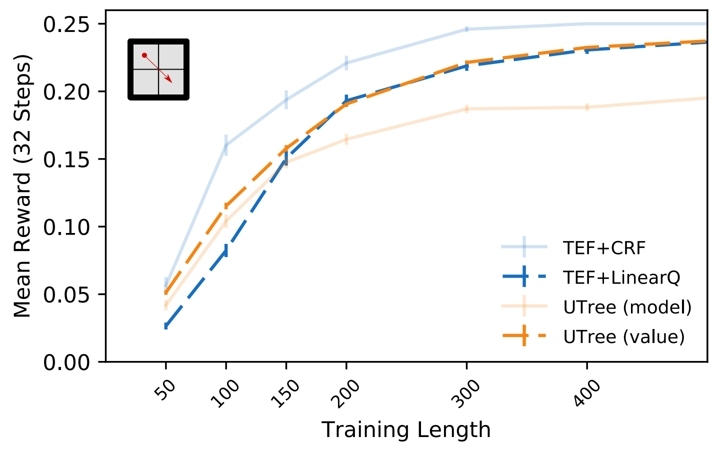


(c) Small Cheese Maze

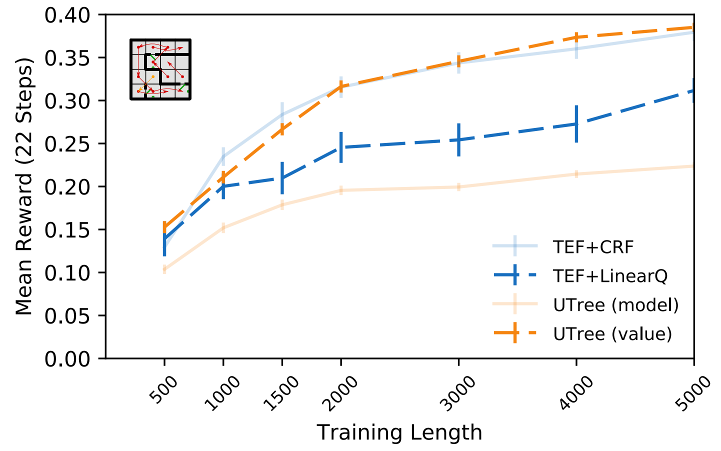


(d) Large Cheese Maze

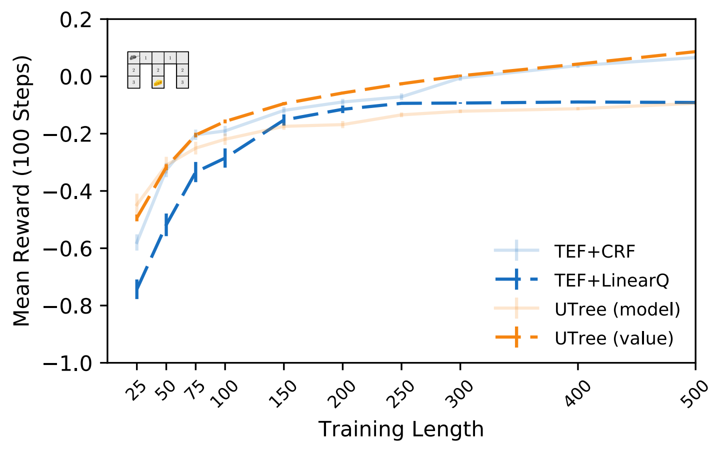
Figure 3.12: Mean reward for the model-based methods, TEF+CRF and U-Tree (model). The number of trials that was collected varies for different methods and trial lengths. The error bars indicate the error of the mean estimator.



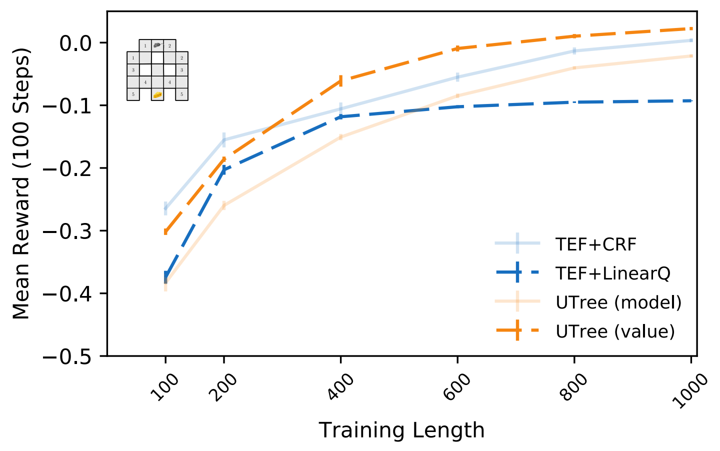
(a) 2×2 Maze



(b) 4×4 Maze



(c) Small Cheese Maze



(d) Large Cheese Maze

Figure 3.13: Mean reward for the model-free methods, TEF+Linear Q and U-Tree (value). The model-based methods, TEF+CRF and U-Tree (model), are shown as semi-transparent plots for comparison. The number of trials that was collected varies for different methods and trial lengths. The error bars indicate the error of the mean estimator.

Reward probability	
✓$_0$	$\theta = -11.91$
×$_0$	$\theta = 11.91$

Correlation between rewards and locations	
✓$_0$ ∧ ◲$_0$	$\theta = 13.50$
×$_0$ ∧ ◰$_{-1}$	$\theta = 8.22$
×$_0$ ∧ ◲$_{-1}$	$\theta = 6.59$
✓$_0$ ∧ ◰$_{-2}$	$\theta = 7.71$
×$_0$ ∧ ◰$_{-2}$	$\theta = -7.71$

What location can be reached from what other location	
◰$_0$ ∧ ◰$_{-1}$	$\theta = 12.95$
◰$_0$ ∧ ◱$_{-1}$	$\theta = 7.15$
◰$_0$ ∧ ◳$_{-1}$	$\theta = 7.03$
◳$_0$ ∧ ◳$_{-1}$	$\theta = 16.02$
◳$_0$ ∧ ◲$_{-1}$	$\theta = 8.18$
◳$_0$ ∧ ◰$_{-1}$	$\theta = 7.73$
◱$_0$ ∧ ◱$_{-1}$	$\theta = 15.60$
◱$_0$ ∧ ◲$_{-1}$	$\theta = 7.97$
◱$_0$ ∧ ◰$_{-1}$	$\theta = 7.37$
◲$_0$ ∧ ◲$_{-1}$	$\theta = 14.36$
◲$_0$ ∧ ◱$_{-1}$	$\theta = 9.34$
◲$_0$ ∧ ◳$_{-1}$	$\theta = 6.28$

What location can be reached with what action	
$\uparrow_0$ ∧ ◰$_0$	$\theta = 14.84$
$\uparrow_0$ ∧ ◳$_0$	$\theta = 15.11$
$\downarrow_0$ ∧ ◱$_0$	$\theta = 15.43$
$\downarrow_0$ ∧ ◲$_0$	$\theta = 15.69$
$\leftarrow_0$ ∧ ◰$_0$	$\theta = 14.86$
$\leftarrow_0$ ∧ ◱$_0$	$\theta = 14.78$
$\rightarrow_0$ ∧ ◲$_0$	$\theta = 15.60$
$\rightarrow_0$ ∧ ◳$_0$	$\theta = 14.96$

Other correlations	
◲$_0$ ∧ ◰$_{-2}$	$\theta = -6.67$
◰$_0$ ∧ ◲$_{-2}$	$\theta = -3.37$
◱$_0$ ∧ ◳$_{-2}$	$\theta = -2.37$
◳$_0$ ∧ ◱$_{-2}$	$\theta = -0.95$
◲$_0$ ∧ $\leftarrow_{-1}$	$\theta = -3.23$
◰$_0$ ∧ $\uparrow_{-1}$	$\theta = 2.78$
◰$_0$ ∧ $\leftarrow_{-1}$	$\theta = 2.56$
◲$_0$ ∧ $\rightarrow_{-1}$	$\theta = 1.31$
×$_0$ ∧ ◳$_{-1}$	$\theta = -1.13$
✓$_0$ ∧ ◳$_{-1}$	$\theta = 1.13$

Table 3.5: Feature set learned by TEF+CRF in the 2×2-maze based on 1500 training steps (37 features): Each row contains a conjunctive feature and its weight θ. The symbols represent the location (◰, ◳, ◱, ◲), the action ($\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$, $\circ$) and whether or not a reward is received (✓, ×). The subscript indicates the time index. For example, the basis feature ◰$_{-1}$ indicates whether the agent was at the top left location in the last time step; ✓$_0$ indicates whether it is (hypothetically) going to get a reward in the next step; and $\leftarrow_{-1}$ indicates that the last move was to the left.

of locations is also uniform. As a consequence, the corresponding features do not appear in the model.

Rewards and Locations: These features express the fact that rewards can only be collected in the bottom right location (✓$_0$ ∧ ◲$_0$) and have to be activated in the top left location two time steps before (✓$_0$ ∧ ◰$_{-2}$, ×$_0$ ∧ ◰$_{-2}$). The remaining two features capture situations where collecting a reward is implicitly impossible because it either could not have been activated (×$_0$ ∧ ◲$_{-1}$) or will not be possible to collect (×$_0$ ∧ ◰$_{-1}$).

Connectivity: The third group of features captures the connectivity structure. Each sub-block lists for a specific location what other locations it can be reached from. The higher weight for features that describe staying in the same location is due to the fact that with the random policy this happens with probability $\frac{3}{5}$.

Location and Actions: The fourth block lists the relation between locations and actions, that is for instance the face that by moving up ($\uparrow$) the agent will end in the top left or top right location, ◰ or ◳.

Other Correlations: The last block collects features of lower importance with weights that are subject to a considerable amount of noise. The first sub-block captures the fact that moving to a diagonally opposite location in two steps is rather unlikely

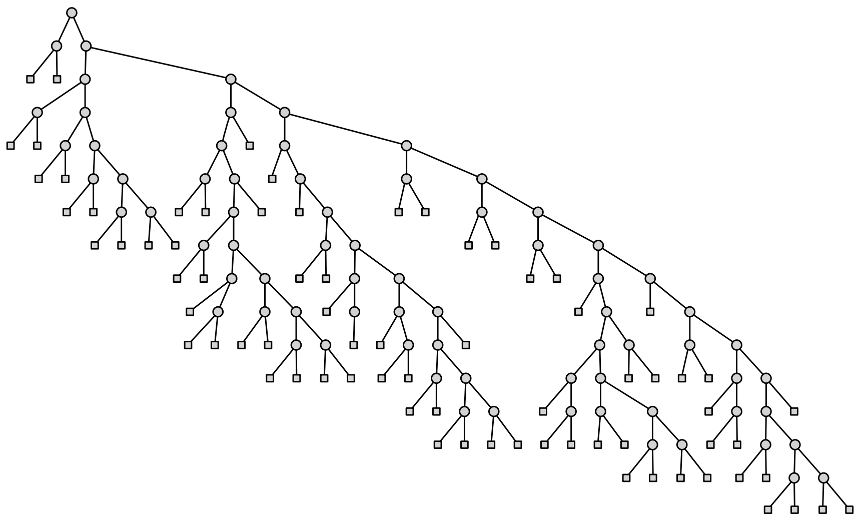


Figure 3.14: U-Tree (model) learned from 1500 training steps in the 2×2-maze (74 leaf nodes; depth 2–15): Each internal node tests for a location, action, or reward at a specific time. Each leaf node contains a tabular representation of the transition probabilities that contains 40 entries (one for each possible combination of an action and an outcomes).

($p = \frac{2}{25}$) under the random policy. The second and third sub-blocks model noisy correlations between the previous action and the next location, and between the previous location and the next reward, respectively.

The grouping of features in Table 3.5 reveals the factorized nature of the learned model. It is clear that dropping all features except for, say, the third block will result in a model that does not, in general, provide accurate predictions but can yield valuable information about reachability. The other blocks yield similarly valuable information about other aspects of the 2×2 maze. In that sense, the features learned by TEF+CRF correspond to an adaptable abstraction as outlined in Section 3.1.

Size of the Models

Finally, we can compare the size of the model learned by TEF+CRF with that one learned by U-Tree (model). The decision tree learned by U-Tree (model) is schematically depicted in Figure 3.14. It contains about twice as many leaf nodes as there are features in the TEF+CRF model. The leaf nodes correspond to mutually exclusive TEFs. Additionally, the depth varies between 2 and 15, meaning that some features correspond to 15-fold conjunctions, which makes them hard to interpret via direct inspection. Moreover, the transition probabilities are maintained in a tabular representation with 40 entries for each leaf node. The model thus has 2960 parameters, which is a factor of 80 more than the

TEF+CRF model. This significant difference in model complexity is indicative for the expressive power of a factorized feature-based representation.

3.5 Temporally Extended Features for Sequential Prediction of Melodies

In this section I will present an application of TEFs and the *PULSE* framework in a different field. As non-Markov dependencies play a crucial role for modeling music, it presents an ideal test bed for TEFs and the *PULSE* framework.[12]

Modeling music is a multifaceted task and learning predictive models for music presents an open challenge. The relevance of this task goes well beyond predicting or generating music. The formation of expectancies constitutes a core capacity of human cognition and plays a major role in the perception of music (Marcus T. Pearce and Wiggins, 2012; Rohrmeier and Koelsch, 2012) and the formation of emotional responses (Huron, 2006; Meyer, 2008). Learning predictive models for music is thus an important component in gaining a better understanding of music itself as well as the processing of complex stimuli in humans. One important strand of research is concerned with the symbolic modeling of monophonic melodies as pitch sequences.

3.5.1 Background on Models for Pitch Sequences

The task of modeling pitch sequences is structurally similar to that of modeling word sequences in computational linguistics and similar approaches have been taken. These involve, most notablely, HMMs (Mavromatis, 2005, 2009), dynamic Bayesian networks (Paiement, S. Bengio, and Eck, 2009; Paiement, Grandvalet, and S. Bengio, 2009; Raczynski et al., 2010), n-gram models (Conklin and Witten, 1995; Marcus T. Pearce and Wiggins, 2004, 2012; Marcus Thomas Pearce, 2005; Whorley et al., 2013), and connectionist approaches (Cherla, Tran, Weyde, et al., 2015; Cherla, Weyde, et al., 2013).

Long- and Short-Term Model

Currently, the models that perform best consist of a combination of a *long-term model* (LTM) and a *short-term model* (STM). The distinction between LTM and STM was suggested by Conklin and Witten (1995). The LTM is trained offline on a corpus of melodies and provides a static time series model for pitch sequences. The STM is trained online on a piece during prediction and provides a sequence model that is able to pick

[12]The content of this section corresponds in large parts to (Langhabel et al., 2017).

	Symbol	Name	Value Range	Description
Viewpoint Features	P	pitch	$\mathcal{P}$	MIDI pitch of the note
	I	interval	$\mathcal{I}$	pitch difference between current and previous note
	C	contour	$\{-1, 0, 1\}$	sign of the interval
	X	extended contour	$\{-2, -1, 0, 1, 2\}$	like C but ± 2 for intervals larger than ± 5 steps
Anchor Features	F_i	i^{th} in piece	$\mathcal{I}$	pitch differences to the i^{th} tone in the current piece
	T	tonic	$\{0, \dots, 11\}$	octave invariant pitch difference to the tonic
	K	key	$\{\mathrm{maj}, \mathrm{min}\} \times \{0, \dots, 11\}$	like T but separate for major and minor keys

Table 3.6: Viewpoint and anchor features. $\mathcal{P}$ is the set of all MIDI pitches occurring in the training corpus and $\mathcal{I} = \{a - b \,|\, a, b \in \mathcal{P}\}$.

up characteristics specific to the given piece like the key or motives. LTM and STM are combined by taking the arithmetic or geometric mean of their predictions weighted based on the entropy of the respective predictive distribution (see Marcus T. Pearce and Wiggins, 2004, for details). Currently, the best performing model employs an n-gram model for the STM (Marcus T. Pearce and Wiggins, 2004) and a *recurrent temporal discriminative restricted Boltzmann machine* (RTDRBM) for the LTM (Cherla, Tran, Garcez, et al., 2015; Cherla, Tran, Weyde, et al., 2015).

Multiple Viewpoint Systems

Instead of working with the raw pitch data it is common to employ *multiple viewpoint systems* (MVS, Conklin and Witten, 1995). A viewpoint may be a feature of the pitch sequence, such as the interval between the two preceding notes, but may also contain additional information, such as the position of a note in the current bar (see Conklin and Witten, 1995; Marcus Thomas Pearce, 2005, for more details). A set of models may be learned based on distinct viewpoints and combined in a mixture model for prediction, thereby forming a MVS.

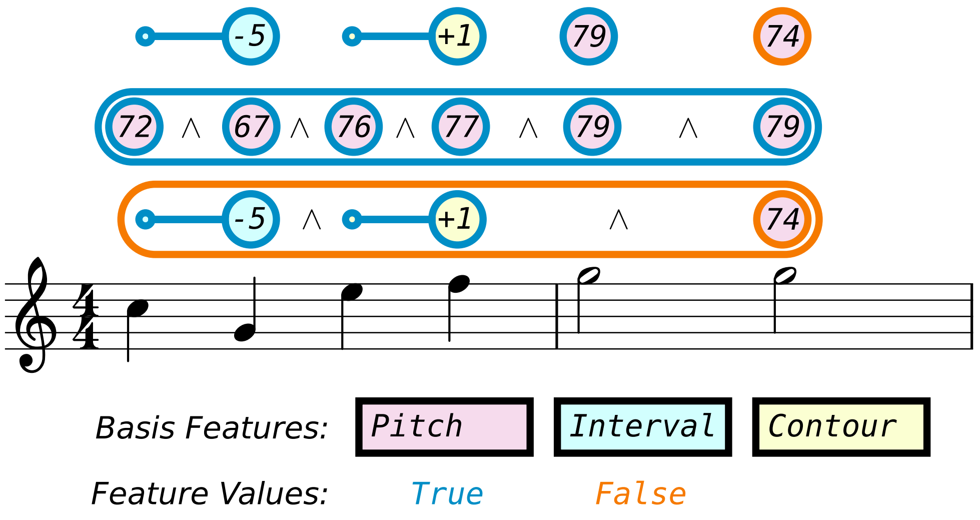


Figure 3.15: Illustration of intermingled/non-intermingled and contiguous/non-contiguous features constructed from three different viewpoints. **Top row:** Examples of tests that can be used in conjunctions. **Middle row:** A contiguous non-intermingled pitch feature. **Bottom row:** A non-contiguous intermingled feature constructed from the pitch (P), interval (I), and contour (C) viewpoint.

3.5.2 Features for Melody Prediction

Temporally Extended Features for Viewpoints

For a given viewpoint the concept of TEFs can be applied in a straight-forward manner. We employed the $\widehat{N}^{+}_{\wedge i}$ operation (3.71) for learning an LTM and the $\widehat{N}^{+}_{\wedge f}$ operation (3.86) for learning an STM. There are two major advantages of discovering TEFs for viewpoints using *PULSE* as compared to MVS. The first advantage is that multiple viewpoints can be mixed freely in one model, either by including separate features for the different viewpoints or by constructing intermingled conjunctions. The second advantage is that the $\widehat{N}^{+}_{\wedge i}$ operation constructs both contiguous conjunctions, which correspond to classical n-grams, as well as non-contiguous conjunctions of variables. This means that TEFs constructed via $\widehat{N}^{+}_{\wedge i}$ are strictly more expressive than n-gram models. To learn TEFs we employed viewpoints derived from the pitch as listed on the upper part of Table 3.6. An illustrative example of the kind of features that can be constructed by a *PULSE* learner is shown in Figure 3.15.

Regularization

As the feature set becomes more diverse with features having different characteristics, for instance, because they are derived from different view points or have a substantially different extent in time, finding an appropriate regularization becomes more involved. In a

PULSE learner the role of regularization is twofold. For one thing, as in any other machine learning method, an appropriate regularization is needed to improve generalization of the model, which becomes a more severe issue for expressive feature sets that have a higher risk to overfit on the training data. For another thing, in *PULSE* the growth of the feature set must be limited to retain practical feasibility, which again becomes a more severe issue for diverse feature sets as the number of possible expansions grows exponentially with the number of viewpoints that are included. It is thus necessary to employ more sophisticated regularization functions that act on different feature types to a different extent. In that respect, it is interesting to note that excluding a certain feature type from expansion in the N^+ operation is the limiting case of a sparseness inducing regularization of that feature type. Our findings for the application of TEFs in modeling monophonic melodies can be summarized as follows:

- Intermingled expansion across multiple viewpoints deteriorates predictive performance if these features are regularized in the same way as single-viewpoint features. We excluded intermingled feature expansion on the N^+-level.
- For some viewpoints expansion beyond the first order deteriorates performance, which was also adjusted on the N^+-level.
- Increasing the L_1-regularization (and the L_2-regularization in case of the STM) exponentially with the temporal extent of the features proved beneficial.
- For the STM starting with a strong L_1- and L_2-regularization and decreasing its strength exponentially during online learning as the data set grows proved beneficial.

The regularization functions contain the overall regularization strengths and the different decay rates as free parameters that need to be adjusted via (cross-)validation. More details can be found in (Langhabel, 2017; Langhabel et al., 2017).

Anchor Features

Another advantage of learning TEFs in the *PULSE* framework is that additional features can easily be included in the model. In tonal music the concept of tonic and key, most notably that of major and minor key, are known to be relevant. In many cases tonic and key are properties of a piece that are established within the first couple of notes and are related to the overall frequency that specific tones occur with (Krumhansl, 1990). As such they are of a substantially different nature than n-gram/k-MDP features. In (Langhabel et al., 2017) we suggested the concept of *anchor features*. An anchor feature is a function from the past events in a piece of music to a viewpoint (pitch in our case) and serves as a reference point for constructing predictive features. We used three different kinds of anchor features as listed in the bottom part of Table 3.6. The F_i features use the i^{th} tone in the piece as anchor. For the T and K features we use the Krumhansl-Schmuckler key finding algorithm (Krumhansl, 1990) with key profiles from (Temperley, 1999) to estimate the tonic and key from the past events. Both feature types use the tonic as anchor, however, the T features ignore the key while the K features are distinct for major and minor keys.

3.5.3 Experiments

We trained our LTM and STM models on a standard benchmark corpus of Bach chorales and folk melodies (Cherla, Tran, Garcez, et al., 2015; Cherla, Tran, Weyde, et al., 2015; Cherla, Weyde, et al., 2013; Marcus T. Pearce and Wiggins, 2004). The corpus contains eight sub-corpora from different cultural backgrounds and models were trained on these sub-corpora independently. Feature weights were optimized using *stochastic gradient descent* (SGD) on the neg-log-likelihood. Performance was evaluated via 10-fold cross-validation using the same folds as in (Cherla, Tran, Garcez, et al., 2015; Marcus T. Pearce and Wiggins, 2004) by first averaging the neg-log-likelihood (i.e. the cross-entropy) over the cross-validation folds within each sub-corpus and then taking the average over the eight sub-corpora. This is the same approach as in previous works (Cherla, Tran, Garcez, et al., 2015; Cherla, Tran, Weyde, et al., 2015; Cherla, Weyde, et al., 2013; Marcus T. Pearce and Wiggins, 2004). Hyper-parameters were optimized separately for each cross-validation fold by leaving out 10% of the training data using Gaussian-process-based global optimization. For the STM we used the configuration PI^*F_1, where an asterisk means that the corresponding features were expanded beyond the first order. The STM hyper-parameters were set based on preliminary runs and were not further optimized. The LTM was tested for the configurations listed in Table 3.8. We tested two configurations for an LTM-STM hybrid by combining the best performing ($\text{PI}^*\text{C}^*\text{K}$)-LTM with the ($\text{PI}^*\text{F}_1$)-STM and a classical n-gram-STM.

Results

Our best performing LTM, STM, and hybrid models significantly improve over the state-of-the-art as reported in (Cherla, Tran, Weyde, et al., 2015; Marcus T. Pearce and Wiggins, 2004). The corresponding performances in terms of cross-entropy are summarized in Table 3.7. For the STM it is noteworthy that this is the first time in more than a decade that an improvement over classical n-gram STMs was achieved. This result was based on an N^+ configuration and hyper-parameter values that were set in preliminary runs

	LTM	STM	Hybrid
Pulse	**2.547**	**3.094**	**2.395**
RTDRBM (Cherla, Tran, Weyde, et al., 2015)	2.712	3.363	2.421
n-gram (Marcus T. Pearce and Wiggins, 2004)	2.878	3.139	2.479

Table 3.7: Comparison of best performing *Pulse*, RTDRBM, and n-gram models.

Pulse-LTM				
P	I*	C	–	2.701
		X*	–	2.692
		C*	–	2.692
			F_1	2.620
			$F_1F_2F_3$	2.602
			T	2.586
			K	**2.547**

Table 3.8: Performance of *Pulse*-LTM for different configurations. An asterisk behind a viewpoint indicates that the corresponding features were expanded beyond the first order.

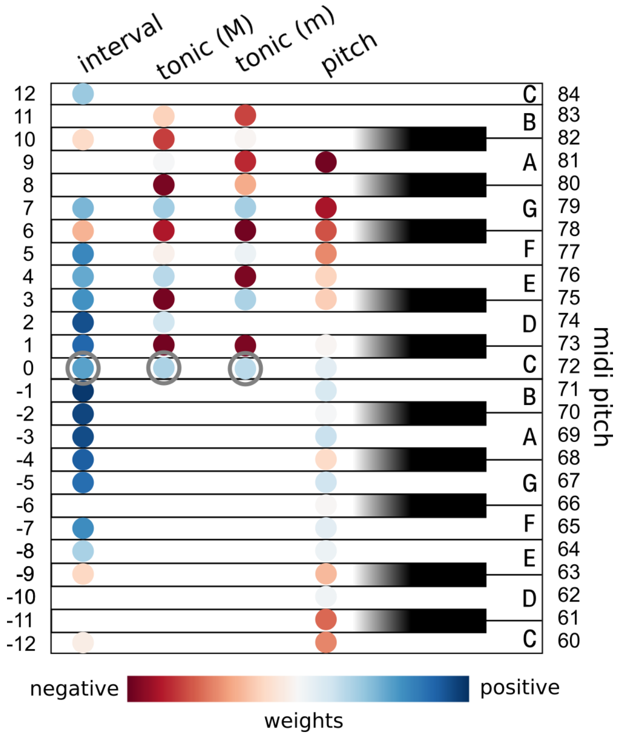


Figure 3.16: Qualitative plot of the feature weights for P and I features of length one as well as K features, learned based on the Bach chorales. For I and K features middle C is chosen as reference tone as marked by the circles. Reproduced from (Langhabel et al., 2017).

and were not optimized thoroughly (which poses a challenge in terms of computational resources). Further improvements are therefore to be expected. Also see (Langhabel, 2017) for a more detailed investigation of STM configurations. For the LTM it can be seen that including prior knowledge about the tonic and key significantly improves performance culminating in the best performing PI*C*K configuration. The PI* configuration as basis for both the LTM and the STM is convincing from a musicological perspective as the P component allows to model a marginal distribution over pitch values (also see Figure 3.16) while the I* component models melodies in a transposition invariant way. The observed improvement for inclusion of contour features C* is also music-theoretically expected. However, the extended contour features X* were not observed to further improve performance. The role of the K features becomes obvious from Figure 3.16, which reveals that they express a preference for tones from the major and minor triad, respectively. For the hybrid model an interesting finding is that replacing our TEF-based (PI*F$_1$)-STM with a classical n-gram-STM improves performance by another 0.03 bits even though the n-gram-STM by itself displays a lower performance than our (PI*F$_1$)-STM (3.139 versus 3.094 bits). We conjecture that the TEF-based LTM and the n-gram-STM have complementary properties and that the n-gram-STM better compensates for missing information. This highlights a weakness in the hybrid approach of separating LTM and STM as well as generally in MVS: By training models separately it is not possible for them to actively learn to complement each other. Therefore, an aim for future research should be to reunite the distinct models. Due to its flexibility the *PULSE* framework is a promising candidate for this task.

Chapter 4

Active Planning

Planning is the task of finding an optimal policy π^* for a given model of the environment

$$\pi^*_{(a\,|\,s)} = \text{argmax}_\pi Q^\pi_{(s,a)} \tag{4.1}$$

$$Q^\pi_{(s,a)} = \sum_{s'} p_{(s'\,|\,s,a)} \left(r_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a'\,|\,s')}\, Q^\pi_{(s',a')} \right) . \tag{4.2}$$

In complex environments, computing the action value Q exactly requires summing an intractably large number of terms generated by unrolling the recursion in Eq. (4.2) over all possible trajectories. The art of planning thus consists in finding ways to compute or approximate Q efficiently.[13] This is usually done in an iterative manner where in each iteration the algorithm takes a *planning action* that consists of querying information from the model and updating the estimate of Q. What actions are at the planners disposal entirely depends on the model's capabilities. The way how planning actions are chosen makes up the essence of a planning algorithm. For example, value iteration (see e.g. Richard S. Sutton and Barto, 1998) uses a predictive model and in each iteration queries all transition probabilities for all state-action pairs. MCTS, in contrast, uses a generative model and queries a particular sequence of simulated transitions. Taking a planning action may be related to taking actions in the environment, as in the case of MCTS, but in general planning actions may be of an entirely different kind as long as the model supports it. This may give an indication of how diverse planning actions and the ways to select them may be.

Just as the agent interacts with the environment, the process of planning can be seen as an interactive process in its own right. As a consequence the planning problem can be reformulated as an active learning problem. In the course of this chapter I will describe

[13]Technically, Q is only an interim step in computing π^*, however, everything transfers to the case where π^* is attempted to be directly computed.

this view of *planning as active learning* in more detail and discuss some of its consequences and applications.

4.1 Planning as Active Learning

To formulate planning as an active learning process we need to identify the components of the planning process that make up the corresponding *interactive process* (IP), which are the actions, observations, the internal state and internal state updates, and the reward/objective. I will use Latin and calligraphic letters ($a \in \mathcal{A}$, $o \in \mathcal{O}$) to denote variables in the environment that planning should be performed in and Gothic letters ($\mathfrak{a} \in \mathfrak{A}$, $\mathfrak{o} \in \mathfrak{O}$) for variables associated to the planning process. The detailed definitions depend on the specific environment, the kind of model that is used for planning, as well as the planning algorithm itself. In the following I describe the active learning formulation of planning on a general level.

Actions: Planning actions can be any kind of query or manipulation to the model. Let the planning actions $\mathfrak{a} \in \mathfrak{A}$ be the set of *atomic queries* to a model that cannot be further broken down. A particular planning algorithm may still perform a sequence of atomic actions before effectively updating its internal state. For a probabilistic model a planning action corresponds to querying the probability for a particular transition; for a generative model a planning action corresponds to querying a simulated transition from a particular state for a particular action; for a feature-based model a planning action corresponds to specifying a subset of features and querying the upper or lower bound for a particular transition

$$\mathfrak{A} \equiv \mathcal{S} \times \mathcal{A} \qquad \textit{(generative model)} \tag{4.3}$$

$$\mathfrak{A} \equiv \mathcal{S}^2 \times \mathcal{A} \qquad \textit{(probabilistic model)} \tag{4.4}$$

$$\mathfrak{A} \equiv \mathcal{S}^2 \times \mathcal{A} \times \{0,1\}^{|\mathcal{F}|} \times \{+,-\} \qquad \textit{(feature-based model)}\ . \tag{4.5}$$

Observations: The observation is the model's response to the planning action, which is a probability for probabilistic and feature-based models and the resulting state from the simulated transition for generative models

$$\mathfrak{O} \equiv \mathcal{S} \qquad p_{\mathfrak{O}} : \mathfrak{A} \rightsquigarrow \mathfrak{O} \qquad \textit{(generative model)} \tag{4.6}$$

$$\mathfrak{O} \equiv [0,1] \qquad p_{\mathfrak{O}} : \mathfrak{A} \to \mathfrak{O} \qquad \textit{(probabilistic model)} \tag{4.7}$$

$$\mathfrak{O} \equiv [0,1] \qquad p_{\mathfrak{O}} : \mathfrak{A} \to \mathfrak{O} \qquad \textit{(feature-based model)}\ . \tag{4.8}$$

Internal State/Internal State Updates: I assume the planner has an internal state $\boldsymbol{\xi}$ consisting of all relevant variables, data structures etc. that it updates after each planning action and observation. In MCTS, for instance, $\boldsymbol{\xi}$ comprises the structure of the search tree and the transition counts and observed returns in all of its nodes. The internal state update is deterministic and depends on the specific planning approach. For planners that use sampled data I will write $\boldsymbol{\xi}[D]$ to indicate that the internal state depends on the data $D \in (\mathfrak{A} \times \mathfrak{O})^*$ collected by the planner. This connects well to the active learning notation used in Section 2.1.3 and also covers the general case of a non-Markov internal state $\boldsymbol{\xi}$ that is not updated but instead directly computed from the data.

Objective: The objective $\mathcal{O}$ is a function of the planner's internal state $\boldsymbol{\xi}$ that quantifies the uncertainty associated with the quantity of interest (Section 4.1.1).

4.1.1 Objective

The goal is to find an optimal policy π^* as in Eq. (4.1), that is, to identify the optimal action

$$a^* = \operatorname{argmax}_a Q_{(s,a)} \tag{4.9}$$

for a given state s. The posterior distribution of a^* is

$$p(a^* \,|\, s, \boldsymbol{\xi}) = \int p(\mathbf{Q}_s \,|\, \boldsymbol{\xi}) \, [\![\forall_{a \in \mathcal{A}} \, Q_{(s,a)} \leq Q_{(s,a^*)}]\!] \, d\mathbf{Q}_s \ , \tag{4.10}$$

where the integral runs over the $|\mathcal{A}|$-dimensional space of action values $\mathbf{Q}_s$ for state s, and for sampling-based planners $p(\mathbf{Q}_s \,|\, \boldsymbol{\xi}[D])$ is estimated from the data $D \in (\mathfrak{A} \times \mathfrak{O})^*$ collected by the planner. The most obvious active learning objective is to minimize the entropy of the optimal action

$$\mathrm{H}(a^* \,|\, \boldsymbol{\xi}) \ . \tag{4.11}$$

If $p(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ takes a particularly simple form it may be possible to solve the integral in Eq. (4.10) exactly (see e.g. Section 5.2.4) but this will not generally be the case. An alternative objective is to instead minimize the entropy of the action values

$$\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi}) \ . \tag{4.12}$$

Minimizing $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ does not guarantee a minimal value of $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$ nor the other way around. On the one hand, the probability mass of $p(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ may be distributed in a region where the maximum action is uniquely defined, which results in a high value of $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ yet in a low value of $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$. On the other hand, $p(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ may exhibit multiple low-entropy modes in areas with different maximum actions, which implies a low value of $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ yet a high value of $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$. Even if $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$ is tractable to compute, minimizing $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ might produce more reliable results in situations where the optimal action is not uniquely defined as it allows quantifying information gain that is not reflected in $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$. A third option is to minimize the entropy of the state value

$$\mathrm{H}(V_s^\pi \,|\, \boldsymbol{\xi}) \;, \tag{4.13}$$

where

$$p(V_s^\pi \,|\, \boldsymbol{\xi}) = \sum_a p(a^* \,|\, s, \boldsymbol{\xi}) \int p(\mathbf{Q}_s \,|\, \boldsymbol{\xi}) \, Q_{(s,a)} \; d\mathbf{Q}_s \;. \tag{4.14}$$

Computing $p(V_s^{\pi^*} \,|\, \boldsymbol{\xi})$ requires $p(a^* \,|\, s, \boldsymbol{\xi})$ as well as $p(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ to be tractable to compute. We may circumvent computing $p(a^* \,|\, s, \boldsymbol{\xi})$ by using an alternative policy $\pi_{(a \,|\, s, \boldsymbol{\xi})}$ that chooses actions based on the planner's current internal state $\boldsymbol{\xi}$ and converges to the optimal policy as the uncertainty of the action values decreases

$$\lim_{\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi}) \to -\infty} \pi_{(a \,|\, s, \boldsymbol{\xi})} = p(a^* \,|\, s, \boldsymbol{\xi}) \;. \tag{4.15}$$

Using $\pi_{(a \,|\, s, \boldsymbol{\xi})}$ instead of $p(a^* \,|\, s, \boldsymbol{\xi})$ the state value distribution $p(V_s^{\pi^*} \,|\, \boldsymbol{\xi})$ may become tractable to compute and we may use $\mathrm{H}(V_s^\pi \,|\, \boldsymbol{\xi})$ as our objective.

As computing $\mathrm{H}(V_s^\pi \,|\, \boldsymbol{\xi})$ still requires $p(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ to be tractable to compute it may seem more intuitive to directly use $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ as our objective in this situation. However, using $\mathrm{H}(V_s \,|\, \boldsymbol{\xi})$ may be advantageous for the following reason: If the value $Q_{(s,a)}$ for a specific action a has a high uncertainty while at the same time a being almost certainly sub-optimal this uncertainty in $Q_{(s,a)}$ contributes to $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$ even though the optimal action can be determined with high certitude, that is, $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$ would have a low value. When we are using $\mathrm{H}(V_s \,|\, \boldsymbol{\xi})$ as our objective this problem may be avoided by employing a carefully chosen policy $\pi_{(a \,|\, s, \boldsymbol{\xi})}$ that assign a low probability to these high-uncertainty sub-optimal actions. Using $\mathrm{H}(V_s \,|\, \boldsymbol{\xi})$ thus allows retrieving some desirable properties of $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$ while being as tractable to compute as $\mathrm{H}(\mathbf{Q}_s \,|\, \boldsymbol{\xi})$.

4.1.2 Action Selection

Let $\mathcal{O}(\boldsymbol{\xi}[D])$ be the objective that we want to minimize, $\boldsymbol{\xi}$ be the planner's internal state, and $D \in (\mathfrak{A} \times \mathfrak{O})^*$ be the data collected by the planner. As discussed in Section 2.1.3 finding an optimal policy for an active learning problem is intractable in most cases and I will therefore assume actions to be chosen by the one-step greedy policy (2.21)

$$\mathfrak{a}^* = \operatorname{argmin}_{\mathfrak{a}} \mathbb{E}\Big[\!\!\Big[\mathcal{O}\Big(\boldsymbol{\xi}[D, \mathfrak{a}, \mathfrak{o}]\Big)\Big]\!\!\Big]_{\mathfrak{o}\,|\,\mathfrak{a},D} \tag{4.16}$$

$$= \operatorname{argmin}_{\mathfrak{a}} \Delta_{\mathfrak{a}}\mathcal{O}\Big(\boldsymbol{\xi}[D]\Big)\ , \tag{4.17}$$

where $\mathfrak{a} \in \mathfrak{A}$ and $\mathfrak{o} \in \mathfrak{O}$ are planning actions and observations and $\Delta_{\mathfrak{a}}$ is a shorthand for the expectation (the letter Δ is used because we are effectively interested in the expected change of the objective). Even for the greedy policy, computing Eq. (4.16) exactly will be intractable in all but the most restricted cases as it involves computing the expectation over all possible observations $\mathfrak{o}$ conditional on $\mathfrak{a}$ of the objective after adding $(\mathfrak{a}, \mathfrak{o})$ to the data D, which in turn requires re-computing the objective for each of these cases.

Linear Approximation

Instead of computing the full expectation in Eq. (4.16) we may approximate changes of our objective linearly, which simplifies computations substantially. Additionally, in many cases the planner's internal state $\boldsymbol{\xi}$ is highly structured and consists of a number of variables, from which a given planning action $\mathfrak{a}$ may affect only a small subset. Let $\boldsymbol{\xi}_{\mathfrak{a}} \subseteq \boldsymbol{\xi}$ be the subset of variables whose values may change when performing the planning action $\mathfrak{a}$. We can then perform a linear approximation of Eq. (4.16)

$$\mathfrak{a}^* = \underset{\mathfrak{a}}{\operatorname{argmin}}\ \mathbb{E}\Big[\!\!\Big[\mathcal{O}\Big(\boldsymbol{\xi}[D, \mathfrak{a}, \mathfrak{o}]\Big)\Big]\!\!\Big]_{\mathfrak{o}\,|\,\mathfrak{a},D} \tag{$\rightarrow$ 4.16}$$

$$= \underset{\mathfrak{a}}{\operatorname{argmin}}\ \mathbb{E}\Big[\!\!\Big[\mathcal{O}\Big(\boldsymbol{\xi}[D, \mathfrak{a}, \mathfrak{o}]\Big) - \mathcal{O}\Big(\boldsymbol{\xi}[D]\Big)\Big]\!\!\Big]_{\mathfrak{o}\,|\,\mathfrak{a},D} \tag{4.18}$$

$$\approx \underset{\mathfrak{a}}{\operatorname{argmin}}\ \mathbb{E}\left[\!\!\left[\frac{\partial\mathcal{O}\Big(\boldsymbol{\xi}[D]\Big)}{\partial\boldsymbol{\xi}_{\mathfrak{a}}}\Big(\boldsymbol{\xi}_{\mathfrak{a}}[D, \mathfrak{a}, \mathfrak{o}] - \boldsymbol{\xi}_{\mathfrak{a}}[D]\Big)\right]\!\!\right]_{\mathfrak{o}\,|\,\mathfrak{a},D} \tag{4.19}$$

$$= \underset{\mathfrak{a}}{\operatorname{argmin}}\ \frac{\partial\mathcal{O}\Big(\boldsymbol{\xi}[D]\Big)}{\partial\boldsymbol{\xi}_{\mathfrak{a}}}\mathbb{E}\Big[\!\!\Big[\boldsymbol{\xi}_{\mathfrak{a}}[D, \mathfrak{a}, \mathfrak{o}]\Big]\!\!\Big]_{\mathfrak{o}\,|\,\mathfrak{a},D}\ , \tag{4.20}$$

where in Eq. (4.19) we assumed the change of $\mathcal{O}$ to be approximately linear in the variables $\boldsymbol{\xi}_{\mathfrak{a}}$ affected by the planning action $\mathfrak{a}$. This principle can be generalized to higher orders by computing the Taylor expansion of the objective around the planner's current internal state $\boldsymbol{\xi}$. For second order approximations this should be tractable at least for a diagonal approximation of the Hessian. Approximations of higher order n are likely to become intractable due to the size of the higher-order tensors of partial derivatives that grows exponentially. The linear decomposition in Eq. (4.20) separates the gradient of the objective for the given data D from the expected change of the planners internal state $\boldsymbol{\xi}$ induced by $(\mathfrak{a}, \mathfrak{o})$. This separation can result in a considerable reduction of computational costs because the gradient does not need to be recomputed inside the expectation. Moreover, we can compute the gradient once for all variables $\boldsymbol{\xi}$, which will usually be more efficient than computing it separately for each subset of affected variables $\boldsymbol{\xi}_{\mathfrak{a}}$.

In Section 4.2 I describe how the formulation of active planning together with the linear approximation in Eq. (4.20) can be applied to enhance classical MCTS. In Chapter 5 the active planning approach is combined with the structured models developed in Chapter 3 to produce a novel kind of planning method where planning actions are performed in the feature space of the model.

4.2 Active Tree Search

Active tree search (ATS) applies the idea of active planning to *Monte-Carlo tree search* (MCTS). The planner has access to a black-box model of the environment from which it can sample transitions to successively build up a forward search tree and estimate the action values at the root node. MCTS embraces the idea of active learning in that the tree policy addresses action selection as a multi-armed bandit problem. The state from which an action is sampled, however, is chosen implicitly as the result of all preceding transitions in the rollout. In contrast, the idea of active planning is to choose each single planning action individually based on a well-defined criterion. Therefore, in MCTS neither a single transition nor the rollout as a whole can be considered a deliberate planning action in the spirit of active planning.

In ATS a planning action corresponds to querying a single transition from a single state[14] $\mathfrak{a} = (s, a)$ from the model, which returns the resulting state and reward as observation $\mathfrak{o} = (s^*, \rho^*)$.

4.2.1 Objective Function

In MCTS the action values at the root node are estimated by performing backups along each newly sampled rollout. For ATS we need to estimate the distribution $p(\mathbf{Q}_s \mid D)$ of

[14]For reasons of simplicity, I will assume that the state s uniquely identifies a node in the tree. Otherwise the planning action has to disambiguate by selecting a node, which then implies the state s.

action values instead. The value estimates of MCTS correspond to the expected value

$$\widehat{Q}_{(s,a)} = \mathbb{E}\left[\!\left[Q_{(s,a)} \mid D \right]\!\right] \ , \tag{4.21}$$

that is the first moment of $p(\mathbf{Q}_s \mid D)$. To approximate the distribution $p(\mathbf{Q}_s \mid D)$ in ATS I also compute the second moments, that is, the variances

$$\widetilde{Q}_{(s,a)} = \mathbb{E}\left[\!\left[(Q_{(s,a)} - \widehat{Q}_{(s,a)})^2 \mid D \right]\!\right] \ , \tag{4.22}$$

where I ignored covariance terms, that is, I assumed the action values for different states or actions to be independent (also see Section 4.2.2). The maximum entropy distribution with mean $\widehat{Q}_{(s,a)}$ and variance $\widetilde{Q}_{(s,a)}$ is a normal distribution with an entropy proportional to $\log \widetilde{Q}_{(s,a)}$ (2.20), that is, I assume $p(\mathbf{Q}_s \mid D)$ to be a multivariate normal distribution with diagonal covariance matrix. Under this assumption solving the integral in Eq. (4.10) for computing $p(a^* \mid s, D)$ involves the error function, which does not have a closed-form solution, so that using the entropy $\mathrm{H}(a^* \mid D)$ (4.11) as objective is unfavorable.[15] The alternative of minimizing $\mathrm{H}(\mathbf{Q}_s \mid D)$ (4.12) is straight forward as

$$\mathrm{H}(\mathbf{Q}_s \mid D) \propto \sum_a \log \widetilde{Q}_{(s,a)} \ . \tag{4.23}$$

However, as mentioned in Section 4.1.1 it may be favorable to use the entropy of the state value $\mathrm{H}(V_s \mid D)$ (4.13) as our objective if we can define an appropriate policy $\pi_{(a \mid s,D)}$. For ATS I use an optimistic soft-max policy

$$\pi_{(a \mid s,D)} \propto \exp \frac{1}{\tau}\left[\widehat{Q}_{(s,a)} + C\left(\sqrt{\widetilde{Q}_{(s,a)} + \epsilon^2} - \epsilon \right) \right] \ , \tag{4.24}$$

where the temperature τ regulates the greediness, C determines the amount of exploration/optimism, and the small constant $\epsilon > 0$ ensures differentiability for zero variance, which will become important when applying the linear decomposition (4.20). For low temperatures $\tau \approx 0$ the optimistic soft-max policy (4.24) fulfills the convergence condition (4.15)

$$\mathrm{H}(\mathbf{Q}_s \mid D) \to -\infty \Rightarrow \forall_a \, \widetilde{Q}_{(s,a)} \to 0 \tag{4.25}$$

$$\Rightarrow \pi_{(a \mid s,D)} \to [\![a = \operatorname{argmax}_{a'} \widehat{Q}_{(s,a')}]\!] = \pi^*_{(a \mid s)} \ , \tag{4.26}$$

[15] An interesting starting point for further investigation arises from the fact that in ATS we only need to be able to compute the derivative of the objective. Since the derivative of the error function is simply the normal distribution using the entropy $\mathrm{H}(a^* \mid D)$ as objective might actually be tractable.

where Eq. (4.25) assumes that $p(\mathbf{Q}_s \,|\, D)$ is non-degenerate. Another advantage of using $\mathrm{H}(V_s \,|\, D)$ with the optimistic soft-max policy is that it is robust against degenerate $p(\mathbf{Q}_s \,|\, D)$ as $\mathrm{H}(V_s \,|\, D)$ remains finite if $\widetilde{Q}_{(s,a)} = 0$ for some actions a whereas $\mathrm{H}(\mathbf{Q}_s \,|\, D)$ collapses to $-\infty$. To compute $\mathrm{H}(V_s \,|\, D)$ I will, as for the action values, approximate $p(V_s \,|\, D)$ as a normal distribution with mean $\widehat{V}_s$ and variance $\widetilde{V}_s$ so that minimizing $\mathrm{H}(V_s \,|\, D)$ is equivalent to minimizing $\widetilde{V}_s$ and the final objective for ATS is

$$\mathcal{O}(\boldsymbol{\xi}[D]) = \widetilde{V}_s \ , \tag{4.27}$$

where s is the state at the root node.

4.2.2 Computing the Variance

In order to compute the variance $\widetilde{V}_s$ at the root node we have to propagate not only the mean values $\widehat{Q}_{(s,a)}$ but also the variances $\widetilde{Q}_{(s,a)}$ through the tree.[16] The state and action values are defined as

$$V_s = \sum_a \pi_{(a\,|\,s)} \underbrace{\sum_{s'} p_{(s'\,|\,s,a)} \Big(r_{(s,a,s')} + \gamma\, V_{s'} \Big)}_{Q_{(s,a)}} \tag*{$(\rightarrow 2.6)$}$$

$$Q_{(s,a)} = \sum_{s'} p_{(s'\,|\,s,a)} \Big(r_{(s,a,s')} + \gamma \underbrace{\sum_{a'} \pi_{(a'\,|\,s')}\, Q_{(s',a')}}_{V_{s'}} \Big) \ . \tag*{$(\rightarrow 2.7)$}$$

All variables (V, Q, p, r) in these equations except for the policy π, which is defined as in Eq. (4.24), are to be understood as random variables with an associated posterior distribution that we need to compute from the data D. Generally, variables associated with different nodes in the tree may be correlated. For instance, the value in different nodes that represent the same state of the environment is perfectly correlated, or the rewards for transitions to different states $r_{(s,a,s')}$ and $r_{(s,a,s'')}$ may be correlated if the reward only depends on s and a but not on the resulting state. In what follows I shall assume that variables associated with different nodes are independent, that is, their covariance is zero. This assumption is implicit in vanilla MCTS, such as UCT (Kocsis and Szepesvári, 2006), where statistics are computed per node. The only exception will be transition probabilities, where $\widetilde{p}_{(s'/s''\,|\,s,a)}$ denotes the covariance of the probabilities of ending in s' and s'', respectively, when performing action a in state s.

[16]The backup equations for the variance derived in this section are not specific to ATS and can also be used to employ variance-based tree policies in classical MCTS with Bellman backups (Lieck, Ngo, and Toussaint, 2017).

To compute mean and variance of V and Q we first need to estimate the mean and variance of p and r.

Transition Probabilities and Expected Rewards

Assuming a Dirichlet distribution for p the mean and variance of p and r are

$$\widehat{p}_{(s'\,|\,s,a)} = \frac{n_{(s'|s,a)}}{n_{(s,a)}} \tag{4.28}$$

$$\widetilde{p}_{(s'\,|\,s,a)} = \frac{\widehat{p}_{(s'\,|\,s,a)}\left(1-\widehat{p}_{(s'\,|\,s,a)}\right)}{n_{(s,a)}+1} \tag{4.29}$$

$$\widetilde{p}_{(s'/s''\,|\,s,a)} = -\frac{\widehat{p}_{(s'\,|\,s,a)}\,\widehat{p}_{(s''\,|\,s,a)}}{n_{(s,a)}+1} \qquad s' \neq s'' \tag{4.30}$$

$$\widehat{r}_{(s,a,s')} = \frac{\Sigma_{\rho_{(s,a,s')}}}{n_{(s'|s,a)}} \tag{4.31}$$

$$\widetilde{r}_{(s,a,s')} = \frac{\Sigma_{\rho^2_{(s,a,s')}}}{n^2_{(s'|s,a)}} - \frac{\widehat{r}^{\,2}_{(s,a,s')}}{n_{(s'|s,a)}}\ , \tag{4.32}$$

where $n_{(s'|s,a)}$ is the number of transitions from (s,a) that ended in s', $n_{(s,a)} = \sum_{s'} n_{(s'|s,a)}$ is the number of all transitions from (s,a), $\Sigma_{\rho_{(s,a,s')}}$ is the sum of rewards for transitions from (s,a) that ended in s', and $\Sigma_{\rho^2_{(s,a,s')}}$ is the corresponding sum of squared rewards. Note that in order to get meaningful values also for a single sample, I decided to use the sample variance for the reward instead of the bias-corrected estimator of the population variance. As an alternative one could use a Bayesian approach and specify a prior for the reward distribution.

State and Action Values

The mean values $\widehat{V}$ and $\widehat{Q}$ follow directly from Eqs. (2.6) and (2.7)

$$\widehat{V}_s = \sum_a \pi_{(a\,|\,s)} \sum_{s'} \widehat{p}_{(s'\,|\,s,a)} \left(\widehat{r}_{(s,a,s')} + \gamma\,\widehat{V}_{s'}\right) \tag{4.33}$$

$$\widehat{Q}_{(s,a)} = \sum_{s'} \widehat{p}_{(s'\,|\,s,a)} \left(\widehat{r}_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a'\,|\,s')}\,\widehat{Q}_{(s',a')}\right)\ . \tag{4.34}$$

In deriving the variance $\widetilde{V}_s$ and $\widetilde{Q}_{(s,a)}$ I will repeatedly use the fact that for two random variables x and y

$$\mathbb{E}[\![xy]\!] = \widehat{x}\,\widehat{y} + \mathrm{Cov}[x,y] \tag{4.35}$$

and that the covariance $\mathrm{Cov}[x, y]$ is zero if x and y are independent (as assumed above for all variables belonging to different nodes except for p).

$$
\begin{aligned}
\widetilde{V}_s &= \mathbb{E}\left[\!\!\left[\Big[\sum_a \pi_{(a\,|\,s)}\, Q_{(s,a)} \Big]^2 \right]\!\!\right] - \widehat{V}_s^2 && (4.36)\\
&= \mathbb{E}\left[\!\!\left[\sum_{a,a'} \pi_{(a\,|\,s)}\, \pi_{(a'\,|\,s)}\, Q_{(s,a)}\, Q_{(s,a')} \right]\!\!\right] - \widehat{V}_s^2 && (4.37)\\
&= \sum_a \pi^2_{(a\,|\,s)}\, \widetilde{Q}_{(s,a)} \ . && (4.38)\\
\widetilde{Q}_{(s,a)} &= \mathbb{E}\left[\!\!\left[\Big[\sum_{s'} p_{(s'\,|\,s,a)} \Big(r_{(s,a,s')} + \gamma\, V_{s'} \Big) \Big]^2 \right]\!\!\right] - \widehat{Q}^2_{(s,a)} && (4.39)\\
&= \mathbb{E}\left[\!\!\left[\sum_{s',s''} p_{(s'\,|\,s,a)}\, p_{(s''\,|\,s,a)} \Big[r_{(s,a,s')}\, r_{(s,a,s'')} + \gamma^2\, V_{s'}\, V_{s''} + \gamma\, r_{(s,a,s')}\, V_{s''} + \gamma\, r_{(s,a,s'')}\, V_{s'} \Big] \right]\!\!\right] - \widehat{Q}^2_{(s,a)} && (4.40)\\
&= \sum_{s'} \Big(\widehat{p}^{\,2}_{(s'\,|\,s,a)} + \widetilde{p}_{(s'\,|\,s,a)} \Big) \Big[\widetilde{r}_{(s,a,s')} + \gamma^2\, \widetilde{V}_{s'} \Big] + \dots \\
&\quad \dots + \sum_{s',s''} \widetilde{p}_{(s'/s''\,|\,s,a)} \Big[\widehat{r}_{(s,a,s')}\, \widehat{r}_{(s,a,s'')} + \gamma^2\, \widehat{V}_{s'}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s')}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s'')}\, \widehat{V}_{s'} \Big] + \dots \\
&\quad \dots + \sum_{s',s''} \widehat{p}_{(s'\,|\,s,a)}\, \widehat{p}_{(s''\,|\,s,a)} \Big[\widehat{r}_{(s,a,s')}\, \widehat{r}_{(s,a,s'')} + \gamma^2\, \widehat{V}_{s'}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s')}\, \widehat{V}_{s''} + \gamma\, \widehat{r}_{(s,a,s'')}\, \widehat{V}_{s'} \Big] - \widehat{Q}^2_{(s,a)} && (4.41)\\
&= \sum_{s'} \Big(\widehat{p}^{\,2}_{(s'\,|\,s,a)} + \widetilde{p}_{(s'\,|\,s,a)} \Big) \Big[\widetilde{r}_{(s,a,s')} + \gamma^2\, \widetilde{V}_{s'} \Big] + \sum_{s',s''} \widetilde{p}_{(s'/s''\,|\,s,a)} \Big[\widehat{r}_{(s,a,s')} + \gamma\, \widehat{V}_{s'} \Big] \Big[\widehat{r}_{(s,a,s'')} + \gamma\, \widehat{V}_{s''} \Big] \ , && (4.42)\\
&= \sum_{s'} \Big(\widehat{p}^{\,2}_{(s'\,|\,s,a)} + \widetilde{p}_{(s'\,|\,s,a)} \Big) \Big[\widetilde{r}_{(s,a,s')} + \gamma^2 \sum_{a'} \pi^2_{(a'\,|\,s')}\, \widetilde{Q}_{(s',a')} \Big] + \dots \\
&\quad \dots + \sum_{s',s''} \widetilde{p}_{(s'/s''\,|\,s,a)} \Big[\widehat{r}_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a'\,|\,s')} \widehat{Q}_{(s',a')} \Big] \Big[\widehat{r}_{(s,a,s'')} + \gamma \sum_{a''} \pi_{(a''\,|\,s'')} \widehat{Q}_{(s'',a'')} \Big] \ , && (4.43)
\end{aligned}
$$

where in Eq. (4.41) the first line collects all variance terms in r and V (covariance terms drop out), the second line collects the covariance terms in p, and the third line cancels out. In Eq. (4.43) the state values are rewritten in terms of the action values so that we can directly update the action value variances when performing backups.

4.2.3 Computing Expected Changes

Let $\mathfrak{a} = (s, a)$ be the planning action and $\mathfrak{o} = (s^*, \rho^*)$ be the planning observation, that is, the resulting state and reward returned by the generative model. For computing the expected change of the objective (4.27) I apply the linear decomposition (4.20), which separates computing the gradient (Section 4.2.4) from computing the expected change of the variables affected by an action. Let $\widehat{r}^*$, $\widetilde{r}^*$, $\widehat{p}^*$, and $\widetilde{p}^*$ be the new values of the existing variables $\widehat{r}$, $\widetilde{r}$, $\widehat{p}$, and $\widetilde{p}$ after performing $\mathfrak{a}$ and observing $\mathfrak{o}$, that is, after adding the transition $(s, a) \to (s^*, \rho^*)$ to the data of the corresponding node. All changes of these variables accumulate in $\widehat{Q}_{(s,a)}$ and $\widetilde{Q}_{(s,a)}$ at the corresponding node and it turns out that we can compute the expected values $\mathbb{E}[\![\widehat{Q}_{(s,a)}]\!]_{s^*,\rho^*}$ and $\mathbb{E}[\![\widetilde{Q}_{(s,a)}]\!]_{s^*,\rho^*}$ in closed form. We can therefore take $\boldsymbol{\xi}_{\mathfrak{a}}$ in Eq. (4.20) to be $\widehat{Q}_{(s,a)}$ and $\widetilde{Q}_{(s,a)}$ for $\mathfrak{a} = (s, a)$.

Transition Probabilities and Expected Rewards

In order to compute $\mathbb{E}[\![\widehat{Q}_{(s,a)}]\!]_{s^*,\rho^*}$ and $\mathbb{E}[\![\widetilde{Q}_{(s,a)}]\!]_{s^*,\rho^*}$ we need to handle the involved expectations over s^* and ρ^*. The state s^* may be one of the previously observed states or an unobserved one, for which a new node in the tree would need to be created. Structurally, this corresponds to a Chinese restaurant process and the choice of how to model this step will generally depend on prior knowledge about the specific environment. I will assume that there is no specific knowledge about particular states and all new states are processed in the same way. As a consequence we can pretend there was only a single unobserved state and write $\mathcal{D}_{(s^* \,|\, s,a)}$ for the probability of observing the previously or newly observed state s^*. Specifically, if $\mathcal{D}$ is a Chinese restaurant process with concentration parameter α we have

$$\mathcal{D}_{(s^* \,|\, s,a)} = \begin{cases} \frac{\alpha}{\alpha + n_{(s,a)}} & \text{if } s^* \text{ was not observed before} \\ \frac{n_{(s^* \,|\, s,a)}}{\alpha + n_{(s,a)}} = \frac{n_{(s,a)}}{\alpha + n_{(s,a)}} \widehat{p}_{(s^* \,|\, s,a)} & \text{else .} \end{cases} \tag{4.44}$$

This allows us to compute the expectations $\mathbb{E}[\![\cdots]\!]_{s^*}$ by explicitly summing over possible outcomes s^*

$$\mathbb{E}[\![\cdots]\!]_{s^*} = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)} \left[\cdots\right] , \tag{4.45}$$

where the sum runs over observed states plus a hypothetical unobserved state and inside the sum we need to use the updated values $\widehat{p}^*$ and $\widetilde{p}^*$ that are (re)computed based on Eqs. (4.28)–(4.30) after temporally adding the transition $(s, a) \to (s^*, \rho^*)$ to the data of the corresponding node (which possibly needs to be created first).

The observed reward ρ^* is continuous so that we would need to integrate over possible values. However, ρ^* only influences the mean $\widehat{r}$ and variance $\widetilde{r}$ of the expected reward. As it turns out the expectations $\mathbb{E}[\![\cdots]\!]_{\rho^*}$ over ρ^* can be expressed in terms of the expected values of the updated variables $\widehat{r}^*$, $\widehat{r}^{*2}$, and $\widetilde{r}^*$, which have closed-form solutions based on Eqs. (4.31) and (4.32)

$$\begin{aligned}
\mathbb{E}\left[\!\!\left[\,\widehat{r}^*\right]\!\!\right]_{\rho^*} &= \mathbb{E}\left[\!\!\left[\frac{n\widehat{r}+\rho^*}{n+1}\right]\!\!\right] = \widehat{r} &(4.46)\\
\mathbb{E}\left[\!\!\left[\,\widehat{r}^{*2}\right]\!\!\right]_{\rho^*} &= \mathbb{E}\left[\!\!\left[\frac{(\rho^*+n\widehat{r})^2}{(n+1)^2}\right]\!\!\right] &(4.47)\\
&= \frac{1}{(n+1)^2}\left[\mathbb{E}[\![\rho^{*2}]\!] + 2n\mathbb{E}[\![\rho^*]\!]\widehat{r} + n^2\widehat{r}^2\right] &(4.48)\\
&= \frac{1}{(n+1)^2}\left[\widehat{r}^2 + n\widetilde{r} + 2n\widehat{r}^2 + n^2\widehat{r}^2\right] &(4.49)\\
&= \widehat{r}^2 + \frac{n}{(n+1)^2}\,\widetilde{r} &(4.50)\\
\mathbb{E}[\![\,\widetilde{r}^*]\!]_{\rho^*} &= \mathbb{E}\left[\!\!\left[\frac{\rho^{*2}+n^2\widetilde{r}+n\widehat{r}^2}{(n+1)^2} - \frac{\widehat{r}^{*2}}{n+1}\right]\!\!\right] &(4.51)\\
&= \frac{\widehat{r}^2+n\widetilde{r}+n^2\widetilde{r}+n\widehat{r}^2}{(n+1)^2} - \frac{\widehat{r}^2}{n+1} - \frac{n\widetilde{r}}{(n+1)^3} &(4.52)\\
&= \frac{\widehat{r}^2+n\widetilde{r}+n^2\widetilde{r}+n\widehat{r}^2}{(n+1)^2} - \frac{\widehat{r}^2}{n+1} - \frac{n\widetilde{r}}{(n+1)^3} &(4.53)\\
&= \frac{n^2(n+2)}{(n+1)^3}\,\widetilde{r}\ , &(4.54)
\end{aligned}$$

where n is the number of transitions observed so far and I made use of the fact that

$$\begin{aligned}
\mathbb{E}[\![\rho^*]\!] &= \widehat{r} &(4.55)\\
\text{and}\qquad \mathbb{E}[\![\rho^{*2}]\!] &= \mathbb{E}[\![\rho^*]\!]^2 + \mathrm{Var}(\rho^*) &(4.56)\\
&= \widehat{r}^2 + n\,\widetilde{r}\ . &(4.57)
\end{aligned}$$

To simplify notation below, I will use

$$\alpha_{(s,a,s'=s^*)} = \begin{cases} \frac{n^2(n+2)}{(n+1)^3} - 1 & \text{if } s' = s'' \\ 0 & \text{else} \end{cases} \tag{4.58}$$

$$\beta_{(s,a,s'=s''=s^*)} = \begin{cases} \frac{n}{(n+1)^2} & \text{if } s' = s'' = s^* \\ 0 & \text{else .} \end{cases} \tag{4.59}$$

State and Action Values

We are now in the position to compute the expected values of of $\widehat{Q}$ and $\widetilde{Q}$

$$\begin{aligned}
&\mathbb{E}\left[\!\left[\widehat{Q}_{(s,a)}\right]\!\right]_{s^*,\rho^*} \\
&\quad = \mathbb{E}\left[\!\left[\sum_{s'} \widehat{p}^*_{(s'|s,a)} \left(\widehat{r}^*_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a' \,|\, s')}\, \widehat{Q}_{(s',a')}\right)\right]\!\right]_{s^*,\rho^*} && (4.60) \\
&\quad = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)} \mathbb{E}\left[\!\left[\sum_{s'} \widehat{p}^*_{(s'|s,a)} \left(\widehat{r}^*_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a' \,|\, s')}\, \widehat{Q}_{(s',a')}\right)\right]\!\right]_{\rho^*} && (4.61) \\
&\quad = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)} \sum_{s'} \widehat{p}^*_{(s'|s,a)} \left(\mathbb{E}\left[\!\left[\widehat{r}^*_{(s,a,s')}\right]\!\right]_{\rho^*} + \gamma \sum_{a'} \pi_{(a' \,|\, s')}\, \widehat{Q}_{(s',a')}\right) && (4.62) \\
&\quad = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)} \sum_{s'} \widehat{p}^*_{(s'|s,a)} \left(\widehat{r}_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a' \,|\, s')}\, \widehat{Q}_{(s',a')}\right) && (4.63) \\
&\quad = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)}\, \widehat{Q}^*_{(s,a)} && (4.64)
\end{aligned}$$

$$\begin{aligned}
&\mathbb{E}\left[\!\left[\widetilde{Q}_{(s,a)}\right]\!\right]_{s^*,\rho^*} \\
&\quad = \mathbb{E}\Bigg[\!\!\Bigg[\sum_{s'} \left(\widehat{p}^{*2}_{(s' \,|\, s,a)} + \widetilde{p}^*_{(s' \,|\, s,a)}\right)\left[\widetilde{r}^*_{(s,a,s')} + \gamma^2 \sum_{a'} \pi^2_{(a' \,|\, s')}\, \widetilde{Q}_{(s',a')}\right] + \ldots \\
&\qquad\qquad \ldots + \sum_{s',s''} \widetilde{p}^*_{(s'/s'' \,|\, s,a)}\left[\widehat{r}^*_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a' \,|\, s')} \widehat{Q}_{(s',a')}\right]\left[\widehat{r}^*_{(s,a,s'')} + \gamma \sum_{a''} \pi_{(a'' \,|\, s'')} \widehat{Q}_{(s'',a'')}\right]\Bigg]\!\!\Bigg]_{s^*,\rho^*} && (4.65) \\
&\quad = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)}\, \mathbb{E}\Bigg[\!\!\Bigg[\sum_{s'} \left(\widehat{p}^{*2}_{(s' \,|\, s,a)} + \widetilde{p}^*_{(s' \,|\, s,a)}\right)\left[\widetilde{r}^*_{(s,a,s')} + \gamma^2 \sum_{a'} \pi^2_{(a' \,|\, s')}\, \widetilde{Q}_{(s',a')}\right] + \ldots \\
&\qquad\qquad \ldots + \sum_{s',s''} \widetilde{p}^*_{(s'/s'' \,|\, s,a)}\left[\widehat{r}^*_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a' \,|\, s')} \widehat{Q}_{(s',a')}\right]\left[\widehat{r}^*_{(s,a,s'')} + \gamma \sum_{a''} \pi_{(a'' \,|\, s'')} \widehat{Q}_{(s'',a'')}\right]\Bigg]\!\!\Bigg]_{\rho^*} && (4.66) \\
&\quad = \sum_{s^*} \mathcal{D}_{(s^* \,|\, s,a)} \Bigg\{ \sum_{s'} \left(\widehat{p}^{*2}_{(s' \,|\, s,a)} + \widetilde{p}^*_{(s' \,|\, s,a)}\right)\left[\mathbb{E}\left[\!\left[\widetilde{r}^*_{(s,a,s')}\right]\!\right]_{\rho^*} + \gamma^2 \sum_{a'} \pi^2_{(a' \,|\, s')}\, \widetilde{Q}_{(s',a')}\right] + \ldots \\
&\qquad \ldots + \sum_{s',s''} \widetilde{p}^*_{(s'/s'' \,|\, s,a)}\Bigg[\mathbb{E}\left[\!\left[\widehat{r}^*_{(s,a,s')}\widehat{r}^*_{(s,a,s'')}\right]\!\right]_{\rho^*} + \mathbb{E}\left[\!\left[\widehat{r}^*_{(s,a,s')}\right]\!\right]_{\rho^*} \gamma \sum_{a''} \pi_{(a'' \,|\, s'')} \widehat{Q}_{(s'',a'')} + \ldots \\
&\qquad \ldots + \mathbb{E}\left[\!\left[\widehat{r}^*_{(s,a,s'')}\right]\!\right]_{\rho^*} \gamma \sum_{a'} \pi_{(a' \,|\, s')} \widehat{Q}_{(s',a')} + \gamma \sum_{a'} \pi_{(a' \,|\, s')} \widehat{Q}_{(s',a')} \gamma \sum_{a''} \pi_{(a'' \,|\, s'')} \widehat{Q}_{(s'',a'')}\Bigg]\Bigg\} && (4.67)
\end{aligned}$$

$$= \sum_{s^*} \mathcal{D}_{(s^*\,|\,s,a)} \Bigg\{ \sum_{s'} \Big(\widehat{p}^{*2}_{(s'\,|\,s,a)} + \widetilde{p}^{*}_{(s'\,|\,s,a)} \Big) \Big[\big(\alpha_{(s,a,s'=s^*)} + 1 \big)\, \widetilde{r}_{(s,a,s')} + \gamma^2 \sum_{a'} \pi^2_{(a'\,|\,s')}\, \widetilde{Q}_{(s',a')} \Big] + \dots$$

$$\dots + \sum_{s',s''} \widetilde{p}^{*}_{(s'/s''\,|\,s,a)} \Big[\widehat{r}_{(s,a,s')} \widehat{r}_{(s,a,s'')} + \beta_{(s,a,s'=s''=s^*)} \widetilde{r}_{(s,a,s^*)} + \widehat{r}_{(s,a,s')} \gamma \sum_{a''} \pi_{(a''\,|\,s'')} \widehat{Q}_{(s'',a'')} + \dots$$

$$\dots + \widehat{r}_{(s,a,s'')} \gamma \sum_{a'} \pi_{(a'\,|\,s')} \widehat{Q}_{(s',a')} + \gamma \sum_{a'} \pi_{(a'\,|\,s')} \widehat{Q}_{(s',a')} \gamma \sum_{a''} \pi_{(a''\,|\,s'')} \widehat{Q}_{(s'',a'')} \Big] \Bigg\} \tag{4.68}$$

$$= \sum_{s^*} \mathcal{D}_{(s^*\,|\,s,a)} \Bigg\{ \sum_{s'} \Big(\widehat{p}^{*2}_{(s'\,|\,s,a)} + \widetilde{p}^{*}_{(s'\,|\,s,a)} \Big) \Big[\widetilde{r}_{(s,a,s')} + \gamma^2 \sum_{a'} \pi^2_{(a'\,|\,s')}\, \widetilde{Q}_{(s',a')} \Big] + \dots$$

$$\dots + \sum_{s',s''} \widetilde{p}^{*}_{(s'/s''\,|\,s,a)} \Big[\widehat{r}_{(s,a,s')} + \gamma \sum_{a'} \pi_{(a'\,|\,s')} \widehat{Q}_{(s',a')} \Big] \Big[\widehat{r}_{(s,a,s'')} + \gamma \sum_{a''} \pi_{(a''\,|\,s'')} \widehat{Q}_{(s'',a'')} \Big] + \dots$$

$$\dots + \Big(\widehat{p}^{*2}_{(s^*\,|\,s,a)} + \widetilde{p}^{*}_{(s^*\,|\,s,a)} \Big) \widetilde{r}_{(s,a,s^*)} \alpha_{(s,a,s^*)} + \widetilde{p}^{*}_{(s^*\,|\,s,a)} \widetilde{r}_{(s,a,s^*)} \beta_{(s,a,s^*)} \Bigg\} \tag{4.69}$$

$$= \sum_{s^*} \mathcal{D}_{(s^*\,|\,s,a)} \Bigg\{ \widetilde{Q}^{*}_{(s,a)} + \Big[\Big(\widehat{p}^{*2}_{(s^*\,|\,s,a)} + \widetilde{p}^{*}_{(s^*\,|\,s,a)} \Big) \Big(\frac{n^2(n+2)}{(n+1)^3} - 1 \Big) + \widetilde{p}^{*}_{(s^*\,|\,s,a)} \frac{n}{(n+1)^2} \Big] \widetilde{r}_{(s,a,s^*)} \Bigg\} \tag{4.70}$$

$$= \sum_{s^*} \mathcal{D}_{(s^*\,|\,s,a)} \Bigg\{ \widetilde{Q}^{*}_{(s,a)} - \frac{1}{(n+1)^3} \Big[(n^2 + 3n + 1)\, \widehat{p}^{*2}_{(s^*\,|\,s,a)} + (2n+1)\, \widetilde{p}^{*}_{(s^*\,|\,s,a)} \Big] \widetilde{r}_{(s,a,s^*)} \Bigg\} \,, \tag{4.71}$$

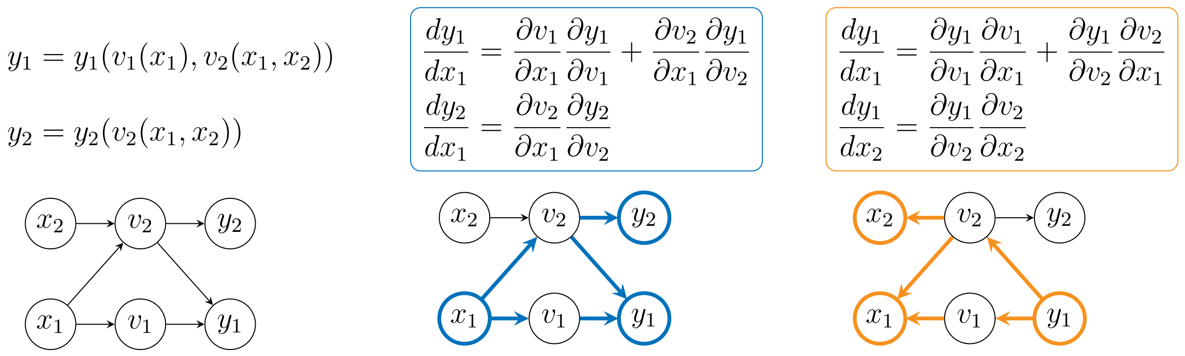
where the starred variables $\widehat{Q}^*$, $\widetilde{Q}^*$ are a shorthand notation to indicate updated/recomputed values of $\widehat{Q}$ and $\widetilde{Q}$ after (temporally) adding the transition $(s, a) \to s^*$ *ignoring the reward*. That is, for $\widehat{Q}^*$ and $\widetilde{Q}^*$ we are using the updated transition probabilities $\widehat{p}^*$ and $\widetilde{p}^*$ but the *old* expected rewards $\widehat{r}$ and $\widetilde{r}$ and then compute $\widehat{Q}^*$ and $\widetilde{Q}^*$ as usual via Eqs. (4.34) and (4.43). The expected change of $\widehat{Q}_{(s,a)}$ and $\widetilde{Q}_{(s,a)}$ when taking planning action $\mathfrak{a} = (s, a)$ is

$$\Delta\widehat{Q}_{(s,a)} = \mathbb{E}\Big[\!\Big[\widehat{Q}_{(s,a)} \Big]\!\Big]_{s^*,\rho^*} - \widehat{Q}_{(s,a)} \tag{4.72}$$

$$\text{and} \quad \Delta\widetilde{Q}_{(s,a)} = \mathbb{E}\Big[\!\Big[\widetilde{Q}_{(s,a)} \Big]\!\Big]_{s^*,\rho^*} - \widetilde{Q}_{(s,a)} \,, \tag{4.73}$$

with the expectations computed according to Eqs. (4.64) and (4.71).[17]

[17]Note that $\widehat{Q}_{(s,a)}$ is defined as the expected value of $Q_{(s,a)}$ so that $\mathbb{E}[\![\widehat{Q}_{(s,a)}]\!]$ is the expected value of an expected value and $\Delta\widehat{Q}_{(s,a)}$ should be zero by definition. However, firstly, the expectation in Eq. (4.72) is not taken with respect to $p(Q\,|\,D)$, which was used to define $\widehat{Q}_{(s,a)}$, and, secondly, structural changes also change the definition of $\widehat{Q}_{(s,a)}$ by adding more terms in the sums, so that effectively $\Delta\widehat{Q}_{(s,a)}$ may be non-zero.


$y_1 = y_1(v_1(x_1), v_2(x_1, x_2))$
$y_2 = y_2(v_2(x_1, x_2))$
$\frac{dy_1}{dx_1} = \frac{\partial v_1}{\partial x_1}\frac{\partial y_1}{\partial v_1} + \frac{\partial v_2}{\partial x_1}\frac{\partial y_1}{\partial v_2}$
$\frac{dy_2}{dx_1} = \frac{\partial v_2}{\partial x_1}\frac{\partial y_2}{\partial v_2}$
$\frac{dy_1}{dx_1} = \frac{\partial y_1}{\partial v_1}\frac{\partial v_1}{\partial x_1} + \frac{\partial y_1}{\partial v_2}\frac{\partial v_2}{\partial x_1}$
$\frac{dy_1}{dx_2} = \frac{\partial y_1}{\partial v_2}\frac{\partial v_2}{\partial x_2}$
x_2
v_2
y_2
x_1
v_1
y_1

(a) Computational Graph **(b)** Forward Accumulation **(c)** Reverse Accumulation

Figure 4.1: Automatic Differentiation: In a computational graph **(a)** for a multivariate function $\mathbf{y}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian can be computed via n passes of forward accumulation **(b)** or m passes of reverse accumulation **(c)**.

4.2.4 Computing the Gradient

The second step in computing the expected change of the objective in the linear decomposition (4.20) is to compute the gradient of the objective with respect to the variables affected by an action. As discussed in Section 4.2.3 $\widehat{r}$, $\widetilde{r}$, $\widehat{p}$, and $\widetilde{p}$ are the independent variables but all effects of a planning action $\mathfrak{a} = (s, a)$ accumulate in $\widehat{Q}_{(s,a)}$ and $\widetilde{Q}_{(s,a)}$ for which we can compute the expected change in closed form. We thus need to compute the gradient of our objective

$$\mathcal{O}(\boldsymbol{\xi}[D]) = \widetilde{V}_s \qquad (\to 4.27)$$
$$= \sum_a \pi^2_{(a\,|\,s)}\, \widetilde{Q}_{(s,a)} \qquad (\to 4.38)$$

with respect to *all* $\widehat{Q}_{(s,a)}$ and $\widetilde{Q}_{(s,a)}$ in the tree, those that immediately occur in Eq. (4.38) and those that indirectly influence $\widetilde{V}_s$ via Eq. (4.43). This task is similar to that of computing the gradient of the loss in a neural network with respect to all weights where the corresponding solution is known as *error backpropagation* or simply *backprop*. We can apply the same general principle, known as *automatic differentiation* (see e.g. Bartholomew-Biggs et al., 2000), and in implementations use the same tools and libraries to efficiently compute the gradient. The idea of automatic differentiation is illustrated in Figure 4.1 for a two-valued function of two variables. For an m-valued function $(y_1, \ldots, y_m)$ of n

variables $x_1, \ldots, x_n$ the Jacobian[18]

$$J = \begin{pmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \vdots \\ \frac{dy_m}{dx_1} & \cdots & \frac{dy_m}{dx_n} \end{pmatrix} \tag{4.74}$$

can be computed via n passes of *forward accumulation* (Figure 4.1b) or m passes of *reverse accumulation* (Figure 4.1c) through the computational graph. To compute the derivative $\frac{dy_i}{dx_j}$ we have to sum over all paths from x_j to y_i and for each path take the product of partial derivatives along the path. Forward and reverse accumulation are two different ways of computing these sums over products while exploiting the fact that a single partial derivative in the computational graph may lie on multiple paths. Let $\mathfrak{a} = (s_n, a_n)$ be a specific planning action and $s_0, a_0, \ldots, s_n, a_n$ be the state-action sequence corresponding the path from the root node to (s_n, a_n). The expected change of the objective can then be approximated as

$$\Delta_{(s_n,a_n)}\mathcal{O}(\boldsymbol{\xi}[D]) \approx \begin{pmatrix} \frac{d\widetilde{V}_{s_0}}{d\widehat{Q}_{(s_0,a_0)}} & \frac{d\widetilde{V}_{s_0}}{d\widetilde{Q}_{(s_0,a_0)}} \end{pmatrix} \begin{pmatrix} \Delta\widehat{Q}_{(s_n,a_n)} \\ \\ \Delta\widetilde{Q}_{(s_n,a_n)} \end{pmatrix} \tag{4.75}$$

$$= \begin{pmatrix} \frac{\partial\widetilde{V}_{s_0}}{\partial\widehat{Q}_{(s_0,a_0)}} & \frac{\partial\widetilde{V}_{s_0}}{\partial\widetilde{Q}_{(s_0,a_0)}} \end{pmatrix} \underbrace{\begin{pmatrix} \frac{d\widehat{Q}_{(s_0,a_0)}}{d\widehat{Q}_{(s_n,a_n)}} & \frac{d\widehat{Q}_{(s_0,a_0)}}{d\widetilde{Q}_{(s_n,a_n)}} \\ \\ \frac{d\widetilde{Q}_{(s_0,a_0)}}{d\widehat{Q}_{(s_n,a_n)}} & \frac{d\widetilde{Q}_{(s_0,a_0)}}{d\widetilde{Q}_{(s_n,a_n)}} \end{pmatrix}}_{J_{(s_0,a_0),(s_n,a_n)}} \begin{pmatrix} \Delta\widehat{Q}_{(s_n,a_n)} \\ \\ \Delta\widetilde{Q}_{(s_n,a_n)} \end{pmatrix} , \tag{4.76}$$

where the expected changes $\Delta\widehat{Q}_{(s_n,a_n)}$ and $\Delta\widetilde{Q}_{(s_n,a_n)}$ are computed from Eqs. (4.72) and (4.73) and

$$\frac{\partial\widetilde{V}_{s_0}}{\partial\widehat{Q}_{(s_0,a_0)}} = 0 \tag{4.77}$$

$$\frac{\partial\widetilde{V}_{s_0}}{\partial\widetilde{Q}_{(s_0,a_0)}} = \pi^2_{(a_0\,|\,s_0)} \ . \tag{4.78}$$

[18]Mathematically, the Jacobian contains partial derivatives. Algorithmically, however, the corresponding functions are computed by a series of function calls. I will use partial derivatives $\frac{\partial\,\cdot}{\partial\,\cdot}$ for derivatives of these algorithmic functions with respect to their arguments and total derivatives $\frac{d\,\cdot}{d\,\cdot}$ whenever the chain of function calls has to be traversed. Also see the discussion of Wegnert lists by Bartholomew-Biggs et al. (2000) in that respect.

The Jacobian

$$J_{(s_0,a_0),(s_n,a_n)} = J_{(s_0,a_0),(s_1,a_1)}\, J_{(s_1,a_1),(s_2,a_2)} \,\cdots\, J_{(s_{n-1},a_{n-1}),(s_n,a_n)} \tag{4.79}$$

$$\text{with} \quad J_{(s_{k-1},a_{k-1}),(s_k,a_k)} = \begin{pmatrix} \frac{\partial \widehat{Q}_{(s_{k-1},a_{k-1})}}{\partial \widehat{Q}_{(s_k,a_k)}} & \frac{\partial \widehat{Q}_{(s_{k-1},a_{k-1})}}{\partial \widetilde{Q}_{(s_k,a_k)}} \\ \frac{\partial \widetilde{Q}_{(s_{k-1},a_{k-1})}}{\partial \widehat{Q}_{(s_k,a_k)}} & \frac{\partial \widetilde{Q}_{(s_{k-1},a_{k-1})}}{\partial \widetilde{Q}_{(s_k,a_k)}} \end{pmatrix} \tag{4.80}$$

has to be computed via forward or reverse accumulation where the partial derivatives follow from Eqs. (4.34) and (4.43).[19]

Reverse Accumulation: In reverse accumulation the product of Jacobians (4.79) is computed from left to right starting with $J_{(s_0,a_0),(s_1,a_1)}$. The accumulated Jacobian $J_{(s_0,a_0),(s_k,a_k)}$ after the first k steps is shared among all nodes in the corresponding subtree so that it can be reused in all branches and only a single pass through the tree is needed to compute the full gradient.

Forward Accumulation: In forward accumulation the product of Jacobians (4.79) is computed from right to left starting with $J_{(s_{n-1},a_{n-1}),(s_n,a_n)}$. The accumulated Jacobian $J_{(s_{n-k},a_{n-k}),(s_n,a_n)}$ after the first k steps is unique to the starting node. While subsequent factors are shared among nodes of the corresponding subtree the accumulated Jacobians for the different nodes have distinct initial values when reaching that point and need to be accumulated separately.

4.2.5 Efficient Updates and Action Selection

Partial Updates

We can compute the expected change of the objective with respect to all planning actions by performing a pass of reverse accumulation through the whole tree. Doing this after each planning action, however, may slow down ATS considerably as compared to classical MCTS especially when the tree grows larger. It also seems inefficient as large parts of the tree are unaffected by a single planning action. Let $\mathfrak{a} = (s_n, a_n)$ be a planning action and (s_k, a_k) an arbitrary state-action pair on the path from (s_n, a_n) to the root node. Using Eq. (4.76) the policy (4.16) can be rewritten as

$$\mathfrak{a}^* = \operatorname{argmin}_{(s_n,a_n)} \Delta_{(s_n,a_n)} \mathcal{O}\Big(\boldsymbol{\xi}[D]\Big) \tag{$\rightarrow$ 4.16}$$

[19]Note that $\partial\widehat{Q}/\partial\widetilde{Q}$ is not zero because the policy π depends on $\widetilde{Q}$. In practice these derivatives can be computed automatically using libraries that provide automatic differentiation facilities such as Theano (Theano Development Team, 2016) for Python.

$$= \operatorname{argmin}_{(s_n,a_n)} \mathbf{dV}_{(s_0,a_0),(s_k,a_k)} \, \boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)} \ , \tag{4.81}$$

where the Jacobian $J_{(s_0,a_0),(s_n,a_n)}$ was split as

$$\underbrace{\begin{pmatrix} \frac{\partial \widetilde{V}_{s_0}}{\partial \widehat{Q}_{(s_0,a_0)}} & \frac{\partial \widetilde{V}_{s_0}}{\partial \widetilde{Q}_{(s_0,a_0)}} \end{pmatrix} \overbrace{\begin{pmatrix} \frac{d\widehat{Q}_{(s_0,a_0)}}{d\widehat{Q}_{(s_k,a_k)}} & \frac{d\widehat{Q}_{(s_0,a_0)}}{d\widetilde{Q}_{(s_k,a_k)}} \\ \frac{d\widetilde{Q}_{(s_0,a_0)}}{d\widehat{Q}_{(s_k,a_k)}} & \frac{d\widetilde{Q}_{(s_0,a_0)}}{d\widetilde{Q}_{(s_k,a_k)}} \end{pmatrix}}^{J_{(s_0,a_0),(s_n,a_n)}=J_{(s_0,a_0),(s_k,a_k)}J_{(s_k,a_k),(s_n,a_n)}}}_{\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}} \underbrace{\begin{pmatrix} \frac{d\widehat{Q}_{(s_k,a_k)}}{d\widehat{Q}_{(s_n,a_n)}} & \frac{d\widehat{Q}_{(s_k,a_k)}}{d\widetilde{Q}_{(s_n,a_n)}} \\ \frac{d\widetilde{Q}_{(s_k,a_k)}}{d\widehat{Q}_{(s_n,a_n)}} & \frac{d\widetilde{Q}_{(s_k,a_k)}}{d\widetilde{Q}_{(s_n,a_n)}} \end{pmatrix} \begin{pmatrix} \Delta\widehat{Q}_{(s_n,a_n)} \\ \Delta\widetilde{Q}_{(s_n,a_n)} \end{pmatrix}}_{\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}} . \tag{4.82}$$

That is, $J_{(s_0,a_0),(s_n,a_n)}$ was split up into $J_{(s_0,a_0),(s_k,a_k)}$ that accumulates partial derivatives from the root node to (s_k, a_k) and $J_{(s_k,a_k),(s_n,a_n)}$ that accumulates partial derivatives from (s_k, a_k) to (s_n, a_n). A planning action $\mathfrak{a} = (s, a)$ affects $\widehat{Q}_{(s,a)}$, $\widetilde{Q}_{(s,a)}$ and, as a consequence, all $\widehat{Q}$, $\widetilde{Q}$ and their partial derivatives along the path to the root node. Let (s_k, a_k) be the lowest common ancestor of (s, a) and (s_n, a_n), then the planning action $\mathfrak{a}$ only affects $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ but not $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$. For each node updates can therefore be restricted to accumulating the Jacobian up to the lowest common ancestor, which lies on the path from (s, a) to the root node where backups need to be performed anyway. It would therefore be possible to perform forward accumulation of the corresponding Jacobians during the backup.

Partial Reverse Accumulation

A caveat with the approach of naive forward accumulation is that we have to start with a distinct initial value for each (s_n, a_n), which would be $J_{(s_k,a_k),(s_n,a_n)}$ for computing only the Jakobian or $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$ for directly computing the expected change. Effectively this requires to run multiple parallel forward accumulation passes for which we need to maintain a list of Jacobians that grows during the backup until it comprises all nodes in the tree when reaching the root node. A significant improvement can be achieved if instead of performing forward accumulation *during* the backup we perform reverse accumulation *after* the backup along the same trace from the root node back to (s, a). Let $\mathbf{d\bar{V}}_{(s_0,a_0),(s_k,a_k)}$ be the old value of $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ at (s_k, a_k) and $\mathbf{dV}^*_{(s_0,a_0),(s_k,a_k)}$ be the new value computed via reverse accumulation. The expected change of the objective for a planning action (s_n, a_n) can then be updated as

$$\Delta_{(s_n,a_n)}\mathcal{O}\big(\boldsymbol{\xi}[D]\big) \leftarrow \mathbf{dV}^*_{(s_0,a_0),(s_k,a_k)} \, \boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)} \tag{4.83}$$

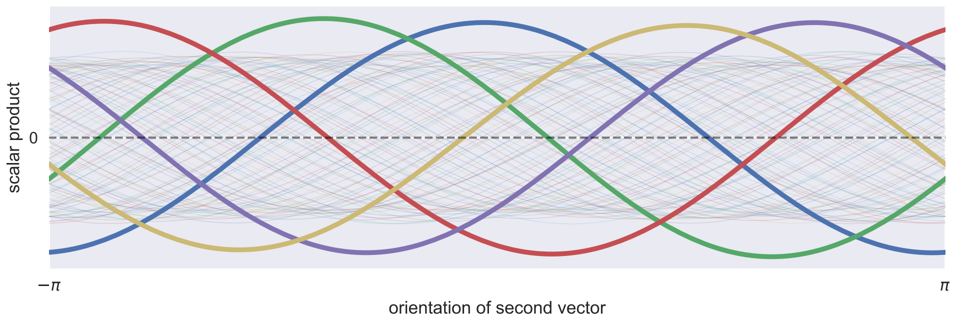


Figure 4.2: Set of vectors with minimal scalar product (4.89). The set may contain arbitrarily many vectors (thin plots) but vectors with a large norm will dominate (thick plots).

$$= \mathbf{dV}^*_{(s_0,a_0),(s_k,a_k)} \, [\mathbf{d\bar{V}}_{(s_0,a_0),(s_k,a_k)}]^{-1} \, \mathbf{d\bar{V}}_{(s_0,a_0),(s_k,a_k)} \, \boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)} \tag{4.84}$$

$$= \mathbf{dV}^*_{(s_0,a_0),(s_k,a_k)} \, [\mathbf{d\bar{V}}_{(s_0,a_0),(s_k,a_k)}]^{-1} \, \Delta_{(s_n,a_n)} \mathcal{O}\Big(\boldsymbol{\xi}[D]\Big) \; . \tag{4.85}$$

That is, the expected change of the objective can be updated by multiplying it by a scalar that is the product of the new value of $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ at the lowest common ancestor (s_k, a_k) and the inverse of its old value. The advantage of Eq. (4.85) as compared to Eq. (4.83) is that $\mathbf{d\bar{V}}_{(s_0,a_0),(s_k,a_k)}$ is a vector that can be stored in each node (s_k, a_k) whereas $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$ is a list of vectors for all *pairs* of nodes (s_n, a_n) and (s_k, a_k). The storage requirements for Eq. (4.83) are thus quadratic in the tree size, whereas that of Eq. (4.85) are linear. Moreover, reverse accumulation after the backup is more efficient than forward accumulation during the backup because we do not need to maintain a list of Jacobians. However, we still need to update all values in the tree.

Lazy Forward Accumulation

An approach that avoids updating all values in the tree is to perform partial forward accumulation in a lazy-evaluation fashion. Assume, for a moment, that $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ and $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$ were scalars instead of vectors. All nodes (s_n, a_n) in the same subtree of (s_k, a_k) share $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ but differ in $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$. When evaluating $\mathrm{argmin}_{(s_n,a_n)}$ for different planning actions $\mathfrak{a} = (s_n, a_n)$ we could thus take $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ out of the $\mathrm{argmin}_{(s_n,a_n)}$ and determine the minimum purely based on $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$. This would allow the following lazy-evaluation scheme: During backups we evaluate the $\mathrm{argmin}_{(s_n,a_n)}$ in each node, perform forward accumulation *only* for the minimum planning action, and store a pointer to the corresponding node. Unfortunately, $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ and $\boldsymbol{\Delta}_{(s_k,a_k),(s_n,a_n)}$ are vectors, which means that we can take the norm of $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}$ out of the $\mathrm{argmin}_{(s_n,a_n)}$

but not its orientation

$$\begin{aligned}
\mathfrak{a}^* &= \operatorname{argmin}_{(s_n,a_n)} \mathbf{dV}_{(s_0,a_0),(s_k,a_k)} \mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)} && (4.86)\\
&= \operatorname{argmin}_{(s_n,a_n)} \cos\left\{\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}, \mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}\right\} \left\|\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}\right\| \left\|\mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}\right\| && (4.87)\\
&= \left\|\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}\right\| \operatorname{argmin}_{(s_n,a_n)} \cos\left\{\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}, \mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}\right\} \left\|\mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}\right\| , && (4.88)
\end{aligned}$$

where $\cos\{\,\cdot\,,\,\cdot\,\}$ is the cosine of the angle between the two vectors. However, we can achieve a similar performance gain as in the scalar-valued case by excluding all $\mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}$ for which the scalar product $\mathbf{dV}_{(s_0,a_0),(s_k,a_k)}\mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}$ cannot become minimal. Let $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots\}$ be a set of vectors and

$$\operatorname{vecmin} V = \{\mathbf{v} \in V \mid \exists_{\mathbf{u}} \forall_{\mathbf{v}' \in V}\, \mathbf{uv} < \mathbf{uv}'\} \tag{4.89}$$

be a set function that returns the subset $\operatorname{vecmin} V \subseteq V$ of vectors $\mathbf{v}$ for which the scalar product $\mathbf{uv}$ may become minimal given an appropriate second vector $\mathbf{u}$. Instead of maintaining a single minimal value as in the scalar-valued case we may maintain a list of minimal vectors. While in pathological cases[20] this list may grow infinitely large, in most practical cases $\mathbf{\Delta}_{(s_k,a_k),(s_n,a_n)}$ will scatter to some extent in magnitude and the size of the list will be independent of the size of the tree (also confer the illustration in Figure 4.2). Furthermore, as the derivative of cos is bounded by ± 1, if we are willing to accept ϵ-sub-optimal planning actions we can always restrict the number of minimal vectors to at most $2\pi/\epsilon$ by confining the orientation of the second vector $\mathbf{u}$ in Eq. (4.89) to a grid with ϵ-spacing. The list of minimal vectors at the root node contains the fully accumulated $\mathbf{\Delta}_{(s_0,a_0),(s_n,a_n)}$ for all planning actions that may be optimal. In order to select the optimal planning action we need to traverse this list and choose the action that has a minimal scalar product with

$$\begin{aligned}
\mathbf{dV}_{(s_0,a_0),(s_0,a_0)} &= \begin{pmatrix} \frac{\partial \widetilde{V}_{s_0}}{\partial \widehat{Q}_{(s_0,a_0)}} & \frac{\partial \widetilde{V}_{s_0}}{\partial \widetilde{Q}_{(s_0,a_0)}} \end{pmatrix} && (4.90)\\
&= \begin{pmatrix} 0 & \pi^2_{(a_0\,|\,s_0)} \end{pmatrix} . && (4.91)
\end{aligned}$$

Using this strategy of *lazy forward accumulation* the order of complexity of ATS is independent of the tree size and equal to that of classical MCTS up to additive and multiplicative constants.

[20]An example of a pathological case would be a set of vectors with identical magnitude but pairwise distinct orientation.

4.2.6 Extensions and Challenges

There is a number of possible extensions of the ATS approach and challenges associated with the method as presented so far as well as with the possible extension.

Structural Changes

Structural changes, that is adding new nodes to the search tree, poses a challenge in several respects.

A structural change may result in a large increase of uncertainty that is hard to model when computing the expected change of our objective, all the more due to the linear approximation. This large increase of uncertainty comes from the fact that a structural change typically occurs when a new action is taken or a new outcome is observed, which means that there is only a single data point associated with the newly added node and the associated uncertainty may diverge. In the ATS implementation, for instance, I decided to use the sample variance (which is zero for a single data point) instead of the bias-corrected estimator of the population variance (which diverges for a single data point) to handle the variance of the reward in these cases. A theoretically more solid method would be to use a Bayesian approach in that case (also see Bayesian Modeling below). This, however, adds more degrees of freedom to ATS, which need to be adjusted appropriately.

Another practical aspect with regard to structural changes is the computation of the gradient. If there were no changes in structure the gradient could be automatically and efficiently computed for arbitrarily complex dependencies (also see Beyond Tree Structures) by once defining the structure and letting automatic differentiation tools figure out the rest.[21] A challenge therefore is to carry over efficiency of computation and ease of implementation to the case of changing structures.

Finally, sequences of structural changes lead to highly correlated effects that are not captured by a one-step greedy policy (also see Beyond One-Step Greedy Policies). A prominent example are rollouts from the leaf nodes that fulfill the crucial role of an initialization heuristic in MCTS.

Beyond One-Step Greedy Policies

In ATS, as in most active learning and RL problems, we are faced with the notorious problem that the full policy is intractable to compute but the one-step greedy policy is blind for correlated long-term effects of actions. In experiments (Section 4.2.7) these

[21]Note that the automatic differentiation tools of state-of-the-art libraries like Theano, TensorFlow etc. that were mainly developed in the wider context of deep learning provide highly optimized code for this case.

effects were most prominent in the case of structural changes but generally they are also relevant for collecting samples within the tree. In principle, the full range of approaches to tackle this problem can be attempted to be transferred to ATS. I will briefly discuss two of them, namely the idea of *optimism in the face of uncertainty* and the concept of *macro actions* (also confer Section 2.1.4).

The principle of *optimism in the face of uncertainty* is frequently implemented in terms of an *intrinsic reward* or an *exploration bonus*. In ATS this means to add an additional bias towards actions that may substantially impact (decrease or increase)[22] the variance in a way that is not captured by the one-step greedy policy. I observed some positive effects using this approach for capturing the influence of rollouts from the leaf nodes but a more thorough investigation is required to provide solid evidence.

The idea of *macro actions*, as also employed in hierarchical RL, is to capture the effect of a series of atomic actions by an abstract macro action. The challenge consists in first defining appropriate macro actions, which is highly problem-specific, and second modeling their effect, which may be non-trivial depending on how they are defined. An advantage in the case of ATS is that we are dealing with a meta-problem in the sense that the problem structure of ATS does not change even if the underlying planning domain changes. It might therefore be worthwhile to develop ATS-specific macro actions. Prominent candidates would be a "rollout macro action" for leaf node initialization or an "expansion macro action" to explicitly model changes of the tree structure.

Bayesian Modeling

The approach of ATS as presented in this work strives for the most straight-forward implementation of *planning as active learning* for MCTS. With this in mind, the goal was to introduce as few new parameters as possible to achieve a robust algorithm that provides a working proof of concept. Bayesian approaches, on the other hand, provide a flexible level of description but involve additional challenges such as choosing appropriate priors and solving the associated inference problems. In that sense Bayesian modeling represents an appropriate next step in the further development of ATS. Two examples where a Bayesian approach seems especially fruitful are structural changes, where the suggested Chinese restaurant process is only the first step, and more advanced reward models that appropriately deal with little data.

[22]Confer the idea of *expected model change* for greedy action selection mentioned in Section 2.1.3.

Computing Expected Changes

Due to the relatively simple model assumptions made in the present implementation of ATS computing expected changes of the variables was possible in closed form. As more advanced models and more complex dependencies between different variables are considered computing the expected changes becomes an increasing challenge. A major advantage of the linear decomposition is that expectations only involve the independent variables and not the entire objective. This substantially increases the freedom concerning the approaches that may be applied for computing expected changes. In addition to methods for exact inference approximate inference via Markov chain Monte-Carlo or variational methods seem to be within the bounds of practical feasibility.

Beyond Tree Structures

Using a tree structure for planning, in a strict sense, ignores the fact that states may be revisited. There are basically two ways to address this shortcoming in trial-based planning. One way is to depart from a tree structure altogether as is done, for instance, in *real-time dynamic programming* (Barto, Bradtke, and Singh, 1995; McMahan, Likhachev, and Gordon, 2005; Sanner et al., 2009) where finite trials from specific states are simulated, similar to MCTS, but backups are performed in the domain's transition graph rather than building up a search tree. Another approach is to maintain a global model for transition probabilities, expected rewards etc. and use the tree structure mainly to organize the data and to coordinate trials (see for instance Silver, Huang, et al., 2016).

The associated challenges for these two approaches are of a different kind. When using a global model the main challenge arises from the fact that the influence of a new sample is not anymore local to single nodes in the tree. While computing the expected change of variables does not necessarily become more difficult, changes of the value and thus of the gradient spread throughout the tree, which makes backups of the value and updates of the gradient much more involved. The challenge associated with non-tree structures is that the typically used dynamic programming updates need to be transferred to computations of the gradient. While chances are that the convergence criteria for Bellman updates of the value transfer easily to updates of the Jacobian, a corresponding proof would need to be provided.

Other Objectives and Backup-Policies

An aspect that is somewhat independent from the ones discussed so far is the role of the objective function and the policy used within the tree to backup values and compute the gradient. Due to the linear decomposition of the objective gradient and the expected change of variables the objective function and the backup-policy can be chosen largely independently of other properties such as the employed models. In the present implementation of ATS I used the value variance as objective, which requires backups of (at least) the first two moments of the value distribution, and facilitates the use of a variance-based backup-policy such as the employed optimistic soft-max policy. Another commonly used technique to assess uncertainty in planning is by computing confidence bounds (Kocsis and Szepesvári, 2006; McMahan, Likhachev, and Gordon, 2005; Toussaint and Lopes, 2017), which in turn requires backups of the bounds and facilitates backup-policies using these bounds (also confer the use of bounds in Chapter 5). Other combinations of objective functions and backup-policies are conceivable and present an interesting topic for further investigation.

Models without Reset

ATS in its present form assumes that a transition can be simulated from an arbitrary state of the environment. For complex domains with generative models that perform involved simulations it might not always be possible to "jump" to a given state of the environment, for instance, because the associated information required to restore the model's internal state may be immense. As for these models an efficient exploitation of simulated transition is particularly desirable it is worthwhile exploring the option to adapt ATS to models without reset. One obvious option is to resort to classical rollout-based tree search and use the objective of ATS to provide an additional exploration bonus of the kind used in Bayesian RL that encourages information gathering. A challenge in that respect is that the expected changes for different nodes along a rollout may be highly correlated (information collected in one node might devaluate the expected information gain of another) so that computing them separately might be misleading.

4.2.7 Experiments

To assess the practical applicability of the ATS approach we performed evaluations in a domain that exhibits a prototypical structure where actively sampling single transitions can be expected to be beneficial as compared to performing contiguous rollouts (Lieck and Toussaint, 2017). The results demonstrate that ATS can be combined with classical MCTS to produce superior performance. They also highlight the importance of structural changes and shortcomings of the one-step greedy policy pointing towards interesting topics for future investigations.

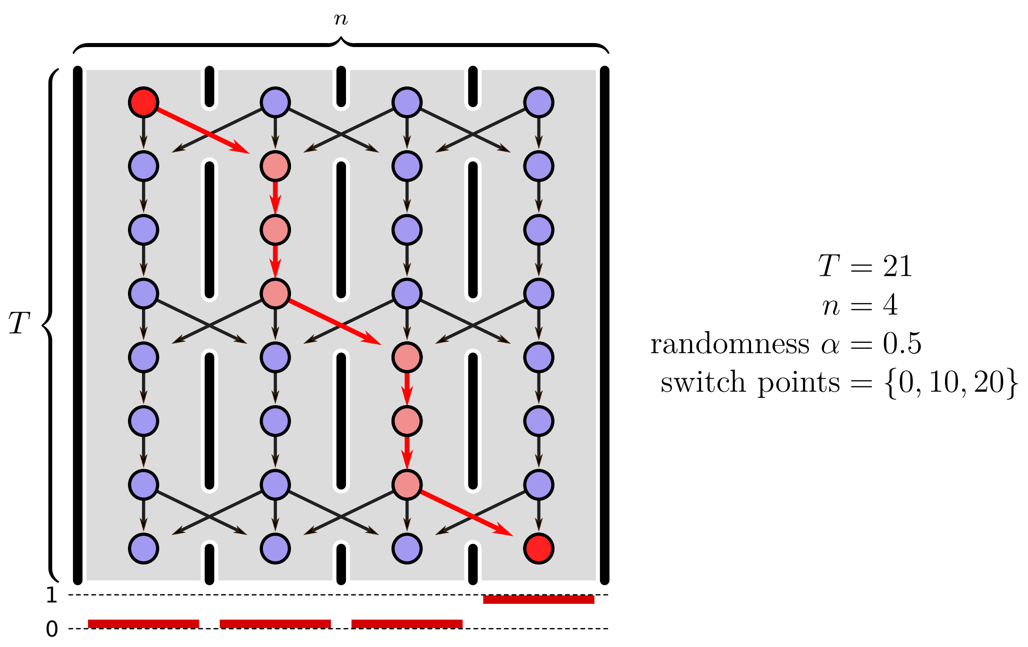


Figure 4.3: Highway Environment

Choosing an Environment

A primary motive for developing the ATS approach was the insight that a contiguous rollout may contain many samples that do not yield relevant information. An environment for demonstrating the applicability of ATS should therefore feature this characteristic. We designed a prototypical environment where the effective number of relevant decisions along a rollout is much smaller than the rollout length. In our *Highway* environment, depicted in Figure 4.3, the agent moves forward along fixed lanes for a number of T time steps. It may change lanes only at specific *switch points* being forced to stay in the same lane at all other times. At switch points changing the lane will fail with probability α directing the agent to the left, right, or the same lane with equal probability. The agent starts in the left-most lane and receives a reward only in the terminal state after T steps: If ending in the right-most lane the reward is uniformly distributed in $[0.9, 1]$ otherwise it is uniformly distributed in $[0, 0.1]$. The Highway environment that we used in our experiments had a depth of $T = 21$, $n = 4$ lanes, a randomness of $\alpha = 0.5$, and switch points at time steps $\{0, 10, 20\}$.

Structural Changes and One-Step Greedy Policy

To increase robustness while using a one-step greedy policy we followed the concept of *expected model change* (Section 2.1.3) and chose actions to maximize the expected change in variance instead of minimizing its expected value. The analysis of our experimental results below confirms that a temporary increase in variance is indeed closely connected to a good performance.

The fact that we are using a greedy policy is also relevant with respect to structural changes. In preliminary runs we observed ATS to be sensitive to the way to how structural changes were modeled. As described in Section 4.2.3 we modeled structural changes as a Chinese restaurant process. Large concentration parameters in the Chinese restaurant process imply a high probability of observing a new outcome and new outcomes initially have a high variance as they are sampled only once. Therefore, large concentration parameters lead to known actions being frequently resampled even if their action value estimates already have a low variance. Additionally, as discussed in Section 4.2.6, the one-step greedy policy does not capture the important effect of a rollout to a terminal state, which would require a coordinated sequence of planning actions. The attempt to account for this effect by distinguishing between terminal and non-terminal states when modeling structural changes lead to somewhat unstable behavior: Assigning a low probability to terminal states did not change the behavior while higher values led to excessive rollouts. This demonstrates that the concept of *optimism in the face of uncertainty* (in this case optimism with respect to reaching a terminal state with zero variance) has the potential to account for the effect of rollouts but requires further investigation.

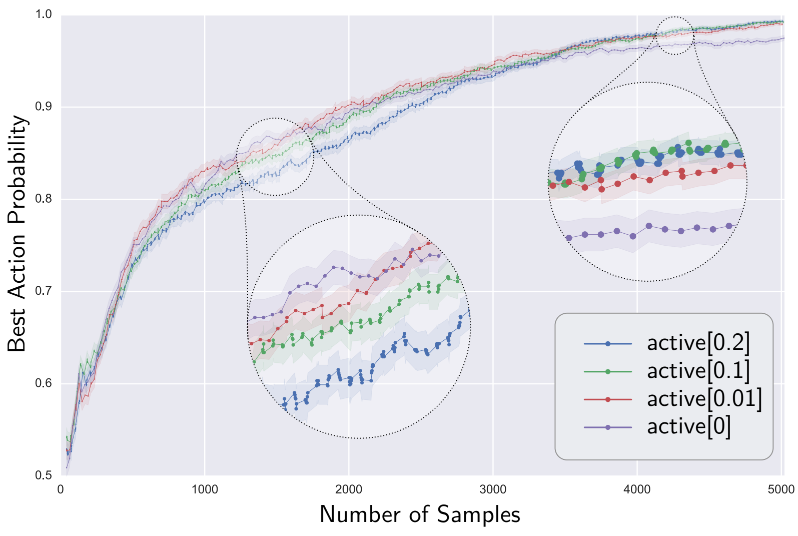
In the light of these ambiguous preliminary results we decided to ignore structural changes altogether by setting the concentration parameter to zero, which effectively drops the terms that account for structural changes in Eqs. (4.64) and (4.71). This simplified version of ATS does not actively build up the search tree and therefore has to be combined with classical rollout-based MCTS. We employed a mixing strategy where the ratio of active samples and rollout samples was kept constant: After each rollout the strategy active[χ] draws a number of active samples until they make up a χ-fraction of all samples. The strategy active[0] corresponds to plain UCT.

Results

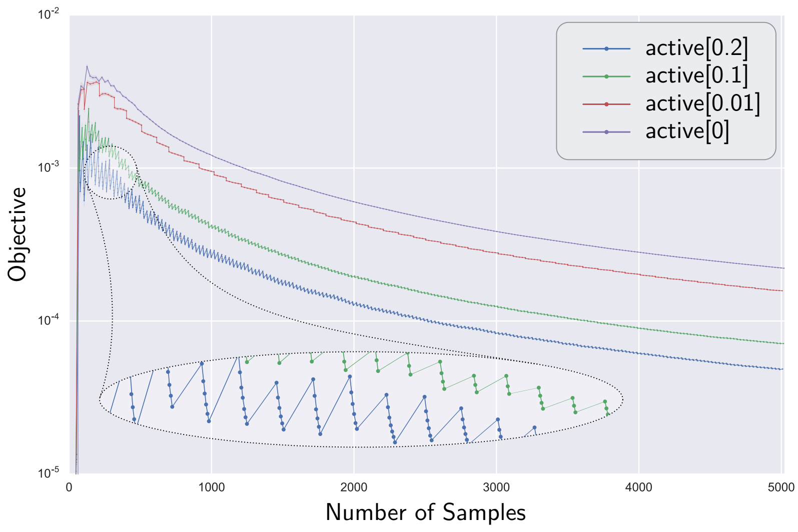
Figure 4.4 show the result for ≈ 2600 runs for mixing strategies with $\chi = \{0\%, 1\%, 10\%, 20\%\}$. The error bands correspond to one standard deviation of the mean estimator. Due to the fixed depth of the *Highway* environment all runs of the same method follow the same pattern of active samples and rollout samples, which allows a detailed analysis.

Figure 4.4a shows the performance of the different methods as measured by their probability of choosing the best action. There are two salient observation to make. First, during the initial planning phase (< 3000 samples), where large parts of the search tree are built up, including a large amount of active samples ($> 1\%$) may deteriorate performance. Second, in the final planning phase, where the probability of choosing the best action approaches 1, all active sampling strategies are on the same level and significantly outperform plain UCT. This is the case even for those strategies that fell behind in the initial planning phase. The mixed results for the initial planning phase confirm the fact that our reduced ATS method does not consider the relevance of structural changes. A larger fraction of active samples therefore delays relevant expansions. The superior performance of active sampling methods in the final planning phase, on the other hand, confirms that ATS reliably chooses relevant transitions for sampling within the tree. A low fraction of 1% of active samples does not significantly impair performance at the early stage but still leads to a significantly better performance at the final stage. The active[0.01] strategy is therefore globally equal or superior to plain UCT.

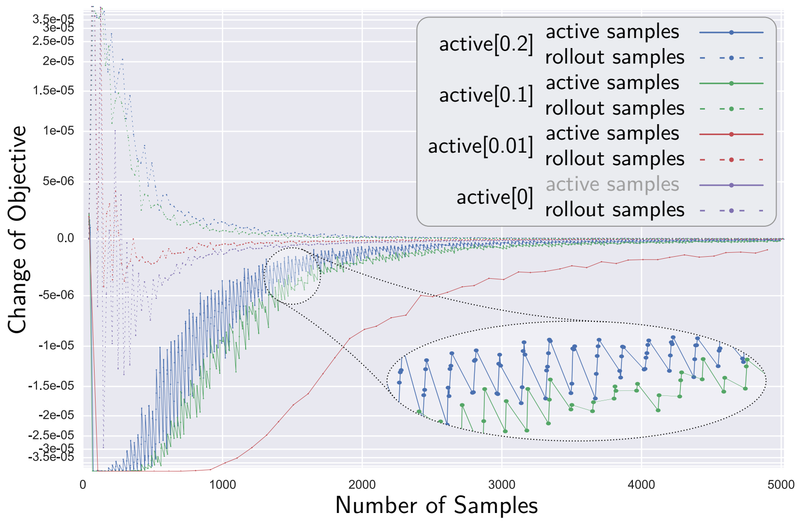
To gain a better understanding of the interplay between rollout-based samples and active samples we can take a closer look at the impact they have on the objective. Figure 4.4b shows the objective value as a function of the number of samples. These results clearly demonstrate that our active sampling strategy optimizes the objective more efficiently than rollout-based samples. This becomes apparent on the global time scale where methods with a higher fraction of active samples converge more quickly. But this can also be seen on the per-samples basis as showcased in the inset: Active samples successively minimize the objective while rollouts lead to an increase (except for active[0.01]), which produces a sawtooth-like pattern for mixed sampling strategies. The interplay of active samples and rollout-based samples becomes more apparent when plotting the change of the objective separately for both types, as done in Figure 4.4c. These results suggest that there is a competition between active samples and rollout samples (and even between successive active samples) on collecting variance-reducing information. On that score, our active learning approach that is specifically designed for that purpose proves superior to rollout-based sampling. In the final planing phase this coincides with a superior ability of identifying the best action, which suggests that striving for a small variance is a suitable overall objective. In the early planning phase, however, a low objective value is not an indicator for a good performance. It seems that in that phase the shortcomings of a myopic one-step greedy policy are particularly pronounced.



(a) Probability of choosing the best action as a function of the number of samples.



(b) Objective value as a function of the number of samples.



(c) Change of the objective value as a function of the number of samples.

Figure 4.4: Results for active tree search with different mixing strategies in the *Highway* environment.

Chapter 5

Planning in Feature Space

In this section I describe how the different components developed so far can effectively be put to work. More precisely, I describe how the structured models developed in Chapter 3 and the approach of active planning introduced in Section 4.1 can be combined and together allow to actively adapt the employed abstractions by performing planning actions in the model's feature space. There are three essential components involved:

1. A model that allows for adaptable abstractions, such as the kind of models developed in Chapter 3.

2. A representation that the planning process is carried out on. This representation has to provide the means to simplify computations of the value based on the abstractions provided by the model. I use *abstract search trees* (ASTs), which generalize classical forward search trees and are described in Section 5.1.

3. A planning algorithm, that is, a process that adapts the model's abstraction and the representation consistently and ensures that the computed values converge to the true value. In Section 5.2 I describe how the principle of active planning from Chapter 4 can be employed for planning in feature space.

This sections establishes the missing link for bridging the gap between classical planning approaches and data-based models by showing how planning in abstract spaces is possible on the basis of learned models. A thorough empirical investigation was beyond the scope of this thesis, however, the presented results demonstrate, I think, the feasibility of the approach and its potential for future developments.

5.1 Abstract Search Trees

In this section I describe *abstract search trees* (ASTs), which serve as the representation to perform active planning with feature-based models. A representation for planning has to employ the abstraction capabilities of the model to simplify computations of the value. To make this connection, I first describe abstraction on an overarching level that connects to both the arithmetics of computing values and the set theoretic structure of abstractions (Section 5.1.1). I then define ASTs (Section 5.1.2) and describe how they can be manipulated during planning (Section 5.1.3). Finally, I draw the connection to feature-based models by describing how the corresponding abstractions can be represented in ASTs (Section 5.1.4).

5.1.1 Abstraction as Trajectory Clustering

Computing action values in the general non-Markov case corresponds to an infinite nested sum-product over all possible trajectories

$$Q^{\pi}_{(h_0,a_1)} = \sum_{o_1} p_{(o_1\,|\,a_1,h_0)} \left(r_{(h_0,a_1,o_1)} + \gamma \sum_{a_2} \pi_{(a_2\,|\,h_1)}\, Q^{\pi}_{(h_1,a_2)} \right) \tag{5.1}$$

$$= \sum_{h} \left[\prod_{t'=0}^{\infty} p_{(o_{t'+1}\,|\,a_{t'+1},h_{t'})}\, \pi_{(a_{t'+2}\,|\,h_{t'+1})} \right] \sum_{t=0}^{\infty} \gamma^{t}\, r_{(h_t,a_{t+1},o_{t+1})} \;, \tag{5.2}$$

where h runs over all trajectories with prefix h_0 and $h_t = \{\ldots, a_1, o_1, \ldots, a_t, o_t\}$ is the history up to point t. Performing these computations explicitly is infeasible and the task in planning consists in coming up with a tractable way of computing (an approximation of) Eq. (5.2).

In principle any invariance of the model allows to simplify computations: If the model is invariant for a specific set of trajectories all these trajectories contribute equally to the value and the corresponding computations need to be performed only once. Applying an abstraction corresponds to assuming additional invariances that allow for further simplification at the cost of a coarser approximation. Additionally, the recursive structure of Eq. (5.2) allows to split trajectories and rearrange terms

$$Q^{\pi}_{(h_0,a_1)} = \sum_{h} \left[\prod_{t'=0}^{\infty} p_{(o_{t'+1}\,|\,a_{t'+1},h_{t'})}\, \pi_{(a_{t'+2}\,|\,h_{t'+1})} \right] \sum_{t=0}^{\infty} \gamma^{t}\, r_{(h_t,a_{t+1},o_{t+1})} \tag{$\rightarrow$ 5.2}$$

$$= \sum_{h} \left[\prod_{t'=0}^{T} p_{(o_{t'+1}\,|\,a_{t'+1},h_{t'})}\, \pi_{(a_{t'+2}\,|\,h_{t'+1})} \right] \left[\sum_{t=0}^{T} \gamma^{t}\, r_{(h_t,a_{t+1},o_{t+1})} + \ldots \right. \tag{5.3}$$

$$\left. \ldots + \sum_{\bar{h}} \left[\prod_{\bar{t}'=0}^{\infty} p_{(o_{\bar{t}'+1}\,|\,a_{\bar{t}'+1},\bar{h}_{\bar{t}'})}\, \pi_{(a_{\bar{t}'+2}\,|\,\bar{h}_{\bar{t}'+1})} \right] \sum_{\bar{t}=0}^{\infty} \gamma^{t+\bar{t}}\, r_{(\bar{h}_{\bar{t}},a_{\bar{t}+1},o_{\bar{t}+1})} \right] , \tag{5.4}$$

where h runs over all length-T trajectories with prefix h_0, $\bar{h}$ runs over all trajectories with prefix h, and time indexing starts at the end of the fixed prefix. This splitting and clustering of trajectories may occur in an interleaved manner. For instance, one subset of trajectories may be split at a specific point and clustered in a certain way while the remaining trajectories may be split or clustered in different ways. A representation for planning has to provide ways of maintaining and adapting this nested clustering of trajectories. In the following I describe how this is realized by ASTs.

5.1.2 Definition of Abstract Search Trees

Let $\mathcal{A}$ be the action space, $\mathcal{O}$ be the observation space, and $\mathcal{F}$ be a set of binary features of action-observation sequences. Then an *abstract search tree* (AST) is a directed acyclic graph with a single source node, called the root, where each node v is associated with

- a subset of actions $\mathcal{A}_v \subseteq \mathcal{A}$
- a subset of observations $\mathcal{O}_v \subseteq \mathcal{O}$
- a subset of features $\mathcal{F}_v \subseteq \mathcal{F}$, called the *active* features
- fixed values for the active features and
- an integer $N_v \geq 0$, called the node's (maximum) *cyclicity*, which may be infinite.

Each node n represents a set of trajectories $\mathcal{H}_v$, where the root node represents the current history and for all other nodes

$$\mathcal{H}_v := \left\{ h' \oplus h \,|\, h' \in \mathcal{H}_{v'}, v' \in \mathcal{N}^{-}(v), h \in (\mathcal{A}_v \times \mathcal{O}_v)^n, 0 < n \leq N_v + 1 \right\} , \tag{5.5}$$

where $\cdot \oplus \cdot$ denotes the concatenation of two trajectories and $\mathcal{N}^{-}(v)$ is the set of direct predecessors of v. That is, the set of trajectories represented by a node consists of all trajectories that can be constructed by taking a trajectory from one of the node's direct predecessors and appending a trajectory constructed from the node's action and observation set while respecting the node's cyclicity, which indicates the maximum number of *additional* steps it may represent.

- An AST is *consistent* if

  1. no trajectory is represented by more than one node (*trajectory-consistent*) and
  2. any active feature in a node evaluates to the corresponding fixed value for all represented trajectories (*feature-consistent*).

- An AST is *complete* if

  1. any trajectory that has the current history as prefix is represented by a node (*trajectory-complete*) and

2. all features that take a unique value for all trajectories represented by a node are active (*feature-complete*).

- The *plan* represented by a consistent AST is the set of all represented trajectories together with the active features and their values in every step of every trajectory. A plan is *refined* by activating one or more additional features in some of the trajectories.

- A consistent AST is *minimal* if it is not possible to merge any nodes without making the AST inconsistent or changing its plan.

Computations of the value in ASTs are performed analogously to classical search trees with two slight modifications:

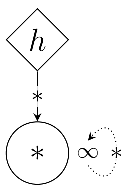
1. The iteration over successor nodes in ASTs corresponds to summing over sets of actions and observations as opposed to single actions and observations as in classical search trees.

2. The value updates for nodes with non-zero cyclicity has to be replaced by Eq. (5.16) for semi-Markov abstractions as detailed in Section 5.1.4.

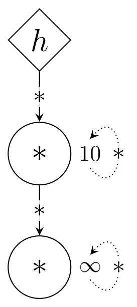
5.1.3 Manipulating Abstract Search Trees

During planning with ASTs we seek to maintain a complete, consistent, minimal AST while successively refining its plan. The smallest complete AST is depicted in Figure 5.1a and consists of a single node (in addition to the root node) that represents all possible trajectories. This AST represents the empty plan for the current history where only features describing the marginal reward distribution conditional on the current history are active. The following operations on an AST allow to manipulate its plan and preserve or reestablish its property of being consistent, complete, and minimal.

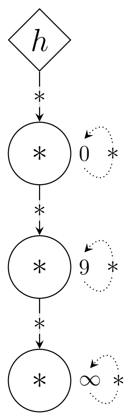
Splitting/Merging Cycles A node with non-zero cyclicity may be split into two nodes, one being connected to all predecessors and one being connected to all successors. The two nodes are connected by a single arc and must have the same actions, observations, active features, and feature values. From a node with infinite cyclicity nodes with arbitrary cyclicity may be split off (Figure 5.1b), otherwise the sum of their cyclicities has to equal that of the original node minus one (Figure 5.1c). The inverse operation consists in merging two such nodes. Splitting a cycle without adding active features produces a non-minimal AST.



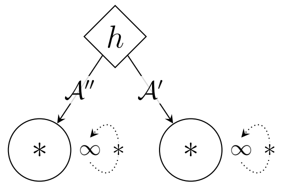
(a) Smallest complete AST.



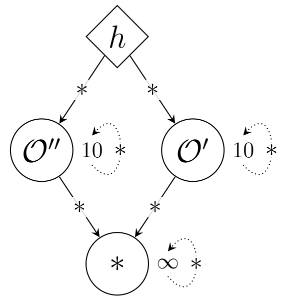
(b) Splitting infinite cycles.



(c) Splitting finite cycles.



(d) Splitting actions.



(e) Splitting observations.

Figure 5.1: Abstract Search Trees: The root is depicted as a diamond-shaped node; each node is labeled with its observation set (expect for the root, which is labeled with the current history); its action set is indicated on the incoming arcs; its cyclicity is indicated next to it; an asterisk ($*$) indicates the complete action and observation set, respectively.

Splitting/Merging Actions and Observations A node may be split into two nodes both being connected to all predecessors and successors. The two nodes must have the same active features and feature values, and complementary actions and observations. That is, the corresponding sets of action-observation pairs have to be mutually exclusive and jointly match that of the original node. Figure 5.1d shows an example of splitting only the action sets, Figure 5.1e one of splitting only the observation set. The inverse operation consists in merging two such nodes. Splitting actions or observation without adding active features produces a non-minimal AST.

Adding/Removing Active Features A node's set of active features may be changed by adding or removing one or more features with a specific value. Generally, not all features may be added while retaining consistency. It may be possible for a non-minimal AST to become minimal after adding features (but this is not necessarily possible) and vice versa.

The procedure for manipulating ASTs used for planning in feature space consists of the following steps:

1. select a node
2. (optionally) split the node's cycle and select one of the resulting nodes instead
3. select a feature to be activated in that node
4. split the action and observation sets of the node and all its predecessors such that adding the respective feature is possible in a consistent way
5. complete the feature sets of all manipulated nodes by activating all features with a unique value.

The first three steps require some criterion for selecting the corresponding node and feature, which is discussed in more detail in Section 5.2. For the last step it is necessary to check all inactive features for consistency, which in the worst case may require to compute their value for all trajectories represented by the respective node. The intermediate step – adapting the AST's structure to retain consistency – may be computationally expensive for general non-Markov features as it may require iterating over the complete set of represented trajectories. However, for conjunctive features, such as the temporally extended features described in Section 3.3.3, this step can be performed efficiently because each condition can be verified locally: Only if all relevant nodes along a specific path can fulfill the local condition we need to split off the respective trajectory bundle, which adds at most k new nodes to the AST for a conjunction reaching k steps into the past. Discussing how to perform this step efficiently for other types of non-Markov features is beyond the scope of this work.

5.1.4 Feature-Based Abstractions

Planning in feature space employs the kind of feature-based models developed in Chapter 3. The strength of these models in conjunction with non-Markov features is their flexibility in representing adaptable abstractions. It is important to understand how these different kinds of abstraction may lead to computational simplifications. For feature-based models we can rewrite Eq. (5.2) to explicitly include the feature set $\mathcal{F}$ in the observation and reward model

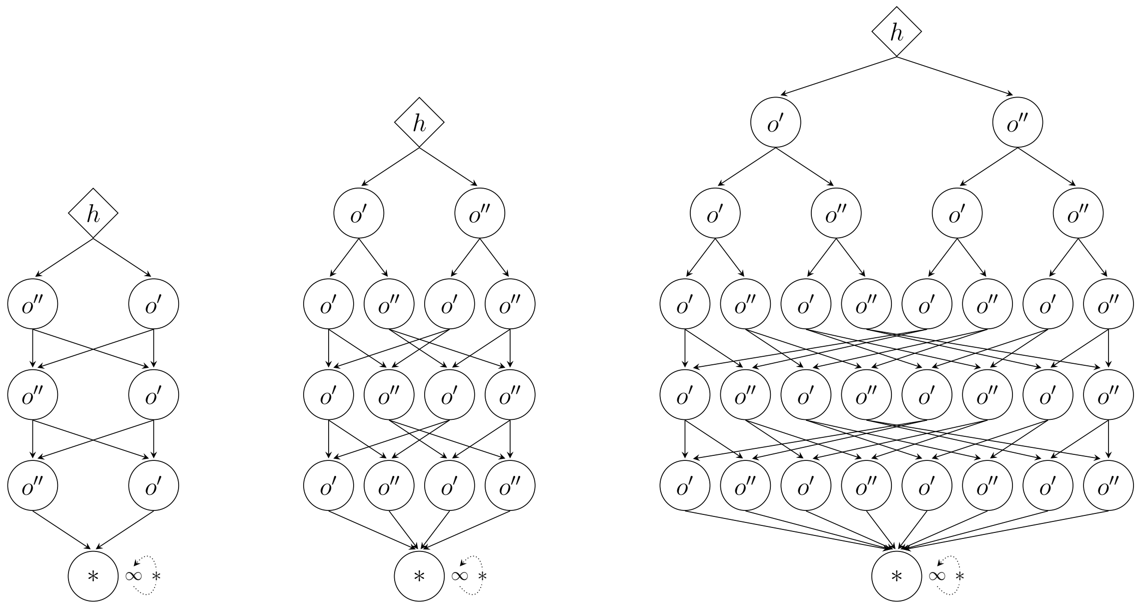
$$Q^{\pi}_{(h_0,a_1)} = \sum_{h} \left[\prod_{t'=0}^{\infty} p_{\mathcal{F}}(o_{t'+1} \,|\, a_{t'+1}, h_{t'})\, \pi_{(a_{t'+2} \,|\, h_{t'+1})} \right] \sum_{t=0}^{\infty} \gamma^t\, r_{\mathcal{F}}(h_t, a_{t+1}, o_{t+1}) \;, \qquad (5.6)$$

where at a specific time in a specific trajectory I assume $p_{\mathcal{F}}$ and $r_{\mathcal{F}}$ to depend on the same feature set. Abstractions are introduced by using a subset of the full feature set so that the model becomes invariant with respect to certain properties of trajectories. How to appropriately choose these subsets is described in Section 5.2. In the following I describe different kinds of abstractions, how they are represented in ASTs, and how they translate to computational simplifications.

Action-Observation Abstraction

Features may be invariant with respect to some actions or observations at a specific point in time. If this is the case we may split trajectories at that point and reduce the corresponding sums to only run over invariant sets of actions and/or observations. If, for instance, there are N clusters of observations for which the model takes distinct values letting the sum run over these clusters instead of all observations reduces the number of computations by a factor of $\frac{|\mathcal{O}|}{N}$. In ASTs each of these clusters is represented as a single node so that the value is computed by summing over all successors of a given node, just as in classical search trees.

We might want to do the same if the model is invariant with respect to sub-sequences of two or more steps – we could split trajectories before and after these sub-sequences and iterate only over the respective clusters of trajectories. However, as opposed to the single-step case the model does not readily yield the respective probabilities for the multi-step case and, in general, computing these probabilities involves additional computational effort. Below I discuss the special case of semi-Markov abstractions, where the probabilities for trajectories of different length can be computed in closed form.



(a) Markov **(b)** 2-Markov **(c)** 3-Markov

Figure 5.2: Abstract search trees to exploit first, second, and third order Markov invariances of the model for computing values. To keep the illustration clear there are only two possible observations o' and o'', and a single action that is not explicitly indicated.

k-Markov Features

An important special case are Markov and k-Markov features that only depend on the last k actions and observations in a trajectory. This means that trajectories that do not differ within the last k steps can be grouped at any given time step and we effectively only need to iterate over $(|\mathcal{A}||\mathcal{O}|)^k$ clusters of trajectories. Rewriting Eq. (5.2) to represent this structure with variable k requires a notation to specify iteration over the corresponding clusters and is omitted because it does not provide further insights. In contrast, a representation in terms of ASTs is straight forward and readily provides the desired computational simplifications. In Figure 5.2 this is illustrated for three steps of first, second, and third order Markov invariances. In practice the corresponding structures are built up as required when activating new features and may be freely intermingled. The fact that in ASTs, as opposed to classical search trees, trajectories may be reunited allows to exploit k-Markov dependencies. Performing a pass through the AST to compute the value offers the same computational savings as dynamic programming on a k-Markov representation with the important difference that in ASTs k may vary throughout the computation.

Non-Markov Features

Non-Markov features may depend on arbitrary properties of a trajectory. Due to this general characteristic it is not possible to specify concrete principles to simplify computations. However, the employed representation has to provide means to incorporate these features. As mentioned above for arbitrary features this may require complex manipulations to the AST and I restrict the discussion to conjunctive features.

Semi-Markov Abstraction

Semi-Markov abstractions are the kind of temporal abstraction that lies at the core of hierarchical RL. In a semi-Markov abstraction the environment is described in terms of an abstract state that fulfills the Markov property, except for the fact that the agent may stay within an abstract state for a variable number of steps. The standard approach taken in the options framework is to encapsulate this variable temporal extent by using a *multi-time model* (Richard S. Sutton, Precup, and Singh, 1999) that combines transition probabilities and discounts. While the options framework requires to define macro actions (initiation set, termination conditions, and their policies) beforehand I will take a slightly different approach that allows to perform semi-Markov abstraction on-the-fly during the planning process. This is possible whenever for a specific assignment of feature values the probability of keeping theses values only depends on the next action and observation and subsequent features do not depend on the exact extent of this condition. More formally, a sufficient condition for semi-Markov abstraction is as follows.

Condition 5 (Semi-Markov Abstraction). *Let $\mathcal{H}$ be a set of prefix trajectories, $\mathcal{F}$ be a set of features and $N \geq 0$ a maximum trajectory length, then a semi-Markov abstraction is possible if*

$$\exists_{\mathcal{A}' \subseteq \mathcal{A}} \, \exists_{\mathcal{O}' \subseteq \mathcal{O}} \, \forall_{h \in \mathcal{H}} \, \forall_{n \leq N} : \tag{5.7}$$

$$\bigwedge \begin{array}{lcr} h^* \in (\mathcal{A}' \times \mathcal{O}')^n & \Rightarrow & \forall_{f \in \mathcal{F}} \; f(h \oplus h^*) = f(h) \quad (5.8) \\ h^* \in (\mathcal{A}' \times \mathcal{O}')^n, a \in \mathcal{A}, o \in \mathcal{O} & \Rightarrow & V_{h \oplus (a,o)} = V_{h \oplus h^* \oplus (a,o)} \quad (5.9) \end{array}$$

where $\cdot \oplus \cdot$ denotes the concatenation of the two trajectories.

That is, there have to exist subsets of actions $\mathcal{A}'$ and observations $\mathcal{O}'$, such that appending trajectories of length $n \leq N$ to any existing trajectory prefix from $\mathcal{H}$ does not change the feature values as long as the actions and observations are taken from the respective subsets and additionally it does not affect subsequent values. Note that $h^* \in (\mathcal{A}' \times \mathcal{O}')^n$ is a sufficient but not a necessary condition for the features to keep their values. This means that the features are still allowed to keep their values if it is not fulfilled and $\mathcal{A}'$ and $\mathcal{O}'$ may be chosen conservatively to make a semi-Markov abstraction possible. The prefix trajectories together with the feature values can be considered an abstract semi-Markov state that the agent spends up to N steps in. An illustrative situation where this condition is fulfilled would be waiting for a specific event to occur or performing the same action over an extended period of time. If Condition 5 is fulfilled we can compute the value of this abstract state in closed form. Let $\mathcal{A}'$ and $\mathcal{O}'$ be as in Condition 5 and $V_h^{(N)}$ be the sought value of the abstract state then

$$\begin{aligned}
V_h^{(N)} &= \Bigg[\sum_{t=0}^{N} \underbrace{\Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{(o'\,|\,a',h)}\Big]}_{\xi_h}^{t} \dots && (5.10)\\
&\qquad \left.\begin{aligned}
&\dots \Big[\sum_{a\notin\mathcal{A}'} \pi_{(a\,|\,h)} \sum_{o\notin\mathcal{O}'} p_{(o\,|\,a,h)} \Big(r_{(h\oplus h^*,a,o)} + \gamma\, V_{h\oplus h^*\oplus(a,o)}\Big) + \dots \\
&\dots + \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{(o'\,|\,a',h)}\, r_{(h\oplus h^*,a',o')}\Big]\Bigg] + \dots
\end{aligned}\right\}\psi_h && \begin{aligned}(5.11)\\(5.12)\end{aligned}\\
&\qquad \dots + \underbrace{\Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{(o'\,|\,a',h)}\Big]^{N+1} V_{h\oplus h^*}}_{\zeta_h} && (5.13)\\
&= \zeta_h + \sum_{t=0}^{N} \psi_h \xi_h^t && (5.14)\\
&= \zeta_h + \psi_h \frac{1-\xi_h^{N+1}}{1-\xi_h} && (5.15)\\
&= \Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{(o'\,|\,a',h)}\Big]^{N+1} V_{h\oplus h^*} + \dots \\
&\qquad \dots + \Big[\sum_{a} \pi_{(a\,|\,h)} \sum_{o} p_{(o\,|\,a,h)} \Big(r_{(h\oplus h^*,a,o)} + [\![a\notin\mathcal{A}' \land o\notin\mathcal{O}']\!] \gamma\, V_{h\oplus h^*\oplus(a,o)}\Big)\Big] \dots \\
&\qquad \dots \frac{1-\Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{(o'\,|\,a',h)}\Big]^{N+1}}{1-\Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{(o'\,|\,a',h)}\Big]}\,, && (5.16)
\end{aligned}$$

where h^* denotes the variable-length part of the trajectory; (5.10) is the (discounted) probability of staying in the abstract state for t steps; (5.11) captures the value of leaving the abstract state; (5.12) captures the reward collected when staying in the abstract state; and (5.13) is the value of leaving the abstract state because of reaching the maximum number of steps N. Using the indicated abbreviations it becomes obvious that this corresponds to a geometric series with an additive term (5.14), which can be written in closed form (5.15). The full update equation is given by Eq. (5.16). In ASTs Condition 5 is fulfilled in all nodes with non-zero cyclicity where the usual value update has to be replaced by Eq. (5.16) and N corresponds to the node's cyclicity. As for $N = 0$ Eq. (5.16) reduces to the case of zero cyclicity it can generally be used for all nodes.

Options with a deterministic termination function β can be incorporated into this formalism by using β as the only feature in $\mathcal{F}$ and fixing π to the option's policy. To incorporate options with stochastic β the observation space has to be augmented with a binary factor that indicates termination of the option (similar to the approach taken in MAXQ where the state space is augmented with the list of active macro actions). The geometric series occurring in the computation of $V_h^{(N)}$ is the equivalent of the geometric series occurring in the multi-time models for options. A fundamental difference between options as they are used in hierarchical RL and the approach presented here is the possibility of interleaving macro actions: As there is no fixed hierarchy the different macro actions may terminate independently.

The extreme case of $N = 0$ corresponds to ignoring the semi-Markov abstraction while $N \to \infty$ corresponds to removing the maximum-length constraint. If the probability of staying in the abstract state ξ_h/γ goes to zero the values for different N converge $\lim_{\xi_h/\gamma \to 0} V_h^{(N)} = V_h^{(0)} = V_h$. If, on the other hand, ξ_h/γ goes to one (and $N = \infty$) the agent remains in the abstract state infinitely long, which corresponds to a value of $\frac{1}{1-\gamma} \sum_a \pi_{(a\,|\,h)} \sum_o p_{(o\,|\,a,h)} r_{(h,a,o)}$. This case is particularly relevant as it is used in the leaf nodes of an AST to represent the stationary distribution for a given policy.

Similar to the case of action-observation abstraction we may want to perform semi-Markov abstraction for models that depend on actions and observations that lie more than one step in the past. And again, while being possible in principle, the challenge consists in representing the respective clusters of trajectories and computing the corresponding transition probabilities, which is a task that I shall not further address in this work.

Parallel Decomposition

Parallel decomposition consists in applying two different abstractions to the *same* set of trajectories independently and combining the respective results afterwards. In this sense parallel decomposition is of a fundamentally different nature than the abstractions discussed so far. A prerequisite for parallel decomposition is the ability to specify different properties of a trajectory independently, such as specifying what the right arm does while ignoring the left and simultaneously specifying what the left arm does while ignoring the right. Feature-based models provide full parallel decomposition capabilities including the possibility to use an abstraction that ignores interactions in order to perform an approximate parallel decomposition. For instance, one subset of features may capture properties of the right arm, a second subset of feature properties of the left arm, and a third subset interactions between the right and the left arm. Ignoring features from the third set allows to perform an approximate parallel decomposition (by only using features from the first set for the right arm and features from the second set for the left arm), which would not be possible using the exact model.

A challenge with parallel decompositions consists in the fact that they do not correspond to a single way of clustering the trajectory space but rather to multiple clusterings at the same time. These clusterings are compatible in the sense that the respective features do not conflict but they are mutually exclusive in the sense that neither one is a refinement of the other. While I expect the approach of planning in feature space presented in this chapter to be extendable to parallel decomposition (also see Section 5.3.3), addressing this issue is beyond the scope of this work.

Partial Orders

Partial orders are a special case of a non-Markov property of a trajectory. The representation of partial orders in a feature-based model is straight-forward and the corresponding abstraction (ignoring the ordering) is part of the abstractions representable by feature-based models. As mentioned above, any representation for planning has to be able to represent such non-Markov properties. However, it may not be able to exploit the computational simplifications that are possible based on the corresponding abstraction. As before the challenge consists in representing and iterating over the respective clusters of trajectories, which I shall not further discuss here.

5.2 Active Planning with Feature-Based Models and Abstract Search Trees

Abstract search trees are capable of representing plans in diverse ways incorporating a wide range of abstractions provided by feature-based models. As discussed above, employing abstractions makes computing the value more efficient but the result is only an approximation of the true value. We therefore seek a procedure to refine a plan by activating features in specific nodes such that the approximation converges as quickly as possible. In the following I describe how this task can be tackled as an active planning problem. The crucial step in doing so consists in defining and computing the corresponding uncertainty measure.

5.2.1 Quantifying Uncertainty

To apply the active planning approach described in Section 4.1 it is necessary to quantify the uncertainty associated with the current internal state $\boldsymbol{\xi}$ of the planner (the AST in the present case) and estimate the reduction in uncertainty associated with a specific change of $\boldsymbol{\xi}$ induced by adding a specific feature in a specific node. As before, the eventual goal is to choose the optimal action a^*. For planning in feature space we rewrite Eq. (4.10) in terms of the history instead of the state and quantify the uncertainty of a^* as

$$p(a^* \,|\, h, \boldsymbol{\xi}) = \int p(\mathbf{Q}_h \,|\, \boldsymbol{\xi}) \, [\![\forall_{a \in \mathcal{A}} \, Q_{(h,a)} \leq Q_{(h,a^*)}]\!] \, d\mathbf{Q}_h \ , \tag{5.17}$$

that is, we have to determine the action value distribution $p(\mathbf{Q}_h \,|\, \boldsymbol{\xi})$ for the current history h. The uncertainty in $p(\mathbf{Q}_h \,|\, \boldsymbol{\xi})$ originates from the applied abstractions, that is, from the inactive features in the AST. As detailed below, we can make use of the known weights of the inactive features to bound $\mathbf{Q}_h$ and choose $p(\mathbf{Q}_h \,|\, \boldsymbol{\xi})$ to be the maximum entropy distribution with the given bounds, that is, we assume $\mathbf{Q}_h$ to be uniformly distributed within the bounds. The bounds of $\mathbf{Q}_h$ can be described as a set of linear constraints and $p(a^* \,|\, s, \boldsymbol{\xi})$ can be computed exactly by determining the volume of the respective polytopes, which allows us to use the entropy

$$\mathrm{H}(a^* \,|\, \boldsymbol{\xi}) \tag{5.18}$$

as our uncertainty measure.

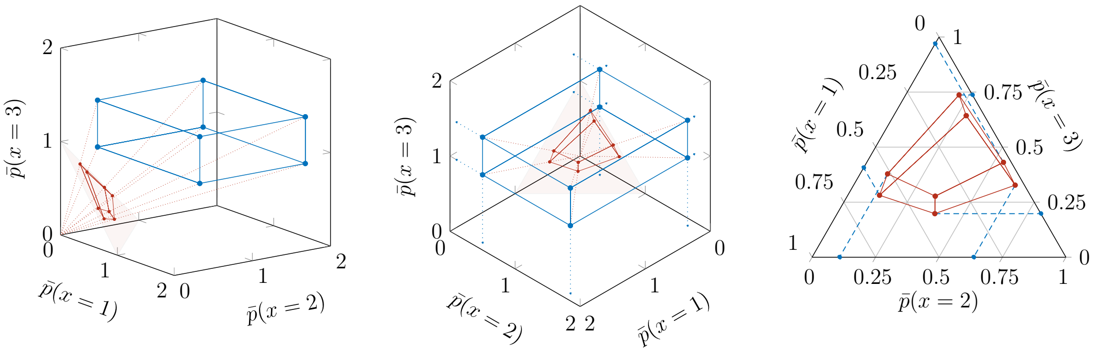
5.2.2 Computing Model Bounds

We can compute value bounds by first computing model bounds locally in each node and then propagating bounds through the tree while computing the value. Let $\mathcal{F}$ be the full feature set, $\widetilde{\mathcal{F}}$ be the set of active features, and consider the log-linear model

$$p_{\mathcal{F}}(o \,|\, a, h; \boldsymbol{\theta}) = \frac{\bar{p}_{\mathcal{F}}(o \,|\, a, h; \boldsymbol{\theta})}{\sum_{o'} \bar{p}_{\mathcal{F}}(o' \,|\, a, h; \boldsymbol{\theta})} \tag{$\rightarrow$ 3.24}$$

$$\bar{p}_{\mathcal{F}}(o \,|\, a, h; \boldsymbol{\theta}) = \exp \sum_{f \in \mathcal{F}} \theta_f f(o, a, h) \ , \tag{5.19}$$

where $\bar{p}_{\mathcal{F}}(o \,|\, a, h; \boldsymbol{\theta})$ is the unnormalized model. When employing the reduced feature set $\widetilde{\mathcal{F}}$ we can compute upper and lower bounds $[\,\cdot\,]^{\pm}$ of the unnormalized model $\bar{p}_{\widetilde{\mathcal{F}}}(o \,|\, a, h; \boldsymbol{\theta})$



(a) View from the side. **(b)** View along (1,1,1). **(c)** Barycentric plot.

Figure 5.3: Distortion of the box constraints of the unnormalized model $\bar p_{\mathcal F}(\mathbf x;\boldsymbol\theta)$ (blue) by the normalization (red). The new box constraints of the normalized model $\bar p_{\mathcal F}(\mathbf x;\boldsymbol\theta)$ are indicated in 5.3c.

for each observation o

$$\left[\bar p_{\widetilde{\mathcal F}}(o\,|\,a,h;\boldsymbol\theta)\right]^{\pm} = \exp\left[\sum_{f\in\widetilde{\mathcal F}}\theta_f f(o,a,h) + \sum_{f\in\mathcal F\setminus\widetilde{\mathcal F}}\theta_f\,[\![\theta_f \gtrless 0]\!]\right]. \tag{5.20}$$

As illustrated in Figure 5.3 box constraints of an unnormalized model are distorted by the normalization. For recovering box constraints after the normalization we have to choose the normalization constant $\sum_{o'}\bar p_{\widetilde{\mathcal F}}(o'\,|\,a,h;\boldsymbol\theta)$ with reversed bounds, that is, minimal within the given constraints to recover upper bounds and maximal for lower bounds. As the bounds of unnormalized model for the specific observation must not be reversed the term has to be handled separately in the normalization. The resulting bounds for the normalized model $p_{\widetilde{\mathcal F}}(o\,|\,a,h;\boldsymbol\theta)$ are

$$\left[p_{\widetilde{\mathcal F}}(o\,|\,a,h;\boldsymbol\theta)\right]^{\pm} = \frac{[\bar p_{\widetilde{\mathcal F}}(o\,|\,a,h;\boldsymbol\theta)]^{\pm}}{[\bar p_{\widetilde{\mathcal F}}(o\,|\,a,h;\boldsymbol\theta)]^{\pm} + \sum_{o'\neq o}[\bar p_{\widetilde{\mathcal F}}(o'\,|\,a,h;\boldsymbol\theta)]^{\mp}}\,. \tag{5.21}$$

Using box constraints makes propagating bounds computationally efficient but results in wider bounds. An alternative is to solve the corresponding linear program exactly in each backup step, which is more expensive but results in exact bounds for the action value.

5.2.3 Computing Value Bounds

The values for history h are computed analogously to Eq. (2.7) and (5.16) as

$$Q_{(h,a)} = \sum_o p_{\widetilde{\mathcal F}}(o\,|\,a,h;\boldsymbol\theta)\left(r_{(h,a,o)} + \gamma V_{h\oplus(a,o)}\right) \tag{5.22}$$

$$\begin{aligned}V_h^{(N)} &= \Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} p_{\widetilde{\mathcal{F}}}(o'\,|\,a',h;\boldsymbol{\theta})\Big]^{N+1} V_{h\oplus h^*} + \dots \\ &\quad \dots + \Big[\sum_a \pi_{(a\,|\,h)} \sum_o p_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})\Big(r_{(h\oplus h^*,a,o)} + [\![a\notin\mathcal{A}'\wedge o\notin\mathcal{O}']\!]\gamma\, V_{h\oplus h^*\oplus(a,o)}\Big)\Big]\dots \\ &\quad \dots \frac{1-\Big[\gamma\sum_{a'\in\mathcal{A}'}\pi_{(a'\,|\,h)}\sum_{o'\in\mathcal{O}'}p_{\widetilde{\mathcal{F}}}(o'\,|\,a',h;\boldsymbol{\theta})\Big]^{N+1}}{1-\Big[\gamma\sum_{a'\in\mathcal{A}'}\pi_{(a'\,|\,h)}\sum_{o'\in\mathcal{O}'}p_{\widetilde{\mathcal{F}}}(o'\,|\,a',h;\boldsymbol{\theta})\Big]}\;. \end{aligned} \tag{5.23}$$

Using box constraints for $p_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})$ the value bounds are

$$\Big[Q_{(h,a)}\Big]^{\pm} = \sum_o \Big\{\Big[p_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})\Big]^{\pm}; \Big[r_{(h,a,o)}\Big]^{\pm} + \gamma\Big[V_{h\oplus(a,o)}\Big]^{\pm}\Big\}_{\oplus} \tag{5.24}$$

$$\begin{aligned}\Big[V_h^{(N)}\Big]^{\pm} &= \Big\{\Big[\gamma \sum_{a'\in\mathcal{A}'} \pi_{(a'\,|\,h)} \sum_{o'\in\mathcal{O}'} [p_{\widetilde{\mathcal{F}}}(o'\,|\,a',h;\boldsymbol{\theta})]^{\pm}\Big]^{N+1}; \Big[V_{h\oplus h^*}\Big]^{\pm}\Big\}_{\oplus} + \dots \\ &\quad \dots + \Big\{\sum_a \pi_{(a\,|\,h)} \sum_o \Big[p_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})\Big]^{\pm}; \Big[r_{(h\oplus h^*,a,o)}\Big]^{\pm} + [\![a\notin\mathcal{A}'\wedge o\notin\mathcal{O}']\!]\gamma \Big[V_{h\oplus h^*\oplus(a,o)}\Big]^{\pm}\Big\}_{\oplus}\dots \\ &\quad \dots \frac{1-\Big[\gamma\sum_{a'\in\mathcal{A}'}\pi_{(a'\,|\,h)}\sum_{o'\in\mathcal{O}'}[p_{\widetilde{\mathcal{F}}}(o'\,|\,a',h;\boldsymbol{\theta})]^{\pm}\Big]^{N+1}}{1-\Big[\gamma\sum_{a'\in\mathcal{A}'}\pi_{(a'\,|\,h)}\sum_{o'\in\mathcal{O}'}[p_{\widetilde{\mathcal{F}}}(o'\,|\,a',h;\boldsymbol{\theta})]^{\pm}\Big]}\;, \end{aligned} \tag{5.25}$$

where

$$\Big\{[x]^{\pm}; y\Big\}_{\oplus} = \begin{cases} [\,x\,]^{\pm}\, y & \text{if } y \le 0 \\ [\,x\,]^{\mp}\, y & \text{else }, \end{cases} \tag{5.26}$$

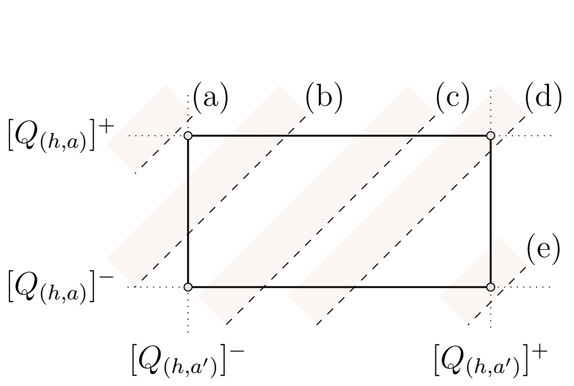
denotes the bounds of a product with the first factor being positive. The reward function $r_{(h,a,o)}$ is either given analytically, in which case the bounds match the given value, or $[r_{(h,a,o)}]^{\pm}$ is computed from the model. If we are using exact bounds for the normalized model $p_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})$ based on the box constraints for the unnormalized model $\bar{p}_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})$ the value bounds are computed by solving the linear program where the model is replaced by a constrained parameterized version

$$\Big[p_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})\Big]^{\pm} \to \frac{\bar{p}_o}{\sum_{o'}\bar{p}_{o'}} \tag{5.27}$$

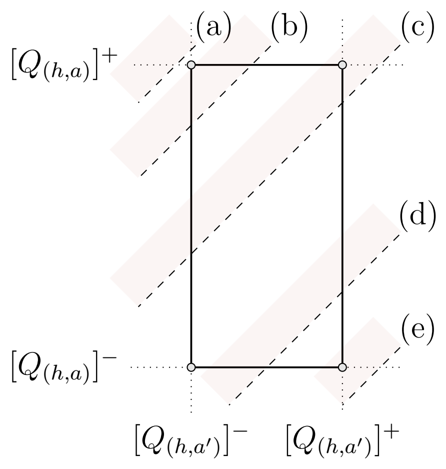
$$s.t.\quad \forall_o\, \bar{p}_o \le \Big[\bar{p}_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})\Big]^{+} \tag{5.28}$$

$$\forall_o\, \bar{p}_o \ge \Big[\bar{p}_{\widetilde{\mathcal{F}}}(o\,|\,a,h;\boldsymbol{\theta})\Big]^{-} \tag{5.29}$$

and Eq. (5.24) and Eq. (5.25) are maximized and minimized over the set of all $\bar{p}_o$ within their given bounds.



(a) Splitting cases for $\frac{Q^+_{(h,a)} - Q^-_{(h,a)}}{Q^+_{(h,a')} - Q^-_{(h,a')}} < 1$.



(b) Splitting cases for $\frac{Q^+_{(h,a)} - Q^-_{(h,a)}}{Q^+_{(h,a')} - Q^-_{(h,a')}} > 1$.

Figure 5.4: The dashed lines denote the boundary of the $Q_{(h,a)} \geq Q_{(h,a')}$ half plain with the red shading indicating the interior of the volume.

5.2.4 Computing the Policy

To compute bounds for the values we also need to take the policy $\pi_{(a \mid h)}$ into account. One option is to choose a deterministic policy, for instance, by maximizing the upper bounds. A disadvantage of a deterministic policy is that it is piecewise constant for changing action values, which is unfortunate for selecting planning actions using the gradient-based approximation suggested in Section 4.1.2. Instead, I use the distribution of the optimal action $p(a^* \mid s, \boldsymbol{\xi})$, which can be computed analytically, provides useful gradient information, and corresponds to a Thompson sampling policy (Agrawal and Goyal, 2012). We have to compute the probability for each given action to be optimal

$$p(a^* \mid s, \boldsymbol{\xi}) = \int p(\mathbf{Q}_h \mid \boldsymbol{\xi}) \, [\![\forall_{a \in \mathcal{A}} \, Q_{(h,a)} \leq Q_{(h,a^*)}]\!] \, d\mathbf{Q}_h \ , \qquad (\rightarrow 5.17)$$

which, for $p(\mathbf{Q}_h \mid \boldsymbol{\xi})$ uniform within the given bounds, yields the Thompson policy

$$\pi_{(a \mid h)} = \prod_{a'} \frac{1}{[Q_{(h,a')}]^+ - [Q_{(h,a')}]^-} \int_{[Q_{(h,a_1)}]^-}^{[Q_{(h,a_1)}]^+} \cdots \int_{[Q_{(h,a_n)}]^-}^{[Q_{(h,a_n)}]^+} \left[\!\!\left[\forall_{a'' \in \mathcal{A}} \, Q_{(h,a'')} \leq Q_{(h,a)} \right]\!\!\right] d\mathbf{Q}^\pi_h \ . \qquad (5.30)$$

The probability $\pi_{(a\,|\,h)}$ of choosing action a is proportional to the volume of the convex polytope defined by the inequalities

$$\forall_{a'\in\mathcal{A}}\ Q_{(h,a')} \leq [Q_{(h,a')}]^{+} \qquad \text{(upper bounds)} \tag{5.31}$$

$$\forall_{a'\in\mathcal{A}}\ Q_{(h,a')} \geq [Q_{(h,a')}]^{-} \qquad \text{(lower bounds)} \tag{5.32}$$

$$\forall_{a'\in\mathcal{A}}\ Q_{(h,a')} \leq Q_{(h,a)} \qquad \text{(diagonals)}\ . \tag{5.33}$$

This volume can be computed analytically, which makes the bound propagation end-to-end differentiable. In the following I describe an algorithm that computes the volume by recursively reducing the problem to lower dimensions, successively splitting along each axis. The basic idea is to split at the intersection of upper or lower bounds with the diagonals such that the resulting sub-volumes are either prismatic along that dimension (if they do not contain the diagonal) or exactly contain the diagonal. Both of these cases are then trivial to compute. When computing the volume for which $Q_{(h,a)}$ is maximal we iterate through all dimensions $a' \neq a$ to adjust the bounds and/or split along a'. There are five different cases to be considered for splitting, as illustrated in Figure 5.4:

(a) If $[Q_{(h,a)}]^{+} \leq [Q_{(h,a')}]^{-}$ the diagonal does not cross the bounds and the volume defined by the bounds lies entirely outside the diagonal constraint. The partial volume is thus empty.

(b) If (a) is not fulfilled but $[Q_{(h,a)}]^{+} \leq [Q_{(h,a')}]^{+}$ and $[Q_{(h,a)}]^{-} \leq [Q_{(h,a')}]^{-}$ the diagonal cuts off a triangle from the bounded volume. We do not need to split the volume but we need to tighten the bounds accordingly.

(c)/(d) If (a) and (b) are not fulfilled but $[Q_{(h,a)}]^{-} < [Q_{(h,a')}]^{+}$ we need to split the volume at least once. We start by splitting the dimension with wider bounds (a or a'), which is always required. For the sub-volume that does not contain the diagonal we only need to adjust the bounds. The other sub-volume may need to be split again along the other dimension (case (d)). The second split as well as the adaptation of the bounds is handled via recursion.

(e) If $[Q_{(h,a)}]^{-} \geq [Q_{(h,a')}]^{+}$ the diagonal does not cross the bounds and the volume defined by the bounds lies entirely within the diagonal constraint. That means, the higher-dimensional volume is prismatic along this dimension. We can decompose the problem into computing the volume of the base polyhedron with reduced dimensionality and multiply the result by the extent along the current dimension to gain the overall volume.

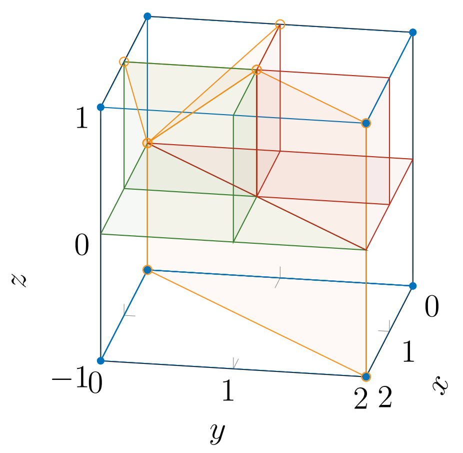
Algorithm 3 Pseudo code for computing the partial volume in Eq. (5.30).

```
Input: $b$, $d_{\max}$                                                    ▷ bounds, max-dimension
Output: VOLUME($b$, $d_{\max}$)                                           ▷ partial volume of max-dimension
function VOLUME($b$, $d_{\max}$, $d_\times$=0)
    if $d_\times = d_{\max}$ then                                         ▷ skip $d_{\max}$
        $d_\times \leftarrow d_\times + 1$
    end if
    if $d_\times \geq |b|$ then                                           ▷ splitting finished
        return $\frac{1}{|b|} \prod b^+ - b^-$                            ▷ return $n^{\text{th}}$ part of $n$-dimensional hypercube
    else                                                                  ▷ split volume if possible
        if $b^+_{d_{\max}} \leq b^-_{d_\times}$ then
            return 0                                                      ▷ case (a): empty volume
        else if $b^-_{d_{\max}} \leq b^-_{d_\times} \wedge b^+_{d_{\max}} \leq b^+_{d_\times}$ then
            return ADAPTBOUNDS($b$, $d_{\max}$, $d_\times$)               ▷ no splitting required
        else if $b^-_{d_{\max}} < b^+_{d_\times}$ then
            return SPLITVOLUME($b$, $d_{\max}$, $d_\times$)               ▷ split volume
        else
            return REDUCEDIMENSIONALITY($b$, $d_{\max}$, $d_\times$)      ▷ no intersection with diagonal
        end if
    end if
end function

function ADAPTBOUNDS($b$, $d_{\max}$, $d_\times$)                         ▷ case (b): only adapt bounds $b$
    init $b' \leftarrow b$
    $b'^-_{d_{\max}} \leftarrow b'^-_{d_\times}$
    $b'^+_{d_\times} \leftarrow b'^+_{d_{\max}}$
    if $b'^-_{d_{\max}} \neq b^-_{d_{\max}} \vee b'^-_{d_\times} \neq b^-_{d_\times}$ then   ▷ bounds changed
        $d'_\times = 0$                                                   ▷ restart splitting with first dimension
    else                                                                  ▷ bounds did not change
        $d'_\times \leftarrow d_\times + 1$                               ▷ continue splitting with next dimension
    end if
    return VOLUME($b'$, $d_{\max}$, $d'_\times$)
end function

function SPLITVOLUME($b$, $d_{\max}$, $d_\times$)                         ▷ case (c/d): at least one split required
    if $b^+_{d_{\max}} - b^-_{d_{\max}} > b^+_{d_\times} - b^-_{d_\times}$ then   ▷ split dimension $d_{\max}$ at $b^+_{d_\times}$
        init $b' \leftarrow b$
        $b'^-_{d_{\max}} \leftarrow b^+_{d_\times}$                       ▷ volume without intersection
        init $b'' \leftarrow b$
        $b''^+_{d_{\max}} \leftarrow b^+_{d_\times}$                      ▷ volume with intersection (may be split a 2nd time)
        return VOLUME($b'$, $d_{\max}$, 0) + VOLUME($b''$, $d_{\max}$, 0)
    else                                                                  ▷ split dimension $d_\times$ at $b^-_{d_{\max}}$
        init $b' \leftarrow b$
        $b'^+_{d_\times} \leftarrow b^-_{d_{\max}}$                       ▷ volume without intersection
        init $b'' \leftarrow b$
        $b''^-_{d_\times} \leftarrow b^-_{d_{\max}}$                      ▷ volume with intersection (may be split a 2nd time)
        return VOLUME($b'$, $d_{\max}$, 0) + VOLUME($b''$, $d_{\max}$, 0)
    end if
end function

function REDUCEDIMENSIONALITY($b$, $d_{\max}$, $d_\times$)                ▷ case (e): no intersection with diagonal
    $x = b^+_{d_\times} - b^-_{d_\times}$                                 ▷ extent along current dimension
    init $b' \leftarrow b$                                                ▷ bounds with reduced dimensionality
    delete $b'_{d_\times}$                                                ▷ delete current dimension
    return $x \cdot$ VOLUME($b'$, $d_{\max}$, $d_\times + 1$)
end function
```



$$
\begin{aligned}
[x]^- &= 0 & [x]^+ &= 2 & V_{x>y,z} &= 2+1+1/2+1/3\\
[y]^- &= 0 & [y]^+ &= 2 & V_{y>x,z} &= 2+1+1/2+1/3\\
[z]^- &= -1 & [z]^+ &= 1 & V_{z>x,y} &= 1/3
\end{aligned}
$$

Figure 5.5: Decomposition of three dimensional volume. Blue lines indicate the bounds, orange lines/planes indicate the diagonal constraints, red and green lines/planes indicate the decomposition of the volume as performed by Algorithm 3 for the respective sub-volumes.

The full algorithm is shown in Algorithm 3. After each adaptation of bounds in (b) and each split in (c)/(d) we have to restart checking from the first dimension to correctly handle adaptation of bounds, splits, and reduction of dimensionality. The recursion terminates when either a volume is empty (case (a)) or it cannot be further split. In the latter case some dimensions may have been reduced (case (e)) while the non-reduced dimensions define an n-dimensional hypercube that is split into n equal parts by the diagonal constraints. The described procedure exploits the specific problem structure (a combination of box constraints and diagonal constraints) to circumvent computing all intersections explicitly. It also avoids unnecessary splits, most notably, for case (d) the second split is only applied to one of the sub-volumes, producing three instead of four components. In Figure 5.5 the resulting decomposition is depicted for a simple 3D example. Note that for $(x > y, z)$ as well as $(y > x, z)$ the first split is along the z-axis, which avoids unnecessary splittings of the $z < 0$ sub-volumes.

5.2.5 Performing Planning Actions

We are now in the position to compute the uncertainty measure for active planning, namely the entropy of the optimal action $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$ given the current state of the AST $\boldsymbol{\xi}$. Except for minor modifications we can follow the exact same scheme as in active tree search for choosing planning actions using a one-step greedy policy, that is,

1. we compute the gradient of $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$ with respect to the value bounds in each node
2. we compute the expected change of the value bounds for activating a specific feature in a specific node

3. we compute the gradient-based linear approximation and choose the node and feature with maximum expected change of $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$.

The gradient can be computed analogously to active tree search. Computing expected changes becomes simpler for planning in feature space because there is no stochastic sampling involved. In principle we could temporally activate a specific feature and readily obtain the corresponding change by recomputing $\mathrm{H}(a^* \,|\, \boldsymbol{\xi})$. This exact method involves performing the corresponding structural changes to the AST and the full computation by performing backups through the tree. It is therefore rather expensive but might still be advantageous if it prevents extensive growth of the AST in the long run. The gradient-based approximation is more efficient in terms of structural changes and computations but introduces inaccuracies due to the linearization and the fact that the effect of implicitly activated features (during the final completion step – see Section 5.1.3) is not taken into account. The best choice will generally depend on the specific model and environment and requires empirical investigations that are beyond the scope of this work.

5.3 Outlook

5.3.1 Solving Planning by Planning

The approach to planning developed in this work consists in framing planning as an active learning problem. As described in Section 2.2.3 active learning itself poses a challenging planning problem. It is therefore appropriate to view planning itself as a planning problem, which seems obvious once a clear distinction between actions in the environment and planning actions is made. In this view different paths in planning space correspond to different valid plans. The available planning resources can be employed to refine either of these plans. This situation is structurally similar to a parallel decomposition (also see Section 5.3.3), which represents a special case of coexisting plans.

5.3.2 Hierarchical Decomposition, Options, and Policy Features

Hierarchical decomposition, most notably the options framework from hierarchical RL, is a relevant form of temporal abstraction. The approach for feature-based semi-Markov abstraction provided by ASTs allows integrating predefined options with a fixed policy. However, this kind of macro actions are somewhat rigid, monolithic building blocks for temporal abstraction. It would be desirable to integrate these structures in a more flexible, adaptable way. One path towards this goal would be *policy features*, that is, the possibility to represent policy information as part of the model. From the modeling side this is conceptually straight forward and can be realized by including the action in the set of predicted variables (together with the observation and the reward). However, as planning is about optimizing the policy it is not obvious how policy features integrate with planning. A view that I deem promising is to consider the policy-part of the model to be a prior distribution over actions that can be overruled by the planner by multiplying the action distribution with a non-uniform planning policy. An open question on the learning side is how to appropriately train such a model. For one thing, to capture macro actions and temporal abstraction the model must be trained on temporally extended sequences as opposed step-by-step samples. For another thing, it is not clear whether maximizing the likelihood is the best objective for learning typical policies and macro behaviors but it certainly represents a sensible starting point for further investigation.

5.3.3 Parallel Decomposition

As already discussed in Section 5.1.4, parallel decomposition requires maintaining several abstractions of the same set of trajectories in a consistent way. This is not compatible with ASTs in their current form as they are required to be consistent in order to compute meaningful values. In principle, however, ASTs are capable of compactly representing multiple parallel plans by using branchings that do not partition the trajectory space as required for consistency of a single plan. For computing the value these branchings have to be treated differently. If the corresponding partial plans are known to be compatible, that is, if the parallel decomposition is exact, values can be summed at the corresponding branching point. In general, however, the interaction or mutual exclusion of the respective partial plans has to be taken into account, which is a highly interesting challenge to be tackled in the future.

5.3.4 Loops

In classical search trees the depth equals the maximum representable trajectory length. In ASTs semi-Markov abstraction allows to compactly represent trajectories of arbitrary length but only under very specific conditions. Allowing for loops in the graph extends the capabilities of representing (infinitely) long trajectories but poses a challenge in two respects. First, ensuring consistency (in the sense of a unique representation of a specific trajectory) is more involved. Second, we have to ensure the convergence of the value and the gradient in a loopy computational graph.

5.3.5 Beyond Macro Actions: Reachability

Macro actions are a powerful approach for representing temporally extended behavior. However, they are bound to a specific policy that has to be followed. In many situations the interesting question for planning is not so much *how* to reach a certain goal but whether it is *reachable at all.* This kind of information allows to plan on a more abstract level and saves resources (that would otherwise be spent on computing/representing the specific policy) if a certain option turns out to be irrelevant. Representing reachability requires some form of modal logic to formalize a statement such as “it is possible to reach goal g” as opposed to probabilistic models that only allow to represent statements such as “the probability of reaching goal g is p”, where p has to be a specific numeric value. The attempt to generalize probabilistic models to include modal logic and to use this kind of model for planning is an extensive and interesting research endeavor.

5.3.6 Complex Temporal Abstraction

Some forms of temporal abstraction are representable by ASTs, most notable, non-Markov dependencies that can be described in terms of action-observation abstraction and semi-Markov abstraction. As mentioned before other forms of temporal abstraction, such as partial orders, are relevant but not compactly representable by ASTs. The bottleneck in dealing with these abstractions consists in representing the corresponding clusters of trajectories in a way that allows to efficiently compute values from the given model. The brute-force approach – iterating over all trajectories and computing the value for each trajectory separately – is conceptually easy but contradicts the whole idea of abstraction. It might be possible to derive special representations tailored to specific kinds of temporal abstraction, similar to the semi-Markov approach presented in this work. Likewise it might be worthwhile trying to learn structured models that build on existing representations for temporal abstraction used in classical planning.

Chapter 6

Conclusion

The overarching endeavor of this thesis is to bridge the gap between the powerful abstract planning methods from classical AI and the modern data-driven approach to learning models for predicting and simulating the environment. The main challenge consists in the fact that the structural information needed to carry out abstract planning is not easily incorporated in a learned model that focuses on prediction in a step-by-step manner.

To bridge the gap, in this thesis, I proceeded in three steps. First, I developed an approach to learning models that allows to incorporate complex structural information as it is needed to represent adaptable abstractions for planning. Second, I reformulated planning as an active learning process, which provides a general basis for making use of structural information and adaptable abstractions provided by a model. Third, I showed how both components – structured feature-based models learned from data and the active planning approach – can be linked to carry out planning in the model's feature space thereby departing from the step-wise simulation-based approach to planning that is commonly used with learned models.

6.1 Contribution

On a theoretical level this thesis contributes by providing a better understanding why classical planning approaches and Markov models learned from interaction data are largely incompatible. Identifying the Markov property as the bottleneck that hinders a unification of both worlds lays the basis for a more focused effort in making a connection. On a practical level, in three chapters I established a basis on both sides and forged an initial link demonstrating how planning in abstract spaces can be combined with probabilistic models learned from data.

6.1.1 Learning Structured Models

The challenge on the modeling side was how to learn models from data that provide structural information that goes beyond merely describing the dynamics of the system on a low level and that does not rely on a well-defined Markov state of the system.

In Chapter 3 I proposed a general structure for models that are flexible and expressive enough to represent various forms of abstraction, on the one hand, but lend themselves to learning from data, on the other hand. A major achievement is the development of the *PULSE* framework that is capable of learning such models, which is investigated theoretically and demonstrated empirically. The *PULSE* framework is an independent contribution to the repertoire of machine learning methods that has an impact beyond the scope of this work as demonstrated in the application for modeling monophonic melodies in music (Section 3.5).

6.1.2 Active Planning

From the planning side, the major challenge was to go beyond step-wise simulation-based planning without relying on specialized model specification languages that render learning a model intractable. In Chapter 4 I developed the *active planning* approach that frames planning as an active learning problem. Importantly, active planning can also be used with conventional Markov models, which is demonstrated in the *active tree search* approach (Section 4.2), the active planning version of Monte-Carlo tree search. The active planning approach is an essential contribution to a unified approach to planning as it provides a general and theoretically sound framework to describe the interaction between a planning algorithm and a model.

6.1.3 Planning in Feature Space

The eventual link is established by describing how the active planning approach from Chapter 4 can be used with the structured models developed in Chapter 3. The crucial step consists in departing from planning actions that are linked to actions in the environment and instead perform planning actions in the abstract feature space of the model. By decoupling planning actions from the underlying representation of the environment the complexity of a planning problem does not depend on parameters such as the state-action space size but rather on the inherent complexity of the task and the quality of the corresponding abstractions provided by the model.

6.2 Outlook

The approaches developed throughout this thesis provide numerous opportunities for further research both in line with the endeavor of combining abstract planning with learned models as well as in other directions.

The *PULSE* framework provides an independent approach for learning model structures and can be applied to various problems and extended in different ways. Some connections to other learning frameworks are discussed in Section 3.2.5 and some alternatives for the underlying model in Section 3.2.6, both of which can serve as a starting point for further research.

The active planning approach is a novel view on the planning problem and may inspire developments in different branches of planning. The use of active planning in MCTS, presented in Section 4.2, is one such example, which may be worthwhile developing further (also see Section 4.2.6).

Finally, the approach for planning in feature space, presented in Chapter 5, offers wide-ranging options for further investigation. For one thing, an empirical evaluation of the presented algorithm is pending. For another thing, this way of intertwining the planning algorithm with the way how abstraction is represented in a learned model possibly represents the starting point for a branch of research dealing with how to improve this interplay both from the planning as well as the modeling side.

Literature

Agrawal, Shipra and Navin Goyal (2012). “Analysis of Thompson Sampling for the Multi-Armed Bandit Problem”. In: *Conference on Learning Theory*, pp. 39–1 (cit. on p. 138).

Amari, Shun’ichi (2001a). “Information Geometry on Hierarchical Decomposition of Stochastic Interactions”. In: *IEEE Transaction on Information Theory* 47, pp. 1701–1711 (cit. on pp. 66, 67).

Amari, Shun’ichi (2001b). “Information Geometry on Hierarchy of Probability Distributions”. In: *IEEE transactions on information theory* 47.5, pp. 1701–1711 (cit. on pp. 66, 73).

Andrychowicz, Marcin et al. (2016). “Learning to Learn by Gradient Descent by Gradient Descent”. In: *Advances in Neural Information Processing Systems*, pp. 3981–3989 (cit. on p. 31).

Audibert, Jean-Yves, Rémi Munos, and Csaba Szepesvári (2009). “Exploration-Exploitation Tradeoff Using Variance Estimates in Multi-Armed Bandits”. In: *Theoretical Computer Science* 410.19, pp. 1876–1902 (cit. on p. 29).

Auer, Peter, Nicolò Cesa-Bianchi, and Paul Fischer (2002). “Finite-Time Analysis of the Multiarmed Bandit Problem”. In: *Machine Learning* 47 (2–3), pp. 235–256 (cit. on pp. 26, 28).

Auersperg, Alice M. I., Alex Kacelnik, and Auguste M. P. von Bayern (2013). “Explorative Learning and Functional Inferences on a Five-Step Means-Means-End Problem in Goffin’s Cockatoos (Cacatua Goffini)”. In: *PLOS ONE* 8.7 (cit. on p. 1).

Bartholomew-Biggs, Michael et al. (2000). “Automatic Differentiation of Algorithms”. In: *Journal of Computational and Applied Mathematics* 124.1, pp. 171–190 (cit. on pp. 109, 110).

Barto, Andrew G., Steven J. Bradtke, and Satinder P. Singh (1995). “Learning to Act Using Real-Time Dynamic Programming”. In: *Artificial Intelligence* 72 (1–2), pp. 81–138 (cit. on pp. 28, 117).

Barto, Andrew G. and Sridhar Mahadevan (2003). “Recent Advances in Hierarchical Reinforcement Learning”. In: *Discrete Event Dynamic Systems* 13.4, pp. 341–379 (cit. on pp. 5, 21).

Bengio, Yoshua (2000). “Gradient-Based Optimization of Hyperparameters”. In: *Neural computation* 12.8, pp. 1889–1900 (cit. on p. 33).

Berry, Donald A. and Bert Fristedt (1985). *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability).* Springer (cit. on pp. 21, 26, 28).

Bertsekas, Dimitri P. et al. (1995). *Dynamic Programming and Optimal Control.* Vol. 1. 2. Athena scientific Belmont, MA (cit. on p. 4).

Bishop, Christopher M. (2007). *Pattern Recognition and Machine Learning (Information Science and Statistics).* 1st ed. 2006. Corr. 2nd printing. Springer (cit. on pp. 16, 31).

Bnaya, Zahy et al. (2015). “Confidence Backup Updates for Aggregating MDP State Values in Monte-Carlo Tree Search”. In: *Eighth Annual Symposium on Combinatorial Search* (cit. on p. 29).

Boutilier, Craig, Thomas Dean, and Steve Hanks (1999). “Decision-Theoretic Planning: Structural Assumptions and Computational Leverage”. In: *Journal of Artificial Intelligence Research* 11, pp. 1–94 (cit. on pp. 25, 39).

Brafman, Ronen I and Moshe Tennenholtz (2003). “R-Max-a General Polynomial Time Algorithm for near-Optimal Reinforcement Learning”. In: *The Journal of Machine Learning Research* 3, pp. 213–231 (cit. on p. 21).

Browne, Cameron B et al. (2012). “A Survey of Monte Carlo Tree Search Methods”. In: *Computational Intelligence and AI in Games, IEEE Transactions on* 4.1, pp. 1–43 (cit. on pp. 26, 28).

Camacho, Eduardo F. and Carlos Bordons Alba (2013). *Model Predictive Control.* Springer Science & Business Media (cit. on p. 4).

Chaloner, Kathryn and Isabella Verdinelli (1995). “Bayesian Experimental Design: A Review”. In: *Statistical Science* 10.3, pp. 273–304 (cit. on p. 17).

Chaslot, GMJB et al. (2007). “Progressive Strategies for Monte-Carlo Tree Search”. In: *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pp. 655–661 (cit. on p. 30).

Cherla, Srikanth, Son N. Tran, Artur d’Avila Garcez, et al. (2015). “Discriminative Learning and Inference in the Recurrent Temporal RBM for Melody Modelling”. In: *International Joint Conference on Neural Networks (IJCNN).* IEEE, pp. 1–8 (cit. on pp. 89, 92).

Cherla, Srikanth, Son N. Tran, Tillman Weyde, et al. (2015). “Hybrid Long-and Short-Term Models of Folk Melodies.” In: *ISMIR*, pp. 584–590 (cit. on pp. 88, 89, 92, 93).

Cherla, Srikanth, Tillman Weyde, et al. (2013). “Learning Distributed Representations for Multiple-Viewpoint Melodic Prediction”. In: *14th International Society for Music Information Retrieval Conference*. ISMIR (cit. on pp. 88, 92).

Conklin, Darrell and Ian H. Witten (1995). “Multiple Viewpoint Systems for Music Prediction”. In: *Journal of New Music Research* 24.1, pp. 51–73 (cit. on pp. 88, 89).

Couëtoux, Adrien et al. (2011). “Continuous Upper Confidence Trees”. In: *Learning and Intelligent Optimization*. Springer, pp. 433–445 (cit. on p. 30).

De Waard, Maarten (2016). “Monte Carlo Tree Search with Options for General Video Game Playing”. MA thesis. Universiteit van Amsterdam (cit. on p. 21).

DeLanda, Manuel (2006). *A New Philosophy of Society: Assemblage Theory and Social Complexity*. A&C Black (cit. on p. 24).

Della Pietra, S., V. Della Pietra, and J. Lafferty (1997). “Inducing Features of Random Fields”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 19.4, pp. 380–393 (cit. on p. 34).

Dietterich, T. G. (2000). “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition”. In: *J. Artif. Intell. Res. (JAIR)* 13, pp. 227–303 (cit. on p. 22).

Dietterich, Thomas G. (1998). “The MAXQ Method for Hierarchical Reinforcement Learning”. In: *In Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann, pp. 118–126 (cit. on p. 22).

Efron, Bradley et al. (2004). “Least Angle Regression”. In: *The Annals of statistics* 32.2, pp. 407–499 (cit. on pp. 35, 76).

Feldman, Zohar and Carmel Domshlak (2014a). “Monte-Carlo Tree Search: To MC or to DP?” In: *Models and Paradigms for Planning under Uncertainty: a Broad Perspective*, p. 11 (cit. on p. 29).

Feldman, Zohar and Carmel Domshlak (2014b). “On MABs and Separation of Concerns in Monte-Carlo Planning for MDPs”. In: *Twenty-Fourth International Conference on Automated Planning and Scheduling* (cit. on p. 26).

Fikes, Richard E. and Nils J. Nilsson (1971). “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. In: *Artificial intelligence* 2 (3–4), pp. 189–208 (cit. on p. 4).

Gadamer, Hans-Georg (2008). *Philosophical Hermeneutics*. Univ of California Press (cit. on p. 44).

Geramifard, Alborz et al. (2011). “Online Discovery of Feature Dependencies”. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Ed. by Lise Getoor and Tobias Scheffer. ICML ’11. ACM, pp. 881–888 (cit. on p. 34).

Ghallab, Malik, Dana Nau, and Paolo Traverso (2016). *Automated Planning and Acting.* Cambridge University Press (cit. on pp. 4, 5, 23, 24, 40).

Golovin, Daniel and Andreas Krause (2011). “Adaptive Submodularity: Theory and Applications in Active Learning and Stochastic Optimization”. In: *Journal of Artificial Intelligence Research* 42, pp. 427–486 (cit. on pp. 26, 34).

Guez, Arthur et al. (2014). “Bayes-Adaptive Simulation-Based Search with Value Function Approximation”. In: *Advances in Neural Information Processing Systems*, pp. 451–459 (cit. on p. 29).

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2008). *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* 2nd ed. Springer (cit. on pp. 31, 34, 35, 76).

Hengst, Bernhard (2002). “Discovering Hierarchy in Reinforcement Learning with HEXQ”. In: *In Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning.* Morgan Kaufmann, pp. 243–250 (cit. on p. 22).

Höfer, Sebastian (2017). “On Decomposability in Robot Reinforcement Learning”. PhD thesis. Computer Science Department, Technische Universität Berlin (cit. on p. 24).

Huron, David Brian (2006). *Sweet Anticipation: Music and the Psychology of Expectation.* MIT press (cit. on p. 88).

Hutter, Marcus (2009). “Feature Reinforcement Learning: Part I. Unstructured MDPs”. In: *Journal of Artificial General Intelligence* 1.1, pp. 3–24 (cit. on p. 82).

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra (1998). “Planning and Acting in Partially Observable Stochastic Domains”. In: *Artificial Intelligence* 101 (1–2), pp. 99–134 (cit. on p. 16).

Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore (1996). “Reinforcement Learning: A Survey”. In: *CoRR* cs.AI/9605103 (cit. on p. 82).

Keller, Thomas and Malte Helmert (2013). “Trial-Based Heuristic Tree Search for Finite Horizon MDPs.” In: *ICAPS* (cit. on pp. 26, 28, 29).

Kocsis, Levente and Csaba Szepesvári (2006). “Bandit Based Monte-Carlo Planning”. In: *Machine Learning: ECML 2006.* Springer, pp. 282–293 (cit. on pp. 28, 102, 118).

Köhler, Wolfgang (1917). *Intelligenzprüfungen an Menschenaffen.* (English version: Wolgang Köhler (1925): *The Mentality of Apes.* Harcourt & Brace, New York.) Springer, Berlin (3rd edition, 1973) (cit. on p. 1).

Kolter, J Zico and Andrew Y Ng (2009). “Near-Bayesian Exploration in Polynomial Time”. In: *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM, pp. 513–520 (cit. on p. 21).

Krause, Andreas and Carlos Guestrin (2007). “Near-Optimal Observation Selection Using Submodular Functions”. In: *AAAI*. Vol. 7, pp. 1650–1654 (cit. on p. 34).

Krumhansl, Carol L. (1990). *Cognitive Foundations of Musical Pitch*. Oxford Psychology 17. Oxford University Press (cit. on pp. 91, 92).

Kulick, Johannes, Robert Lieck, and Marc Toussaint (2016). “Cross-Entropy as a Criterion for Robust Interactive Learning of Latent Properties”. In: *NIPS Workshop on the Future of Interactive Learning Machines* (cit. on p. 26).

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira (2001). “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data”. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. Morgan Kaufmann Publishers Inc., pp. 282–289 (cit. on p. 76).

Lagoudakis, Michail G. and Ronald Parr (2003). “Least-Squares Policy Iteration”. In: *J. Mach. Learn. Res.* 4, pp. 1107–1149 (cit. on p. 82).

Lake, Brenden M. et al. (2016). “Building Machines That Learn and Think Like People”. In: *Behavioral and Brain Sciences* (cit. on p. 31).

Langhabel, Jonas (2017). “Learning a Predictive Model for Music Using PULSE”. MA thesis. Computer Science Department, Technische Universität Berlin (cit. on pp. 91, 94).

Langhabel, Jonas et al. (2017). “Feature Discovery for Sequential Prediction of Monophonic Music”. In: *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*. International Society for Music Information Retrieval Conference (cit. on pp. 77, 88, 91, 93).

Li, Ke and Jitendra Malik (2016). “Learning to Optimize”. In: *arXiv preprint* (cit. on p. 31).

Li, Lihong et al. (2011). “Knows What It Knows: A Framework for Self-Aware Learning”. In: *Mach Learn* 82.3, pp. 399–443 (cit. on p. 21).

Lieck, Robert, Vien Ngo, and Marc Toussaint (2017). “Exploiting Variance Information in Monte-Carlo Tree Search”. In: *ICAPS Workshop on Heuristics and Search for Domain-Independent Planning* (cit. on pp. 29, 102).

Lieck, Robert and Marc Toussaint (2016). “Temporally Extended Features in Model-Based Reinforcement Learning with Partial Observability”. In: *Neurocomputing* 192, pp. 49–60 (cit. on p. 79).

Lieck, Robert and Marc Toussaint (2017). “Active Tree Search”. In: *ICAPS Workshop on Planning, Search, and Optimization* (cit. on p. 118).

Ljung, Lennart (1998). “System Identification”. In: *Signal Analysis and Prediction*. Springer, pp. 163–173 (cit. on p. 5).

Lusena, Christopher, Judy Goldsmith, and Martin Mundhenk (2001). “Nonapproximability Results for Partially Observable Markov Decision Processes”. In: *J. Artif. Intell. Res.(JAIR)* 14, pp. 83–103 (cit. on p. 16).

Madani, Omid, Steve Hanks, and Anne Condon (1999). “On the Undecidability of Probabilistic Planning and Infinite-Horizon Partially Observable Markov Decision Problems”. In: *AAAI/IAAI: Proceedings of the Sixteenth National Conference on Artificial Intelligence.* American Association for Artificial Intelligence, pp. 541–548 (cit. on p. 16).

Mannor, Shie et al. (2004). “Dynamic Abstraction in Reinforcement Learning via Clustering”. In: *Proceedings of the Twenty-First International Conference on Machine Learning.* ICML ’04. ACM (cit. on p. 22).

Maron, Oded and Tomás L. Pérez (1998). “A Framework for Multiple-Instance Learning”. In: *Advances in Neural Information Processing Systems.* Ed. by Michael I. Jordan, Michael J. Kearns, and Sara A. Solla. Vol. 10. The MIT Press, pp. 570–576 (cit. on p. 25).

Mavromatis, Panayotis (2005). “A Hidden Markov Model of Melody Production in Greek Church Chant”. In: *Computing in Musicology* 14, pp. 93–112 (cit. on p. 88).

Mavromatis, Panayotis (2009). “Minimum Description Length Modelling of Musical Structure”. In: *Journal of Mathematics and Music* 3.3, pp. 117–136 (cit. on p. 88).

McCallum, A. (2002). “Efficiently Inducing Features of Conditional Random Fields”. In: *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence.* Morgan Kaufmann Publishers Inc., pp. 403–410 (cit. on p. 34).

McCallum, Andrew Kachites (1996). “Reinforcement Learning with Selective Perception and Hidden State”. PhD thesis. Computer Science Department, University of Rochester (cit. on pp. 80, 82).

McDermott, Drew et al. (1998). *PDDL – The Planning Domain Definition Language.* URL: `http://www.citeulike.org/group/13785/article/4097279` (cit. on p. 4).

McGovern, Amy and Andrew G. Barto (2001). “Automatic Discovery of Subgoals in Reinforcement Learning Using Diverse Density”. In: *In Proceedings of the Eighteenth International Conference on Machine Learning.* Morgan Kaufmann, pp. 361–368 (cit. on p. 25).

McMahan, H. Brendan, Maxim Likhachev, and Geoffrey J. Gordon (2005). “Bounded Real-Time Dynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees”. In: *Proceedings of the 22nd International Conference on Machine Learning.* ACM, pp. 569–576 (cit. on pp. 28, 117, 118).

Menache, Ishai, Shie Mannor, and Nahum Shimkin (2002). “Q-Cut – Dynamic Discovery of Sub-Goals in Reinforcement Learning”. In: *Machine Learning: ECML 2002.* Ed.

by Tapio Elomaa, Heikki Mannila, and Hannu Toivonen. Vol. 2430. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 187–195 (cit. on pp. 22, 25).

Meyer, Leonard B. (2008). *Emotion and Meaning in Music*. University of Chicago Press (cit. on p. 88).

Nemhauser, George L., Laurence A. Wolsey, and Marshall L. Fisher (1978). “An Analysis of Approximations for Maximizing Submodular Set Functions—I”. In: *Mathematical Programming* 14.1, pp. 265–294 (cit. on pp. 26, 34).

Nguyen, Phuong, Peter Sunehag, and Marcus Hutter (2012). “Context Tree Maximizing Reinforcement Learning”. In: *Proc. of the 26th AAAI Conference on Artificial Intelligence*, pp. 1075–1082 (cit. on p. 82).

Nikolić, Danko (2015). “Practopoiesis: Or How Life Fosters a Mind”. In: *Journal of Theoretical Biology* 373, pp. 40–61 (cit. on p. 32).

Paiement, Jean-Francois, Samy Bengio, and Douglas Eck (2009). “Probabilistic Models for Melodic Prediction”. In: *Artificial Intelligence* 173.14, pp. 1266–1274 (cit. on p. 88).

Paiement, Jean-Francois, Yves Grandvalet, and Samy Bengio (2009). “Predictive Models for Music”. In: *Connection Science* 21 (2–3), pp. 253–272 (cit. on p. 88).

Parr, Ronald, Lihong Li, et al. (2008). “An Analysis of Linear Models, Linear Value-Function Approximation, and Feature Selection for Reinforcement Learning”. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML ’08. ACM, pp. 752–759 (cit. on p. 82).

Parr, Ronald and Stuart Russell (1997). “Reinforcement Learning with Hierarchies of Machines”. In: *Advances in Neural Information Processing Systems*. Ed. by Michael I. Jordan, Michael J. Kearns, and Sara A. Solla. Vol. 10. The MIT Press (cit. on p. 22).

Pavlov, Ivan Petrovich (2003). *Conditioned Reflexes*. (Republication of the work originally published by the Oxford University Press, London, in 1927.) Dover Publications (cit. on p. 1).

Pearce, Marcus T. and Geraint A. Wiggins (2004). “Improved Methods for Statistical Modelling of Monophonic Music”. In: *Journal of New Music Research* 33.4, pp. 367–385 (cit. on pp. 88, 89, 92, 93).

Pearce, Marcus T. and Geraint A. Wiggins (2012). “Auditory Expectation: The Information Dynamics of Music Perception and Cognition”. In: *Topics in cognitive science* 4.4, pp. 625–652 (cit. on p. 88).

Pearce, Marcus Thomas (2005). “The Construction and Evaluation of Statistical Models of Melodic Structure in Music Perception and Composition”. City University London (cit. on pp. 88, 89).

Pickett, Marc and Andrew G. Barto (2002). “PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning”. In: *Proceedings of the Nineteenth International Conference on Machine Learning.* Morgan Kaufmann, pp. 506–513 (cit. on p. 25).

Raczynski, Stanislaw et al. (2010). “Multiple Pitch Transcription Using DBN-Based Musicological Models”. In: *11th International Society for Music Information Retrieval Conference*, pp. 363–368 (cit. on p. 88).

Rohrmeier, Martin A. and Stefan Koelsch (2012). “Predictive Information Processing in Music Cognition. A Critical Review”. In: *International Journal of Psychophysiology* 83.2, pp. 164–175 (cit. on p. 88).

Russell, Stuart J. and Peter Norvig (2003). *Artificial Intelligence: A Modern Approach.* 2nd ed. Pearson Education (cit. on p. 4).

Saffidine, Abdallah, Tristan Cazenave, and Jean Méhat (2012). “UCD: Upper Confidence Bound for Rooted Directed Acyclic Graphs”. In: *Knowledge-Based Systems* 34, pp. 26–33 (cit. on p. 28).

Sanner, Scott (2010). “Relational Dynamic Influence Diagram Language (Rddl): Language Description”. In: *Unpublished ms. Australian National University* (cit. on p. 4).

Sanner, Scott et al. (2009). “Bayesian Real-Time Dynamic Programming”. In: *Twenty-First International Joint Conference on Artificial Intelligence* (cit. on pp. 28, 117).

Schierwagen, Andreas (2012). “On Reverse Engineering in the Cognitive and Brain Sciences”. In: *Natural Computing* 11.1, pp. 141–150 (cit. on p. 24).

Settles, Burr (2010). *Active Learning Literature Survey.* Computer Sciences Technical Report 1648. University of Wisconsin–Madison (cit. on pp. 17, 26).

Shibuya, Takeshi and Tomoki Hamagami (2011). “Complex-Valued Reinforcement Learning: A Context-Based Approach for POMDPs”. In: *Advances in Reinforcement Learning Learning.* INTECH Open Access Publisher (cit. on p. 80).

Silver, David, Aja Huang, et al. (2016). “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587, pp. 484–489 (cit. on pp. 26, 28, 117).

Silver, David, Richard S Sutton, and Martin Müller (2012). “Temporal-Difference Search in Computer Go”. In: *Machine learning* 87.2, pp. 183–219 (cit. on p. 29).

Silver, David and Joel Veness (2010). “Monte-Carlo Planning in Large POMDPs”. In: *Advances in Neural Information Processing Systems*, pp. 2164–2172 (cit. on p. 30).

Simon, Herbert A. (1996). *The Sciences of the Artificial.* 3rd ed. MIT Press (cit. on pp. 24, 25).

Şimşek, Özgür and Andrew G. Barto (2004). “Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning”. In: *Proceedings of the 21st Inter-*

national Conference on Machine Learning. ACM Press, pp. 751–758 (cit. on pp. 22, 25).

Sussman, Gerald Jay (1975). *A Computer Model of Skill Acquisition.* Vol. 1. American Elsevier Publishing Company New York (cit. on p. 6).

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning).* The MIT Press (cit. on pp. 20, 23, 82, 95).

Sutton, Richard S., Doina Precup, and Satinder P. Singh (1999). "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112 (1–2), pp. 181–211 (cit. on pp. 21, 131).

Taylor, Matthew E and Peter Stone (2009). "Transfer Learning for Reinforcement Learning Domains: A Survey". In: *Journal of Machine Learning Research* 10, pp. 1633–1685 (cit. on p. 22).

Temperley, David (1999). "What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered". In: *Music Perception: An Interdisciplinary Journal* 17.1, pp. 65–100 (cit. on p. 92).

Theano Development Team (2016). "Theano: A Python Framework for Fast Computation of Mathematical Expressions". In: *arXiv e-prints* abs/1605.02688 (cit. on p. 111).

Thrun, Sebastian and Anton Schwartz (1995). "Finding Structure in Reinforcement Learning". In: *Advances in Neural Information Processing Systems 7.* MIT Press, pp. 385–392 (cit. on p. 25).

Toussaint, Marc (2004). "Notes on Information Geometry and Evolutionary Processes". In: *arXiv preprint nlin/0408040* (cit. on p. 66).

Toussaint, Marc and Manuel Lopes (2017). "Multi-Bound Tree Search for Logic-Geometric Programming in Cooperative Manipulation Domains". In: *IEEE International Conference on Robotics and Automation (ICRA).* IEEE, pp. 4044–4051 (cit. on pp. 5, 118).

Ure, N Kemal et al. (2012). "Adaptive Planning for Markov Decision Processes with Uncertain Transition Models via Incremental Feature Dependency Discovery". In: *Machine Learning and Knowledge Discovery in Databases.* Springer, pp. 99–115 (cit. on p. 34).

Veness, Joel et al. (2010). "Reinforcement Learning via AIXI Approximation". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence.* AAAI (cit. on p. 82).

Vlassis, Nikos et al. (2012). "Bayesian Reinforcement Learning". In: *Reinforcement Learning.* Springer, pp. 359–386 (cit. on p. 20).

Wang, Yizao, Jean-Yves Audibert, and Rémi Munos (2009). “Algorithms for Infinitely Many-Armed Bandits”. In: *Advances in Neural Information Processing Systems*, pp. 1729–1736 (cit. on p. 30).

Whorley, Raymond Peter et al. (2013). “The Construction and Evaluation of Statistical Models of Melody and Harmony”. PhD thesis. Goldsmiths, University of London (cit. on p. 88).

Willems, Frans MJ, Yuri M Shtarkov, and Tjalling J Tjalkens (1995). “The Context-Tree Weighting Method: Basic Properties”. In: *Information Theory, IEEE Transactions on* 41.3, pp. 653–664 (cit. on p. 82).

Wolpert, D. H. and W. G. Macready (1997). “No Free Lunch Theorems for Optimization”. In: *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82 (cit. on p. 44).

Wolpert, David H., William G. Macready, et al. (1995). *No Free Lunch Theorems for Search*. Technical Report SFI-TR-95-02-010, Santa Fe Institute. URL: `http://delta.cs.cinvestav.mx/~ccoello/compevol/nfl.pdf` (cit. on p. 44).

Zhou, Kemin, John Comstock Doyle, Keith Glover, et al. (1996). *Robust and Optimal Control*. Vol. 40. Prentice Hall New Jersey (cit. on p. 4).

Acronyms

RL reinforcement learning. iii, 2, 11, 20–23, 25, 26, 34, 79, 115, 116, 118, 131, 133, 142

RTDRBM recurrent temporal discriminative restricted Boltzmann machine. 89, 93

SGD stochastic gradient descent. 92

STM short-term model. 88–92, 94

TEF temporally extended feature. iii, 76, 78–80, 82, 83, 87, 88, 90, 91, 94, 128

UCT upper confidence bounds applied to trees. 102, 120, 121

U-Tree utility tree. 82–85, 87

Index