Applications of Probabilistic Inference to Planning & Reinforcement Learning

Thomas Furmston

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University of London.

Department of Computer Science

University College London

2012

Declaration

I, Thomas John Fairfax Furmston confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Optimal control is a profound and fascinating subject that regularly attracts interest from numerous scientific disciplines, including both pure and applied Mathematics, Computer Science, Artificial Intelligence, Psychology, Neuroscience and Economics. In 1960 Rudolf Kalman discovered that there exists a duality between the problems of filtering and optimal control in linear systems [84]. This is now regarded as a seminal piece of work and it has since motivated a large amount of research into the discovery of similar dualities between optimal control and statistical inference. This is especially true of recent years where there has been much research into recasting problems of optimal control into problems of statistical/approximate inference. Broadly speaking this is the perspective that we take in this work and in particular we present various applications of methods from the fields of statistical/approximate inference to optimal control, planning and Reinforcement Learning. Some of the methods would be more accurately described to originate from other fields of research, such as the *dual decomposition* techniques used in chapter(5) which originate from convex optimisation. However, the original motivation for the application of these techniques was from the field of approximate inference. The study of dualities between optimal control and statistical inference has been a subject of research for over 50 years and we do not claim to encompass the entire subject. Instead, we present what we consider to be a range of interesting and novel applications from this field of research.

Acknowledgements

I would like to thank my supervisor David Barber for all the time he has spent assisting in my research and the many times he has pointed me on the right track throughout the course of my Ph.D. I would also like to thank him for the huge amount of leeway he has allowed me in the topics and directions of research I have taken during my time at University College London. I would also like to thank my wife who has been very supportive throughout. I'm sure I can't have been easy to live with at certain points through my Ph.D. and I'm grateful for the patience she has shown in such times. I would also like to thank Edward Challis, Chris Bracegirdle, Guy Lever, David Silver, Nicolas Hees, Arthur Guez and Steffen Grunewalder for interesting and fruitful discussions. I would also like to thank Nikos Vlassis for helpful discussions regarding my work with dual decomposition techniques and for highlighting a relevant reference in the area of finite horizon Markov Decision Processes. I would like to thank Stephen Hailes for the patience he has shown while I complete this thesis. Finally I would like to thank the Engineering and Physical Research Council (EPSRC) for the funding scholarship and the opportunity to study for my Ph.D. at such a remarkable research centre in Machine Learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Broadly speaking the problems of optimal control, planning and reinforcement learning are concerned with the optimisation of sequential decision making processes. Another way of stating this, which is perhaps more accurate in the case of optimal control, is as the problem of optimising the dynamics of a given system, *w.r.t.* a control variable or functional, so that the state of the system is brought to some desired state at an optimal cost. There are numerous real-world problems that can be cast in such a framework and examples include: Optimal play in games such as chess, backgammon and go; The optimisation of financial portfolios to maximise the expected return of the portfolio; Network management, which in, for example, urban traffic networks consists of minimising the amount of congestion in the network; The optimal control of physical systems, such as robotic equipment, to perform some mechanical task at a minimal cost to the system. These are but a few of the vast range of possible applications and this, as well the mathematical intricacies of the subject, has led to a vast amount of research in this area, especially in the last 60 years.

Optimal control is primarily concerned with the optimisation of systems that are continuous in both in time and space and, as elegantly argued in [159], can be dated back to 1697 with Bernoulli's solution to the *brachystochrone problem*[1]. Optimal control has since matured greatly, from Lagrange's derivation of the *Euler-Lagrange* equations and the birth of the *calculus of variations* (a branch of mathematics in its own right) up until the twentieth century with the introduction of Pontryagin's Maximum principal [127] and Bellman's dynamic programming [22]. Planning, which we consider as the discrete time counterpart to optimal control, is a younger subject, but it is one that has come into prominence in the last 60 years, especially since the advent of the computer. Due to the discrete time formalism of planning the solutions techniques are necessarily iterative in nature and the advent of the computer has allowed the implementation of solution techniques that would have been tedious or impossible to compute otherwise. Additionally, the advent of the computer has brought to prominence the study of problems that are most naturally formalised in the planning framework, such as games like chess and backgammon. While

[1]The *brachystochrone problem* can be stated as follows: *If in a vertical plane two points A and B are given, then it is required to specify the orbit AMB of the movable point M, along which it, starting from A, and under the influence of its own weight, arrives at B in the shortest possible time*. The solution of the brachystochrone problem is given by a cycloid, which is a curve that is described by a point P on a circle that rolls on an axis in such a way that P passes through first A and then B. See [159] for details on Bernoulli's solution and a detailed introduction to the Maximum principal.

there is no equivalent to the maximum principal in the planning framework the dynamic programming paradigm is easily transferable and is, in fact, one of the cornerstones of the subject. However, while dynamic programming provides a theoretical basis for planning and optimal control it has several well-known limitations, which include the *curse of dimensionality*, the restriction that the environment is Markovian and the requirement for a model of the environment.

The curse of dimensionality is a core computational bottleneck of dynamic programming, where the computational complexity scales exponentially in the dimension of the environment. This restricts the exact application of dynamic programming to relatively small problems and alternative optimisation methods are required for larger, more realistic environments. This has led to the introduction of other solution techniques, such as *approximate dynamic programming* methods [26, 41] and policy-search methods [182, 40, 162, 19, 83], which include gradient-based methods. While, on the most basic level at least, the application of gradient-based methods to Markov Decision Processes is relatively easy, there are the invariable difficulties of applying these methods to complicated large scale environments over a possibly infinite planning horizon. These issues include the poor scaling of the gradient, where the magnitude of change in the objective function varies dramatically along different components of the gradient, which necessitates the application of more sophisticated methods, such as Expectation Maximisation [44] and natural gradient ascent [6, 3, 5, 4]. An additional issue is the actual evaluation of the gradient, or similar terms in methods such as Expectation Maximisation, which, due to issues such as non-linearities in the system dynamics or the high dimensions of the environment, is generally intractable. These and other difficulties, along with the the strong performance of these policy-search methods in real-world applications, have led to a large amount of research in this area and we shall consider these methods in detail in chapter(2). In particular we shall consider both the problem of scaling and of inference and propose some novel methods to both problems.

In most models considered in the literature the reward function and dynamics of the system are assumed to be Markovian, *i.e.* the rewards and transitions are only dependent on the current state of the environment and not all previous states of the environment. This assumption is made for computational reasons and the optimisation problems quickly becomes intractable when this condition fails to hold. While a strong assumption it is still possible to model complex dynamics under this framework and so is often accepted in practice. Another condition that is necessary for the application of dynamic programming is that the conditioning set of the controller, that is the variables upon which an action are decided, forms a separator set between the current action variable and the previous states of the environment. When this condition fails to hold dynamic programming is inapplicable and the optimisation problem necessarily becomes significantly more difficult in general. This second restriction fails to hold in many models of interest, such as those introduced in sections(1.5 & 1.4), and alternative optimisation techniques are required. In chapter(5) we shall consider one such model (namely a finite horizon Markov Decision Process with a stationary policy) and apply dual decomposition techniques, which originate from convex optimisation, to enable the application of dynamic programming to a relaxed form of the original planning problem. We shall also briefly suggest some possible extensions to other planning

models where, similarly, dynamic programming is inapplicable, leaving the explicit construction of such extensions as a point of future research.

Both optimal control and planning consider a model of the environment to be known and when this is not the case the controller optimisation problem is known as reinforcement learning.[2] Typical planning algorithms, such as dynamic programming, require a model of the environment and so in the reinforcement learning framework it is not possible to directly apply dynamic programming, or other planning techniques. One solution is to create a model of the environment from any available data and then perform dynamic programming, or some other form of planning, using this model. There are situations where this approach is either undesirable, perhaps because the construction of the model is expensive, or simply not possible due to issues of system identification. When this is the case reinforcement learning methods focus on optimising the controller in an on-line manner, while directly interacting with the environment. These methods often attempt to estimate the quantities of interest in dynamic programming, such as the value function, directly through the use of samples and without explicit construction of a model. Prominent examples include Q-learning [179], *SARSA* [139] and *temporal difference learning* [160]. A general high-level overview is given in *e.g.* [163]. We shall generally consider planning in this work, but we shall consider a Bayesian form of reinforcement learning in chapter(6). More specifically we shall optimise the controller *w.r.t.* a Bayesian objective that incorporates uncertainty in the model by simultaneously accounting for all possible models of the environment (that are within the support of the posterior) given the environmental data.

In this chapter we shall introduce the various models that we shall consider during the course of this work, while also providing an in depth overview of dynamic programming. The organisation of the chapter shall be as follows: In section(1.2) we shall introduce Markov Decision Processes, which are the core model for fully observable environments, in both discrete and continuous time; In section(1.3) we shall introduce dynamic programming for some of the standard formulations of the Markov Decision Process, as well as providing a summary of the strengths and weaknesses of dynamic programming; In section(1.4) we shall introduce a richer class of planning models, known generally as *Partially Observable Markov Decision Processes*, that can model more complex environments and are necessarily harder to optimise; In section(1.5) we shall introduce *decentralised transition independent Markov Decision Processes*, which are a popular model for multi-agent systems where there are restrictions on the communication between the agents; Finally, in section(1.6) we shall summarise the chapter.

1.2 Markov Decision Processes

In this section we describe the planning and control frameworks for environments that are fully observable and Markovian, which are generally referred to as Markov Decision Processes. In section(1.2.1) we shall introduce the discrete time framework, while in section(1.2.2) we shall introduce the continuous time counterpart.

[2] There are alternative, closely related, forms of reinforcement learning that are more interested in different issues, such as the learning mechanisms of animals or humans. Our interest in reinforcement learning is solely from the optimisation perspective and we don't detail these other forms of reinforcement learning.

1.2.1 Discrete Time Control

The discrete time Markov Decision Process[3] (MDP) is a very general mathematical framework with which to model optimal control problems. The generality and flexibility of the MDP framework has led to it being one the most widely used and successful frameworks for discrete time control, being successfully applied to an almost ubiquitous range of problems, from robotics [124, 94, 93, 124, 168, 166, 138, 43], games [61, 60, 152, 164, 144, 174], finance [18] and network management [136, 71]. Informally an MDP considers the optimality of an *agent's* controlled movements through a given environment. The agent is able to direct its movements around the environment (within the restriction of the agent's dynamics and the physical constraints of the environment) by selecting an action at each time point. At each time point the agent receives a scalar reward that usually depends on the agents current position in the environment and the action it just performed. The aim of the MDP framework is to optimise the agent's behaviour so that it can expect to receive the maximal amount of reward during its trajectory through the environment. The MDP framework is very general and allows for a large range possibilities in the modelling of the control problem; including discrete and continuous environments (or a mixture of the two) as well as various ways to model the planning horizon, such finite, episodic or infinite horizons. The main constraint of the MDP framework is that the transitions of the agent evolve in a Markovian manner, a point which will be made more precise shortly. We now give an intuitive example of an MDP, along with some of the modelling possibilities, before proceeding to a more formal definition.

Example 1. *Consider the problem of optimising the behaviour of an agent that is located within a maze environment. An example maze environment is given in fig(1.1), where the walls of the maze are depicted by the black lines. In such a problem the state space could either be modelled continuously or in a discrete manner. In an MDP the agent makes its decision based on its current state, which in this example corresponds to its current position in the maze. The possible actions that the agent can perform depends on how the environment is modelled: in a continuous model a reasonable range of actions is to move in any direction in the* 360° *range; where as in a discrete model a reasonable range of actions is a discrete set of directions, such as* $\{up, down, left, right\}$*. A typical objective is to train the the agent to move from the beginning of the maze, denoted by start, to the end of the maze, denoted by finish, in the fastest possible time. A possible reward function that would achieve a global optimum at this desired behaviour would be to give the agent a positive reward upon completing the maze, while receiving a zero reward at all other time points. There are several immediate possibilities to modelling the planning horizon; two examples are a finite horizon or an infinite horizon episodic environment, where the agent is replaced at the start of the maze every time it completes the maze.*

Formally an MDP is described by the tuple $\{\mathcal{S}, \mathcal{A}, H, p_1, p, \pi, R\}$, where $\mathcal{S}$ and $\mathcal{A}$ are sets known respectively as the state and action space, $H \in \mathbb{N}$ is the planning horizon and $\{p_1, p, \pi, R\}$ are functions that take the following form

[3] We shall generally use the simpler terminology Markov Decision Process when it is clear that we are considering the discrete time framework.

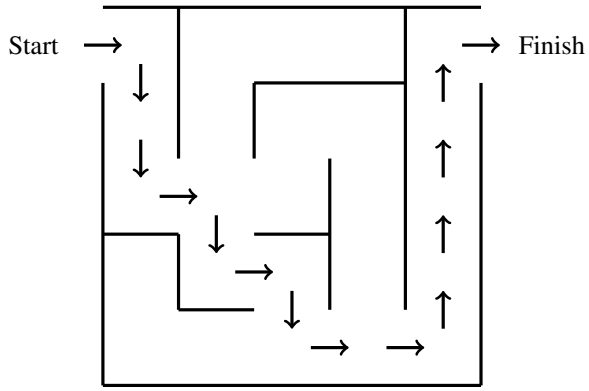


Figure 1.1: A graphical illustration of a typical MDP maze problem, where the walls are depicted by the solid black lines and the initial and end positions are appropriately labeled. A possible trajectory of the agent through the maze is depicted by the arrows.

$$p_1(\boldsymbol{s}) : \mathcal{S} \to [0,1], \qquad \text{initial state distribution,}$$

$$p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) : \mathcal{S}^2 \times \mathcal{A} \to [0,1], \qquad \text{transition dynamics,}$$

$$\pi(\boldsymbol{a}|\boldsymbol{s}) : \mathcal{A} \times \mathcal{S} \to [0,1], \qquad \text{policy,}$$

$$R(\boldsymbol{s},\boldsymbol{a}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \qquad \text{reward function.}$$

In general the state and action spaces can be arbitrary sets, but we restrict our attention to either discrete sets or subsets of $\mathbb{R}^n$, where $n \in \mathbb{N}$. When $\mathcal{S}$ and $\mathcal{A}$ are discrete then the distributions $p_1(\cdot)$ and $p(\cdot|\boldsymbol{a},\boldsymbol{s})$, $(\boldsymbol{a},\boldsymbol{s}) \in \mathcal{A} \times \mathcal{S}$, are constrained to the $|\mathcal{S}|$-simplex, while for each $\boldsymbol{s} \in \mathcal{S}$ the distribution $\pi(\cdot|\boldsymbol{s})$ is constrained to the $|\mathcal{A}|$-simplex. A similar situation holds in the continuous case where these (conditional) distributions are assumed properly defined. A common choice of transition dynamics and policy is a Gaussian with a mean that is a function of the conditioning variables. It is also assumed that the reward function is bounded. We use boldface notation to represent a vector and also use the notation $\boldsymbol{z} = (\boldsymbol{s},\boldsymbol{a})$ to denote an state-action pair.

Given an MDP the trajectory of the agent is determined by the following recursive procedure: Given the agent's state, $\boldsymbol{s}_t$, at a given time-point, $t \in \mathbb{N}_H$, an action is selected according to the policy, $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t = \boldsymbol{s})$; The agent will then transition to a new state according to the transition dynamics, $\boldsymbol{s}_{t+1} \sim p(\cdot|\boldsymbol{a}_t,\boldsymbol{s}_t)$; This process is iterated sequentially through all of the time-points in the planning horizon, where the state of the initial time-point is determined by the initial state distribution $\boldsymbol{s}_1 \sim p_1(\cdot)$. The reward of the trajectory can be obtained in conjunction to calculating a trajectory of the agent; Given the current state-action pair $(\boldsymbol{s}_t,\boldsymbol{a}_t)$ the reward at the current time-point is obtained from the reward function $R(\boldsymbol{s}_t,\boldsymbol{a}_t)$. An influence diagram representation [151] of the MDP framework for a finite planning horizon and non-stationary rewards and policies is given in fig(1.2).

As mentioned earlier the objective of the MDP framework is to maximise the amount of reward the agent can expect to receive during the course of the planning horizon. As the general MDP framework

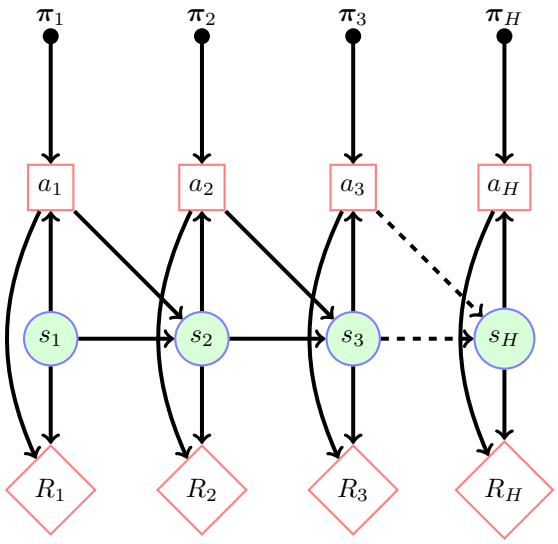


Figure 1.2: An influence diagram representation of an unconstrained finite horizon H MDP. In influence diagram notation circular nodes represent random variables, square nodes represent decision variables and diamond nodes represent (possibly stochastic) functions. The black point nodes, in this case the π's, represent the functions to be maximised. The dashed arrows are used to highlight the possibility of intermediate nodes not shown in the diagram.

allows for stochastic transition dynamics, policies and initial state distribution the most logical objective function is the total expected reward of the agent, where the expectation is taken over all possible trajectories. Given a policy, π, the total expected reward is given by

$$U(\pi) = \sum_{t=1}^{H} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a}|\pi)}\Big[R(\boldsymbol{s},\boldsymbol{a})\Big], \tag{1.1}$$

where the notation $p_t(\boldsymbol{s},\boldsymbol{a}|\pi)$ is used to represent the marginal $p(\boldsymbol{s}_t,\boldsymbol{a}_t|\pi)$ of the joint state-action trajectory distribution

$$p(\boldsymbol{s}_{1:H},\boldsymbol{a}_{1:H}|\pi) = p(\boldsymbol{a}_H|\boldsymbol{s}_H,\pi)\Big\{\prod_{t=1}^{H-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)p(\boldsymbol{a}_t|\boldsymbol{s}_t,\pi)\Big\}p_1(\boldsymbol{s}_1). \tag{1.2}$$

Under the assumption of a bounded reward function it follows that the objective function (1.1) is bounded for finite planning horizons. This is not necessarily the case when the planning horizon is infinite and the objective function has to be altered to ensure boundedness. The two most popular methods of handling infinite planning horizon are the *discounted rewards* and *average rewards*. In the discounted rewards setting the reward received at a time point is scaled down in such a way to ensure boundedness of the objective function, where the amount of scaling depends on the time point. More precisely a scalar factor, $\gamma \in [0,1)$, known as the discount factor, is introduced and the reward received at time t is scaled by γ^{t-1}. Including the introduction of this discount factor the objective function now takes the form

$$U(\pi) = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a}|\pi)}\Big[\gamma^{t-1}R(\boldsymbol{s},\boldsymbol{a})\Big]. \tag{1.3}$$

It is simple to see, through the use of a geometric progression, that if $|R(\boldsymbol{s}, \boldsymbol{a})| \leq M$ then the discounted reward objective function satisfies the bound

$$\frac{-M}{1-\gamma} \leq \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a}|\pi)}\Big[\gamma^{t-1} R(\boldsymbol{s}, \boldsymbol{a})\Big] \leq \frac{M}{1-\gamma}.$$

In the average rewards framework the objective function takes the form

$$U(\pi) = \lim_{H\to\infty} \frac{1}{H} \sum_{t=1}^{H} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a}|\pi)}\Big[R(\boldsymbol{s}, \boldsymbol{a})\Big]. \tag{1.4}$$

In the average rewards framework it is assumed that the Markov chain induced by any policy is ergodic, see *e.g.* [69]. Under this assumption for any given policy there is a unique stationary state-action distribution, denoted by $p(\boldsymbol{s}, \boldsymbol{a}; \pi)$, and (1.4) can be written in the equivalent form

$$U(\pi) = \mathbb{E}_{p(\boldsymbol{s},\boldsymbol{a}|\pi)}\Big[R(\boldsymbol{s}, \boldsymbol{a})\Big].$$

It is clear that under the assumption of a bounded reward the objective function for the average reward framework is well defined and bounded.

It is worth noting that (1.1, 1.3 & 1.4) are not the only reasonable objective functions for the MDP framework. In particular these objective functions are, what is commonly referred to as, *risk-insensitive*. In other words they only take into account the expected reward and not any sort of volatility, or risk, of the reward. Other variants of the MDP framework have been constructed to take into account certain measures of risk, such as the variance of the total expected reward. A typical example is the objective function,

$$U_{\text{rs}}^{\lambda}(\pi) = \mathbb{E}_{p(\boldsymbol{s}_{1:H},\boldsymbol{a}_{1:H}|\pi)}\Big[\exp\lambda \sum_{t=1}^{H} R(\boldsymbol{s}_t, \boldsymbol{a}_t)\Big], \tag{1.5}$$

where λ is the risk-sensitive parameter. Up to first order variation in λ we have

$$\lambda^{-1} \log U_{\text{rs}}^{\lambda}(\pi) = \mathbb{E}_{p(\boldsymbol{s}_{1:H},\boldsymbol{a}_{1:H}|\pi)}\Big[\sum_{t=1}^{H} R(\boldsymbol{s}_t, \boldsymbol{a}_t)\Big] + \lambda \text{Var}\Big[\sum_{t=1}^{H} R(\boldsymbol{s}_t, \boldsymbol{a}_t)\Big] + \mathcal{O}(\lambda^2),$$

so that, intuitively, $\lambda > 0$ encourages risk-averse behaviour, while $\lambda < 0$ encourages risk-seeking behaviour. We don't consider any such objective in this work, see *e.g.* [31] for more details on risk-sensitive control, but mention it simply to highlight the generality of the MDP framework.

1.2.2 Continuous Time Control

Having detailed some of the standard frameworks for discrete time Markov Decision Processes we now detail the the continuous time counterparts. In continuous time control the time index is usually denoted by a real-valued variable, $t \in \mathbb{R}$. The control system is considered between some initial time point, t_0 and a final time point, t_f, which can be infinite. Invariably with the transition from discrete to continuous

time the transition dynamics are now described by a set of (possibly stochastic) differential equations,

$$d\boldsymbol{s} = \boldsymbol{b}(\boldsymbol{s}(t), \boldsymbol{a}(t))dt + d\boldsymbol{\xi}, \tag{1.6}$$

where $\boldsymbol{s}$, $\boldsymbol{b}$, $d\boldsymbol{\xi}$ and $d\boldsymbol{s}$ are n-dimensional vectors and $\boldsymbol{a}$ is an m-dimensional vector which defines the control. The term $\boldsymbol{b}(\boldsymbol{s}(t), \boldsymbol{a}(t))dt$ describes the deterministic part of the transition dynamics, and is usually referred to as the drift component. The stochasticity of the differential equation comes from the term $d\boldsymbol{\xi}$, which follows a Wiener process and is known as the diffusion component. Deterministic systems are retrieved by letting the diffusion process tend to zero. More details on stochastic differential equations can be found in *e.g.* [122].

There are various possibilities on the formulation of the control problem in continuous time. The simplest possibility, and the one that will be considered here, is a fixed finite horizon in an unconstrained state space. Other possibilities include control until exit from a closed region of the state space, or control with a constraint on the state, either at the final time point or throughout the trajectory, see *e.g.* [52].

In the simplest case of control in an unconstrained state space with a finite planning horizon the optimal control problem is to find the control $\boldsymbol{a}(t)$, $t \in [t_0, t_f)$, that minimises the total expected cost,

$$U(\boldsymbol{s}_0, t_0, \boldsymbol{a}(t_0 \to t_f)) = \mathbb{E}_{p(\boldsymbol{s}(t_0 \to t_f)|\boldsymbol{a}(t_0 \to t_f), \boldsymbol{s}(0) = \boldsymbol{s}_0)}\left[\phi(\boldsymbol{s}(t_f)) + \int_{t_0}^{t_f} dt f(\boldsymbol{s}(t), \boldsymbol{a}(t), t)\right],$$

where the expectation is taken over all possible trajectories that pass through the state $\boldsymbol{s}_0$ at time t_0. The functions ϕ and f are known as the terminal and intermediate cost functions, respectively. In deterministic optimal control this expectation vanishes as all the mass is placed on a single trajectory. We now detail a simple example, based on an example in [52], to illustrate these ideas.

Example 2. *Consider the production planning of a factory producing n commodities. Using the notation $x_i(t)$, $u_i(t)$ and $d_i(t)$ to respectively denote the inventory, production and demand levels of stock i at time t. As we are considering the fully observable case the demand levels are assumed to be known to the planner. The rate of change of the inventory level is given by the differential equation*

$$\frac{d}{dt}\boldsymbol{s}(t) = \boldsymbol{a}(t) - \boldsymbol{d}(t).$$

Given an initial inventory level, $\boldsymbol{s}(t_0)$, the control problem is to optimise the production rate to minimise

$$\int_{t_0}^{t_f} \boldsymbol{s}^\top(t) A \boldsymbol{s}(t) + \boldsymbol{a}^\top(t) B \boldsymbol{a}(t) dt + \phi(\boldsymbol{s}(t_f)),$$

where A and C are positive diagonal matrices that correspond to the storage and production costs respectively, while ϕ is the terminal cost function.

At the moment there are no constraints on the state or action space, which is obviously unrealistic in such a problem. For example, the inventory levels should be restricted to be non-negative as it is not possible for a factory to store a negative amount of a commodity. Additionally there are constraints on

the amount of commodities that a factory can produce per time unit, which is usually expressed through the constraint

$$\boldsymbol{c}^\top \boldsymbol{a} \leq 1,$$

for a given constant vector $\boldsymbol{c} \succeq 0$.

1.3 Dynamic Programming

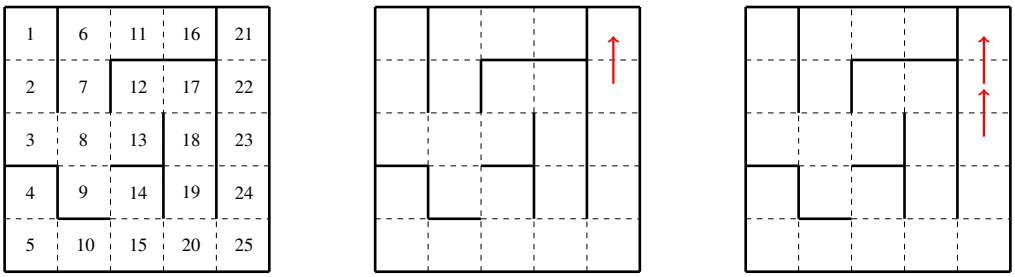
Having introduced both discrete and continuous time MDPs we now give a detailed discussion of dynamic programming, which was introduced by Richard Bellman [22], and is one of the cornerstones of optimal control and planning. Dynamic programming is very powerful tool and yet on an intuitive level it is amazingly simple. Fundamentally it is based on idea that, given the current state of the environment, the optimal action is independent of any past actions and instead only dependent on possible future actions and their effects. This idea is summarised by Bellman's *principal of optimality* [22]

> Principle of Optimality: An optimal policy has the property that whatever the current state and current decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the current decision.

To obtain an intuitive understanding of the dynamic programming principal we now give an informal description of its application to the maze problem in example(1).

Example 3. *In this example we consider a discrete version of the maze problem where there are* 25 *states and* 4 *actions, which correspond to moving right, left, up and down. The initial state is* $s = 1$*, the goal state is* $s = 21$*, see fig(1.3a), and the transition dynamics are deterministic. We consider the problem where the agent continues to move around the maze indefinitely until it reaches the goal state. The objective is to get the agent to the goal state in the shortest amount of time and this corresponds to a shortest path problem. Dynamic programming iterates backward in time and, as we are considering a shortest path problem, the state of the penultimate time-point is given by state* $s = 22$*. Given this state the optimal action is clearly to move upwards to the goal state, which is depicted in fig(1.3b). In the next iteration the agent must either be in state* $s = 22$ *or* $s = 23$*, but as the optimal action in state* $s = 22$ *is to move upwards the agent must be in* $s = 23$*. The possible next states are* $s = 22$*,* 23 *or* 24 *and due to the previous iterations of dynamic programming it is clear that the optimal action is to move upwards to state* $s = 22$*. This process is iterated backward in time until the optimal policy for the entire maze is obtained. We depict the sixth iteration of this process in fig(1.3d) and the final optimal policy is given in fig(1.3e). Note that dynamic programming gives the optimal policy for all states, even those states that will not be visited under the optimal policy. This is an aspect of dynamic programming, where the policy obtained is optimal for all possible initial state distributions.*

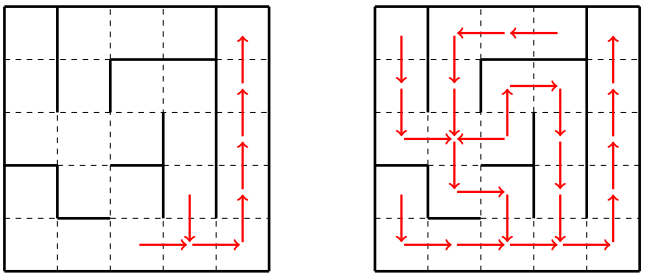
Unsurprisingly the dynamic programming solutions of the discrete and continuous time problems are fundamentally different: In discrete time problems dynamic programming leads to either a recursive update equation or a set of fixed point equations, depending on whether the planning horizon is finite or



(a) A Maze Shortest Path Problem.

(b) First Iteration of Dynamic Programming.

(c) Second Iteration of Dynamic Programming.



(d) Sixth Iteration of Dynamic Programming.

(e) Optimal Policy.

Figure 1.3: A graphical illustration of dynamic on a shortest path maze problem. (a) A graphical depiction of the Maze, with the state numbers given, where the initial and goal states given by 1 and 21. (b) The first iteration of dynamic programming gives the optimal policy for state $s = 22$, which is to move upwards to the final state. (c) The second iteration of dynamic programming gives the optimal policy for state $s = 23$, which again is to move upwards. (d & e) Dynamic programming continues to iterate backwards in time, with the sixth iteration given in (d) and the final policy given in (e).

infinite respectively; In continuous time problems the dynamic programming principal leads to a non-linear partial differential equation. We now detail the dynamic programming framework for these two cases in the next two sections.

1.3.1 Discrete Time Control

In this section we will discuss the dynamic programming paradigm for some of the standard models for the discrete time MDP framework, in particular the finite planning horizon and the discounted infinite planning horizon. There are several other models which will not be considered here, such as average reward over an infinite planning horizon, or infinite horizon problems with periodic rewards and transition dynamics, see *e.g.* [24]. To avoid obfuscating the overall simplicity of dynamic programming we ignore some of the more technical issues, such as issues of measurability.

Finite Planning Horizon

In the case where $H < \infty$ the MDP optimisation problem over $\pi_{1:H}$ takes the form

$$\max_{\pi_{1:H}} U(\pi_{1:H}) = \max_{\pi_{1:H}} \sum_{t=1}^{H} \mathbb{E}_{p_t(s,a;\pi_{1:t})} \Big[R_t(s,a) \Big]. \tag{1.7}$$

Now before proceeding to the actual optimisation problem we introduce a function that will play a key role in finite horizon dynamic programming, the *value function*, also sometimes known as the *optimal*

value function. Given a point t_1 in the planning horizon the value function *at time* t_1 is defined as

$$V^*_{t_1}(s') = \max_{\pi_{t_1:H}} \sum_{t=t_1}^{H} \mathbb{E}_{p_t(s,a;\pi_{1:t})}\Big[R_t(s,a)\big|s_{t_1}=s'\Big]. \tag{1.8}$$

As can be seen from the definition the value function gives the optimal amount of reward the agent can expect to receive, from the current time point onwards, given that it is currently in state s' at time t_1. Due to the Markovian structure of the transition dynamics (1.2) the optimal value function can be written in the form

$$V^*_{t_1}(s') = \max_{\pi_{t_1:H}} \sum_{t=t_1}^{H} \mathbb{E}_{p_t(s,a|s_{t_1}=s';\pi_{t_1:t})}\Big[R_t(s,a)\Big],$$

and is easily seen to be independent of the policy at earlier time points. Additionally, between successive time points the optimal value function satisfies the recursive relation

$$V^*_{t_1}(s) = \max_{\pi_{t_1}} \Big\{\mathbb{E}_{\pi_{t_1}(a|s)}\Big[R_{t_1}(s,a) + \mathbb{E}_{p(s'|s,a)}\Big[V^*_{t_1+1}(s')\Big]\Big]\Big\}. \tag{1.9}$$

This recursion can be shown as follows,

$$\begin{aligned}
V^*_{t_1}(s) &= \max_{\pi_{t_1}} \Big\{\mathbb{E}_{p_{t_1}(s,a|s_{t_1}=s;\pi_{t_1:t})}\Big[R_{t_1}(s,a)\Big] + \max_{\pi_{t_1+1:H}} \sum_{t=t_1+1}^{H} \mathbb{E}_{p_t(s',a'|s_{t_1}=s;\pi_{t_1:t})}\Big[R_t(s',a')\Big]\Big\}, \\
&= \max_{\pi_{t_1}} \Big\{\mathbb{E}_{p_{t_1}(s,a|s_{t_1}=s;\pi_{t_1:t})}\Big[R_{t_1}(s,a)\Big] + \mathbb{E}_{p_{t_1+1}(s'|s_{t_1}=s,a_{t_1}=a;\pi_{t_1:t})}\big[V^*_{t_1+1}(s')\big]\Big\}, \\
&= \max_{\pi_{t_1}} \Big\{\mathbb{E}_{\pi_{t_1}(a|s)}\Big[R_{t_1}(s,a) + \mathbb{E}_{p(s'|s,a)}\Big[V^*_{t_1+1}(s')\Big]\Big]\Big\}.
\end{aligned}$$

The first line uses the definition of the value function along with the independence of the first term to future polices, which allows the maximisation over future policies to be pulled through the summation. The second line conditions on the state-action pair at time t_1 and uses the definition of the value function at time $t_1 + 1$. The final line uses the Markovian dynamics, which gives

$$p_{t_1}(s,a|s_{t_1}=s;\pi_{t_1:t}) = \pi_{t_1}(a|s), \qquad p_{t_1+1}(s'|s_{t_1}=s,a_{t_1}=a;\pi_{t_1:t}) = p(s'|s,a)\pi_{t_1}(a|s).$$

The recursive formulae (1.9) forms the core of the dynamic programming principal for finite horizon problems and is known as the *Bellman equation*, or *finite horizon Bellman equation*. The fact that for each given state the policy is a distribution over $\mathcal{A}$ means that the maximum of (1.9) occurs when the policy is deterministic[4] and takes the form[5]

$$\pi^*_{t_1}(a|s) = \begin{cases} 1 & \text{if } a = \operatorname*{argmax}_{a\in\mathcal{A}} \Big\{R_{t_1}(s,a) + \mathbb{E}_{p(s'|s,a)}\Big[V^*_{t_1+1}(s')\Big]\Big\}, \\ 0 & \text{otherwise.} \end{cases}$$

[4]It is possible that there can be multiple optimal actions, in which case the policy can be stochastic. For simplicity of notation we assume that the optimum is always unique and the optimal policy is deterministic.

[5]In the case of continuous state-action spaces this distribution doesn't exist as there is no well defined distribution on a subset of the Euclidean space that has all of its mass on a single point. In this case one should really drop the notion of optimal policy and instead consider the optimal controller, *i.e.* the optimal action, but it is retained here for simplicity of exposition.

$$\pi^*_H(a|s) = \begin{cases} 1 & \text{if } a = \underset{a\in\mathcal{A}}{\operatorname{argmax}}\ R_H(a,s) \\ 0 & \text{otherwise.} \end{cases}$$

$$V^*_H(s) = \mathbb{E}_{\pi^*_H(a|s)}\Big[R_H(s,a)\Big]$$

for $t = H-1,\ldots,1$ **do**

$$\pi^*_t(a|s) = \begin{cases} 1 & \text{if } a = \underset{a\in\mathcal{A}}{\operatorname{argmax}}\ R_t(s,a) + \mathbb{E}_{p(s'|s,a)}\Big[V^*_{t+1}(s')\Big] \\ 0 & \text{otherwise.} \end{cases}$$

$$V^*_t(s) = \mathbb{E}_{\pi^*_t(a|s)}\left[R_t(a,s) + \mathbb{E}_{p(s'|s,a)}\left[V^*_{t+1}(s')\right]\right]$$

end for

Algorithm 1.1: Dynamic Programming algorithm for finite horizon MDPs with non-stationary policy.

The Bellman equation can now be written in the form

$$V^*_{t_1}(s) = \max_{a\in\mathcal{A}}\left\{R_{t_1}(s,a) + \mathbb{E}_{p(s'|s,a)}\Big[V^*_{t_1+1}(s')\Big]\right\}. \tag{1.10}$$

The Bellman equation allows one to recursively calculate the value function backwards in time. It remains to obtain a formulae for the value function at the final time point, which is required at the start of these recursions. This is simple and can be immediately obtained from (1.8), which gives

$$V^*_H(s) = \max_{\pi_H}\mathbb{E}_{\pi_H(a|s)}\Big[R_H(s,a)\Big] = \max_{a\in\mathcal{A}} R_H(a,s). \tag{1.11}$$

Hence the value functions can be calculated in linear time *w.r.t.* the planning horizon: the recursion begins at the final time point with (1.11) and then recurses backwards in time using (1.10).

Having introduced the value function and the Bellman equation we now return to the original optimisation problem. We first observe that the objective function (1.7) can be written in terms of the value function corresponding to the initial time point as follows

$$\max_{\pi_{1:H}} U(\pi_{1:H}) = \max_{\pi_{1:H}}\sum_{t=1}^{H}\mathbb{E}_{p_t(s,a;\pi_{1:t})}\Big[R_t(s,a)\Big] = \mathbb{E}_{p_1(s)}\Big[V^*_1(s)\Big].$$

Hence, to calculate the maximal value of $U(\pi_{1:H})$ it suffices to calculate the value function of the initial time point, which is most naturally done through the Bellman equation. As well as providing the maximal value of $U(\pi_{1:H})$ the value functions also provide the point at which the maximum is achieved, *i.e.* the optimal policy. This completes the dynamic programming solution to the finite horizon MDP problem and is summarised in algorithm(1.1).

Discounted Infinite Planning Horizon

While in finite horizons the dynamic programming paradigm results in the system of recursive equations (1.10) it is quite different in an infinite planning horizon with discounted rewards, where instead a fixed

point equation is obtained. The optimal controller is then obtained through the solution of this fixed point equation, which is done using either the theory of contraction mappings, and in particular the *contraction mapping fixed point theorem*, or linear programming. In this section we detail the derivation of this fixed point equation, known as the *discounted reward Bellman equation*, along with its various solutions. First, however, we introduce the idea of a value function in the discounted rewards framework.

We saw that when the planning horizon was finite the value function depended explicitly on time, or more specifically on the amount of time left in the planning horizon. In the infinite horizon setting this no longer makes sense because, regardless of the current time point, there will always be an infinite number of time steps remaining. So instead of a value function corresponding to each time point there will be a single, stationary, value function defined for all time points simultaneously. With this in mind the optimal value function for discounted rewards framework is defined as

$$V^*(s') = \max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{p_t(s,a;\pi)} \Big[\gamma^{t-1} R(s,a) \big| s_1 = s' \Big]. \tag{1.12}$$

Similarly, given a policy π the corresponding value function is defined as

$$V^{\pi}(s') = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(s,a;\pi)} \Big[\gamma^{t-1} R(s,a) \big| s_1 = s' \Big]. \tag{1.13}$$

We note that in (1.12) and (1.13) the summation begins at $t = 1$, which should be taken to mean the summation from the current time point onwards rather than the from the initial time point.

While dynamic programming in finite horizon problems centres around the set of recursive equations (1.10) in the discounted rewards problem it centres around a fixed point equation. The derivation is more protracted in the case of an infinite horizon with discounted rewards and we omit the details, but see *e.g.* [24], and the fixed-point equation takes the form

$$V^*(s) = \max_{a \in \mathcal{A}} \Big\{ R(s,a) + \gamma \mathbb{E}_{p(s'|s,a)} \Big[V^*(s') \Big] \Big\}, \tag{1.14}$$

where an analogous equation holds for V^{π}. The fixed point equation (1.14) is known as the *discounted reward Bellman equation*, or simply the Bellman equation when the context is obvious. By writing the discounted reward objective function (1.3) in terms of the optimal value function

$$\max_{\pi} U(\pi) = \max_{\pi} \sum_{t=1}^{\infty} \mathbb{E}_{p_t(s,a;\pi)} \Big[\gamma^{t-1} R(s,a) \Big] = \mathbb{E}_{p_1(s)} \Big[V^*(s) \Big],$$

it is clear that solving the original planning problem is equivalent to solving (1.14), which is usually done either through the use of contraction mappings or by linear programming. Before proceeding to these solutions we give a brief description of the theory of contraction mappings.

Intuitively a contraction mapping is a function with property that the distance between any two points in the domain is greater than the distance between the image of those points, where distance is measured by some given metric. A nice property of contraction mappings, known as the contraction

mapping fixed point theorem, is that the iterative application of a contraction mapping will lead (in the limit) to a unique fixed point. Contraction mapping methods solve the Bellman equation by defining a suitable contraction mapping on functions over $\mathcal{S}$ such that the optimal value function is the fixed point of the contraction mapping. The optimal value function, and hence the optimal controller, is then obtained by iterative use of this contraction mapping. We now makes these ideas more formal.

Given a set $\mathcal{S}$ we denote the set of all bounded real-valued functions on $\mathcal{S}$ by $\mathcal{B}(\mathcal{S})$. Note that the presence of the discount factor ensures that $V^* \in \mathcal{B}(\mathcal{S})$ and also that the value function $V^\pi \in \mathcal{B}(\mathcal{S})$ for any given policy, π. By introducing the supremum norm $\|\cdot\| : \mathcal{B}(\mathcal{S}) \to \mathbb{R}$, *i.e.*

$$\|V\| = \sup_{s \in \mathcal{S}} |V(s)|,$$

the pair $(\mathcal{B}(\mathcal{S}), \|\cdot\|)$ becomes a Banach space, *i.e.* a complete normed vector space. The formal definition of a contraction mapping is as follows:

Definition. *Given a Banach space $(\mathcal{B}(\mathcal{S}), \|\cdot\|)$ a mapping $T : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ is said to be a contraction mapping if there exists $\gamma \in [0, 1)$* s.t.

$$\|T(V) - T(V')\| \leq \gamma \|V - V'\|, \qquad \forall\, V, V' \in \mathcal{B}(\mathcal{S}).$$

As can be seen from this definition a contraction mapping reduces the distance between points in $\mathcal{B}(\mathcal{S})$, where the distance is measured by the metric $\|\cdot\|$. We now state the *contraction mapping fixed point theorem*, the proof of which can be found in standard analysis textbooks [130], which forms the basis for solving the Bellman equation via contraction mappings.

Contraction Mapping Fixed Point Theorem. *If $T : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$ is a contraction mapping then there exists a unique fixed point of T,* i.e. *there exists a unique function $V^*(s) \in \mathcal{B}(\mathcal{S})$* s.t.

$$T(V^*) = V^*.$$

Additionally, for any $T \in \mathcal{B}(\mathcal{S})$

$$\lim_{k \to \infty} \|T^k(V) - V^*\| = 0,$$

T^k denotes k successive compositions of the function T. In other words $T^k(V)$ converges uniformly to V^*.

To solve the Bellman equation through the contraction mapping fixed point theorem we now need to define a suitable contraction mapping that has the optimal value function as its fixed point. Looking at the structure of the Bellman equation an immediate candidate for such a contraction mapping is the following.

Definition. *Given a function $V : \mathcal{S} \to \mathbb{R}$ then the optimal Bellman operator, $T^* : \mathcal{B}(\mathcal{S}) \to \mathcal{B}(\mathcal{S})$, is*

defined by

$$T^*V(s) = \max_{a\in\mathcal{A}} \left[R(a,s) + \gamma \mathbb{E}_{p(\cdot|s,a)} \Big[V(s') \Big] \right].$$

A proof that T^* is a contraction mapping and that its fixed point is the optimal value function can be found in *e.g.* [24]. The two most prominent methods which use this contraction mapping are *value iteration* [22] and *policy iteration* [80]. There are more complicated hybrid versions of these algorithms, like modified policy iteration [131], that are designed for improved convergence but we don't detail them here.

Value Iteration

As the name suggests value iteration works in the space of value functions. Starting at an arbitrary initial value function, V_1, value iteration generates a sequence of value functions, $\{V_k\}_{k\in\mathbb{N}}$, through repeated application of the optimal Bellman operator, *i.e.* $V_{k+1} = T^*V_k$. Provided that $V_1 \in \mathcal{B}(\mathcal{S})$ then it is immediate that this sequence will converge to V^*. In practice it can be difficult to determine the convergence of value iteration and typically one performs iterations until either a predefined number of iterations have been completed or until a given stopping criterion is satisfied. For example one could continue performing iterations until the *Bellman residual*

$$\max_{s\in\mathcal{S}} \Big| V_{k+1}(s) - V_k(s) \Big|$$

is below some threshold, $\epsilon' = \epsilon(1-\gamma)/2\gamma$. Once the iterations have been completed the final policy is obtained

$$\pi_{k_{\text{final}}}(a|s) = \begin{cases} 1 & \text{if } a = \operatorname*{argmax}_{a\in\mathcal{A}} \Big[R(a,s) + \gamma \mathbb{E}_{p(\cdot|s,a)} V_{k_{\text{final}}}(s') \Big] \\ 0 & \text{otherwise.} \end{cases}$$

where $V_{k_{\text{final}}}$ is the value function at the final iteration. The advantage of using the Bellman residual as a convergence criterion is that the final policy will be ϵ-optimal [24], *i.e.* the total expected reward of the policy will be within ϵ of the optimum. A summary of value iteration is given in algorithm(1.2).

The computational complexity of value iteration is equal to the number of iterations necessary for optimality times the cost of performing each iteration. It is clear to see that one iteration of value iteration requires $\mathcal{O}(\mathcal{S}^2\mathcal{A})$ operations. In general there is no guarantee that the value function will converge in a finite number of iterations but it can be shown that the associated policy will converge in a finite number of iterations [24]. In fact this result can be strengthened to convergence occurring in a polynomial number of iterations [172].

Policy Iteration

As previously noted value iteration works in the space of value functions. A common alternative is policy iteration which works directly in the policy space. Starting at some initial policy, π^1, policy

Initialise an arbitrary initial value function $V \in \mathcal{B}(\mathcal{S})$.

repeat

for $s \in \mathcal{S}$ **do**

$V(s) = \max_{a \in \mathcal{A}} \left\{ R(a,s) + \gamma \mathbb{E}_{p(s'|s,a)} \left[V(s') \right] \right\}$

end for

until convergence

Obtain optimal policy using the formulae

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[R(a,s) + \gamma \mathbb{E}_{p(\cdot|s,a)} V(s') \right] \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm 1.2: Value Iteration Algorithm for Infinite Horizon MDPs with Discounted Rewards.

iteration alternates between *policy evaluation* and *policy improvement* until convergence. During policy evaluation the value function corresponding to the current policy, *i.e.* V^{π_k}, is calculated. This can be done in one of two ways: either by solving the system of linear equations

$$V^{\pi_k}(s) = \mathbb{E}_{\pi^k(a|s)} \left[R(s,a) + \gamma \mathbb{E}_{p(s'|s,a)} \left[V^{\pi_k}(s') \right] \right], \tag{1.15}$$

or by successive applications of the operator T_{π^k}

$$V_{i+1}^{\pi_k}(s) = T_{\pi^k} V_i^{\pi_k}(s) = \mathbb{E}_{\pi^k(a|s)} \left[R(s,a) + \gamma \mathbb{E}_{p(s'|s,a)} \left[V_i^{\pi_k}(s') \right] \right],$$

to some initial estimate of the value function. It is easy to show that T_{π^k} is a contraction mapping with a fixed point at V^{π_k}, again see [24], so that

$$V^{\pi^k} \approx T_{\pi^k}^N V_1^{\pi^k},$$

for arbitrary $V_1^{\pi^k} \in \mathcal{B}(\mathcal{S})$ and sufficiently large $N \in \mathbb{N}$. During the policy improvement stage the policy is updated through the application of the optimal Bellman operator to the value function of the current policy, *i.e.*

$$\pi^{k+1}(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[R(a,s) + \gamma \mathbb{E}_{p(\cdot|s,a)} \left[V_{\pi^k}(s') \right] \right] \\ 0 & \text{otherwise.} \end{cases}$$

A summary of the policy iteration algorithm is given in algorithm(1.3).

The computational complexity of policy iteration depends on three terms, the cost of policy evaluation, the cost of policy improvement and number of iterations required for optimality. If policy evaluation is done by solving the linear system (1.15) then the complexity will in general be $\mathcal{O}(\mathcal{S}^3)$, although this can be reduced if the linear system is sparse or has a certain structure. Alternatively, if the policy is

Initialise an arbitrary initial policy π^1 and set iteration count $k = 1$.

repeat

Evaluate policy by either solving the system of linear equations (1.15) or by iteratively applying the contraction mapping T_{π^k} on an approximate value function until convergence to the fixed point V_{π^k}.

Improve policy using the formulae

$$\pi^{k+1}(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[R(a,s) + \gamma \mathbb{E}_{p(\cdot|s,a)} \left[V_{\pi^k}(s') \right] \right] \\ 0 & \text{otherwise.} \end{cases}$$

Set $k \leftarrow k + 1$.

until convergence of policy

Algorithm 1.3: Policy Iteration Algorithm for Infinite Horizon MDPs with Discounted Rewards.

evaluated through iterative use of the contraction mapping T_π then the run-time is $\mathcal{O}(N\mathcal{S}^2\mathcal{A})$, where N is the number of iterations of T_π performed. It is clear that policy improvement can be performed in $\mathcal{O}(\mathcal{A}\mathcal{S}^2)$ operations. An immediate upper bound on the number of iterations required by policy iteration can be obtained by observing that there are $\mathcal{A}^\mathcal{S}$ distinct deterministic stationary policies and an iteration of policy improvement strictly doesn't decrease the value of the policy. This means that at most an exponential number of policy improvement steps are required to obtain optimality. It is too complex to detail here but this bound can be actually reduced to a polynomial bound, see *e.g.* [108] for an overview.

Linear Programming

When the state-action space is discrete there is an alternative to the contraction mapping methods, which it is solve the Bellman equation through linear programming [45]. Associating a positive scalar, $\mu(s) \geq 0$, with each state, $s \in \mathcal{S}$, the primal version of the linear program takes the form

$$\begin{aligned} &\min_{\boldsymbol{\mu}} \sum_{s \in \mathcal{S}} \mu(s) V(s) \\ &\textit{s.t. } V(s) \geq R(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) V(s'), \qquad (s,a) \in \mathcal{S} \times \mathcal{A}, \end{aligned}$$

while the dual formulation takes the form

$$\begin{aligned} &\max_{\boldsymbol{\lambda}} \sum_{a \in \mathcal{A}, s \in \mathcal{S}} \lambda(a,s) R(a,s) \\ &\textit{s.t. } \mu(j) = \sum_{a \in \mathcal{A}} \lambda(a,j) - \gamma \sum_{a \in \mathcal{A}, s \in \mathcal{S}} \lambda(a,s) p(j|a,s), \quad \forall j \in \mathcal{S}, \end{aligned}$$

where $\boldsymbol{\lambda}$ are Lagrange multipliers, which are introduced to enforce the Bellman equation constraint, and are constrained to be positive.

Observe that the primal problem has $|\mathcal{S}|$ variables and $|\mathcal{A}| \times |\mathcal{S}|$ constraints, while the dual problem has $|\mathcal{A}| \times |\mathcal{S}|$ variables and $|\mathcal{S}|$ constraints. Linear programs are solvable in polynomial time [89, 86] so

this formulation provides a polynomial time algorithm (in $\mathcal{S}$ and $\mathcal{A}$) to solve the MDP problem. However, when the linear programming solution was first introduced linear program solvers were slow in practice, even for moderately sized MDP problems, and therefore didn't enjoy the popularity of the contraction mapping methods. The speed of these methods has improved vastly with the advancement of linear program methods, such as interior point methods [32], but contraction mapping methods (or approximations thereof) still dominate the literature. One area where the linear programming formulation has enjoyed greater success than contraction mapping methods is in its elegant handling of constraints, see *e.g.* [2]. It has also enjoyed some success in the approximate dynamic programming literature, with the natural extension of this linear program to *approximate linear programs* [41].

1.3.2 Continuous Time Control

As in the discrete time framework the dynamic programming solution techniques will involve the *optimal value function*, which in this case is defined as

$$V(\boldsymbol{s}, t) = \min_{\boldsymbol{a}(t \to t_f)} U(\boldsymbol{s}, t, \boldsymbol{a}(t \to t_f)).$$

As in the discrete time framework the optimal value function satisfies a recursive formulae relating the functions at different time-points. Consider two time-points t,$t' \in [t_0, t_f)$, *s.t.* $t \leq t'$, then the optimal value function $V(\boldsymbol{s}, t)$ can be written in the form

$$V(\boldsymbol{s}, t) = \min_{\boldsymbol{a}(t \to t')} \mathbb{E}_{p(\boldsymbol{s}(t \to t') | \boldsymbol{a}(t \to t'))} \left[\int_{t_0}^{t_f} dt f(\boldsymbol{s}(t), \boldsymbol{a}(t), t) + V(\boldsymbol{s}(t'), t') \right]. \tag{1.16}$$

This recursive formulae for $V(\boldsymbol{s}, t)$ follows easily by using the linearity of integration, pulling through the minimisation and then using the definition of the optimal value function.

Considering the time-point $t' = t + dt$ and calculating the Taylor expansion of $V(\boldsymbol{s}(t'), t')$ around t, which is done to first order in dt and second order in $d\boldsymbol{s}$ due to standard Ito calculus and ignoring higher order terms, gives

$$V(\boldsymbol{s}(t'), t') = \mathbb{E}_{p(\boldsymbol{s}(t \to t') | \boldsymbol{a}(t \to t'), \boldsymbol{s}(t) = \boldsymbol{s})} \Big[V(\boldsymbol{s}, t) + \partial_t V(\boldsymbol{s}, t) dt + (\partial_{\boldsymbol{s}} V(\boldsymbol{s}, t))^\top d\boldsymbol{s} + \frac{1}{2} \text{Tr}(\partial_{\boldsymbol{s}}^2 V(\boldsymbol{s}, t) d\boldsymbol{s}^2) \Big].$$

Using the definitions of the transition dynamics (1.6) and noise process this Taylor expansion can be rewritten in the form

$$V(\boldsymbol{s}(t'), t') = V(\boldsymbol{s}, t) + \partial_t V(\boldsymbol{s}, t) dt + (\partial_{\boldsymbol{s}} V(\boldsymbol{s}, t))^\top \boldsymbol{b}(\boldsymbol{s}(t), \boldsymbol{a}(t)) dt + \frac{1}{2} \text{Tr}(\partial_{\boldsymbol{s}}^2 V(\boldsymbol{s}, t) \nu(\boldsymbol{s}, \boldsymbol{a}, t)) dt.$$

Substituting this Taylor expansion into the recursive formulae (1.16), dividing by dt and taking the limit $dt \to 0$, gives

$$-\partial_t V(\boldsymbol{s}, t) = \min_s \left(f_0(\boldsymbol{s}, \boldsymbol{a}, t) + \boldsymbol{b}(\boldsymbol{s}(t), \boldsymbol{a}(t))^\top \partial_{\boldsymbol{s}} V(\boldsymbol{s}, t) + \frac{1}{2} \text{Tr}(\nu(\boldsymbol{s}, \boldsymbol{a}, t) \partial_{\boldsymbol{s}}^2 V(\boldsymbol{s}, t)) \right). \tag{1.17}$$

This partial differential equation is known as the *Stochastic Hamilton-Jacobi-Bellman* (HJB) equation, with boundary condition $V(\boldsymbol{s}, t_f) = \phi(\boldsymbol{s})$, and forms the basis of dynamic programming stochastic optimal control in continuous time. The stochastic partial differential equation (1.17) is non-linear and, in general, very difficult to solve.

1.3.3 Linear-Quadratic Control

In the discrete time formulation it appears that dynamic programming is completely intractable as it is necessary to both store the value function and to perform a global optimisation over the action space, both of which will, in general, be intractable. Additionally, we saw in section(1.3.2) that in continuous time problems the optimal control problem can be solved through the HJB equation, which is generally difficult to solve. This, in general, makes the dynamic programming solution to continuous (space) systems intractable. There is, however, one important exception and this is the class of linear-quadratic control (LQC) problems. There are various formulations of LQC, such as discrete or continuous time, or deterministic or stochastic dynamics, but for simplicity we shall consider discrete time and deterministic transitions, where the derivations for the other formulations can be found in *e.g.* [156]. A system is said to be linear-quadratic if the transition dynamics are linear and the cost function is quadratic, *i.e.*

$$\boldsymbol{s}_{t+1} = \Phi_t \boldsymbol{s}_t + \Lambda_t \boldsymbol{a}_t \qquad \text{(transition dynamics)}$$

$$f(\boldsymbol{s}, \boldsymbol{a}, t) = \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{a} \end{bmatrix}^\top \begin{bmatrix} Q_t & M_t \\ M_t^\top & R_t \end{bmatrix} \begin{bmatrix} \boldsymbol{s} \\ \boldsymbol{a} \end{bmatrix} \qquad \text{(intermediate costs)}$$

$$\phi(\boldsymbol{s}) = \boldsymbol{s}^\top \phi_H \boldsymbol{s} \qquad \text{(terminal costs)}$$

where Φ_t, Λ_t, M_t, Q_t, R_t and ϕ_H are all matrices of the appropriate dimensions. As the transition dynamics are deterministic it is no longer necessary to take the expectation over the trajectory and the objective function takes the form

$$U(u_{1:H-1}) = \frac{1}{2}\boldsymbol{s}_H^\top \phi_H \boldsymbol{s}_H + \frac{1}{2}\sum_{t=1}^{H-1} \begin{bmatrix} \boldsymbol{s}_t \\ \boldsymbol{a}_t \end{bmatrix}^\top \begin{bmatrix} Q_t & M_t \\ M_t^\top & R_t \end{bmatrix} \begin{bmatrix} \boldsymbol{s}_t \\ \boldsymbol{a}_t \end{bmatrix}.$$

As before a dynamic programming solution is obtained by iterating backwards from the final time-point to the initial time-point. Suppose that the control from $\boldsymbol{a}^*_{H-1}(\boldsymbol{s}_{H-1})$ to $\boldsymbol{a}^*_{t+1}(\boldsymbol{s}_{t+1})$ has been optimised and that it now remains to optimise $\boldsymbol{a}_{1:t}$. Making the assumption that the maximisation problem takes the form

$$\max_{u_{1:H-1}} U(u_{1:H-1}) = \max_{u_{1:t}} \left\{ \frac{1}{2}\boldsymbol{s}_{t+1}^\top P_{t+1} \boldsymbol{s}_{t+1} + \frac{1}{2}\sum_{\tau=1}^{t} \begin{bmatrix} \boldsymbol{s}_\tau \\ \boldsymbol{a}_\tau \end{bmatrix}^\top \begin{bmatrix} Q_\tau & M_\tau \\ M_\tau^\top & R_\tau \end{bmatrix} \begin{bmatrix} \boldsymbol{s}_\tau \\ \boldsymbol{a}_\tau \end{bmatrix} \right\}, \qquad (1.18)$$

where P_{t+1} is a positive definite matrix, for which a recursive update will be detailed shortly. Similar to

section(1.3.1) the only terms in (1.18) that depend on $\boldsymbol{a}_t$ are

$$Q_t(\boldsymbol{s}_{t+1}, \boldsymbol{s}_t, \boldsymbol{a}_t) = \frac{1}{2}\boldsymbol{s}_{t+1}^\top P_{t+1}\boldsymbol{s}_{t+1} + \frac{1}{2}\begin{bmatrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{bmatrix}^\top \begin{bmatrix}Q_t & M_t \\ M_t^\top & R_t\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{bmatrix}. \tag{1.19}$$

The fact that the transition dynamics are deterministic and linear means that this can be rewritten into the form

$$Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t) = \frac{1}{2}\begin{bmatrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{bmatrix}^\top \begin{bmatrix}\Phi_t^\top P_{t+1}\Phi_t & \Phi_t^\top P_{t+1}\Lambda_t \\ \Lambda_t^\top P_{t+1}\Phi_t & \Lambda_t^\top P_{t+1}\Lambda_t\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{bmatrix} + \frac{1}{2}\begin{bmatrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{bmatrix}^\top \begin{bmatrix}Q_t & M_t \\ M_t^\top & R_t\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_t \\ \boldsymbol{a}_t\end{bmatrix}. \tag{1.20}$$

As (1.20) is quadratic in $\boldsymbol{a}_t$ it can be optimised directly and results in the optimal control law, which is linear in $\boldsymbol{s}$,

$$\boldsymbol{a}_t^*(\boldsymbol{s}_t) = -(R_t + \Lambda_t^\top P_{t+1}\Lambda_t)^{-1}(M_t^\top + \Lambda_t^\top P_{t+1}\Phi_t)\boldsymbol{s}_t, \tag{1.21}$$

$$= -C_t\boldsymbol{s}_t, \tag{1.22}$$

where C_t is known as the gain matrix and, as it is independent of $\boldsymbol{s}_t$, can be calculated off-line.

To complete the derivation it remains to calculate the recursive update of the matrices $\{P_t\}_{t=2}^H$ and to show that the assumption made in (1.18) holds. Substituting the optimal control (1.21) into (1.20) gives

$$Q_t(\boldsymbol{s}_t) = \boldsymbol{s}_t^\top \left(Q_t + \Phi_t^\top P_{t+1}\Phi_t - (M_t + \Phi_t^\top P_{t+1}\Lambda_t)(R_t + \Lambda_t^\top P_{t+1}\Lambda_t)^{-1}(M_t + \Phi_t^\top P_{t+1}\Lambda_t) \right)\boldsymbol{s}_t.$$

It can now be seen that the maximisation over u_t in (1.18) leads to

$$\max_{u_{1:H-1}} U(u_{1:H-1}) = \max_{u_{1:t-1}} \left\{ \frac{1}{2}\boldsymbol{s}_t^\top P_t\boldsymbol{s}_t + \frac{1}{2}\sum_{\tau=1}^{t-1}\begin{bmatrix}\boldsymbol{s}_\tau \\ \boldsymbol{a}_\tau\end{bmatrix}^\top \begin{bmatrix}Q_\tau & M_\tau \\ M_\tau^\top & R_\tau\end{bmatrix}\begin{bmatrix}\boldsymbol{s}_\tau \\ \boldsymbol{a}_\tau\end{bmatrix} \right\},$$

where

$$P_t = Q_t + \Phi_t^\top P_{t+1}\Phi_t - (M_t + \Phi_t^\top P_{t+1}\Lambda_t)(R_t + \Lambda_t^\top P_{t+1}\Lambda_t)^{-1}(M_t + \Phi_t^\top P_{t+1}\Lambda_t).$$

This justifies the assumption that was made in (1.18) and also provides the update equation for the matrices $\{P_t\}_{t=2}^H$, where

$$P_H = \phi_H.$$

A summary of the LQ-control algorithm for discrete finite planning horizons and deterministic transition dynamics given in algorithm(1.4).

Two of the main problems with applying dynamic programming to continuous state-action problems

$P_H = \phi_H$

for $t = H-1, \ldots, 1$ **do**

Calculate gain matrix, C_t,

$$C_t = (R_t + \Lambda_t^\top P_{t+1} \Lambda_t)^{-1} (M_t + \Lambda_t^\top P_{t+1} \Phi_t).$$

Calculate value function, P_t

$$P_t = Q_t + \Phi_t^\top P_{t+1} \Phi_t - (M_t + \Phi_t^\top P_{t+1} \Lambda_t)(R_t + \Lambda_t^\top P_{t+1} \Lambda_t)^{-1} (M_t + \Phi_t^\top P_{t+1} \Lambda_t).$$

end for

At any time-point, t, the optimal control is given by

$$\boldsymbol{a}_t^* = -C_t \boldsymbol{s}_t.$$

Algorithm 1.4: Dynamic Programming algorithm for deterministic Linear-Quadratic MDPs with a discrete finite horizon.

are finding the optimal control for each given state and representing the value function. It can now be seen that neither of these issues arise in linear-quadratic problems. Firstly the optimal control can be found for all states simultaneously and results in a linear controller that is defined through the gain matrix (1.22). Secondly, substitution of the optimal linear controller into the objective function maintains the quadratic nature of the objective function, which results in value functions that are quadratic in the state variable.

The constraints of linear dynamics and quadratic costs is obviously very restrictive in practice and extensions to more general frameworks is an area of active research. Recent examples include the *iterative Linear-Quadratic Gaussian* (iLQG) algorithm [166] and the *approximate inference control* (AICO) algorithm [168]. At each iteration of the iLQG algorithm a second order approximation of the cost function around the *maximum a posteriori* estimate of the trajectory, given the current controller, is calculated. This approximate cost function is then optimised using typical linear-quadratic control and this process is iterated until convergence. In the AICO algorithm applies approximate inference techniques, in particular Expectation Propagation [115], to a probability distribution that is closely related, but not equivalent, to the original cost function. Another approach is the so-called *path integral control*, see *e.g.* [76] and follow up papers. In this case the transition dynamics are assumed to be linear *w.r.t.* the control and of an arbitrary form *w.r.t.* the state. Similarly the cost is assumed to be quadratic *w.r.t.* the control and of an arbitrary form *w.r.t.* the state. Under this form of the transition dynamics and reward function it is possible to solve the HJB equation in the control, which leads to a non-linear partial differential equation. To remove this non-linearity [76] consider, under suitable assumptions, a particular transform which results in a linear partial differential equation that can be solved through various methods, such as sampling techniques.

1.3.4 Summary

While dynamic programming can be used to optimise discrete time MDPs (with either a finite planning horizon and non-stationary policy or an infinite planning horizon and a stationary policy) it has some severe limitations that restrict its application to either relatively small discrete domains or linear-quadratic control. Firstly, at each stage of dynamic programming it is necessary to iterate through the entire state space, either to find an optimal action or to update the value function. While this is feasible in small discrete problems it is prohibitively expensive in large environments, or unfeasible in non-linear continuous systems. Additionally, the number of states in a discrete environments increases exponentially in terms of the dimension of the state space. This means that computational complexity of dynamic programming scales exponentially in terms of the dimension of the problem, which is commonly referred to as the *curse of dimensionality*.

An additional issue concerns the application of dynamic programming to continuous control problems with either non-linear transition dynamics or non-quadratic costs. As the transition dynamics and reward function of continuous systems can be highly non-linear this results in a highly complex, non-convex, value functions. This makes the representation of the value function, through *e.g.* non-linear regression or function approximation techniques, a complex problem. The curse of dimensionality is again an issue here and the complexity of any such representation of the value scales exponentially in the dimension of the state space. Furthermore, dynamic programming also requires performing a global optimisation over the action space, which due to the non-convexity of the value function, will be an intractable problem in complex continuous systems.

It was noted in (1.3.1) and (1.3.2) that the derivation of dynamic programming requires that the transition dynamics are Markovian. It is also required that the conditioning set of the policy forms a separator set between the current action variable and the previous portion of the trajectory, and when a policy satisfies this property it is said to *Markovian*. This requirement of a Markovian policy is essential to the derivation of dynamic programming as it allows the maximisation to be pulled through the expectation in the Bellman equation, see *e.g.* the derivation of (1.9). Unfortunately, this restriction of dynamic programming to Markovian policies is severe and only holds for a few, albeit important, models. Some important models where this property doesn't hold include *blind controllers*, *memoryless controllers* and *finite state controllers*, which are typical models for partially observable environments and shall be introduced in section(1.4). In such models it is necessary to consider alternative optimisation techniques such as Expectation Maximisation or gradient-based methods, which shall be discussed in detail in chapter(2), branch and bound techniques [75, 110], sequential quadratic programs [7] and local search methods [128, 150]. An additional model where dynamic programming is also inapplicable is the *decentralised transition independent Markov Decision Process*, which is a standard model for multi-agent systems and shall be introduced in section(1.5). Typically these models are optimised either through Expectation Maximisation, gradient-based methods or multi-linear programming [126].

A final major issue with dynamic programming, already touched upon in section(1.1), is a need for a complete model of the environment. This is an important issue as the creation of a model is in itself a

difficult and complex task for many complex control systems. Additionally, any errors in the model may have an adverse affect on any controller that is obtained through the model. This is especially important in algorithms such as dynamic programming, where the highly non-linear $\max$-operator in the Bellman equation makes the algorithm sensitive to minor flaws in the model.

1.4 Partially Observable Markov Decision Processes

While the Markov Decision Process is a very general model it assumes that the agent has complete knowledge of the environment when making its decisions, which can be an unrealistic assumption in many applications. To illustrate the point let us reconsider the maze problem considered in example(1). When we considered this example from the MDP perspective we assumed that the agent was always aware of its current state, *i.e.* its position in the maze. Obviously this is a major assumption in the model that wouldn't always hold true in real life; For example the agent might instead have access only to limited local representations of the maze, *e.g.* the surrounding wall configurations, or it could simply become confused and lose track of its position. Problems such as these, where the agent has only incomplete knowledge of the current state of the environment, are known generally as partially observable problems. The main model for partially observable environments is the *Partially Observable Markov Decision Process* (POMDP).

As the agent has only partial knowledge of the environment it is typical to introduce a latent variable to represent the agent's memory, or belief. This latent variable, usually referred to as the *belief*, is used to model the agent's 'belief' of the true state of the environment, *i.e.* the agent's understanding of the true state of the environment. Again referring to example(1) this belief would in some way correspond to the agent's ideas about its position in the maze. There are various methods to model the agent's belief, and we shall consider several such methods shortly, but the underlying structure of the environment is assumed to be the same regardless of the agent's internal belief mechanism. In a POMDP this underlying structure is given by the initial state distribution, transition dynamics, reward function and observation process. The first three of the functions take the same form as in the MDP, *i.e.*

$$p_{s_1}(\boldsymbol{s}) : \mathcal{S} \to [0,1], \qquad \text{initial state distribution,}$$

$$p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) : \mathcal{S}^2 \times \mathcal{A} \to [0,1], \qquad \text{state transition dynamics,}$$

$$R(\boldsymbol{a},\boldsymbol{s}) : \mathcal{A} \times \mathcal{S} \to \mathbb{R}, \qquad \text{reward function.}$$

Now, however, the agent doesn't observe the state of the environment but instead some observation from an observation space, $\mathcal{O}$. The observation at each time point depends on the current state of the environment and follows a (possibly stochastic) observation process given by

$$p(\boldsymbol{o}|\boldsymbol{s}) : \mathcal{O} \times \mathcal{S} \to [0,1], \qquad \text{observation process.}$$

The objective of the agent remains the same and consists of modelling its behaviour in such a manner so as to maximise the total expected reward. The exact form of the objective function depends on the

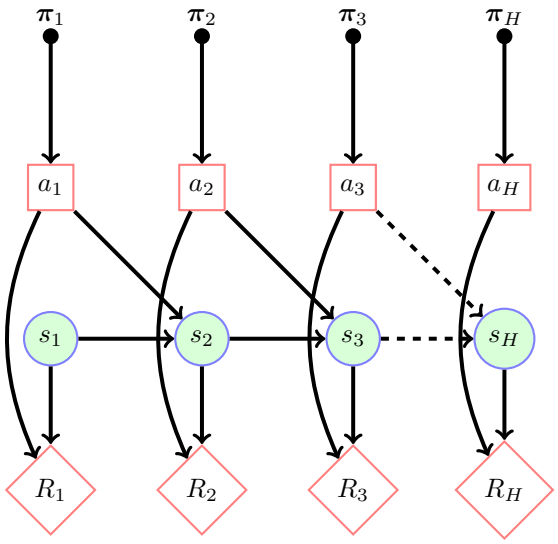


Figure 1.4: An influence diagram representation of an unconstrained finite horizon H blind Controller. The notation is the same as that of fig(1.2).

manner in which the agent's belief is modelled, which also has a dramatic effect on the optimisation process. We shall now consider the following different methods for modelling the agent's belief: *blind controllers*, *memoryless controllers*, *finite state controllers*. This list is not exhaustive and indeed it doesn't include the most general model for partially observable environments, also known as the partially observable Markov Decision Processes, where the belief is modelled as a distribution over the state space. This is the original formulation of planning under partial observability, see *e.g.* [82] for an overview, but we do not consider this model in this work and so a description of the model is omitted.

1.4.1 Blind Controllers

A *blind controller* (BC) is the simplest model of a partially observable environment where the agent is not only unaware of the current state, but also makes no observation of the environment and has no internal belief mechanism. In other words the conditioning set of the policy is empty and the policy takes the form $\pi(\boldsymbol{a})$. In this simple framework the objective function takes the form

$$U(\pi) = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a};\pi)} \Big[\gamma^{t-1} R(\boldsymbol{s}, \boldsymbol{a}) \Big],$$

where the trajectory distribution is given by

$$p(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}; \pi) = p(\boldsymbol{a}_H; \pi) \Big[\prod_{t=1}^{H-1} p(\boldsymbol{s}_{t+1} | \boldsymbol{s}_t, \boldsymbol{a}_t) p(\boldsymbol{a}_t; \pi) \Big] p_{s_1}(\boldsymbol{s}_1).$$

A influence diagram representation of the blind controller framework is given in fig(1.4).

1.4.2 Memoryless Controllers

A *memoryless controller* (MC) still has no internal belief mechanism, but its policy has an additional level of complexity and decisions are now based on the current observation. In other words the condi-

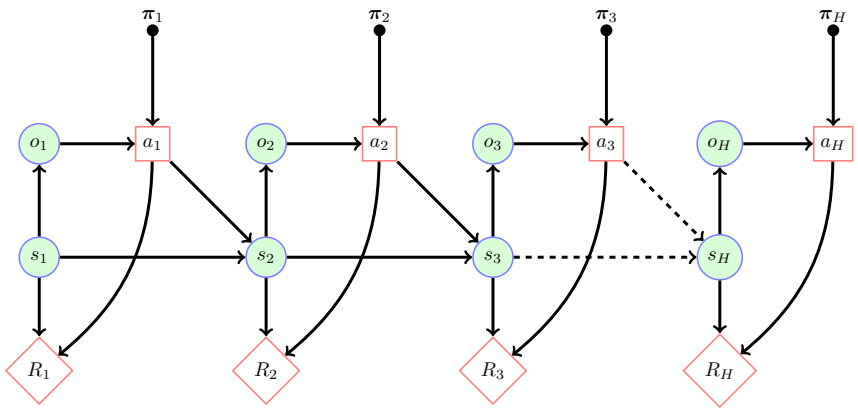


Figure 1.5: An influence diagram representation of an unconstrained finite horizon memoryless controller. Again the influence diagram notation is used, see fig(1.2). In this case the random variables are states and observations, while the decision variables are the actions. The reward function can be seen to have the same structure are in the MDP framework.

tioning set of the policy now consists of the observation and the policy takes the form $\pi(\boldsymbol{a}|\boldsymbol{o})$. In the memoryless controller framework the objective function takes the form

$$U(\pi) = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a};\pi)}\Big[\gamma^{t-1} R(\boldsymbol{s},\boldsymbol{a})\Big],$$

where the trajectory distribution is given by

$$p(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, \boldsymbol{o}_{1:H}; \pi) = p(\boldsymbol{a}_H|\boldsymbol{o}_H;\pi)p(\boldsymbol{o}_H|\boldsymbol{s}_H)\Big[\prod_{t=1}^{H-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t,\boldsymbol{a}_t)p(\boldsymbol{a}_t|\boldsymbol{o}_t;\pi)p(\boldsymbol{o}_t|\boldsymbol{s}_t)\Big]p_{s_1}(\boldsymbol{s}_1).$$

An influence diagram representation of the memoryless controller framework is given in fig(1.5).

1.4.3 Finite State Controllers

In a *finite state controller* (FSC) the agent's belief is modelled using an auxiliary discrete variable $\boldsymbol{b} \in \mathcal{B}$, where $\mathcal{B}$ is a discrete set. The agent's belief at the initial time point is determined by the initial belief distribution, $\nu(\boldsymbol{b})$, which is a parameter of the system. As the agent is unaware of the current state it instead makes its decision based on the its current belief and the latest observation, *i.e.* $\pi(\boldsymbol{a}|\boldsymbol{b},\boldsymbol{o})$. The agent then updates its belief, again based on the its current belief and the latest observation, $\eta(\boldsymbol{b}'|\boldsymbol{b},\boldsymbol{o})$, and this is a parameter of the system which must be optimised. In summary we have the functions

$$\nu(\boldsymbol{b}) : \mathcal{B} \to [0,1], \qquad \text{initial belief distribution,}$$

$$\pi(\boldsymbol{a}|\boldsymbol{b},\boldsymbol{o}) : \mathcal{A}\times\mathcal{B}\times\mathcal{O} \to \mathbb{R}, \qquad \text{policy,}$$

$$\eta(\boldsymbol{b}'|\boldsymbol{b},\boldsymbol{o}) : \mathcal{B}^2\times\mathcal{O} \to [0,1], \qquad \text{belief transition dynamics.}$$

These functions are parameters of the model to be optimised *w.r.t.* the objective function being used in the model. We have given stationary versions of these function, but of course it is possible to use

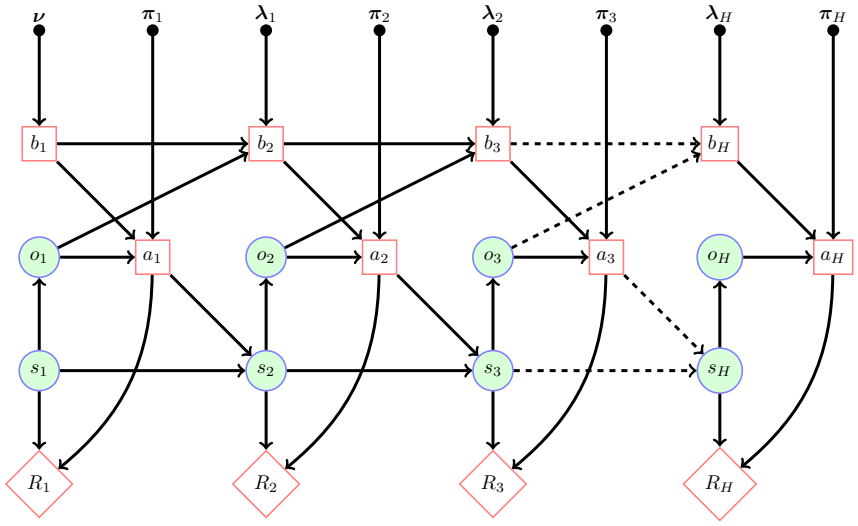


Figure 1.6: An influence diagram representation of an unconstrained finite horizon FSC. Again the influence diagram notation is used, see fig(1.2). In this case the random variables are states and observations, while the decision variables are the beliefs and actions. The introduction of additional decision variables leads to the introduction of extra functions that need to be optimised, *i.e.* the λ's and ν. The reward function can be seen to have the same structure are in the MDP framework.

non-stationary versions. As in the MDP model the objective function typically used is the total expected reward, which now takes the form

$$U(\nu, \eta, \pi) = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{s},\boldsymbol{a}|\nu,\eta,\pi)} \left[\gamma^{t-1} R(\boldsymbol{s}, \boldsymbol{a}) \right], \tag{1.23}$$

where the trajectory distribution is given by

$$\begin{aligned} p(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H}, \boldsymbol{o}_{1:H}, \boldsymbol{b}_{1:H}|\nu, \eta, \pi) &= p(\boldsymbol{a}_H|\boldsymbol{b}_H, \boldsymbol{o}_H; \pi) p(\boldsymbol{o}_H|\boldsymbol{s}_H) \\ &\times \left[\prod_{t=1}^{H-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) p(\boldsymbol{b}_{t+1}|\boldsymbol{b}_t, \boldsymbol{o}_t; \eta) p(\boldsymbol{a}_t|\boldsymbol{b}_t, \boldsymbol{o}_t; \pi) p(\boldsymbol{o}_t|\boldsymbol{s}_t) \right] p_{s_1}(\boldsymbol{s}_1) p_{b_1}(\boldsymbol{b}_1; \nu). \end{aligned} \tag{1.24}$$

A influence diagram representation of the FSC model is given in fig(1.6).

1.5 Decenteralised Transition Independent Markov Decision Processes

A final model that we shall consider at various points in this work is the Transition Independent Decenteralised Markov Decision Process (DEC-MDP). This is a model from the multi-agent planning literature, see *e.g.* [126], that has received a lot of research interest in recent years. The model makes two primary assumptions: (i) The transition dynamics of the agents are mutually independent; (ii) The actions of each agent are based only upon its internal representation of the environment and there is no communication between the agents. A typical example of a DEC-MDP is the *mars rover* problem, see *e.g.* [126], where a group of planetary robots have to explore the surface of an unknown planet in an optimal manner. The robots are so small in comparison to the search area the transition independence is a reasonable assumption, while communication between the robots is extremely expensive and hence undesirable.

As in POMDPs there are various possibilities in the modelling of each agent's policy, such as finite state controllers, but for simplicity we consider a policy that is based on the agent's current state. Denoting the state-action pair of the i^{th} agent by $\boldsymbol{z}^i = (\boldsymbol{s}^i, \boldsymbol{a}^i)$ the decentralised assumption means that the collective policy of N agents takes the constrained form

$$p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{\pi}) = \prod_{n=1}^{N} \pi^n(\boldsymbol{a}^n|\boldsymbol{s}^n), \tag{1.25}$$

which, due to the assumption of transition independence, means the trajectory distribution takes the form

$$p(\boldsymbol{s}_{1:H}, \boldsymbol{a}_{1:H};\boldsymbol{\pi}) = \prod_{n=1}^{N} p(\boldsymbol{s}^n_{1:H}, \boldsymbol{a}^n_{1:H};\pi^n).$$

To obtain cooperation between the agents a global reward function is defined and the DEC-MDP objective function takes the form

$$U(\pi) = \sum_{t=1}^{\infty} \mathbb{E}_{\prod_{n=1}^{N} p(\boldsymbol{s}^n_{1:H}, \boldsymbol{a}^n_{1:H};\pi^n)} \Big[\gamma^{t-1} R(\boldsymbol{s}_t, \boldsymbol{a}_t)\Big].$$

As with many of the other models for partially observability the DEC-MDP cannot be optimised through dynamic programming. A naive application of dynamic programming to this model would break the independence constraint (1.25). Popular optimisation techniques for these models are the policy-search methods of chapter(2), which only offer local optimality, or multi-linear programming methods [126], which offer global optimality but are NP-hard to solve in general.

1.6 Summary

This chapter has provided a very brief overview of the various models, both in the fully and partially observable environments, that we shall consider during the course of this work. There are models that we have mentioned only in passing, such as the most general form of POMDP, and some models that we have not mentioned at all, such as *predictive state representations* [109]. Additionally, we have provided a detailed discussion of dynamic programming. This is not an exhaustive introduction into dynamic programming, having omitted issues such as measurability or periodicity of transition dynamics, but it is sufficient to lay the groundwork for the rest of this text. We have also highlight some of the more prominent flaws in dynamic programming and in section(1.1) we touched upon the main contributions of this work.

Chapter 2

Parametric Policy Search Methods : Introduction

2.1 Introduction

Although dynamic programming is one of the cornerstones of planning and control it is infeasible to implement in many cases of interest. One of the main difficulties in implementing dynamic progamming is the *curse of dimensionality*, where the complexity of constructing the value function scales exponentially in the dimension of the state-action space. Furthermore, the extension of dynamic programming to non-linear systems is intractable in practice, where the non-linear dynamics perclude a closed form representation of the value function and typically cause it to have a complex functional form. Additionally it is not possible to apply dynamic programming to models with a non-Markovian policies, such as POMDPs where the belief is modeled through a finite state controller. While there are numerous alternative optimisation methods, such as approximate dynamic programming methods [26] or Monte-Carlo tree-search methods [95], we shall concentrate on *parametric policy search* methods in this chapter and chapters(3 & 4). There are advantages and disadvantages to all of these different optimisation methods, which we do not detail here, but parametric policy search methods are a popular approach and have been successfully applied to a wide range of complex problems.

Policy search method is a general term used to describe MDP optimisation techniques that work directly in the policy space. We use the term *parametric policy search methods* to include gradient-based methods, such as steepest gradient ascent [66, 67, 134, 135, 19, 113, 162, 29, 99, 180, 68] and natural gradient ascent [83, 123, 29, 13], along with Expectation Maximisation [40, 170, 171, 102, 176, 77, 94, 93, 57, 56], which is a bound optimisation technique from the Statistics literature [44]. In these methods the policy is given some differentiable parametric representation, which results in the MDP objective function being defined over the parameter space. The resulting objective function is then directly optimsed, either by taking steps in a direction of ascent, *w.r.t.* the objective function, or by optimising a lower-bound on the objective function. There are several immediate advantages to such approaches, which include stability of the policy performance during the training process, which can be important in on-line learning scenarios, as well as general convergence guarantees and the ease with which these methods can be applied to other planning models.

Like many iterative optimisation techniques, such as policy iteration, parametric policy search methods can be naturally considered as a two stage iterative procedure, alternating between an *evaluation stage* and an *improvement stage*. In the evaluation stage, which is analogous to the policy evaluation stage of policy iteration, the statistics necessary for a parameter update, such as the gradient of the objective function, are calculated. Typically, when considering complex real world control and planning problems, this evaluation stage is an intractable problem and has led to much research into various approximation techniques. Model-free stochastic approximations [113, 19] have been the methods of choice since their introduction, along with their corresponding variance reduction [180, 68] and function approximation techniques [162, 29, 99]. More recently there has been much research in model-based inference methods [170, 77, 171, 169, 168, 167, 104, 102, 59] which are based on probabilistic inference methods. In the improvement stage, which is analogous to the policy improvement stage of policy iteration, the policy parameters are updated by taking a step in the parameter space, where the update depends on the particular parametric policy search method. As we have just stated, in most MDPs of interest the search direction, or the objective function, for that matter, cannot be evaluated exactly. These difficulties cause problems in terms of the gradient-based algorithm that can be applied to such MDP objectives. In particular we are restricted to methods that are applicable to non-concave optimisation problems where, additionaly, neither the objective nor the search direction can be evaluated exactly. This precludes many advanced optimisation techniques, such as non-linear conjugate-gradient [55] or quasi-Newton methods [38, 53]. Stochastic versions of non-linear conjugate gradients and quasi-Newton methods exist, see *e.g.* [147, 146], but they require certain restrictive properties to hold. For instance, stochastic quasi-Newton methods require the objective to be strongly concave to ensure that the secant equation is satisfied. This condition is not typically met in the MDP framework. Due to these issues the parametric policy search methods typically considered in the literature are steepest gradient ascent, natural gradient ascent and Expectation Maximisation.

In this work we present three theoretical contributions to this area of parametric policy search methods. The first contribution, which we shall consider in chapter(3), is a new family of model-based techniques to perform the evaluation stage of parametric policy search methods. The second and third contributions are related to the search direction of current parametric policy search methods and shall be considered in chapter(4). The first of these two contributions is a novel analysis of the search directions of current parametric policy search methods. In particular, we relate the search directions of natural gradient ascent and Expectation Maximisation to a particular form of approximate Newton method. Motivated by this analysis we then make the third and final contribution in this area, the introduction of a novel parametric policy search method. The method is analogous to the Gauss-Newton method [121] for non-linear least squares problems, where only certain terms of the Hessian, rather than the entire Hessian itself, are used when preconditioning the gradient.

We complete this chapter by providing the details of current parametric policy search techniques necessary for an understanding of our contributions to this area of research. In particular in section(2.2) we shall detail steepest gradient ascent, while in section(2.3) we shall detail natural gradient ascent and

then we shall introduce Expectation Maximisation in section(2.4). As we are considering parametric optimisation algorithms we shall assume that the policy is given some differentiable parametric form, which we denote by $\pi(\cdot|\cdot;\boldsymbol{w})$, where the parameter vector is denoted by $\boldsymbol{w} \in \mathcal{W}$ and $\mathcal{W}$ is used to denoted the parameter space. The following derivations of steepest gradient ascent, natural gradient ascent and Expectation Maximisation are applicable, with appropriate minor alterations, to all three frameworks considered in chapter(1), *i.e.* either a finite horizon or an infinite planning horizon with either discounted or average rewards, but for notational ease we concern ourselves with the infinite horizon framework with discounted rewards. Writing the objective function and trajectory distribution directly in terms of the parameter vector then, for any $\boldsymbol{w} \in \mathcal{W}$, the objective function takes the form

$$U(\boldsymbol{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{a},\boldsymbol{s};\boldsymbol{w})}\Big[\gamma^{t-1} R(\boldsymbol{a},\boldsymbol{s})\Big], \tag{2.1}$$

where used the notation $p_t(\boldsymbol{a},\boldsymbol{s};\boldsymbol{w})$ to represent the marginal $p(\boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a};\boldsymbol{w})$ of the joint state-action trajectory distribution

$$p(\boldsymbol{a}_{1:H}, \boldsymbol{s}_{1:H};\boldsymbol{w}) = \pi(\boldsymbol{a}_H|\boldsymbol{s}_H;\boldsymbol{w})\Big\{\prod_{t=1}^{H-1} p(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t,\boldsymbol{s}_t)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t;\boldsymbol{w})\Big\}p_1(\boldsymbol{s}_1), \quad H \in \mathbb{N}. \tag{2.2}$$

Before we proceed to the derivation of the various parametric policy search algorithms we introduce the following technical assumption on the policy parameterisation.

Assumption 1. For each $t \in \mathbb{N}$, for each trajectory in the state-action space, $\boldsymbol{z}_{1:t}$, and for all $\boldsymbol{w} \in \mathcal{W}$ the derivative, $\nabla_{\boldsymbol{w}} p(\boldsymbol{z}_{1:t};\boldsymbol{w})$, exists. Additionally, the components of $\nabla \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})$ are uniformly bounded by some $M \in \mathbb{R}$.

2.2 Steepest Gradient Ascent

In this section we introduce steepest gradient ascent for the Markov Decision Processes objective. The derivation is based on the likelihood-ratio method [66, 67, 182, 19], which originates from the statistics literature and is also commonly refered to as the *log-trick*. There is also an equivalent method that is widely used to calculate the derivative of the $\log$-marginal likelihood in latent variable models, such as the Hidden Markov Model and the Linear Dynamical System, see *e.g.* [140, 15]. These techniques can easily be extended to models where dynamic programming is inapplicable, and to highlight this point we shall also briefly detail their application to POMDPs where the belief is modeled through a finite state controller. The extension of these techniques to the other planning models introduced in chapter(1) follows through similar arguments.

Steepest gradient ascent optimises (2.1) by taking steps in the parameter space in the direction of the gradient of (2.1), *i.e.*

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w} + \alpha \nabla_{\boldsymbol{w}} U(\boldsymbol{w}), \tag{2.3}$$

where $\alpha \in \mathbb{R}^+$ is the step-length parameter. When the updates (2.3) are calculated exactly and the step-length sequence is selected in an appropriate manner, *e.g.* through a line search satisfying the

Wolfe conditions, then this procedure is globally convergent[1] [121]. Similarly, in the stochastic setting convergence (in probability) is guaranteed under certain technical conditions, such as using a step-size sequence satisfying the Robbins-Monro conditions. See [103] for more details. As the state-action occupancy marginals depend on the parameter vector in a highly non-linear manner there is no simple closed form expression for the gradient of (2.1). As previously mentioned, however, it is possible to calculate the gradient using iterative, message-passing type, procedures based on likelihood-ratio type methods. We now formalise this point with the following theorem, which is generally known as the *policy gradient theorem* [162].

Theorem 1. Suppose that the policy is given some differentiable parametric form, where the parameter vector is given by $\boldsymbol{w} \in \mathcal{W}$. Provided that the reward function is uniformly bounded over the state-action space and the policy parameterisation satisfies assumption(1) then the gradient of the objective function for the infinite horizon discounted reward framework takes the following form,

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big], \tag{2.4}$$

where we use the expectation notation $\mathbb{E}[\cdot]$ to denote the integral/summation *w.r.t.* a non-negative function. The term $p_\gamma(\boldsymbol{z};\boldsymbol{w})$ is a geometric weighted average of state-action occupancy marginals given by

$$p_\gamma(\boldsymbol{z};\boldsymbol{w}) = \sum_{t=1}^{\infty} \gamma^{t-1} p_t(\boldsymbol{z};\boldsymbol{w}),$$

while the term $Q(\boldsymbol{z};\boldsymbol{w})$ is referred to as the *state-action value function* and is equal to the total expected future reward from the current time-point onwards, given the current state-action pair, $\boldsymbol{z}$, and parameter vector, $\boldsymbol{w}$, *i.e.*

$$Q(\boldsymbol{z};\boldsymbol{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p_t(\boldsymbol{z}';\boldsymbol{w})}\Big[\gamma^{t-1} R(\boldsymbol{z}')\Big|\boldsymbol{z}_1 = \boldsymbol{z}\Big].$$

Proof. The first point of the proof is to note is that for any $t \in \mathbb{N}$ we have the following identity, often referred to as the 'log-trick',

$$\nabla_{\boldsymbol{w}} p(\boldsymbol{z}_{1:t};\boldsymbol{w}) = p(\boldsymbol{z}_{1:t};\boldsymbol{w}) \nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}),$$

where this equality holds under assumption(1). Upon interchanging the order of integration and differentiation the gradient takes the form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p(\boldsymbol{z}_{1:t};\boldsymbol{w})}\Big[\gamma^{t-1} R(\boldsymbol{z}_t) \nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big].$$

Due to the Markovian structure of the trajectory distribution (2.2) this derivative can be written in the

[1] Here, as in [121], we use the term globally convergent to mean that the sequence, $\{\|\nabla U(\boldsymbol{w}_k)\|\}_{k\in\mathbb{N}}$, is guaranteed to satisfy the limit $\lim_{k\to\infty} \|\nabla U(\boldsymbol{w}_k)\| = 0$, for any $\boldsymbol{w} \in \mathcal{W}$. This means that sequence generated by such a procedure is gauranteed to converge to a stationary point of the objective function.

equivalent form

$$\begin{aligned}\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) &= \sum_{t=1}^{\infty}\sum_{\tau=1}^{t}\mathbb{E}_{p(\boldsymbol{z}_\tau,\boldsymbol{z}_t;\boldsymbol{w})}\Big[\gamma^{t-1}R(\boldsymbol{z}_t)\nabla_{\boldsymbol{w}}\log\pi(\boldsymbol{a}_\tau|\boldsymbol{s}_\tau;\boldsymbol{w})\Big],\\ &= \sum_{\tau=1}^{\infty}\sum_{t=\tau}^{\infty}\mathbb{E}_{p(\boldsymbol{z}_\tau,\boldsymbol{z}_t;\boldsymbol{w})}\Big[\gamma^{t-1}R(\boldsymbol{z}_t)\nabla_{\boldsymbol{w}}\log\pi(\boldsymbol{a}_\tau|\boldsymbol{s}_\tau;\boldsymbol{w})\Big],\end{aligned}$$

where the second line follows from the first through an interchange of the summations. The chain structure of the trajectory distribution allows the expectation over the marginals of the two time-points of the trajectory distribution to be written as follows

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \sum_{\tau=1}^{\infty}\mathbb{E}_{p_\tau(\boldsymbol{z};\boldsymbol{w})}\Bigg[\sum_{t=\tau}^{\infty}\mathbb{E}_{p_t(\boldsymbol{z}';\boldsymbol{w})}\Big[\gamma^{t-1}R(\boldsymbol{z}')\Big|\boldsymbol{z}_\tau=\boldsymbol{z}\Big]\nabla_{\boldsymbol{w}}\log\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Bigg], \tag{2.5}$$

where we have used the notation $p_\tau(\boldsymbol{z};\boldsymbol{w}) \equiv p(\boldsymbol{z}_\tau=\boldsymbol{z};\boldsymbol{w})$, for $\tau\in\mathbb{N}$. The summation over the inner expectation in (2.5) can be seen to be equal to the state-action value function scaled by $\gamma^{\tau-1}$, *i.e.*

$$\gamma^{\tau-1}Q(\boldsymbol{z};\boldsymbol{w}) = \sum_{t=\tau}^{\infty}\mathbb{E}_{p_t(\boldsymbol{z}';\boldsymbol{w})}\Big[\gamma^{t-1}R(\boldsymbol{z}')\Big|\boldsymbol{z}_\tau=\boldsymbol{z}\Big].$$

Inserting this form for this inner expectation into (2.5) gives

$$\begin{aligned}\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) &= \sum_{\tau=1}^{\infty}\mathbb{E}_{p_\tau(\boldsymbol{z};\boldsymbol{w})}\Big[\gamma^{\tau-1}Q(\boldsymbol{z};\boldsymbol{w})\nabla_{\boldsymbol{w}}\log\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big],\\ &= \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}}\log\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big],\end{aligned}$$

where the second line follows from the definition of $p_\gamma(\boldsymbol{z};\boldsymbol{w})$. This completes the derivation of (2.4). □

Although we are not going to provide the analagous derivations for the gradient of the finite horizon and infinite horizon average reward frameworks it is useful to note the form of these gradients. In particular we have the following two theorems.

Theorem 2. Suppose that the policy is given some differentiable parametric form, where the parameter vector is given by $\boldsymbol{w}\in\mathcal{W}$. Provided that the reward function is uniformly bounded over the state-action space and the policy parameterisation satisfies assumption(1) then the gradient of the objective function for the finite horizon framework takes the following form,

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \sum_{t=1}^{H}\mathbb{E}_{p_t(\boldsymbol{z};\boldsymbol{w})Q_t(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}}\log\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big], \tag{2.6}$$

where the term $Q_t(\boldsymbol{z};\boldsymbol{w})$ is referred to as the *state-action value function* and is equal to the total expected future reward from the t^{th} time-point onwards, given that the state-action pair at the t^{th} time-point is given

$\boldsymbol{z}$ and parameter vector is $\boldsymbol{w}$, *i.e.*

$$Q_t(\boldsymbol{z};\boldsymbol{w}) = \sum_{\tau=t}^{H} \mathbb{E}_{p_\tau(\boldsymbol{z}';\boldsymbol{w})}\Big[R(\boldsymbol{z}')\Big|\boldsymbol{z}_t = \boldsymbol{z}\Big].$$

Proof. The proof is analagous to the proof of theorem 1 and is omitted. □

Theorem 3. Suppose that the policy is given some differentiable parametric form, where the parameter vector is given by $\boldsymbol{w} \in \mathcal{W}$. Provided that the reward function is uniformly bounded over the state-action space, the policy parameterisation satisfies assumption(1) and there exists a unique stationary state-action occupancy distribution for each $\boldsymbol{w} \in \mathcal{W}$ then the gradient of the objective function for the infinite horizon average reward framework takes the following form,

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \mathbb{E}_{p(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big], \tag{2.7}$$

where the term $p(\boldsymbol{z};\boldsymbol{w})$ is the stationary state-action occupancy distribution induced by the policy parameters, $\boldsymbol{w}$, *i.e.*

$$p(\boldsymbol{z};\boldsymbol{w}) = \lim_{t\to\infty} p_t(\boldsymbol{z};\boldsymbol{w}),$$

while the term $Q(\boldsymbol{z};\boldsymbol{w})$ is referred to as the *state-action value function* and is equal to the average expected reward from the current time-point onwards, given the current state-action pair, $\boldsymbol{z}$, and parameter vector, $\boldsymbol{w}$, *i.e.*

$$Q(\boldsymbol{z};\boldsymbol{w}) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}_{p_t(\boldsymbol{z}';\boldsymbol{w})}\Big[R(\boldsymbol{z}')\Big|\boldsymbol{z}_1 = \boldsymbol{z}\Big].$$

Proof. The proof is analagous to the proof of theorem 1 and is omitted. □

It can be seen from (2.4) that to calculate the gradient of (2.1) it is sufficient to calculate the expectation of the derivative of the log-policy, where the expectation is taken *w.r.t.* to the non-negative function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$. In complex real world problems, such as difficult non-linear robotic manipulation tasks, for example, the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$ is highly complex and performing this expectation is intractable. This means that the expectation in (2.4) instead has to be approximated using various techniques, such as sample-based methods. We do not give a detailed dicussion of such approximation techniques here, instead delaying such a discussion to chapter(3), but there one point that is worth highlighting now. In particular, there is an interesting and important difference between the evaluation stage of parametric policy search methods and the analagous evaluation stage in dynamic programming methods, such as policy iteration. To perform an update of the policy in a method such as policy iteration it is necessary to calculate the value function over the entire state space. By contrast, in parametric policy search algorithms it is necessary only to calculate the projection of the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$ onto the space spanned by $\nabla_{\boldsymbol{w}} \log \pi(\cdot|\cdot;\boldsymbol{w})$. For example, in a problem with a continuous state-action space and a linear controller, *i.e.* $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) = \mathcal{N}(\boldsymbol{a}|K\boldsymbol{s};\Sigma^{-1})$ where the policy parameters are given by $\boldsymbol{w} = K$, then it is necessary to calculate only the first two moments of the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$ to perform a parameter update.

In (2.4) we wrote the gradient as the expectation of the derivative of the $\log$-policy, where the expectation is taken *w.r.t.* to the non-negative function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$. Considering that this equation could just as well have been written in the form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) Q(\boldsymbol{z};\boldsymbol{w})\Big],$$

this choice of notation is in need of some justification. We do not have the necessary background in the material to provide this justification at present, but we shall do so during the course of this chapter and chapters(3 & 4).

Finite State Controllers

Although we derived steepest gradient ascent in terms of Markov Decision Processes it is easy to extend these algorithms to other planning models. As an example let us consider a POMDP where the belief is modeled with a finite state controller, where extensions to the other partially observable models in sections(1.4 & 1.5) follow similarly. In this case we have that $\boldsymbol{z} = (\boldsymbol{a}, \boldsymbol{s}, \boldsymbol{b}, \boldsymbol{o})$ and the controllers to be optimised are the initial belief distribution, belief transition dynamics and the policy, where we denote the parameters of these controllers by $\boldsymbol{w}_\nu$, $\boldsymbol{w}_\eta$ and $\boldsymbol{w}_\pi$ respectively. As the derivations are essentially the same as for MDPs we avoid going into the details and simply state the derivatives for infinite horizon discounted rewards framework, which are as follows

$$\begin{aligned}
\nabla_{\boldsymbol{w}_\nu} U(\boldsymbol{w}_\nu, \boldsymbol{w}_\eta, \boldsymbol{w}_\pi) &= \mathbb{E}_{p_1(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}_\nu} \log \nu(\boldsymbol{b};\boldsymbol{w}_\nu)\Big], \\
\nabla_{\boldsymbol{w}_\eta} U(\boldsymbol{w}_\nu, \boldsymbol{w}_\eta, \boldsymbol{w}_\pi) &= \mathbb{E}_{p_\gamma(\boldsymbol{z}',\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z}';\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}_\eta} \log \eta(\boldsymbol{b}'|\boldsymbol{b},\boldsymbol{o};\boldsymbol{w}_\eta)\Big], \\
\nabla_{\boldsymbol{w}_\pi} U(\boldsymbol{w}_\nu, \boldsymbol{w}_\eta, \boldsymbol{w}_\pi) &= \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}_\pi} \log \pi(\boldsymbol{a}|\boldsymbol{b},\boldsymbol{o};\boldsymbol{w}_\pi)\Big],
\end{aligned}$$

where $Q(\boldsymbol{z};\boldsymbol{w})$ is defined analogously to the state-action value function in Markov Decision Processes and $p_\gamma(\boldsymbol{z}', \boldsymbol{z};\boldsymbol{w}) = \sum_{\tau=1}^{\infty} \gamma^\tau p_{\tau+1,\tau}(\boldsymbol{z}', \boldsymbol{z};\boldsymbol{w})$. It can be seen that these gradients are very similar in nature to the gradient of the MDP objective, where they take the form of an expectation of the derivative of the $\log$-controller where the expectation is taken *w.r.t.* the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$. In terms of the evaluation stage of parametric policy search methods the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$, and analagous terms, again plays a central role. The specific functional form of these terms differ slightly in this model, due to the differences in the trajectory distribution, but the role they play in these algorithms is analagous. More details of parametric policy search methods for this model can be found in [170], where, in particular, Expectation Maximisation is considered.

2.3 Natural Gradient Ascent

Natural gradient ascent is a gradinet-based optimisation algorithm that originated in the neural network and blind source seperation literature [3, 4, 5, 6] and was introduced to help alleviate some of the negative

aspects of steepest gradient ascent. One of the main motivations for natural gradient ascent is that steepest gradient ascent implicitly assumes that the parameter space has an Euclidean structure, a point that we shall discuss in more detail in chapter(4), and it can be beneficial to instead consider the parameter space as having a manifold structure. In the case of neural networks, for example, instead of measuring the distance between two parameter vectors through the Euclidean norm it is possible to measure the distance through some measure of difference between the neural networks that these two parameter vector generate. In the case where the parameter vector defines a generative model a common choice of norm is the (local) quadratic norm defined through the Fisher information of the generative model, *i.e.*

$$\|\boldsymbol{w}\|^2_{\text{natural}} = \boldsymbol{w}^\top G(\boldsymbol{w})\boldsymbol{w}, \tag{2.8}$$

where we denote the Fisher information matrix by $G(\boldsymbol{w})$, for any $\boldsymbol{w} \in \mathcal{W}$. See *e.g.* [3, 4, 5, 6] for more details and alternative examples. The application of natural gradient ascent to Markov Decision Processes was introduced by [83] and in this case the parameter update takes the form

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k G^{-1}(\boldsymbol{w}_k)\nabla_{\boldsymbol{w}} U(\boldsymbol{w}_k),$$

where $G(\boldsymbol{w})$ is the Fisher information of the trjectory distribution. In the infinite horizon discounted rewards framework the Fisher information matrix of the trajectory distribution takes the form

$$G(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\nabla^\top_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big], \tag{2.9}$$

which, when the Fisher regularity conditions are satisfied, is equivalent to

$$G(\boldsymbol{w}) = -\mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}}\nabla^\top_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big].$$

The Fisher information takes a similar form in the finite horizon and infinite horizon average reward frameworks. More details on the Fisher information matrix for Markov Decision Processes can be found in [13]. At present this completes our description of natural gradient ascent, but further considerations shall be made in chapter(4).

2.4 Expectation Maximisation

An alternative optimisation procedure that has recently been the centre of much research in the planning, control and reinforcement learning communities is the EM-algorithm [40, 171, 170, 94, 93, 77, 57, 56]. This is an extremely powerful optimisation technique from the statistics and machine learning literature, see *e.g.* [44, 107, 119], that has been successfully applied to a large number of problems. See [15] for a general overview of some of the applications of the algorithm in the machine learning literature. Among the strengths of the algorithm are its guarantee of improving the likelihood (or objective) at each iteration, its often simple update equations and its generalisation to highly intractable models through

variational Bayes approximations[2] [142]. To obtain an intuitive understanding of EM we consider a simple example of a maximum likelihood problem. We then proceed to extend this derivation to Markov Decision Processes.

Suppose we are given a probabilistic model $p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{w})$ of the random variables $(\boldsymbol{y}, \boldsymbol{z})$ that is parameterised by $\boldsymbol{w}$. A typical problem in statistics is to optimise the model parameters given observations of $\boldsymbol{y}$ but no observations of $\boldsymbol{z}$, *e.g.* because these variables are latent or simply because they have been unobserved during data collection. This problem is simply a maximum likelihood problem of the marginal $\log$-likelihood function $\log p(\boldsymbol{y}; \boldsymbol{w})$. However, due to the structure of the objective

$$\log p(\boldsymbol{y}; \boldsymbol{w}) = \log \int d\boldsymbol{z} p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{w}),$$

it will not be possible in general to obtain a closed form solution for the maximum of $\log p(\boldsymbol{y}; \boldsymbol{w})$ and an iterative solution is instead necessary. Assuming that it would be easy to optimise the complete log-likelihood, $\log p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{w})$, if all the data were available the EM-algorithm works by iteratively optimising a lower bound on $\log p(\boldsymbol{y}; \boldsymbol{w})$. This lower bound is not difficult to obtain and essentially relies on the use of Jensen's inequality. We start by noting that for any $\boldsymbol{w}, \boldsymbol{w}' \in \mathcal{W}$ we have the equality

$$\log p(\boldsymbol{y}; \boldsymbol{w}) = \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\Big[\log p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{w})\Big] - \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\Big[\log p(\boldsymbol{z}|\boldsymbol{y}; \boldsymbol{w})\Big]. \tag{2.10}$$

An application of Jensen's inequality gives the bound

$$\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\Big[\log p(\boldsymbol{z}|\boldsymbol{y}; \boldsymbol{w}')\Big] \geq \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\Big[\log p(\boldsymbol{z}|\boldsymbol{y}; \boldsymbol{w})\Big],$$

which again holds $\forall \boldsymbol{w}, \boldsymbol{w}' \in \mathcal{W}$. Using this bound in (2.10) gives the desired lower bound on the objective function

$$\log p(\boldsymbol{y}; \boldsymbol{w}) \geq \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\big[\log p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{w})\big] - \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\big[\log p(\boldsymbol{z}|\boldsymbol{y}; \boldsymbol{w}')\big], \tag{2.11}$$

which is commonly written in the form

$$\log p(\boldsymbol{y}; \boldsymbol{w}) \geq \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}') + \mathcal{H}_{\text{entropy}}(\boldsymbol{w}', \boldsymbol{w}'),$$

where $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}') = \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')}\big[\log p(\boldsymbol{y}, \boldsymbol{z}; \boldsymbol{w})\big]$ and $\mathcal{H}_{\text{entropy}}(\boldsymbol{w}', \boldsymbol{w}')$ is the entropy function applied to the posterior distribution, $p(\boldsymbol{z}|\boldsymbol{y}; \boldsymbol{w}')$. Note that both functions are written so that the first variable occurs inside the expectation while the second variable defines the distribution *w.r.t.* which the expectation is taken.

The EM-algorithm works by iteratively maximising the lower bound (2.11) in a coordinate-wise manner *w.r.t.* $\boldsymbol{w}$ and $\boldsymbol{w}'$. As we have assumed that the complete $\log$-likelihood is easy to maximise we can see that the optimisation of (2.11) *w.r.t.* $\boldsymbol{w}$ (whilst holding $\boldsymbol{w}'$ fixed) will be easy provided that we

[2]The use of the variational Bayes approximation no longer guarantees an increase of the likelihood, but it does guarantee an increase of a lower bound of the likelihood.

can calculate the necessary statistics of $p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}')$. The optimum of (2.11) *w.r.t.* $\boldsymbol{w}'$ (this time holding $\boldsymbol{w}$ fixed) occurs at the point $\boldsymbol{w}$, which can be seen directly from (2.10). As the optimisation *w.r.t.* $\boldsymbol{w}'$ is trivial, in practice one considers only the parameters $\boldsymbol{w}$ and each iteration of the algorithm can essentially be seen as a two step procedure;

E-step Compute the statistics of $p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}_k)$ necessary to perform an M-step, where $\boldsymbol{w}_k$ are the parameter settings at the current iteration.

M-step Find the maximum $\boldsymbol{w}^*$ of $\mathcal{Q}(\boldsymbol{w},\boldsymbol{w}_k)$ *w.r.t.* the first variable and set $\boldsymbol{w}_{k+1}=\boldsymbol{w}^*$.

An alternative but equivalent derivation that is perhaps more intuitive can be obtained through a simple application of Bayes rule and the non-negativity of the Kullback-Leibler divergence. If we first note that the posterior of $\boldsymbol{z}$ given $\boldsymbol{y}$ has the form

$$p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w}) = \frac{p(\boldsymbol{y}|\boldsymbol{z};\boldsymbol{w})p(\boldsymbol{z};\boldsymbol{w})}{p(\boldsymbol{y};\boldsymbol{w})},$$

then we can see that the objective function actually occurs as the normalisation constant of this posterior. Therefore by taking the Kullback-Leibler divergence between some distribution q (over $\mathcal{Z}$) and $p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w})$ then we obtain the same lower bound (2.11). The distribution q is often referred to as the variational distribution. From this perspective we can see that optima $\boldsymbol{w}'=\boldsymbol{w}$ that occurred in the previous derivation makes intuitive sense because it is the minimiser of the Kullback-Leibler divergence between $q(\boldsymbol{z})$ and $p(\boldsymbol{z}|\boldsymbol{y};\boldsymbol{w})$. Up to a second order expansion this is a local metric in the space of distributions parameterised through $\boldsymbol{w}$, see *e.g.* [37].

Although the maximisation of (2.1) has no immediate relation to the maximum likelihood problem considered in the previous example it is clear that there is at least one major similarity. In particular, a main source of intractability both optimisation problems occurs due to the integral operation in the objective function. It is therefore natural to extend the ideas of the EM-algorithm to the Markov Decision Process framework. As noted previously, such an extension can be obtained by constructing a distribution, parameterised through $\boldsymbol{w}\in\mathcal{W}$, such that the normalisation constant of this distribution is given by $U(\boldsymbol{w})$, for any given $\boldsymbol{w}\in\mathcal{W}$. Given that we can assume *w.l.o.g.* that the reward function is non-negative it is not difficult to construct such a distribution. In particular, we define the distribution as follows

$$\hat{p}(\boldsymbol{z}_{1:t},t;\boldsymbol{w}) = \frac{1}{U(\boldsymbol{w})}\gamma^{t-1}R(\boldsymbol{z}_t)p(\boldsymbol{z}_{1:t};\boldsymbol{w}). \tag{2.12}$$

It can easily be seen from (2.1) and (2.2) that the normalisation constant of this distribution is equal to $U(\boldsymbol{w})$. Taking the Kullback-Leibler divergence between the variational distribution $q(\boldsymbol{z}_{1:t},t)$ and $\hat{p}(\boldsymbol{z}_{1:t},t;\boldsymbol{w})$ gives the following lower bound on the logarithm of the objective function,

$$\log U(\boldsymbol{w}) \geq \mathcal{H}_{\text{entropy}}(q(\boldsymbol{z}_{1:t},t)) + \mathbb{E}_{q(\boldsymbol{z}_{1:t},t)}\Big[\log\gamma^{t-1}R(\boldsymbol{z}_t)p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big]. \tag{2.13}$$

An EM-algorithm is obtained from the bound in (2.13) by iterative coordinate-wise maximisation:

E-step For fixed $\boldsymbol{w}_k$ find the best $q(\boldsymbol{z}_{1:t}, t)$ that maximises the *r.h.s.* of (2.13), which for an unconstrained $q(\boldsymbol{z}_{1:t}, t)$ gives $q(\boldsymbol{z}_{1:t}, t) \equiv \hat{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w}_k)$. Then compute the statistics of $\hat{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w}_k)$ necessary to perform a parameter update.

M-step For fixed $q(\boldsymbol{z}_{1:t}, t)$ find the best $\boldsymbol{w}$ that maximises the *r.h.s.* of (2.13). Using similar arguments to those in section(2.2) this be seen to be equivalent to maximising *w.r.t.* $\boldsymbol{w}$ the 'energy' contribution

$$\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w}_k)Q(\boldsymbol{z};\boldsymbol{w}_k)}\Big[\log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w})\Big]. \tag{2.14}$$

Note that $\mathcal{Q}$ is a two-parameter function, where the first parameter occurs inside the expectation and the second parameter defines the 'distribution' *w.r.t.* the expectation is taken. We note that the functional form of $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ is one of the reasons why it is natural to chose to denote the gradient of (2.1) in the form given in (2.4). The decoupling of the parameters in $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ allows the maximisation over $\boldsymbol{w}$ to be performed in many cases of interest. For example, when the log-policy is quadratic in $\boldsymbol{w}$ the maximisation problems is equivalent to a least-squares problem and the optimum can be found through solving a linear system of equations. When it is not possible to directly solve the maximisation of $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ *w.r.t.* $\boldsymbol{w}$ it is often possible to instead calculate the derivative $\nabla_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ and then take a step in the parameter space using this as a search direction. Such a technique is known as a *generalised* EM-algorithm [44] and although the lower bound (2.13) is no longer maximised at each iteration it is easy to see that provided

$$\mathcal{Q}(\boldsymbol{w}_{k+1}, \boldsymbol{w}_k) \geq \mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k),$$

for each $k \in \mathbb{N}$, it will still be increased. Under similar conditions to the EM-algorithm the generalised EM-algorithm is guaranteed to converge to a local maxima [44]. Details on the exact form of the parameter update can be found in the literature for various systems, such as continuous systems [94, 77], discrete systems [171] and partially observable and multi-agent systems [169, 102].

An issue with the EM-algorithm is that the rate of convergence of the algorithm can be prohibitively slow in many cases. Theoretically the rate of convergence for the EM-algorithm can range from anywhere between quadratic to sub-linear depending on the eigenvalues of the Jacobian of the EM-operator in the vicinity of a local optimum, see *e.g.* [44, 141] for more details. Typically it is difficult to categorise the behaviour of these eigenvalues in terms of quantities of interest[3], such as the structure of the reward function, but it well-known that the EM-algorithm can be prohibitively slow in practice. Various authors have attempted to increase the performance of the EM-algorithm in the case of Markov Decision Processes, as well as similar planning models. Using the knowledge that the space of deterministic policies is sufficient to obtain optimality in a Markov Decision Process [170, 171] restricted the policy space of the EM-algorithm to deterministic policies, where the resulting algorithm is referred to as 'greedy EM-algorithm'. However, this 'greedy EM-algorithm' is not a true EM-algorithm, but instead a refor-

[3]While such a categorisation of the eigenvalues is difficult it is possible to analysis the behaviour of the policy update in terms of the problem structure. In appendix(B) we provide some novel analysis of the EM-algorithm in this respect. For instance it is possible to categorise the effect of a multi-modal reward function on the policy update of the EM-algorithm. This analysis doesn't include results pertaining to the rate of convergence of the EM-algorithm, but it does give an intuitive understanding of the algorithm.

mulation of policy iteration. A more formally correct formulation of this EM-algorithm for deterministic policies is detailed in appendix(D), but in practice it has been found that the EM-algorithm loses much of its desirability in terms of robustness to local optima when this restriction to the policy search space is made. Additionally this restriction to the space of deterministic policies can cause freezing of the parameter updates in deterministic, or close to deterministic, environments. Furthermore, this restriction to the space of deterministic policies only makes sense in the case of fully observable environments and it is not applicable to partially observable environments, where it is not necessarily true that optimal control can be obtain from a deterministic controller. An alternative, motivated by the analysis in appendix(B), would be to attempt to reshape the reward function so as to maintain the same optimums of the original objective, but to obtain superior performance in the EM-algorithm. Similar methods have been considered previously, see *e.g.* [181], in relation to other optimisation algorithms with some success. Finally, a heuristic that some authors consider is to use a *softened* greedy M-step, see *e.g.* [169, 102], which has been used with some success.

Chapter 3

Parametric Policy Search Methods : Search Direction Evaluation

3.1 Introduction

It was seen in chapter(2) that a core aspect of parametric policy search methods is the *evaluation stage*, which corresponds to calculating the statistics necessary to perform a parameter update. In the case of steepest gradient ascent, applied to the infinite planning horizon framework with discounted rewards, this corresponds to calculating the following integral

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big],$$

with similar integrals being necessary in either the finite horizon or infinite horizon average reward frameworks. Similar integrals are also necessary for both natural gradient ascent and Expectation Maximisation. In some instances it is possible to calculate these integrals exactly, for instance in a MDP where the state-action space is discrete and sufficiently small that enumeration over the state-action space is feasible. However, in the complex real world problems that are typically of interest to practitioners, such as difficult robotic manipulation tasks, these integrals will be intractable. To obtain a broader understanding of some of the possible sources of intractabilities we now discuss several possible examples. Firstly, it may be the case that the state-action space is too large for enumeration over all state-action pairs to be feasible. For instance, in the game of Tetris, which we shall consider in chapter(4), there are approximately 7×2^{hw} states, where h and w are the height and width of the board respectively. In a typical game of Tetris, where the board is of height 20 and width 10, enumeration over the state-action space in completely infeasible. Another example where such integrals are, in general, intractable is in problems where the state-action space is continuous and the transition dynamics are non-linear. In this case the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$ is highly complex and, in general, has no closed form. This can be seen from the observation that the calculation of the state-action occupancy marginals, $p_t(\cdot;\boldsymbol{w})$, for $t \in \mathbb{N}$, is equivalent to a non-linear filtering problem, which is in itself an intractable problem and the subject of much research in areas such as control and time-series analysis, see *e.g.* [156, 15, 49]. Given that the function $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$ requires knowledge of the state-action occupancy marginals it is clear

that, in general, it is an intractable problem to the calculate this function when the transition dynamics are non-linear. A final example where the inference necessary to perform a parameter update can be intractable is in high-dimensional factored Markov Decision Processes. In the most general case the complexity of representing/calculating either $p_\gamma(\cdot; \boldsymbol{w})$ or $Q(\cdot; \boldsymbol{w})$ will scale exponentially in the number of factors. Unless there is a particular, restrictive, form of sparsity in the level of interaction between the different factors of the MDP, such as in DEC-MDPs considered section(1.5), then the induced tree-width of the state-action occupancy marginals will become prohibitively large, for sufficiently large $t \in \mathbb{N}$, and this will make the inference intractable. In these last two examples we have seen that calculating the state-action occupancy marginals is itself intractable. The same is also true of the state-action value function, which is also generally intractable to calculate in both of these examples. Also note that for similar reasons the evaluation of the objective function, for any given $\boldsymbol{w} \in \mathcal{W}$, is also intractable for the three examples just considered.

We have seen that the evaluation stage of parametric policy search methods is typically an intractable problem. As a result there has been much research into approximate solutions to perform the integrals necessary for a parameter update, including model-free sample-based methods [66, 67, 134, 135, 19, 113, 39, 180, 68, 124], actor-critic methods [99, 98, 162, 28, 29] and model-based message-passing methods [170, 171, 104, 102, 169, 77, 58, 59]. These various approximation methods, along with my own research in this area, are the subject of the present chapter. As my research in this area has focused on model-based message-passing techniques the most significant part of the present chapter will necessarily be devoted to these methods. We note that this bias in the amount of space devoted respectively to model-based and model-free methods is not in any way indicative of a bias towards model-based methods in the literature. In fact, when parametric policy search methods first came to prominence[1], see *e.g.* [66, 67, 134, 135, 19, 113], there was a preference to perform inference in a model-free sample-based manner. This preference towards model-free sample-based methods has persisted, with techniques such variance reduction [39, 180, 68, 124] and actor-critic methods [99, 98, 162, 28, 29] being predominant. However, while sample-based methods are very general and are still the prevalent method in parametric policy search methods they do suffer from various undesirable aspects, such as requiring an excessive amount of samples and suffering from large variance in the estimates of the search direction. It is only more recently that there has been an upsurge of interest in model-based inference routines, see *e.g.* [170, 171, 104, 102, 169, 77, 58, 59], which use methods from probabilistic and approximate inference[2] to obtain efficient inference routines to perform the integrals necessary for a parameter update. A core aspect of probabilistic and approximate inference techniques is in the calculation of marginals, or moments, of high-dimensional distributions, where many of the techniques in this area exploit underlying sparsity in the (graphical structure of the) distribution to obtain efficient and accurate inference routines. Many interesting and complex planning problems satisfy similar sparsity properties during the evaluation stage of parametric policy search methods (a point we

[1] There are several approaches to calculating the gradient of the objective function for Markov Decision Processes, such as finite differences or infinitesimal perturbation analysis, we focus exclusively on likelihood ratio methods.

[2] An introduction to probabilistic and approximate inference can be found in *e.g.* [178, 15].

shall discuss in more detail in section(3.2)) and so it is of theoretical and practical interest to extend the ideas of probabilistic and approximate inference to this field of research.

As previously mentioned, the novel theoretical contribution of this chapter is in the area of model-based inference methods. We shall introduce current model-based inference routines in section(3.2), where we shall also detail our novel contributions in this area. Firstly, we shall detail how model-based methods are closely related to inference routines in latent variable time-series models, such as the Hidden Markov Model [15]. In this respect all of the current model-based inference methods in literature can be seen to be exclusively of the form of *forward-backward* algorithms, while the Rauch-Tung-Striebel (RTS) smoother [133], which plays a prominent role in latent variable time-series models, has been overlooked. The novel theoretical contribution of this chapter is the consideration of RTS smoothing techniques to the evaluation stage of parametric policy search methods, where we shall find that this simple reformulation has some interesting consequences that do not occur when applying RTS inference techniques to latent variable time-series models. We shall also consider several particular classes of planning problem where this form of inference is particularly well-suited, and where there are several important and desirable advantages in comparison to forward-backward inference.

The rest of the chapter shall be organised as follows: In section(3.2) we shall consider model-based inference techniques for the evaluation stage of parametric policy search algorithms, where in section(3.2.1) we shall provide an overview of current forward-backward techniques and in section(3.2.2) we shall introduce our novel family of RTS inference routines; For future reference we shall introduce the prominent model-free sample-based inference methods in section(3.3); In section(3.4) we shall perform various experiments comparing our RTS inference methods with the corresponding forward-backward inference methods; Finally, in section(3.5) we shall provide an overview of the contributions of this chapter.

3.2 Model-Based Evaluation Techniques

Before proceeding to the description of model-based inference routines we first reintroduce a concept that was first introduced in section(2.4), during the derivation of Expectation Maximisation, but will now prove useful notationally and will also as provide insights into the differences between forward-backward and Rauch-Tung-Striebel inference methods. Recall that, as the reward can be assumed to be non-negative, it is possible to introduce the following distribution

$$\hat{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w}) = \frac{1}{U(\boldsymbol{w})} \gamma^{t-1} R(\boldsymbol{z}_t) p(\boldsymbol{z}_{1:t}; \boldsymbol{w}). \tag{3.1}$$

We also denote the unnormalised version of this distribution by $\tilde{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w})$, *i.e.* $\tilde{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w}) = \gamma^{t-1} R(\boldsymbol{z}_t) p(\boldsymbol{z}_{1:t}; \boldsymbol{w})$. For any, $t \in \mathbb{N}$, the term $\tilde{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w})$ equals the probability of the trajectory up until the t^{th} time-point, given the parameter vector, weighted by the discounted reward received at the t^{th} time-point. For which reason we refer to (3.1) as the unnormalised reward weighted trajectory distribution, or just the reward weighted trajectory distribution when the context is clear. In the finite horizon framework we define the reward weighted trajectory distribution in an analogous manner, but the reward

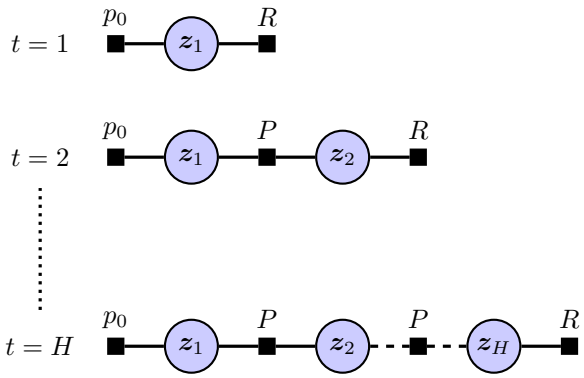


Figure 3.1: A factor graph representation of the reward weighted trajectory distribution.

function is now no longer discounted. The graphical structure of the reward weighted trajectory distribution is given by a (possibly infinite) mixture of chain distributions, each corresponding to a different time-point at which a reward is received. Note that each component of this mixture distribution contains a different number of variables and such a distribution is generally known as a trans-dimensional distribution. A factor graph representation of the reward weighted trajectory distribution is given in fig(3.1). Additionally, the weight of each component in the normalised version of the reward weighted trajectory distribution corresponds to the proportion of the total expected reward obtained at the time-point of the current component, *e.g.* in the finite horizon case we have

$$\hat{p}(t; \boldsymbol{w}) = \frac{\mathbb{E}_{p_t(\boldsymbol{z};\boldsymbol{w})}\big[R(\boldsymbol{z})\big]}{U(\boldsymbol{w})}.$$

It is convenient to note that in the case of a finite planning horizon we have

$$\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = p_\tau(\boldsymbol{z}; \boldsymbol{w}) \mathbb{E}_{p_t(\boldsymbol{z}';\boldsymbol{w})}\Big[R(\boldsymbol{z}')\Big|\boldsymbol{z}_\tau = \boldsymbol{z}\Big],$$

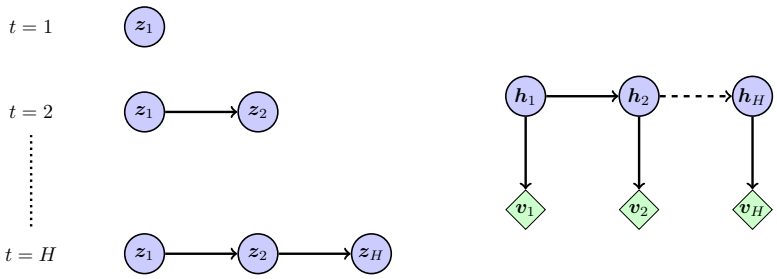
where we have used the notation $\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) \equiv \tilde{p}(\boldsymbol{z}_\tau, t; \boldsymbol{w})$. An analogous equation holds in the infinite horizon discounted rewards framework. In terms of the integrals necessary to perform a parameter update it is not difficult to see that we can write the gradient in the form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \sum_{t=1}^{\infty} \sum_{\tau=t}^{\infty} \mathbb{E}_{\tilde{p}(\boldsymbol{z},\tau,t;\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w})\Big],$$

where the integrals necessary for the EM-algorithm and natural gradient ascent can be written in a similar fashion. Having written these integrals in this manner it can now be seen that the evaluation stage of parametric policy search methods is equivalent to performing inference (*i.e.* calculating the marginals) of the reward weighted trajectory distribution, where these marginals are determined by the derivative of the log-policy.

The observation that the evaluation stage of parametric policy search methods is equivalent to inference in a graphical model has led to a recent surge in model-based inference techniques[3]

[3]This observation has also lead to some novel model-free inference techniques [176, 78, 79] but we do not discuss these here



(a) Reward Weighted Trajectory Distribution (b) Hidden Markov Model

Figure 3.2: (a) A dynamic Bayesian network representation of the reward weighted trajectory distribution (b) A dynamic Bayesian network representation of the Hidden Markov Model.

[170, 169, 171, 104, 102, 77, 59]. These methods aim to use tools from probabilistic and approximate inference to obtain efficient inference routines that are either exact or approximate, typically by exploiting certain aspects of the reward weighted trajectory distribution. An important example is when the state-action space is high-dimensional and, due to some underlying sparsity structure in the problem, the reward weighted trajectory distribution has a sparse graphical structure. There are numerous approximate inference techniques that exploit sparsity in the graphical structure of a distribution to obtain accurate estimates of marginals. It is desirable, therefore, to extend these techniques to the evaluation stage of parametric policy search algorithms. Additionally, this view of the evaluation stage as a problem of probabilistic inference allows for various data modelling techniques from the time-series literature to be employed, such as switching linear dynamical systems or infinite Hidden Markov Models.

In model-based inference the model of the environment is used directly to calculate the marginals of the reward weighted trajectory distribution. Due to the Markov assumption and the temporal nature of Markov Decision Processes the graphical structure of the reward weighted trajectory distribution has close similarities to the graphical structure of latent variable time-series models, such as the Hidden Markov Model. Recall that the structure of the reward weighted trajectory distribution is given by a (possibly infinite) mixture of chain distributions, where each component of the mixture corresponds to a point in the planning horizon. While the Hidden Markov Model does not have this mixture structure it does have the chain structure that is present in each component of the reward weighted trajectory distribution. The Hidden Markov Model also has a visible variable emitted from each hidden variable in the chain, but these are not important to to our discussion. A factor graph representation of the reward weighted trajectory distribution and the Hidden Markov Model is given in fig(3.2). From this perspective the evaluation stage of parametric policy search algorithms, such as steepest gradient ascent or Expectation Maximisation, can be seen to be equivalent to performing inference in a latent variable time-series model, albeit one with a mixture structure over the points in the planning horizon. It is important to note that, while the evaluation stage of parametric policy search algorithms has similarities to inference routines in a latent variable time-series model, the actual form of the inference routines is significantly different. This is due, in main, to the mixture structure of the reward weighted trajectory distribution, which is absent in models such as the Hidden Markov Model. Additionally, the planning

as our focus is on model-based methods.

horizon is often infinite in a Markov Decision Process, which is typically not the case in the Hidden Markov Model. These are important points and necessarily they make the construction of efficient inference routines markedly different. A final point of interest is that applying the procedures from chapter(2) to the risk-sensitive objective (1.5) results in a reward weighted trajectory distribution where this mixture structure is absent. In this case the structure of this distribution would be a single chain, like the Hidden Markov Model. This is due to the fact that the reward structure of (1.5) is multiplicative instead of additive, as it is in (2.1).

We now proceed to some current examples of evaluation methods that are based on ideas from probabilistic inference, which can all be seen as forward-backward inference methods, before proceeding to our new *Rauch-Tung-Striebel* type evaluation procedure.

3.2.1 Forward-Backward Inference

In typical applications of forward-backward methods, say in a Hidden Markov Model, the distribution is chain structured and inference is performed by first passing a set of messages both forward and backward in the chain. Given these sets of messages the marginals are obtained by combining the appropriate forward and backward messages, see *e.g.* [15] for more details and the application of the methods to Hidden Markov Models. As the structure of each component in the reward weighted trajectory distribution is chain structured it is possible to apply forward-backward methods to each of the components individually. While this naive application of forward-backward methods is possible it has undesirable properties, for example the number of messages that need to be calculated scales quadratically *w.r.t.* to the planning horizon, when the planning horizon is finite, and thus the complexity of the algorithm scales quadratically with the planning horizon in this case. It was noted in [170] that when the transition dynamics, policy and reward function are stationary it is only necessary to calculate a linear number of messages, using the 'time-to-go' formulation of the message-passing routine. In [171] it is noted that in discrete systems, where it is possible to enumerate over the state-action space, there is another source of efficiency that comes from combining the backward messages from different components of the reward weighted trajectory distribution. In [171] this observation is made solely *w.r.t.* Expectation Maximisation, being referred to 'incremental-EM', but it is applicable to other similar parametric policy search methods.

Markov Decision Processes

The simplest case to consider is the single variable Markov Decision Process where it is necessary to calculate marginals of the form $\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w})$, for all $t \in \mathbb{N}_H$ and $\tau \in \mathbb{N}_t$. This is the case considered in [170, 171], again in terms of Expectation Maximisation, where it is assumed that it is possible to enumerate over the state-action space. We highlight the forward-backward procedure for a finite planning horizon, where details of the extension to an infinite planning horizon with discounted rewards can be found in [170, 171].

Consider the case where the transition dynamics, policy and reward function are non-stationary and forward-backward inference is performed along each component of the reward weighted trajectory

distributions individually. As the policy is non-stationary we denote the parameters for the policy at the t^{th} time-point by $\boldsymbol{w}_t$. For the component corresponding to the t^{th} time-point, $t \in \mathbb{N}_H$, there are t forward messages and t backward messages, where these messages take the form

$$\alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w}) = p_\tau(\boldsymbol{z}; \boldsymbol{w}), \qquad \beta_t(\boldsymbol{z}, \tau; \boldsymbol{w}) = \mathbb{E}_{p_t(\boldsymbol{z}'; \boldsymbol{w})}\Big[R_t(\boldsymbol{a}', \boldsymbol{s}')\big|\boldsymbol{z}_\tau = \boldsymbol{z}\Big].$$

The initial forward and backward messages of the component corresponding to the t^{th} time-point take the form

$$\alpha_1(\boldsymbol{z}, t; \boldsymbol{w}) = p_1(\boldsymbol{s})p(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w}_1), \qquad \beta_t(\boldsymbol{z}, t; \boldsymbol{w}) = R_t(\boldsymbol{a}, \boldsymbol{s}).$$

These messages are then simultaneously propagated both forward and backward in the chain, where the updates are respectively given by the equations

$$\begin{aligned}\alpha_{\tau+1}(\boldsymbol{z}', t; \boldsymbol{w}) &= \sum_{\boldsymbol{z} \in \mathcal{Z}} P_{\tau+1}(\boldsymbol{z}'|\boldsymbol{z}; \boldsymbol{w}_{\tau+1})\alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w}), \\ \beta_\tau(\boldsymbol{z}, t; \boldsymbol{w}) &= \sum_{\boldsymbol{z}' \in \mathcal{Z}} P_{\tau+1}(\boldsymbol{z}'|\boldsymbol{z}; \boldsymbol{w}_{\tau+1})\beta_{\tau+1}(\boldsymbol{z}', t; \boldsymbol{w}),\end{aligned}$$

and $P_{\tau+1}(\boldsymbol{z}'|\boldsymbol{z}; \boldsymbol{w}_{\tau+1})$ is the non-stationary state-action transition matrix given by

$$P_{\tau+1}(\boldsymbol{z}'|\boldsymbol{z}; \boldsymbol{w}_{\tau+1}) = p(\boldsymbol{a}'|\boldsymbol{s}'; \boldsymbol{w}_{\tau+1})p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}).$$

Given the messages, $\alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w})$ and $\beta_\tau(\boldsymbol{z}, t; \boldsymbol{w})$, the 'marginal', $\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w})$, is given by

$$\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = \alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w})\beta_\tau(\boldsymbol{z}, t; \boldsymbol{w}). \tag{3.2}$$

It can be seen that the forward messages are independent of the mixture component so that there are a linear number of forward messages. We write $\alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w}) \equiv \alpha_\tau(\boldsymbol{z}; \boldsymbol{w})$, for all $t \in \mathbb{N}_H$. In contrast the backward messages are dependent on the mixture component and so the number of backward messages that need to be calculated scales quadratically with the planning horizon. In terms of these forward-backward messages the statistics necessary for a policy update take the form

$$\begin{aligned}\sum_{\tau=1}^{H}\sum_{t=\tau}^{H} \tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) &= \sum_{\tau=1}^{H}\sum_{t=\tau}^{H} \alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w})\beta_\tau(\boldsymbol{z}, t; \boldsymbol{w}), \\ &= \sum_{\tau=1}^{H} \alpha_\tau(\boldsymbol{z}; \boldsymbol{w}) \sum_{t=\tau}^{H} \beta_\tau(\boldsymbol{z}, t; \boldsymbol{w}).\end{aligned}$$

Due to the quadratic number of backward messages that need to be calculated the complexity of this algorithm is quadratic in the planning horizon. As noted in [170] when the reward function, transition dynamics and policy are stationary it is only necessary to calculate a linear number of backward messages. This is because in this case the backward messages depend only on the difference in time, $t - \tau$, or the 'time-to-go' as it is refereed to in [170], and so the same set of backward messages can be used in

all components of the reward weighted trajectory distribution. Under the assumption that it is possible to enumerate over the state-action space a linear time inference routine can be obtained for the policy evaluation problem, which is possible by constructing a recursive relation for the summation of backward messages. The extension to an infinite planning horizon with discounted rewards using the 'time-to-go' argument is given in [170, 171].

The above procedure performs inference along the various components of (3.1) independently, which is unnecessary under the assumption that it is possible to enumerate over the entire state-action space. While it is still possible to obtain an efficient inference routine in this manner, using the 'time-to-go' argument, it requires stationarity in the reward function, transition dynamics and policy. An alternative, more general, inference procedure is given in [171]. Recall that the 'marginal', $\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w})$, can be written in the form

$$\tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = p(\boldsymbol{z}, \tau; \boldsymbol{w}) \mathbb{E}_{p(\boldsymbol{z}', t; \boldsymbol{w})} \Big[R_t(\boldsymbol{z}') \Big| \boldsymbol{z}_\tau = \boldsymbol{z} \Big].$$

The forward and backward component of the marginal (3.2) correspond to $p(\boldsymbol{z}, \tau; \boldsymbol{w})$ and $\mathbb{E}_{p(\boldsymbol{z}', t; \boldsymbol{w})}[R(\boldsymbol{z}')|\boldsymbol{z}_\tau = \boldsymbol{z}]$ respectively. As the statistic necessary for a parameter update takes the form

$$\sum_{\tau=1}^{H} \sum_{t=\tau}^{H} \tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = \sum_{\tau=1}^{H} p(\boldsymbol{z}, \tau; \boldsymbol{w}) \sum_{t=\tau}^{H} \mathbb{E}_{p(\boldsymbol{z}', t; \boldsymbol{w})} \Big[R_t(\boldsymbol{z}') \Big| \boldsymbol{z}_\tau = \boldsymbol{z} \Big],$$

it can be seen that the summation of backward messages $\sum_{t=\tau}^{H} \beta_\tau(\boldsymbol{z}, t; \boldsymbol{w})$ is equivalent to the state-action value function, $Q_\tau(\boldsymbol{z}; \boldsymbol{w})$. Therefore, under the assumption that it is possible to enumerate over the state-action space, it is clear that the following recursive equation can be obtained for the summation of backward messages

$$Q_\tau(\boldsymbol{z}; \boldsymbol{w}) = R_\tau(\boldsymbol{z}) + \sum_{\boldsymbol{z}' \in \mathcal{Z}} P_{\tau+1}(\boldsymbol{z}'|\boldsymbol{z}; \boldsymbol{w}) Q_{\tau+1}(\boldsymbol{z}'; \boldsymbol{w}). \tag{3.3}$$

In this formulation the extension to an infinite planning horizon with discounted rewards is trivial and directly corresponds to the Bellman equation over state-action value functions, where convergence is given by the contraction mapping theorem. Additionally, it is clear that formulation doesn't require the reward function, transition dynamics and policy to be stationary in order to obtain an efficient inference routine, unlike the 'time-to-go' argument. Written in this form it can be seen that standard formulations of the gradient, such as (2.4), are written in a forward-backward form, and the state-action occupancy marginals correspond to the forward term, while the state-action value function corresponds to the backward term.

Finally, we note that ordinarily in a forward-backward algorithm it is of interest to obtain the marginals over the hidden variables and so the product of the forward-backward messages needs to be normalised. If one were to normalise the marginal in (3.2) then one would obtain the state-action marginal for the τ^{th} time-point in the t^{th} component of (3.1), *i.e.*

$$\hat{p}(\boldsymbol{z}, \tau | t; \boldsymbol{w}) = \frac{1}{\mathbb{E}_{p(\boldsymbol{z}, t; \boldsymbol{w})}\big[R_t(\boldsymbol{z})\big]} \alpha_\tau(\boldsymbol{z}, t; \boldsymbol{w}) \beta_\tau(\boldsymbol{z}, t; \boldsymbol{w}).$$

The marginal, $\hat{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w})$, is obtained from the equation $\hat{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = \hat{p}(\boldsymbol{z}, \tau | t; \boldsymbol{w}) \hat{p}(t; \boldsymbol{w})$, where $\hat{p}(t; \boldsymbol{w})$ is given by

$$\hat{p}(t; \boldsymbol{w}) = \frac{\mathbb{E}_{p(\boldsymbol{z}, t; \boldsymbol{w})}\big[R_t(\boldsymbol{z})\big]}{U(\boldsymbol{w})},$$

which can be seen from (3.1). In gradient-based algorithms it is only necessary to calculate the search direction up to a positive scalar, the scalar being absorbed into the step-length, while in the EM-algorithm the normalisation constant of (3.1) doesn't affect the argmax of the function (2.14). As a result it is possible to consider either the normalised or the unnormalised marginals in the parametric policy search methods discussed in chapter(2).

This completes the description of the forward-backward routines in single variable Markov Decision Processes. In both formulations of these forward-backward algorithms it was necessary to be able to enumerate over the state-action space in order to obtain an efficient inference routine. This property allows the merging of backward messages, which effectively corresponds to performing inference over multiple components of the reward weighted trajectory distribution simultaneously. This property clearly doesn't hold in all Markov Decision Processes. For instance, it doesn't hold for any of the Markov Decision Processes considered in section(3.1). Furthermore, it doesn't hold in systems with a continuous state-action space, where the transition dynamics and policy are linear. A model-based forward-backward algorithm [77] has been constructed for such systems, but the inability to combine the backward messages results in it having a quadratic run-time *w.r.t.* the finite planning horizon. In terms of structured Markov Decision Processes [170, 171] suggest two possible methods to perform inference in (3.1). The first method is to eliminate variables that are not in the separator cliques of the time-slices in the trajectory distribution and then perform forward-backward inference over this reduced set of variables, while the second method is to apply the junction tree algorithm to the reward weighted trajectory distribution. While these techniques have been successfully applied to several interesting domains, such as a POMDP where the belief is modelled through a finite state controller [170, 171, 169], they have a run-time that scales exponentially in the induced tree-width of the graphical model. We now consider several different models where forward-backward inference has been successfully applied, after which we shall detail our RTS inference paradigm.

Finite State Controllers

An additional example considered in [170, 171] is the POMDP where the agents' belief is modeled through a finite state controller. This model is interesting because it has been used to demonstrate the possibility of using various data modelling techniques from the time-series literature to construct more sophisticated policy structures. In [169] the technique of hierarchical Hidden Markov Models is used to model the policy of a finite state controller with a hierarchical structure. Additionally, this example is interesting because it illustrates the efficiency gains that are possible by performing inference over the separator sets of the Markovian trajectory distribution. It can be seen in fig(1.6) that the state-belief variables form a separator set between time-points of the trajectory distribution. It is therefore more efficient to perform forward-backward inference in terms of the state-belief variables. The state-belief

transition matrix can be calculated as follows

$$P(\boldsymbol{b}', \boldsymbol{s}'|\boldsymbol{b}, \boldsymbol{s}; \boldsymbol{w}_\pi, \boldsymbol{w}_\nu) = \sum_{\boldsymbol{a}\in\mathcal{A}} \sum_{\boldsymbol{o}\in\mathcal{O}} p(\boldsymbol{s}'|\boldsymbol{a}, \boldsymbol{s}) p(\boldsymbol{b}'|\boldsymbol{b}, \boldsymbol{o}; \boldsymbol{w}_\nu) p(\boldsymbol{a}|\boldsymbol{b}, \boldsymbol{o}; \boldsymbol{w}_\pi) p(\boldsymbol{o}|\boldsymbol{s}),$$

while the initial messages take the form

$$\alpha_1(\boldsymbol{b}, \boldsymbol{s}; \boldsymbol{w}_\eta) = p_0(\boldsymbol{b}; \boldsymbol{w}_\eta) p_0(\boldsymbol{s}), \qquad \beta_t(\boldsymbol{b}, \boldsymbol{s}, t; \boldsymbol{w}_\pi) = \sum_{\boldsymbol{a}\in\mathcal{A}} \sum_{\boldsymbol{o}\in\mathcal{O}} R_t(\boldsymbol{a}, \boldsymbol{s}) p(\boldsymbol{a}|\boldsymbol{b}, \boldsymbol{o}; \boldsymbol{w}_\pi) p(\boldsymbol{o}|\boldsymbol{s}).$$

As in the single variable Markov Decision Process these forward and backward messages are propagated simultaneously, while the following recursion is easily obtainable

$$Q_\tau(\boldsymbol{b}, \boldsymbol{s}; \boldsymbol{w}_\pi, \boldsymbol{w}_\nu) = \beta_\tau(\boldsymbol{b}, \boldsymbol{s}, \tau; \boldsymbol{w}_\pi) + \sum_{\boldsymbol{b}'\in\mathcal{B}} \sum_{\boldsymbol{s}'\in\mathcal{S}} Q_{\tau+1}(\boldsymbol{b}', \boldsymbol{s}'; \boldsymbol{w}_\pi, \boldsymbol{w}_\nu) P(\boldsymbol{b}', \boldsymbol{s}'|\boldsymbol{b}, \boldsymbol{s}; \boldsymbol{w}_\pi, \boldsymbol{w}_\nu),$$

where the term $Q_\tau(\boldsymbol{b}, \boldsymbol{s}; \boldsymbol{w}_\pi, \boldsymbol{w}_\nu)$ is analogous to the state-action value function.

Transition Independent Decenteralised Markov Decision Processes - Additive Sparsity

Another example where forward-backward inference has recently been applied is in transition independent decentralised Markov Decision Processes [102, 101]. For the sake of simplicity we detail the simple model introduced in section(1.5) but more complex models were also considered in [102], such as modelling each agent's belief through a finite state controller. Only Expectation Maximisation is considered in [102, 101] but, as previously mentioned, the same inference routine is also applicable to steepest gradient ascent and similar methods. Due to the transition independence and decentralised nature of the policy[4] these models have a natural sparsity structure in the trajectory distribution. Under the assumption that the trajectory distribution of any individual agent is tractable the only source of intractability in this model is the global reward function. This makes these models an ideal candidate for model-based inference techniques and tractable inference routines will be possible provided that the reward function exhibits a sufficient amount of sparsity.

In [102] a sparse additive structure is placed on the reward function, which is achieved by writing the global reward function into a summation over *local* sparse reward functions. In particular suppose there is some finite index set, $\mathcal{F}$, such that the global reward function can be written in the form

$$R(\boldsymbol{z}) = \sum_{f\in\mathcal{F}} R_f(\boldsymbol{z}^f),$$

where $\boldsymbol{z}^f$ is the subset of the state-action variables corresponding to the agents in the domain of the local reward function R_f. In this case the total expected reward takes the form

$$U(\boldsymbol{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p(\boldsymbol{z}_t;\boldsymbol{w})}\left[\gamma^{t-1} \sum_{f\in\mathcal{F}} R_f(\boldsymbol{z}_t^f)\right] = \sum_{f\in\mathcal{F}} \sum_{t=1}^{\infty} \mathbb{E}_{p(\boldsymbol{z}_t^f;\boldsymbol{w})}\left[\gamma^{t-1} R_f(\boldsymbol{z}_t^f)\right] = \sum_{f\in\mathcal{F}} U_f(\boldsymbol{w}^f),$$

[4]Note that in terms of parametric policies the decentralised policy (1.25) is written in the form $p(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w}) = \prod_{n=1}^{N} \pi^n(\boldsymbol{a}^n|\boldsymbol{s}^n; \boldsymbol{w}^n)$, where $\boldsymbol{w}^n$ are the parameters of the n^{th} agent.

where we use $U_f(\boldsymbol{w}^f)$ to denote the lower-dimensional MDP corresponding to local reward function f. In terms of the reward weighted trajectory distribution this decomposition of the objective function corresponds to a mixture distribution over local reward functions, *i.e.*

$$\tilde{p}(\boldsymbol{z}^f_{1:t}, t, f; \boldsymbol{w}) = R_f(\boldsymbol{z}^f_t) p(\boldsymbol{z}^f_{1:t}; \boldsymbol{w}^f),$$

while the gradient of the objective *w.r.t.* the parameters of the i^{th} agent takes the form

$$\nabla_{\boldsymbol{w}^i} U(\boldsymbol{w}) = \sum_{f \in \mathcal{F}_i} \sum_{t=1}^{H} \sum_{\tau=1}^{t} \mathbb{E}_{\tilde{p}(\boldsymbol{z}^i, \tau, t, f; \boldsymbol{w})} \Big[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}^i | \boldsymbol{s}^i; \boldsymbol{w}^i) \Big],$$

where $\mathcal{F}_i$ is the subset of local functions that have a dependence upon the i^{th} agent. In terms of inference this means that message-passing can be done over each of the local reward function mixture components individually. In [102] it is assumed that each of the local functions contains a sufficiently small number of agents that exact inference in each of the local reward function mixture components is possible using forward-backward inference. The forward-messages correspond to the state-action occupancy distributions, over subsets of the agents, while the (summation of) backward-messages correspond to the state-action value functions of the lower dimensional MDPs.

3.2.2 Rauch-Tung-Striebel Inference

Forward-backward techniques have been successfully applied to some interesting planning models, see *e.g.* [170, 171, 102, 104, 169, 77]. However, to obtain efficient inference routines with these forward-backward techniques it is necessary to be able to combine the backward messages along the different components of the reward weighted trajectory distribution. A good illustration of this point is given by the model-based forward-backward algorithm for continuous systems in [77]. As the systems considered in [77] are continuous it is not possible to combine the backward messages, which results in a forward-backward algorithm whose computational complexity scales quadratically with the (finite) planning horizon. This limitation of forward-backward techniques has thus far restricted their (efficient) application to models where it is either possible to enumerate over the state-action space [170, 171], the components of the reward weighted distribution have a sufficiently small induced tree-width that the junction tree algorithm is feasible [169], or the reward weighted trajectory distribution has a particular form of sparsity *s.t.* it is only necessary to enumerate over subspaces of the state-action space [102]. As an alternative we present an inference technique that is applicable to continuous and high-dimensional systems, where it is either not possible to enumerate over the state-action space or the induced tree-width of each component of the reward weighted trajectory distribution is prohibitively large. The underlying idea of the inference procedure is analogous to performing *Rauch-Tung-Striebel* (RTS) smoothing [133] on (3.1), for which reason we refer to this family of inference algorithms as RTS-inference. The derivation is complicated by the fact that (3.1) has a mixture structure over the points in the planning horizon, which isn't present in standard latent variable time-series models for which the RTS smoother was originally designed. As we shall see, however, it is possible to overcome this difficulty by exploiting the

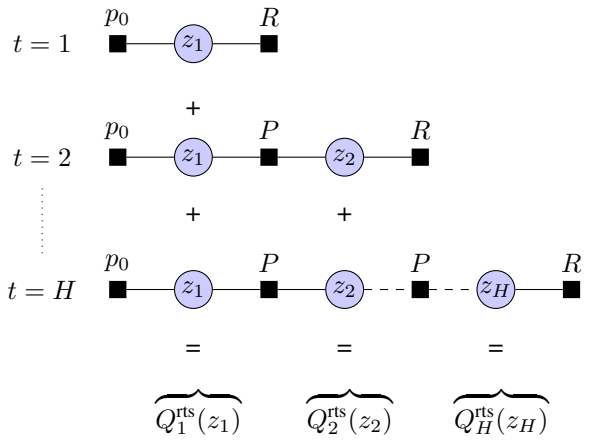


Figure 3.3: An example of how the finite horizon reward weighted trajectory distribution (3.1) splits into Rauch-Tung-Striebel state-action value functions.

special structure of (3.1) that allows inference to be performed over multiple mixture components simultaneously. An additional complication is the possibly infinite planning horizon, which we handle through the derivation of an appropriate fixed-point equation.

We currently focus on finite planning horizons, where we shall detail the formulation for the infinite horizon discounted rewards framework shortly. We first prove the following simple lemma which shows that, conditioned on $t \in \mathbb{N}_H$ and $\boldsymbol{z}_{\tau+1}$, where $\tau \in \mathbb{N}_{t-1}$, the reward weighted trajectory distribution over $\boldsymbol{z}_\tau$ is independent of t and is equal to the system reversal dynamics.

Lemma 1. *Given any $t \in \mathbb{N}_H$ and $\tau \in \mathbb{N}_{t-1}$ the unnormalised distribution $\tilde{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau+1}, t; \boldsymbol{w})$ is independent of t and takes the form*

$$\tilde{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau+1}, t; \boldsymbol{w}) = p(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau+1}; \boldsymbol{w}) \tag{3.4}$$

where $p(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau+1}; \boldsymbol{w})$ is the marginal of the trajectory distribution (2.2).

Proof. For any given $\tau \in \mathbb{N}_t$ the marginal of the reward weighted trajectory distribution $\tilde{p}(\boldsymbol{z}_{\tau:t}, t; \boldsymbol{w})$ takes the form

$$\tilde{p}(\boldsymbol{z}_{\tau:t}, t; \boldsymbol{w}) = p(\boldsymbol{z}_\tau; \boldsymbol{w}) \Bigg\{ \prod_{\tau'=\tau}^{t-1} p(\boldsymbol{z}_{\tau'+1}|\boldsymbol{z}_{\tau'}; \boldsymbol{w}) \Bigg\} R(\boldsymbol{z}_t),$$

As $\tau < t$ we have a similar expression for the marginal $\tilde{p}(\boldsymbol{z}_{\tau+1:t}, t; \boldsymbol{w})$. Using the Markovian structure of the reward weighted trajectory distribution means that the conditional distribution takes the form

$$\tilde{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau+1}, t; \boldsymbol{w}) = \frac{p(\boldsymbol{z}_\tau; \boldsymbol{w}) p(\boldsymbol{z}_{\tau+1}|\boldsymbol{z}_\tau; \boldsymbol{w})}{p(\boldsymbol{z}_{\tau+1}; \boldsymbol{w})} = p(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau+1}; \boldsymbol{w}).$$

□

We now introduce a new set of state-action value functions, which we refer to as Rauch-Tung-Striebel state-action value functions, or just RTS state-action value functions. These new state-action value functions play a prominent role in the RTS-inference formulation and are similar to the standard

state-action value function, with the exception of a prefactor of the state-action occupancy distribution. In particular, for each $\tau \in \mathbb{N}_H$ we define the function

$$Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = \sum_{t=\tau}^{H} \tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = p_\tau(\boldsymbol{z}; \boldsymbol{w}) Q_\tau^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w}), \tag{3.5}$$

where $Q_\tau^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w})$ is the standard state-action value function. Note that the summation of $\boldsymbol{z}$-marginals of (3.1) that appears in the parameter update equation of parametric policy search methods can be written in terms of these RTS state-action value functions as follows

$$\sum_{t=1}^{H} \sum_{\tau=1}^{t} \tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = \sum_{\tau=1}^{H} \sum_{t=\tau}^{H} \tilde{p}(\boldsymbol{z}, \tau, t; \boldsymbol{w}) = \sum_{\tau=1}^{H} Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}), \tag{3.6}$$

so that an efficient calculation of these functions is sufficient to provide an efficient routine to calculate the statistics necessary for a policy update. An illustration of how the marginals of the reward weighted trajectory distribution can be written in terms of these RTS state-action value functions is given in fig(3.3). To obtain an efficient method for the calculation of the RTS state-action value functions we use lemma 1 to obtain the following recursive relationship.

Lemma 2. *Given $\tau \in \mathbb{N}_{H-1}$, the function $Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ satisfies*

$$Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = p(\boldsymbol{z}, \tau; \boldsymbol{w}) R(\boldsymbol{z}) + \sum_{\boldsymbol{z}'} p(\boldsymbol{z}_\tau = \boldsymbol{z} | \boldsymbol{z}_{\tau+1} = \boldsymbol{z}'; \boldsymbol{w}) Q_{\tau+1}^{\text{rts}}(\boldsymbol{z}'; \boldsymbol{w}). \tag{3.7}$$

Proof. We start by rewriting the function $Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ as

$$Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = \tilde{p}(\boldsymbol{z}, \tau, \tau; \boldsymbol{w}) + \sum_{t=\tau+1}^{H} \sum_{\boldsymbol{z}'} \tilde{p}(\boldsymbol{z}, \boldsymbol{z}', \tau, \tau+1, t; \boldsymbol{w}), \tag{3.8}$$

where we have introduced the $\boldsymbol{z}$-variable of the next time-step, $\boldsymbol{z}_{\tau+1}$. Now, by lemma 1, we have that for each $t \in \{\tau+1, \ldots, H\}$

$$\begin{aligned} \tilde{p}(\boldsymbol{z}, \boldsymbol{z}', \tau, \tau+1, t; \boldsymbol{w}) &= \tilde{p}(\boldsymbol{z}_\tau = \boldsymbol{z} | \boldsymbol{z}_{\tau+1} = \boldsymbol{z}', t; \boldsymbol{w}) \tilde{p}(\boldsymbol{z}_{\tau+1}, t; \boldsymbol{w}), \\ &= p(\boldsymbol{z}_\tau = \boldsymbol{z} | \boldsymbol{z}_{\tau+1} = \boldsymbol{z}'; \boldsymbol{w}) \tilde{p}(\boldsymbol{z}_{\tau+1}, t; \boldsymbol{w}). \end{aligned}$$

Substituting this into (3.8) we obtain

$$Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = p(\boldsymbol{z}, \tau; \boldsymbol{w}) R(\boldsymbol{z}) + \sum_{\boldsymbol{z}'} p(\boldsymbol{z}_\tau = \boldsymbol{z} | \boldsymbol{z}_{\tau+1} = \boldsymbol{z}'; \boldsymbol{w}) \sum_{t=\tau+1}^{H} \tilde{p}(\boldsymbol{z}', \tau+1, t; \boldsymbol{w}),$$

where we have used the fact that $p(\boldsymbol{z}_\tau | \boldsymbol{z}_{\tau+1})$ depends only upon τ and not upon t. The result now follows from the definition of $Q_{\tau+1}^{\text{rts}}(\boldsymbol{z}'; \boldsymbol{w})$. □

The recursive equation (3.7) can be seen as a new form of Bellman equation. The standard Bellman equation (3.3) is a backward equation in a forward-backward inference routine, while (3.7) can be seen

as a *forward-then-backward* equation. It is now clear how to construct an efficient inference algorithm for the calculation of the RTS state-action value functions using the recursive equation (3.7). Firstly, due to the Markovian assumption, the trajectory distribution (2.2) is chain structured and all the $\boldsymbol{z}$-marginals can be calculated in linear time [178]. Given the marginals of the trajectory distribution the first term in (3.7) is easy to calculate, which also means that $Q_H^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ is easy to calculate because it takes the simple form

$$Q_H^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = p(\boldsymbol{z}, H; \boldsymbol{w}) R(\boldsymbol{z}).$$

Once the function $Q_H^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ has been calculated all of the remaining functions $\{Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})\}_{\tau \in \mathbb{N}_{H-1}}$ can be computed by repeated use of the recursion (3.7).

Before continuing we briefly highlight some important differences between performing inference in terms of the RTS state-action value functions defined in (3.5) in comparison to the classical state-action value functions from the planning literature. An important difference is the direction of the transition dynamics in the two recursions (3.7) and (3.3), where in (3.7) the transition dynamics are going backwards in time, while in (3.3) the transition dynamics are going forward in time. This is a subtle but important point and, as we shall see, it has some important consequences. For instance in continuous systems it allows for the construction of recursions over the moments of the RTS state-action value functions, as opposed to the functions themselves. We shall soon consider the RTS framework when applied to linear dynamical systems with a possibly non-linear reward structure. The recursions over the moments of the RTS state-action value functions result in an inference algorithm with a computational complexity that scales linearly *w.r.t.* the planning horizon, in the case of a finite planning horizon, as opposed to the quadratic scaling of forward-backward inference in this model [77]. The change in the direction of the transition dynamics is possible due to the prefactor of the state-action occupancy distribution, showing that this prefactor actually plays an vital role in the derivation of (3.7). Another important difference is the *normalisation constant*[5] of the two forms of state-action value function and their relation to the component weights of the reward weighted trajectory distribution. Note that we have $\sum_{\boldsymbol{z}} Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = \tilde{p}(t \geq \tau; \boldsymbol{w})$, for each $\tau \in \mathbb{N}_H$, so that the normalisation constant of the RTS state-action value functions can be directly obtained from the component weights of the reward weighted trajectory distribution. Given the forward messages, or approximations thereof, these component weights are typically easy to calculate. This means that it is only necessary to calculate/approximate the RTS state-action value functions up to a positive multiplicative constant. This is a useful property in, for example, high-dimensional systems where inference in the reward weighted trajectory distribution is intractable. In such cases it is sufficient to consider the distributions

$$\hat{Q}_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) \propto Q_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}), \qquad \tau \in \mathbb{N}_H,$$

and these distributions can be directly approximated through various approximate inference techniques. Shortly we shall consider such a high-dimensional discrete system where it is possible to use this

[5] By normalisation of a non-negative function, $f(\boldsymbol{z})$, we mean the scalar $Z = \sum_{\boldsymbol{z}} f(\boldsymbol{z})$. It is assumed that f is not identically zero so this term is always positive.

property to construct a recursion over these distributions that is analogous to (3.7). In contrast the normalisation constants of the standard state-action value functions have no relation to the component weights of the reward weighted trajectory distribution, with the forward messages providing no information about these functions. Therefore if one were to consider a corresponding set of distributions for the standard state-action value functions, so as to apply the similar approximate inference routines, it would be necessary to also approximate the normalisation constant. This would be an intractable problem in general and so an additional set of approximations would be required using forward-backward inference. Before detailing the application of RTS-inference to various planning models we first formalise the extension to the discounted infinite horizon framework.

Infinite Discounted Planning Horizons

To extend RTS-inference to the discounted infinite horizon framework we rely on the fact that, given the system is stable, stationarity of the state-action occupancy distribution will be reached in a finite amount of time. Given that stationarity is reached by the time-point $\hat{\tau}$ it is straightforward to show that for any $\tau \geq \hat{\tau}$ we have the relation

$$Q^{\text{rts}}_{\tau+1}(\boldsymbol{z}; \boldsymbol{w}) = \gamma Q^{\text{rts}}_{\tau}(\boldsymbol{z}; \boldsymbol{w}). \tag{3.9}$$

This relation can now be used to obtain a formulation for calculating the marginals over $\mathcal{Z}$ for the infinite number of time-points of (3.1). Firstly we split the infinite summation in the parameter update function into the terms before and after stationarity of the trajectory has been reached, *i.e.*

$$\sum_{t=1}^{\infty} Q^{\text{rts}}_{t}(\boldsymbol{z}; \boldsymbol{w}) = \sum_{t=1}^{\hat{\tau}-1} Q^{\text{rts}}_{t}(\boldsymbol{z}; \boldsymbol{w}) + \sum_{t=\hat{\tau}}^{\infty} Q^{\text{rts}}_{t}(\boldsymbol{z}; \boldsymbol{w}). \tag{3.10}$$

The relation (3.9) suggests that from the $\hat{\tau}^{\text{th}}$ time-point onwards the RTS state-action value functions are, up to scaling, time-invariant. It is therefore natural to introduce a time-invariant RTS state-action value function, $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$, which is defined by $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = \gamma^{1-\hat{\tau}} Q^{\text{rts}}_{\hat{\tau}}(\boldsymbol{z}; \boldsymbol{w})$. Note that by (3.9) we have $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = \gamma^{1-t} Q^{\text{rts}}_{t}(\boldsymbol{z}; \boldsymbol{w})$, for all $t \geq \hat{\tau}$, so that $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ is indeed independent of time and this definition is well defined. The infinite summation that occurs (3.10) can now be performed analytically as follows

$$\sum_{t=\hat{\tau}}^{\infty} Q^{\text{rts}}_{t}(\boldsymbol{z}; \boldsymbol{w}) = Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) \sum_{t=\hat{\tau}}^{\infty} \gamma^{t-1} = \frac{\gamma^{\hat{\tau}-1}}{1-\gamma} Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}). \tag{3.11}$$

To perform the summation in (3.10) it now remains to obtain an analytic solution to $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$. This solution is obtained from the following recursion, which almost immediately follows from the relations (3.7) and (3.9) and the definition of $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$,

$$Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = p(\boldsymbol{z}; \boldsymbol{w}) R(\boldsymbol{z}) + \gamma \sum_{\boldsymbol{z}'} \overleftarrow{p}(\boldsymbol{z}|\boldsymbol{z}') Q^{\text{rts}}(\boldsymbol{z}'; \boldsymbol{w}), \tag{3.12}$$

Calculate Forward Messages: Iterate the forward-message recursion until the forward-messages converge to the stationary distribution.

Calculate Stationary RTS state-action value function: Use the stationary occupancy distribution and the stationary system reversal dynamics to calculate the stationary RTS state-action value function, $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$, using either (3.13) or the fixed-point equation (3.12).

Calculate Backward Messages: Use the recursive equation (3.7) to propagate the RTS state-action value functions backwards in time $Q_t^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$, for $t = \hat{\tau} - 1, \ldots, 1$.

Algorithm 3.1: Infinite Horizon RTS-Inference

where $p(\boldsymbol{z}; \boldsymbol{w})$ is the stationary occupancy distribution and $\overleftarrow{p}(\boldsymbol{z}|\boldsymbol{z}')$ is the stationary system reversal dynamics. An algebraic solution for $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ is obtained from (3.12) by observing that

$$Q^{\text{rts}} = (I - \gamma \overleftarrow{P})^{-1} \boldsymbol{\mu}, \tag{3.13}$$

where $\boldsymbol{\mu}$ is the point-wise product of the stationary occupancy distribution with the reward function. An alternative solution to $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ can be obtained by iterating the fixed-point equation (3.12) until convergence, which may be preferable in systems where the matrix inversion is expensive or infeasible. Note that the presence of the discount factor in (3.12) makes this fixed-point equation a contraction mapping so that convergence to a unique fixed-point is guaranteed. The complete algorithm for calculating the infinite number of marginals of (3.1) required for a parameter update is detailed in algorithm(3.1).

Continuous Models

In continuous problems it will generally only be possible to maintain an analytical model-based inference procedure for linear systems with Gaussian noise. While a forward-backward inference procedure has been derived for this model [77] it has a run-time that is quadratic in the planning horizon and is only applicable to finite planning horizon problems. In this section we will detail the RTS-inference procedure for this model, which has a run-time that is linear in the planning horizon. Additionally we provide a analytical procedure for discounted infinite horizon problems, which has a run-time that is determined by the eigenvalue of the state-action transition matrix with largest magnitude.

In a linear dynamical system the initial state distribution, transition dynamics and policy take the form

$$\begin{aligned}
p(\boldsymbol{s}_1) &= \mathcal{N}(\boldsymbol{s}_1|\mu_0, \Sigma_0), \\
p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) &= \mathcal{N}(\boldsymbol{s}_{t+1}|A\boldsymbol{s}_t + B\boldsymbol{a}_t, \Sigma), \\
p(\boldsymbol{a}_t|\boldsymbol{s}_t; K, m, \pi_\sigma) &= \mathcal{N}(\boldsymbol{a}_t|K\boldsymbol{s}_t + \boldsymbol{m}; \pi_\sigma),
\end{aligned}$$

where all the matrices and vectors are assumed to be of appropriate size. Note that the mean of these Gaussians is a linear combination of the conditioning variable, which maintains tractability. Under these policies the quantities needed to perform a policy update are the first two moments of the RTS state-

action value functions. In particular we need to calculate the terms

$$\sum_{\tau=1}^{H} \mathbb{E}_{Q_\tau^{\text{rts}}(\boldsymbol{z};\boldsymbol{w})}\big[\boldsymbol{z}\big], \qquad \sum_{\tau=1}^{H} \mathbb{E}_{Q_\tau^{\text{rts}}(\boldsymbol{z};\boldsymbol{w})}\big[\boldsymbol{z}\boldsymbol{z}^\top\big],$$

which, as we shall see, can be calculated in linear time.

On first sight it would appear that the reward function also has to have some restricted Gaussian form in order to maintain tractability. However, as noted in [77], the objective function (2.1) is linear in the reward function and so it is possible to handle arbitrarily complex reward structures through a linear mixture of Gaussians. This means the reward function can take the form

$$R(\boldsymbol{z}) = \sum_{j=1}^{J} w_j \bar{\mathcal{N}}(\boldsymbol{y}_j | M\boldsymbol{z}, L_j),$$

where $\bar{\mathcal{N}}$ denotes an unnormalised Gaussian. It is easy to see that the statistics required for the various parametric policy search algorithms considered in chapter(2) can now be obtained by performing inference in the J mixtures independently. For this reason the rest of the derivation will assume only a single component.

To perform RTS-inference one first calculates the forward-messages, *i.e.* the state-action marginals of the trajectory distribution. As the policy is considered fixed during inference this is equivalent to a LDS in the state-action space and so the forward-messages simply follow from standard LDS recursions [15]. In particular if we denote the mean and covariance of the state-action marginal at time t by $\boldsymbol{\mu}_t$ and Σ_t respectively then we have the recursions

$$\boldsymbol{\mu}_{t+1} = F(K)\boldsymbol{\mu}_t + \bar{\boldsymbol{m}}, \quad \Sigma_{t+1} = F(K)\Sigma_t F(K)^\top + \bar{\Sigma}, \tag{3.14}$$

where

$$\bar{\boldsymbol{m}} = \begin{bmatrix} 0 \\ \boldsymbol{m} \end{bmatrix} \qquad \bar{\Sigma} = \begin{bmatrix} \Sigma & \Sigma K^\top \\ K\Sigma & K\Sigma K^\top + \pi_\sigma I_{n_a} \end{bmatrix}, \qquad F(K) = \begin{bmatrix} A & B \\ KA & KB \end{bmatrix},$$

and the initial message takes the form

$$\boldsymbol{\mu}_1 = \begin{bmatrix} \boldsymbol{\mu}_0 \\ K\boldsymbol{\mu}_0 + \boldsymbol{m} \end{bmatrix}, \qquad \Sigma_1 = \begin{bmatrix} \Sigma_0 & \Sigma_0 K' \\ K\Sigma_0 & K\Sigma_0 K^\top + \pi_\sigma^2 I_{n_a} \end{bmatrix}. \tag{3.15}$$

Once the forward-messages have been calculated it is necessary to to calculate the statistics of the system reversal dynamics, which are required to perform the RTS-inference recursions (3.7). As the system is linear the reversal dynamics can be calculated using standard conditional Gaussian formulae, see *e.g.* [15]. Given the statistics of the state-action marginals the system reversal dynamics are given by

$$p(\boldsymbol{z}_t | \boldsymbol{z}_{t+1}; \boldsymbol{w}) = \mathcal{N}(\boldsymbol{z}_t | \overleftarrow{G}_t \boldsymbol{z}_{t+1} + \overleftarrow{\boldsymbol{m}}_t, \overleftarrow{\Sigma}_t), \tag{3.16}$$

where $\overleftarrow{G}_t$, $\overleftarrow{\boldsymbol{m}}_t$ and $\overleftarrow{\Sigma}_t$ are given by

$$\begin{aligned}
\overleftarrow{G}_t &= \Sigma_t F(K)^\top (F(K)\Sigma_t F(K)^\top + \bar{\Sigma})^{-1}, \\
\overleftarrow{\boldsymbol{m}}_t &= \boldsymbol{\mu}_t - G_t(F(K)\boldsymbol{\mu}_t + \bar{\boldsymbol{m}}), \\
\overleftarrow{\Sigma}_t &= \Sigma_t - \Sigma_t F(K)^\top (F(K)\Sigma_t F(K)^\top + \bar{\Sigma})F(K)\Sigma_t.
\end{aligned}$$

For convenience we will often write this distribution with the notation $p(\boldsymbol{z}_t = \boldsymbol{z}|\boldsymbol{z}_{t+1} = \boldsymbol{z}'; \boldsymbol{w}) \equiv \overleftarrow{p}_t(\boldsymbol{z}|\boldsymbol{z}'; \boldsymbol{w})$. Additionally, to calculate the moments of the RTS state-action value functions it is necessary to calculate expectations of the form

$$\mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\big[R(\boldsymbol{z})\boldsymbol{z}\big], \qquad \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\big[R(\boldsymbol{z})\boldsymbol{z}\boldsymbol{z}^\top\big].$$

As these are Gaussian integrals they are easy to calculate, but for completeness we give their explicit form. Denoting these moments respectively as $\boldsymbol{\mu}_t^R$ and Σ_t^R, then we have

$$\begin{aligned}
\boldsymbol{\mu}_t^R &= R(t)\Big(\boldsymbol{\mu}_t + \Sigma_t M^\top ((M\Sigma_t M^\top + L)^{-1}(\boldsymbol{y} - M\boldsymbol{\mu}_t))\Big), \\
\Sigma_t^R &= R(t)\Big(\Sigma_t - \Sigma_t M^\top (M\Sigma_t M^\top + L)^{-1} M\Sigma_t + R(t)^{-2}\boldsymbol{\mu}_t^R\big(\boldsymbol{\mu}_t^R\big)^\top\Big),
\end{aligned}$$

where $R(t)$ denotes the expected reward at t^{th} time-point.

Finally, having calculated the forward-messages and the system reversal dynamics it is then possible to obtain the first two moments of the RTS state-action value functions using (3.7). As $\boldsymbol{z}_t$ depends on $\boldsymbol{z}_{t+1}$ linearly in the reversal dynamics it means that the first two moments of $Q_t^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ take the form

$$\begin{aligned}
\mathbb{E}_{Q_t^{\text{rts}}(\boldsymbol{z};\boldsymbol{w})}\big[\boldsymbol{z}\big] &= \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\Big[\boldsymbol{z}R(\boldsymbol{z})\Big] + \mathbb{E}_{Q_{t+1}^{\text{rts}}(\boldsymbol{z}';\boldsymbol{w})}\Big[\mathbb{E}_{\overleftarrow{p}_t(\boldsymbol{z}|\boldsymbol{z}';\boldsymbol{w})}\big[\boldsymbol{z}\big]\Big], \\
&= \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\Big[\boldsymbol{z}R(\boldsymbol{z})\Big] + \mathbb{E}_{Q_{t+1}^{\text{rts}}(\boldsymbol{z}';\boldsymbol{w})}\Big[\big(\overleftarrow{G}_t\boldsymbol{z}' + \overleftarrow{\boldsymbol{m}}_t\big)\Big].
\end{aligned}$$

and

$$\begin{aligned}
\mathbb{E}_{Q_t^{\text{rts}}(\boldsymbol{z};\boldsymbol{w})}\big[\boldsymbol{z}\boldsymbol{z}^\top\big] &= \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\Big[\boldsymbol{z}\boldsymbol{z}^\top R(\boldsymbol{z})\Big] + \mathbb{E}_{Q_{t+1}^{\text{rts}}(\boldsymbol{z}';\boldsymbol{w})}\Big[\mathbb{E}_{\overleftarrow{p}_t(\boldsymbol{z}|\boldsymbol{z}';\boldsymbol{w})}\big[\boldsymbol{z}\boldsymbol{z}^\top\big]\Big], \\
&= \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\Big[\boldsymbol{z}\boldsymbol{z}^\top R(\boldsymbol{z})\Big] + \mathbb{E}_{Q_{t+1}^{\text{rts}}(\boldsymbol{z}';\boldsymbol{w})}\Big[\big(\overleftarrow{G}_t\boldsymbol{z}' + \overleftarrow{\boldsymbol{m}}_t\big)\big(\overleftarrow{G}_t\boldsymbol{z}' + \overleftarrow{\boldsymbol{m}}_t\big)^\top + \overleftarrow{\Sigma}_t\Big].
\end{aligned}$$

The recursions for the first two moments of the RTS state-action value functions can now be immediately read off these equations to give

$$\boldsymbol{\mu}_t^Q = \boldsymbol{\mu}_t^R + Z_{t+1}\overleftarrow{\boldsymbol{m}}_t + G_t\boldsymbol{\mu}_{t+1}^Q, \tag{3.17}$$

$$\boldsymbol{\Sigma}_t^Q = \Sigma_t^R + Z_{t+1}(\overleftarrow{\Sigma}_t + \overleftarrow{\boldsymbol{m}}_t\overleftarrow{\boldsymbol{m}}_t^\top) + G_t(\Sigma_{t+1}^Q + \boldsymbol{\mu}_{t+1}^Q\overleftarrow{\boldsymbol{m}}_t^\top + \overleftarrow{\boldsymbol{m}}_t(\boldsymbol{\mu}_{t+1}^Q)^\top)G_t^\top, \tag{3.18}$$

Calculate Forward Messages: Calculate the initial state-action marginal (3.15) and iterate the forward-message recursions (3.14) up until the end of the planning horizon.

Calculate System Reversal Dynamics: Using the statistics of the forward-messages calculate the system reversal dynamics (3.16).

Calculate Moments of RTS state-action value functions: Calculate the first two moments of the RTS state-action value function at the final time-point and then use the recursive equations (3.17) & (3.18) to calculate the first two moments of the remaining RTS state-action value functions.

Algorithm 3.2: Finite Horizon Inference of RTS state-action value functions in a Linear Dynamical System

where we have used the notation $Z_{t+1} = \sum_{\tau=t+1}^{H} R(t)$ as well as using the fact that

$$\mathbb{E}_{Q^{\text{rts}}_{t+1}(\boldsymbol{z};\boldsymbol{w})}\big[1\big] = Z_{t+1}.$$

This completes the description of the RTS-inference procedure for this model in finite horizon problems. A summary of the algorithm is given in algorithm(3.2).

To extend this algorithm to discounted infinite horizon problems it is necessary for the system to converge to a stationary distribution, which requires the spectrum of $F(K)$ to be contained within the unit circle. We shall talk about this criterion shortly but for now we assume that this is the case. In this situation one calculates the forward-messages in the same manner as in the finite horizon problem, now however instead of terminating the iterations at the final time-point they should be terminated once the messages have converged to stationary distribution. One then obtains the stationary system reversal dynamics, again using the standard Gaussian conditional formulae. Denoting the first and second moment of $Q(\boldsymbol{z})$ by $\boldsymbol{\mu}^Q$ and $\boldsymbol{\Sigma}^Q$ respectively, then applying the same techniques used to derive (3.17) and (3.18) but now using the recursive equation (3.12) gives

$$\boldsymbol{\mu}^Q = \boldsymbol{\mu}^R + \gamma\big(Z\overleftarrow{\boldsymbol{m}} + G\boldsymbol{\mu}^Q\big), \tag{3.19}$$

$$\boldsymbol{\Sigma}^Q = \Sigma^R + \gamma\bigg(Z(\overleftarrow{\Sigma} + \overleftarrow{\boldsymbol{m}}\overleftarrow{\boldsymbol{m}}^\top) + G(\Sigma^Q + \boldsymbol{\mu}^Q\overleftarrow{\boldsymbol{m}}^\top + \overleftarrow{\boldsymbol{m}}(\boldsymbol{\mu}^Q)^\top)G^\top\bigg). \tag{3.20}$$

One can either obtain the moments $\boldsymbol{\mu}^Q$ and $\boldsymbol{\Sigma}^Q$ either by solving the linear systems (provided this is possible) or by iterative application of these equations to some initial estimate. As previously mentioned these mapping are contraction mappings so that this iterative solution will be unique regardless of the initial estimate. A summary of the infinite horizon algorithm is given in algorithm(3.3).

It is important to note some of the characteristics of the set of policy parameters that lead to a state-action transition matrix with all of its eigenvalues lying within the unit circle. Not only is this necessary for the infinite horizon recursions but it is also important in terms of numerical stability of finite horizon problems. When the magnitude of any of the eigenvalues exceed unity the system will diverge exponentially quickly and it will only be possible in practice to handle a limited planning horizon. It is important to note that this is a property of the system and not the inference technique and as such is

Calculate Forward Messages: Calculate the initial state-action marginal (3.15) and iterate the forward-message recursions (3.14) up until convergence of the occupancy distribution.

Calculate System Reversal Dynamics: Using the statistics of the forward-messages calculate the system reversal dynamics before and after convergence of the occupancy distribution.

Calculate Moments of Time-Invariant RTS state-action value function: Calculate the first two moments of the time-invariant RTS state-action value function, $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$, using the recursive equations (3.19) & (3.20). Additionally calculate the moments of $\sum_{t=\hat{\tau}}^{\infty} Q_t^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ using the relation (3.11).

Calculate Moments of Time-Variant RTS state-action value functions: Using the statistics of $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ calculate the first two moments of $Q_{\hat{\tau}}^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$ through the relation $Q^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w}) = \gamma^{1-\hat{\tau}} Q_{\hat{\tau}}^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})$. Then propagate the moments of the RTS state-action value functions back for all $\tau < \hat{\tau}$ using the standard recursive equation (3.7).

Algorithm 3.3: Infinite Horizon Inference of RTS state-action value functions in a Linear Dynamical System

true both of model-based and model-free algorithms. Obviously this set depends only upon K and is characterised as follows

$$\mathcal{K} = \left\{ K \in \mathbb{R}^{n_a \times n_s} \middle\| \rho(F(K)) < 1 \right\},$$

where ρ is the spectral radius operator. The first thing to note is that $\mathcal{K}$ is convex. Indeed suppose that $K_1, K_2 \in \mathcal{K}$, then given any $\lambda \in [0, 1]$ we have

$$\begin{aligned} F(\lambda K_1 + (1-\lambda)K_2) &= \begin{bmatrix} A & B \\ \big(\lambda K_1 + (1-\lambda)K_2\big)A & \big(\lambda K_1 + (1-\lambda)K_2\big)B \end{bmatrix}, \\ &= \lambda \begin{bmatrix} A & B \\ K_1 A & K_1 B \end{bmatrix} + (1-\lambda) \begin{bmatrix} A & B \\ K_2 A & K_2 B \end{bmatrix}, \\ &= \lambda F(K_1) + (1-\lambda) F(K_2). \end{aligned}$$

Since $\rho(F(K_1)) < 1$ and $\rho(F(K_2)) < 1$ it is easy to see that $\rho(F(\lambda K_1 + (1-\lambda)K_2)) < 1$ and hence $\lambda K_1 + (1-\lambda)K_2 \in \mathcal{K}$. The convexity of $\mathcal{K}$ is obviously a desirable property as it means that, provided that the initial policy paramterisation is in $\mathcal{K}$, it is possible (at least in theory) to reach any optima of the objective function without leaving the set $\mathcal{K}$, outside of which these algorithms will become numerically unstable in large planning horizons. Another important characteristic of interest is the boundedness of $\mathcal{K}$. Generally speaking this set will not be bounded as a bounded spectrum does not imply that the elements of the matrix are bounded.

Another issue is obtaining uniform samples from $\mathcal{K}$, which is important in terms of initialisation of these algorithms. This is important because one could easily restrict oneself to a poor part of the parameter space by sampling in a non-uniform manner, especially as these methods are local optimisation techniques. Additionally, obtaining uniform samples is important for fair comparison of algorithms. It is easy to imagine a situation where the performance of two algorithms differ in two parts of the parameters space, so by using non-uniform samples it is easily possible that results could be unfairly biased. Two

immediate possibilities are rejection sampling and Gibbs sampling, see *e.g.* [62]. In rejection sampling one would first obtain a set $\tilde{\mathcal{K}} \subset \mathbb{R}^{n_a \times n_s}$ *s.t.* $\mathcal{K} \subset \tilde{\mathcal{K}}$ and from which it is easy to obtain uniform samples. Given $\tilde{\mathcal{K}}$ one then obtains uniform samples from $\mathcal{K}$ by first sampling from $\tilde{\mathcal{K}}$ and then either accepting or rejecting this sample depending on whether it is in $\mathcal{K}$ or not. Gibbs sampling is an iterative process that would yield uniform samples upon convergence. At each iteration a uniform sample would be taken for a small subset of the parameters of K, while keeping the others fixed. This process would then be iterated, alternating the parameters held fixed, until the algorithm has converged to a uniform distribution over $\mathcal{K}$. The main idea is that by keeping all but a small subset of the parameters fixed during each sampling stage it will be easier to obtain a characterisation of this subset of parameters. See [62] for a more detailed description. Both of these sampling procedures require $\mathcal{K}$ to be bounded and when this is not the case it is not possible to obtain uniform samples over $\mathcal{K}$.

This completes the discussion of inference in linear systems with a reward structure described by a mixture of Gaussians. It may appear at first sight that such systems are limited in their applicability due to the constraint that the dynamics and policy are both linear. This is not true, however, and there are several immediate methods that allow the modelling of non-linear systems. For example it is possible to model a wide range of non-linear systems through a process known as *feedback-linearisation*, which we shall discuss in further detail in section(4.3). Other possibilities are to take approaches similar to either [166] or [168], which were detailed in section(1.3.3). For example one could use any number of approximate inference routines, such as Expectation Propagation [115], to approximate the trajectory distribution with a Gaussian, which could then be used in the RTS-inference recursions. Finally an interesting avenue of possible future research is to model an original non-linear control problem through a switching linear dynamical system. There has been a large amount of research on inference of such systems in the statistics, machine learning and control literature [14, 63, 1], but there seem to have been little attempt at optimal control through these models. While exact inference becomes intractable [15] there are a large range of approximate inference techniques for these models, including VB approximations [63], Gaussian sum filtering [1] and expectation correction techniques [14]. These techniques result in inference recursions that are closely related to standard LDS recursions and one can expect a similar situation in the control setting. To the best of our knowledge such an approach has never been undertaken and could be an interesting avenue of research.

Before proceeding to another model we consider the difficulty of performing model-based inference in this model under the typical forward-backward paradigm. Suppose we wish to calculate the first moment of $p(\boldsymbol{z}, t; \boldsymbol{w})Q_t^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w})$, where $Q_t^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w})$ is the standard state-action value function, then it is necessary to calculate the integral

$$\mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\left[\boldsymbol{z}Q_t^{\text{fb}}(\boldsymbol{z};\boldsymbol{w})\right] = \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\left[\boldsymbol{z}R(\boldsymbol{z})\right] + \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\left[\boldsymbol{z}\mathbb{E}_{p(\boldsymbol{z}'|\boldsymbol{z};\boldsymbol{w})}\left[Q_{t+1}^{\text{fb}}(\boldsymbol{z}';\boldsymbol{w})\right]\right]. \tag{3.21}$$

The first term on the *r.h.s.* of (3.21) is easy and is equivalent to the corresponding term in RTS-inference.

The second term on the *r.h.s.* of (3.21) can be written in the equivalent form

$$\mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\left[\boldsymbol{z}\mathbb{E}_{p(\boldsymbol{z}'|\boldsymbol{z};\boldsymbol{w})}\left[Q^{\text{fb}}_{t+1}(\boldsymbol{z}';\boldsymbol{w})\right]\right] = \mathbb{E}_{p_Q(\boldsymbol{z},t;\boldsymbol{w})}\left[\boldsymbol{z}\right],$$

where $p_Q(\boldsymbol{z}, t; \boldsymbol{w})$ is given by

$$p_Q(\boldsymbol{z}, t; \boldsymbol{w}) = p(\boldsymbol{z}, t; \boldsymbol{w})\mathbb{E}_{p(\boldsymbol{z}'|\boldsymbol{z};\boldsymbol{w})}\left[Q^{\text{fb}}_{t+1}(\boldsymbol{z}';\boldsymbol{w})\right].$$

The unnormalised distribution $p_Q(\boldsymbol{z}, t; \boldsymbol{w})$ is an unnormalised mixture of Gaussians with $(H - t)$ components, which can be seen from the definition of the state-action value function and the fact that the system is linear Gaussian. Therefore, to calculate the expectation $\mathbb{E}_{p_Q(\boldsymbol{z},t;\boldsymbol{w})}\left[\boldsymbol{z}\right]$ it is necessary to calculate the expectation over each component of the mixture individually. This is equivalent to the procedure that [77] perform and its run-time it quadratic in the planning horizon. Additionally, there is no easy extension to infinite planning horizons, which would require calculating the expectation over an infinite number of components in a infinite mixture of Gaussians. In contrast RTS-inference uses the linearity of the system reversal dynamics to obtain a linear-time recursion for the calculation of the expectation over all of the mixtures simultaneously. Additionally, the infinite horizon recursion (3.12) calculates the expectation over an infinite number of mixtures and convergence is guaranteed through the contraction mapping theorem.

High-Dimensional Transition-Dependent Markov Decision Processes

In this section we demonstrate RTS-inference in high-dimensional discrete systems where inference in each component of the reward weighted trajectory distribution is intractable. We consider a specific application, both for notational convenience and to motivate the our approach, but the arguments hold in more general settings. At present we only highlight the general procedure and the advantages of RTS-inference, leaving the explicit construction of particular approximate inference algorithms for these systems as a point of future work. We restrict our attention to a finite planning horizon, where the extension to the discounted infinite horizon framework is possible through the fixed-point equation (3.12).

A popular class of Markov Decision Processes where inference in the reward weighted trajectory distribution is intractable is the class of factored multi-agent systems , see *e.g.* [71, 73, 72]. A typical example of such a system is an urban traffic network, where each agent corresponds to a traffic junction in the network and the objective is to minimise congestion. In this class of MDPs the state-action vector is given by

$$\boldsymbol{z}_t = \left(s^1_t, ..., s^N_t, a^1_t, ..., a^N_t\right).$$

where N is equal to the number of agents in the system and $(s^i, a^i) \in \mathcal{S}^i \times \mathcal{A}^i$ corresponds to the state-action space of the i^{th} agent. The transition dynamics are typically sparse in these systems because each agent's dynamics are affected by only a subset of the other agents, *e.g.* neighbouring junctions in a traffic network. To formalise this idea slightly, while trying not to over complicate the point, we

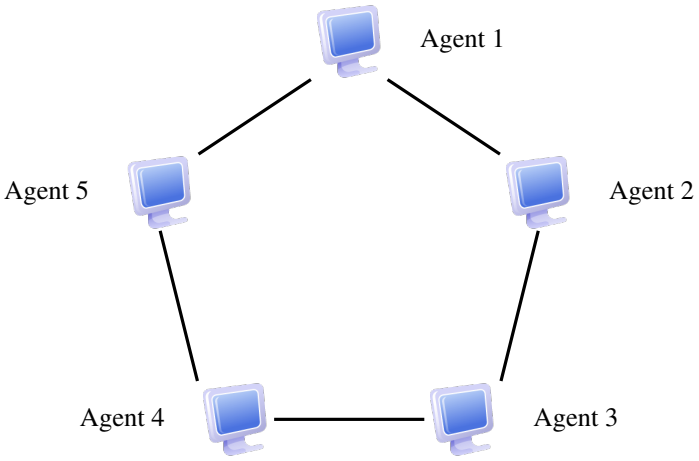


Figure 3.4: The neighbourhood graph of an example system network factored MDP.

define the *neighbourhood graph* to be a graph with N nodes, with each node corresponding to an agent in the system. There is a directed edge from node i to node j if the transition dynamics of j^{th} agent is dependent on the state-action pair of the i^{th} agent. An undirected edge is used for codependent agents. An example of such a neighbourhood graph is given in fig(3.4), which is a computer network system and each agent corresponds to a computer in the network. In this example the transition dynamics of the first agent are dependent on its own state and the state of agents 2 & 5, *i.e.* $p(s^1_{\text{next}}|\boldsymbol{z}) = p(s^1_{\text{next}}|z^1, z^2, z^5)$. A final assumption that is typically made in these systems is that the reward function has a sparse additive structure, *e.g.*

$$R(\boldsymbol{z}) = \sum_{i=1}^{N} R_i(s^i, a^i). \tag{3.22}$$

In the example of a traffic network this reward function may correspond to the amount of congestion at each junction in the network. For the remainder of this section we shall assume that the reward function has the form given in (3.22).

In order to employ approximate inference routines we assume that there is some reasonable sparse conditioning structure for the policies of the various agents. For example assume that there is a set of functions $\{\phi^i\}_{i\in\mathbb{N}_N}$ that map the state space to some lower-dimensional space *s.t.*

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \prod_{i=1}^{N} \pi^i(a^i|\phi^i(\boldsymbol{s})).$$

In this example the actions along the various dimensions of the action space are independent given the current state. This is not necessary and it is possible to model more sophisticated policies, but for simplicity we consider this simple case. In these systems there is often a natural selection of $\{\phi^i\}_{i\in\mathbb{N}_N}$ in terms of the structure of the neighbourhood graph, *e.g.* $\phi^i(\boldsymbol{s}) = \boldsymbol{s}^d$, where $\boldsymbol{s}^d$ is the vector of state variables that are a distance less than d away from i in the neighbourhood graph.

Given the reward function (3.22) the reward weighted trajectory distribution (3.1) is now a mixture over time-points and agents, *i.e.* $\hat{p}(\boldsymbol{z}_{1:t}, t, i; \boldsymbol{w}) \propto R_i(z^i_t)p(\boldsymbol{z}_{1:t}; \boldsymbol{w})$. A factor graph representation of

(3.1) for the example factored MDP introduced in fig(3.4) is given in fig(3.5). We alter definition of an RTS state-action value function accordingly

$$Q_\tau^{\text{rts}}(\boldsymbol{z};\boldsymbol{w}) = \sum_{t=\tau}^{H}\sum_{i=1}^{N}\tilde{p}(\boldsymbol{z},\tau,t,i;\boldsymbol{w}),$$

where $\tilde{p}(\boldsymbol{z},\tau,t,i;\boldsymbol{w}) = \tilde{p}(\boldsymbol{z}_\tau,t,i;\boldsymbol{w})$. Note that the product structure in the policy means that in order to update the policy, π^i, it is necessary to calculate the term

$$\sum_{\tau=1}^{H} Q_\tau^{\text{rts}}(a^i,\phi^i(\boldsymbol{s});\boldsymbol{w}),$$

so that it is only necessary to calculate marginals over small subsets of state-action variables. Additionally, due to the sparsity properties of the problem class, each component of the reward weighted trajectory distribution has a sparse graphical structure. Hence the policy evaluation stage of policy search methods in these problems is amenable, at least in theory, to approximate inference techniques. However, the construction of efficient inference algorithms is non-trivial due to the large number of components in this mixture distributions, where there are NH components in this example.

As in continuous systems the first step in RTS-inference is to perform (approximate) inference on the trajectory distribution to calculate the forward messages and the system reversal dynamics. Due to the high dimensional state-action space and the transition dependence between the agents of the system this inference will be intractable in general and so an approximate inference procedure is necessary, such as belief propagation or the cluster variational method, see *e.g.* [100] or [85]. The Markovian structure of the trajectory distribution suggests that a simple forward inference procedure is appropriate. Once the forward messages and system reversal dynamics have been calculated/approximated it is necessary to calculate the component weights in the reward weighted trajectory distribution, which in this example correspond to the expected reward terms for each time-point in the planning horizon and each agent in the system. Given the approximation to the trajectory distribution it is easy to calculate the expected reward for each time-point and each agent because to the sparse reward structure (3.22). These expected reward terms are important as they form the component weights in a mixture distribution that is central to the inference. We define the following terms

$$Z_{i,\tau} = \mathbb{E}_{p(z^i,\tau;\boldsymbol{w})}\Big[R_i(s^i,a^i)\Big], \qquad \forall i \in \mathbb{N}_N, \forall \tau \in \mathbb{N}_H,$$

$$Z_\tau = \textstyle\sum_{i=1}^{N} \mathbb{E}_{p(z^i,\tau;\boldsymbol{w})}\Big[R_i(s^i,a^i)\Big], \qquad \forall \tau \in \mathbb{N}_H,$$

$$Z_{\tau\to H} = \textstyle\sum_{t=\tau}^{H}\sum_{i=1}^{N} \mathbb{E}_{p(z^i,t;\boldsymbol{w})}\Big[R_i(s^i,a^i)\Big], \qquad \forall \tau \in \mathbb{N}_H.$$

Note that these terms are closely connected to the various component weights in the reward weighted trajectory distribution. For instance $\hat{p}(i,\tau;\boldsymbol{w}) \propto Z_{i,\tau}$, while $\hat{p}(\tau;\boldsymbol{w}) \propto Z_\tau$ and $\hat{p}(t \geq \tau;\boldsymbol{w}) \propto Z_{\tau\to H}$.

Given the approximation to the trajectory distribution, system reversal dynamics and the expected

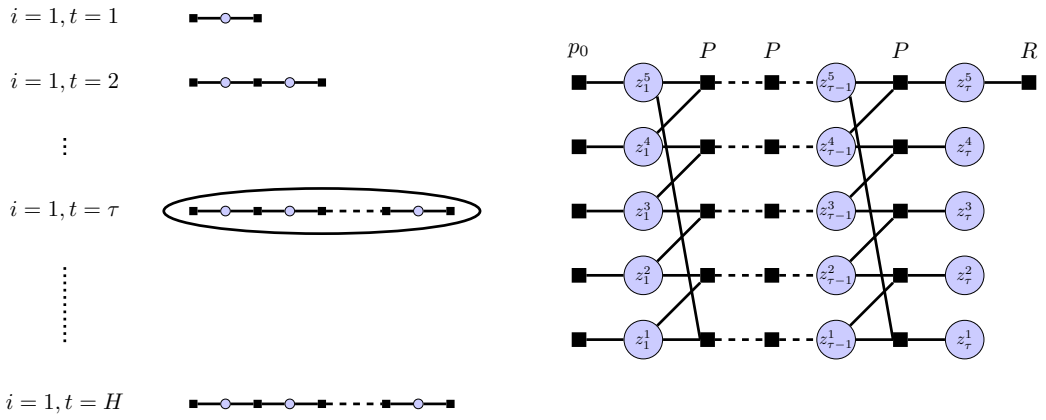


Figure 3.5: A factor graph representation of $\tilde{p}(\boldsymbol{z}_{1:\tau}, t, i; \boldsymbol{w})$ for the factored MDP considered in fig(3.4), with a detailed illustration of the structure for the time-component, $t = \tau$, and agent-component, $i = 1$, given on the right of the figure.

reward terms it remains to approximate the RTS state-action value functions. As the normalisation constants of these functions are known (note that $Z_{\tau\to H} = \sum_{\boldsymbol{z}} Q^{\text{rts}}_\tau(\boldsymbol{z}; \boldsymbol{w})$, $\forall \tau \in \mathbb{N}_H$) it is sufficient to approximate these functions up to a positive scaling. It is therefore sufficient to approximate the distributions

$$\hat{Q}^{\text{rts}}_\tau(\boldsymbol{z}; \boldsymbol{w}) = \frac{1}{Z_{\tau\to H}} Q^{\text{rts}}_\tau(\boldsymbol{z}; \boldsymbol{w}), \tag{3.23}$$

which can be approximated through numerous approximate inference techniques. These distributions are still complicated functions and to obtain an efficient inference routine we consider the following recursion, which is the analogue of the recursion (3.7), over these distributions

$$\hat{Q}^{\text{rts}}_\tau(\boldsymbol{z}; \boldsymbol{w}) = \frac{1}{Z_{\tau\to H}} \Big\{ Z_\tau q_{\tau,1}(\boldsymbol{z}; \boldsymbol{w}) + Z_{\tau+1\to H} q_{\tau,2}(\boldsymbol{z}; \boldsymbol{w}) \Big\}, \tag{3.24}$$

where the distributions $q_{\tau,1}(\boldsymbol{z}; \boldsymbol{w})$ and $q_{\tau,2}(\boldsymbol{z}; \boldsymbol{w})$ take the form

$$q_{\tau,1}(\boldsymbol{z}; \boldsymbol{w}) = \hat{p}(\boldsymbol{z}_\tau = \boldsymbol{z}|\tau; \boldsymbol{w}), \qquad q_{\tau,2}(\boldsymbol{z}; \boldsymbol{w}) = \sum_{\boldsymbol{z}'} p_{\tau\leftarrow\tau+1}(\boldsymbol{z}|\boldsymbol{z}'; \boldsymbol{w}) \hat{Q}^{\text{rts}}_{\tau+1}(\boldsymbol{z}'; \boldsymbol{w}).$$

The first point to note is that $q_{\tau,1}(\boldsymbol{z}; \boldsymbol{w})$ and $q_{\tau,2}(\boldsymbol{z}; \boldsymbol{w})$ are both distributions. The first term is a distribution by definition, while the second term is a distribution due to the direction of the transition dynamics in the term $p_{\tau\leftarrow\tau+1}(\boldsymbol{z}|\boldsymbol{z}'; \boldsymbol{w})$ and the fact that $\hat{Q}^{\text{rts}}_{\tau+1}(\boldsymbol{z}'; \boldsymbol{w})$ is a distribution. Written in this form it is clear that $\hat{Q}^{\text{rts}}_\tau(\boldsymbol{z}; \boldsymbol{w})$ is a two component mixture distribution, with the first component corresponding to the expected reward of the current time-point and the second component corresponding to the total expected reward from future time-points. Additionally, the weights of the two components are known and equal $Z_\tau Z^{-1}_{\tau\to H}$ and $Z_{\tau+1\to H} Z^{-1}_{\tau\to H}$ respectively, where it is easy to see that $Z_\tau + Z_{\tau+1\to H} = Z_{\tau\to H}$. Taking these points into account it is clear that (3.24) forms the basis for any of a number of approximate inference routines. We have yet to construct any such an inference routines explicitly, and this is a point

of future work, but even the most basic inference algorithms, such as belief propagation, are immediately applicable.

As a motivational example consider the case where the RTS state-action functions are approximated through distributions with a tractable graphical structure, where we denote the underlying graph by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Approximate inference techniques of this form are known as structural mean field methods, or variational Bayes approximations, see *e.g.* [143, 17]. The graphical structure is selected to maintain tractability of the inference algorithm, while allowing complex correlations to be modelled into the state-action function. Consider the case the underlying graphical structure is given by a tree where, denoting the approximate state-action value function by $\tilde{Q}^{\text{rts}}_{\tau}(\boldsymbol{z}; \boldsymbol{w})$, we have

$$\tilde{Q}^{\text{rts}}_{\tau}(\boldsymbol{z}; \boldsymbol{w}) = \prod_{s \in \mathcal{V}} \mu^{\tau}_{s}(z^s) \prod_{(s,t) \in \mathcal{E}} \frac{\mu^{\tau}_{st}(z^s, z^t)}{\mu^{\tau}_{s}(z^s)\mu^{\tau}_{t}(z^t)}, \tag{3.25}$$

and $\{\mu^{\tau}_{s}\}_{s \in \mathcal{V}}$ and $\{\mu^{\tau}_{st}\}_{(s,t) \in \mathcal{E}}$ are the approximate singleton and pairwise marginals. One possibility to model the RTS state-action value function in this manner would be to approximate $q_{\tau,1}(\boldsymbol{z}; \boldsymbol{w})$ and $q_{\tau,2}(\boldsymbol{z}; \boldsymbol{w})$ with the same graphical structure, *i.e.*

$$\tilde{q}_{\tau,i}(\boldsymbol{z}; \boldsymbol{w}) = \prod_{s \in \mathcal{V}} \mu^{\tau,i}_{s}(z^s) \prod_{(s,t) \in \mathcal{E}} \frac{\mu^{\tau,i}_{st}(z^s, z^t)}{\mu^{\tau,i}_{s}(z^s)\mu^{\tau,i}_{t}(z^t)}, \qquad i = 1, 2.$$

It then follows through (3.24) and matching of terms that the singleton and pairwise marginal terms of $\tilde{Q}^{\text{rts}}_{\tau}(\boldsymbol{z}; \boldsymbol{w})$ can be calculated as follows

$$\mu^{\tau}_{s}(z^s) = \frac{1}{Z_{\tau \to H}} \left(Z_{\tau} \mu^{\tau,1}_{s}(z^s) + Z_{\tau+1 \to H} \mu^{\tau,2}_{s}(z^s) \right),$$
$$\mu^{\tau}_{st}(z^s, z^t) = \frac{1}{Z_{\tau \to H}} \left(Z_{\tau} \mu^{\tau,1}_{s,t}(z^s, z^t) + Z_{\tau+1 \to H} \mu^{\tau,2}_{st}(z^s, z^t) \right),$$

where the consistency of the marginals $\{\mu^{\tau}_{s}, \mu^{\tau}_{st}\}_{s \in \mathcal{V}, (s,t) \in \mathcal{E}}$ follows from the consistency of the marginals, $\{\mu^{\tau,i}_{s}, \mu^{\tau,i}_{st}\}^{i=1,2}_{s \in \mathcal{V}, (s,t) \in \mathcal{E}}$. Due to the fact that the component weights of (3.24) are known this procedure corresponds to first approximating $\hat{Q}^{\text{rts}}_{\tau}(\boldsymbol{z}; \boldsymbol{w})$ through a two component mixture model, where the component weights of the approximation are assumed to be known and equal to the component weights of $\hat{Q}^{\text{rts}}_{\tau}(\boldsymbol{z}; \boldsymbol{w})$ and each component is assumed to be tree-structured. The approximation (3.25) is then obtained by marginalising over these two components under the assumption that the resulting marginal is again tree-structured.

We now consider other possible methods of performing inference in (3.1) and highlight the advantages of the RTS-inference paradigm. Firstly, consider the graphical structure of (3.1) and the *naive* direct application of approximate inference routines. As previously mentioned (3.1) has a mixture distribution over time-components and agent-components. In these high-dimensional transition-dependent factored MDPs each of these components will typically contain a large number of variables and inference in each component will be intractable. In such cases calculating the marginals of each component will be intractable and direct application of forward-backward methods will be infeasible. Instead, a

possibility is to directly apply any of a number of approximate inference algorithms to each of the components individually. However, such an approach will be highly inefficient and will not exploit any of the similarities between the structure of the components. Additionally, such a method will have a run-time that is quadratic in the planning horizon with no clear extension to an infinite planning horizon. An alternative approach for factored MDPs, suggested in [170, 171], is to use a variable elimination procedure. However, variable elimination has a run-time that is exponential in the induced tree-width of the graphical model, which makes such a procedure generally infeasible in these high-dimensional transition-dependent factored MDPs.

Another possibility is to apply approximate inference techniques to the standard state-action value function, which in terms of forward-backward algorithms corresponds to performing approximate inference on the summation of backward messages. In order to use approximate inference techniques one would consider the normalised version of the standard state-action value function,

$$\hat{Q}_\tau^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w}) = \frac{1}{Z_\tau^{\text{fb}}} Q_\tau^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w}), \tag{3.26}$$

where $Z_\tau^{\text{fb}} = \sum_{\boldsymbol{z}} Q_\tau^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w})$ is the normalisation constant. An important point to note is that, in contrast to the RTS state-action value function, the normalisation constants $\{Z_\tau^{\text{fb}}\}_{\tau \in \mathbb{N}_H}$ are not related to the component weights of the reward weighted trajectory distribution and nor are they obtainable from the forward messages. It is therefore also necessary to approximate these normalisation constants in this case and so an additional level of approximation is required using forward-backward inference in comparison to RTS-inference. This is a significant point as the terms that need to be estimated can be seen to be equal to the normalisation constant of a non-trivial high-dimensional distribution, which is an intractable problem in general. For example, in distributions from the exponential family the normalisation constant, or the $\log$-partition function, is a sufficient statistic of the distribution and is NP-hard to calculate in general. Obtaining bounds or approximations of the $\log$-partition function is a core aspect of much approximate inference research, see *e.g.* [178] for an overview, and the estimation of the normalisation constants $\{Z_\tau^{\text{fb}}\}_{\tau \in \mathbb{N}_H}$ is a significant challenge. Note that these issues persist when writing $\hat{Q}_\tau^{\text{fb}}(\boldsymbol{z}; \boldsymbol{w})$ in the form of a recursive equation, analogous to (3.3), which would require approximating the weights of the mixture components.

In summary there are two immediate advantages to RTS-inference over forward-backward inference in models of this form. Firstly, the fact that the normalisation constants of the RTS state-action value functions can be directly obtained from the forward messages means that it is only necessary to approximate these functions up to a positive scaling. This makes it sufficient to consider the distributions, $\{\hat{Q}_\tau^{\text{rts}}(\boldsymbol{z}; \boldsymbol{w})\}_{\tau \in \mathbb{N}_H}$, which can be approximated through any number of approximate inference techniques. The second advantage is that the direction of the transition dynamics in the RTS recursions mean that it is possible to obtain a recursive equation (3.24) that has the form of a two component mixture model, which allows for the efficient approximation of these distributions.

Summary

This completes our description of the RTS-inference method for model-based inference techniques to perform the integrals necessary for a parameter update. We have seen that this form inference technique has several important advantages over typical forward-backward inference methods. In terms of computational complexity the only difference between RTS-inference methods and forward-backward techniques is the need to calculate the system reversal dynamics. The advantages of RTS-inference come at this additional computational cost, but the construction of the system reversal dynamics is typically easily possible from the forward messages. The preference between the two forms of inference therefore depends on the additional computational cost of calculating the system reversal dynamics and the benefits of RTS-inference. We have considered some important models where these advantages of RTS-inference makes it attractive alternative.

3.3 Model-Free Evaluation Techniques

In section we give a brief review of some of the predominant sampling-based methods for estimating the gradient. No new theoretical work is provided in this section and instead we give an overview of these methods to provide a reference for future discussions. In model-free methods the statistics necessary to perform a parameter update are calculated in a stochastic, sampling-based manner. There are several reason why such an approach may be taken; for instance a model of the system dynamics may not be available and instead one has access only to samples of the system through either direct interaction or simulation. On the other-hand even when a model of the system dynamics is available it is often an intractable problem to calculate the expectations necessary to perform a parameter update. Model-free procedures attempt to sidestep these problems by providing sample-based estimates of these expectations. Sampling-based methods are often very simple to implement as well as requiring no knowledge of the system dynamics. Additionally, given enough sampling time model-free methods will converge, in probability, to the true value of the statistics. One of the issues with sampling-based methods is that they tend to have high variance in the estimands, which introduces a fresh set of problems into the optimisation procedure. For example, line searches lose much of their desirability because only noisy estimates of the objective and its derivative are available. Instead a predefined, or possibly adaptive, step-size sequence is typically used instead. Algorithms such as steepest gradient ascent are particularly sensitive to poor scaling of the search direction and in many problems defining an appropriate step-size sequence can in itself be a challenging task. While the use of a step-size sequence $\{\alpha_t\}_{t=1}^{\infty}$ satisfying the conditions

$$\alpha_t > 0, \qquad \sum_{t=1}^{\infty} \alpha_t = \infty, \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty,$$

is guaranteed to converge to a local maxima the rate of convergence can be prohibitively slow in practice.

In this section we shall highlight some of the main techniques used to perform model-free inference. We shall focus on forward-sampling techniques that obtain samples through direct forward simulation of the system dynamics. More complex sampling methods, such as Markov Chain Monte-Carlo techniques [79, 78] and nonparametric Bayesian regression techniques [65], have also been successfully applied.

Additionally we shall highlight two of the main techniques used to reduce the variance of the estimates, including *baseline* methods [68, 180] and actor-critic methods [99, 98, 162, 29, 28]. Other methods for reducing the variance of estimates also exist, such parameter-based exploration [149, 116], but we do not detail them here.

Forward-Sampling

In terms of sampling the simplest situation is where the planning horizon is finite and it is possible to directly simulate trajectories in the environment. It can be seen from the relation (2.6) that there is a simple generative method for obtaining a sample-based estimate of the gradient. For instance, one can simply sample a trajectory of length H from the trajectory distribution (2.2), while at each time-point obtaining a sample from the reward distribution conditional on the current point in the trajectory. This process can then repeated, say N times, and the results used to obtain an unbiased estimate of the gradient. Denote these sample trajectories by $\{\boldsymbol{z}_{1:H}^l\}_{l=1}^N$, while the samples of the reward distribution are denoted by $\{\boldsymbol{r}_{1:H}^l\}_{l=1}^N$. In the example of a finite horizon MDP we have that the model-free estimate of the gradient takes the simple form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) \approx \frac{1}{N} \sum_{n=1}^{N} \left\{ \sum_{\tau=1}^{H} \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}_\tau^n | \boldsymbol{s}_\tau^n; \boldsymbol{w}) \left\{ \sum_{t=\tau}^{H} r_t^n \right\} \right\}.$$

This completes the most basic application of sampling-based methods to perform inference in finite horizon problems. Inference for natural gradient ascent and Expectation Maximisation follows similarly.

Unsurprisingly the average reward and discounted infinite reward frameworks are necessarily more complex than the finite planning horizon. In this framework it is generally assumed that the estimates will be obtained from a single sample path of the system so that at some given time, T, the current sample will consist of $\boldsymbol{z}_{1:T}$ and $r_{1:T}$. A common technique is to make use of a recurrent state, denoted by $\boldsymbol{z}^*$, by which is meant a state with an almost sure finite hitting time. If we let t_n be the time-point of the n^{th} visit to the recurrent state, then the sample sequence $\boldsymbol{z}_{t_n:t_{n+1}}$ is known as the n^{th} regenerative cycle, and its length is given by

$$T_n = t_{n+1} - t_n.$$

Holding $\boldsymbol{w}$ fixed the random variables T_n are independent and identically distributed, with finite mean denoted by $T^{\boldsymbol{w}}$.

As we have seen the functions $\nabla_{\boldsymbol{w}} \log p(\cdot|\cdot\,; \boldsymbol{w})$ play a special role in gradient-based methods and it is useful to maintain a summary statistic of these functions evaluated over the course of each regenerative cycle. At the beginning of the n^{th} regenerative cycle the term $\boldsymbol{\Lambda}_1^n$ is initialised to zero and then through the course of this cycle the following recursive update is applied

$$\boldsymbol{\Lambda}_{t+1}^n = \begin{cases} \boldsymbol{\Lambda}_t^n + \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}_{t+1} | \boldsymbol{s}_{t+1}; \boldsymbol{w}), & \text{if } \boldsymbol{z}_{t+1} \neq \boldsymbol{z}^*, \\ 0, & \text{if } \boldsymbol{z}_{t+1} = \boldsymbol{z}^*. \end{cases}$$

The term $\boldsymbol{\Lambda}_t^n$ is generally referred to as the eligibility trace of the n^{th} cycle at time t, or just the eligibility

trace when the context is clear. Additionally it is helpful to rewrite the samples of the reward function in terms of the regenerative cycles, *i.e.* r_t^n is used to denote the sample reward received at the t^{th} time-point of the n^{th} regenerative cycle. If the samples of N regenerative cycles are to be used in the estimate then an unbiased estimate of (2.7) is given by

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} r_t^n \boldsymbol{\Lambda}_t^n.$$

While algorithms that rely on the use of a recurrent state are able to provide unbiased estimates of (2.7) they can also be problematic in a number of ways. For instance the variance of the estimate is dependent on the recurrence time, which will typically grow as the state space increases. Additionally, the recurrence time of a state is dependent on the control parameters and so is liable to change dramatically through the course of the optimisation process, which means that the selection of a good recurrent state may be a difficult problem in certain systems. Various techniques have been devised to overcome this reliance on the use of recurrent states, of which two prominent examples are truncating and discounting of the eligibility trace [137, 67]. While these methods generally introduce bias into the estimate, they also offer a certain amount of control on the variance. This reduced variance is usually at the cost of increased bias and so these methods have a natural bias-variance tradeoff property built into their mechanism.

Methods that truncate the eligibility trace, see *e.g.* [137, 67], have an additional integer parameter, n, and the eligibility trace is calculated using the previous n sample points. This means that up until the n^{th} time-point the eligibility trace is calculated in the normal recursive manner, *i.e.*

$$\boldsymbol{\Lambda}_t(n) = \sum_{\tau=1}^{t} \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}_\tau | \boldsymbol{s}_\tau; \boldsymbol{w}),$$

and then from the $(n+1)^{\text{th}}$ time-point onwards the truncated eligibility trace is calculated

$$\boldsymbol{\Lambda}_{t+1}(n) = \boldsymbol{\Lambda}_t(n) + \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}_{t+1} | \boldsymbol{s}_{t+1}; \boldsymbol{w}) - \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}_{t+1-n} | \boldsymbol{s}_{t+1-n}; \boldsymbol{w}).$$

An estimate of (2.7), which we denote by $\hat{\nabla}_{\boldsymbol{w}}^n U(\boldsymbol{w})$, is obtained after T time-points ($T \geq n$) through the truncated summation

$$\hat{\nabla}_{\boldsymbol{w}}^n U(\boldsymbol{w}) = \frac{1}{T-n+1} \sum_{t=n}^{\top} r_t \boldsymbol{\Lambda}_t(n).$$

Unless n exceeds the maximum recurrence time, which is infinite in ergodic Markov chains, $\hat{\nabla}_{\boldsymbol{w}}^n U(\boldsymbol{w})$ will be a biased estimate. The bias of $\hat{\nabla}_{\boldsymbol{w}}^n U(\boldsymbol{w})$ tends to zero as $n \to \infty$, however its variance will diverge in the limit. Hence there is a natural trade-off between the bias and variance of the estimate when selecting the truncation parameter. The parameter n has to be selected so as not to introduce too much bias, while avoiding excessive amounts of variance.

An alternative to truncating the eligibility trace is to instead consider a discounted recursive update, which was the subject of [19]. Introducing a bias-variance trade off parameter $\beta \in [0, 1)$, also known as a discount parameter, the discounted eligibility trace is updated according to the following recursive

formulae

$$\boldsymbol{\Lambda}_{t+1}(\beta) = \beta\boldsymbol{\Lambda}_t(\beta) + \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1}; \boldsymbol{w}).$$

It was shown in [19] that

$$\hat{\nabla}^{\beta}_{\boldsymbol{w}} U(\boldsymbol{w}) = \frac{1}{T}\sum_{t=1}^{\top} r_t \boldsymbol{\Lambda}_t(\beta),$$

gives a biased estimate of the gradient for $\beta \in [0, 1)$, while

$$\lim_{\beta\to 1}\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{\top} r_t\boldsymbol{\Lambda}_t(\beta) = \lim_{T\to\infty}\frac{1}{T}\sum_{\tau=1}^{\top}\sum_{t=\tau}^{\top}\mathbb{E}_{\tilde{p}(\boldsymbol{z},\tau,t;\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}}\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big].$$

As with the truncated estimate there is a natural bias-variance tradeoff in this discounted estimate. As $\beta \to 1$ the bias of $\hat{\nabla}^{\beta}_{\boldsymbol{w}} U(\boldsymbol{w})$ tends to zero while its variance increases, eventually diverging at the point $\beta = 1$. However, it was additionally shown in [19] that the accuracy of the biased gradient estimate also depends on the mixing time of the Markov chain induced by the current parameter setting. In particular they obtain a bound relating this mixing time to the normalised inner product of (2.7) and its biased estimate, see Theorem 3 of [19], so that it is not always necessary to have a value of β close to unity to obtain a good approximation. Similar estimates where also proposed in [91, 92, 113]. In [91, 92] the term r_t is replaced by $r_t - b$, where b is, known as a *baseline*, used to help reduce the variance of the estimate while not effecting the bias. The use of baselines as a variance reduction technique is a subject that we shall approach shortly. In [113] the term r_t is replaced by $r_t - \hat{U}(\boldsymbol{w})$, where $\hat{U}(\boldsymbol{w})$ is an estimate of the average reward, and they make use of an identifiable recurrent state to zero out the eligibility trace. An advantage of the discounting approach over the truncation method is that it is no longer necessary to store the last n sample points, which can be a big advantage if it is necessary to have a prohibitively large n to obtain an acceptably small amount of bias. The use of this estimate in steepest gradient ascent is usually referred to as the *GPOMDP* algorithm [19] and there has been a general preference in the literature to its use instead of the truncation method.

Baselines

While sampling-based methods are often easy to implement they generally suffer from high variance in the estimands. This is a well-known problem and various techniques have been devised to help reduce this variance, the simplest of which is perhaps the use of a *baseline*. In this case an additional term is introduced into the gradient equation in such a manner that it doesn't effect the bias of the estimate, while having an effect on the variance. It is then possible to select this additional term in such a way so as to either reduce or minimise the variance. The work [39] considered the effect baselines had on the variance of gradient estimates for binary immediate reward problems. Similarly the immediate reward problem was considered in [182], this time in the problem of connectionist networks, and found that the introduction of a various baseline didn't introduce bias into the estimand, although no basis was found for selecting this baseline. These results were extended to the average reward *GPOMDP* gradient estimator in [180, 68], where various optimality results were found for the use of a baseline. To observe the main point of the baseline method consider the case of a finite horizon Markov Decision Process. In

this instance the notion of an optimal baseline stems from the observation that for an arbitrary function $b : \mathcal{S} \to \mathbb{R}$ the gradient can be written in the equivalent form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \sum_{t=1}^{H} \mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\big[Q_t(\boldsymbol{z};\boldsymbol{w}) - b(\boldsymbol{s})\big]\Big],$$

so that the incorporation of this additional term does not introduce any bias into sample-based estimate of the gradient. This is simple to see because for each $t \in \mathbb{N}_H$ we have

$$\mathbb{E}_{p(\boldsymbol{z},t;\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) b(\boldsymbol{s})\Big] = \mathbb{E}_{p(\boldsymbol{s},t;\boldsymbol{w})}\Big[b(\boldsymbol{s})\nabla_{\boldsymbol{w}} \sum_{\boldsymbol{a}\in\mathcal{A}} p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big] = \mathbf{0}.$$

While the incorporation of this additional term, which is usually referred to as a *baseline*, doesn't effect the bias of any sample-based estimate it does effect the variance. It is therefore natural to find the optimal baseline, where optimality is measured in the terms of variance. There are various possibilities in the functional form of the baseline, such as constant baselines or baselines that depend on the state, and the explicit form of such baselines can be found in *e.g.* [68, 180, 124].

Actor-Critic Methods

In the sampling schemes considered thus far no use is made of the simulations from previous parameter updates, this information being essentially thrown away. This deficiency has led to the introduction of the so called *Actor-Critic* methods [99, 98, 162] which additionally maintain an estimate of the state-action value function through the use of *function approximation*. These methods are then formed of a two stage iterative process, alternating between updates of the control parameters, known as the *actor*, and updates of the approximate value function, known as the *critic*. While these methods attempt to increase the rate of convergence of gradient-based methods by reducing the variance of the estimands, they also have an important property in terms of the low complexity of the function approximation required by the critic. In particular, as shown independently in [98] and [162], the critic only needs to maintain a low-dimensional projection of the state-action value function, instead of the entire state-action value function. These methods have been further extended to employ Bayesian inference methods [64] and natural gradient actor-critic methods [123, 28, 29]. We shall postpone the discussion of these natural gradient actor-critic methods until natural gradient methods have been introduced in section(4.1).

Consider the discounted infinite horizon framework for Markov Decision Processes, where the extensions to the finite horizon and average reward frameworks are straightforward. As we've seen the vector-valued function $\nabla_{\boldsymbol{w}} \log p(\cdot|\cdot;\boldsymbol{w}) : \mathcal{A}\times\mathcal{S} \to \mathbb{R}^{n_{\boldsymbol{w}}}$ is of prime importance in gradient-based methods. Following standard notation in the literature we make the identification $\boldsymbol{\psi}^{\boldsymbol{w}} \equiv \nabla_{\boldsymbol{w}} \log p(\cdot|\cdot\,;\boldsymbol{w})$, so that $\psi_i^{\boldsymbol{w}}(\boldsymbol{a},\boldsymbol{s}) = \frac{d}{dw_i} \log p(\boldsymbol{a}|\boldsymbol{s}\,;\boldsymbol{w})$. Consider the vector space of functions defined on $\mathcal{A}\times\mathcal{S}$ and denote the subspace spanned by the functions $\{\psi_i^{\boldsymbol{w}}\}_{i=1}^{n_{\boldsymbol{w}}}$ by $\boldsymbol{\Psi}^{\boldsymbol{w}}$. For any instantiation of the parameter vector define the inner product $\langle\cdot,\cdot\rangle_{\boldsymbol{w}}$ of any two real valued functions f_1, f_2 on $\mathcal{A}\times\mathcal{S}$ to be given by

$$\big\langle f_1, f_2\big\rangle_{\boldsymbol{w}} = \sum_{t=1}^{\infty} \mathbb{E}_{p_\gamma(\boldsymbol{z},t;\boldsymbol{w})}\Big[f_1(\boldsymbol{a},\boldsymbol{s}) f_2(\boldsymbol{a},\boldsymbol{s})\Big].$$

Under this notation the gradient for the infinite horizon discounted rewards framework can be written in the form

$$\frac{d}{dw_i}U(\boldsymbol{w}) = \left\langle Q^{\boldsymbol{w}}, \psi_i^{\boldsymbol{w}} \right\rangle_{\boldsymbol{w}}, \tag{3.27}$$

where for notational convenience we use $Q^{\boldsymbol{w}}$ to denote the state-action value function. Thus the dependence on $Q^{\boldsymbol{w}}$ is only through its projection into the subspace $\boldsymbol{\Psi}^{\boldsymbol{w}}$ and instead of attempting to learn the state-action value function it is instead sufficient to learn its projection in $\boldsymbol{\Psi}^{\boldsymbol{w}}$. This can be seen by defining the projection operator $\Pi_{\boldsymbol{w}} : \mathbb{R}^{|\mathcal{A}|\times|\mathcal{S}|} \to \boldsymbol{\Psi}^{\boldsymbol{w}}$ as follows

$$\Pi_{\boldsymbol{w}}(f) = \operatorname*{argmin}_{\hat{f}\in\boldsymbol{\Psi}^{\boldsymbol{w}}} \|f - \hat{f}\|,$$

then it is simple to show that, see *e.g.* [162],

$$\left\langle Q^{\boldsymbol{w}}, \psi_i^{\boldsymbol{w}} \right\rangle_{\boldsymbol{w}} = \left\langle \Pi_{\boldsymbol{w}}(Q^{\boldsymbol{w}}), \psi_i^{\boldsymbol{w}} \right\rangle_{\boldsymbol{w}}.$$

It is clear from (3.27) that it suffices to use the projection in $\Pi_{\boldsymbol{w}}(Q^{\boldsymbol{w}})$ in place of $Q^{\boldsymbol{w}}$ in (2.4), where the projection $\Pi_{\boldsymbol{w}}(Q^{\boldsymbol{w}})$ is often known a *compatible function* approximation [162]. As $\Pi_{\boldsymbol{w}}(Q^{\boldsymbol{w}}) \in \boldsymbol{\Psi}^{\boldsymbol{w}}$ there exists $\boldsymbol{\theta} \in \mathbb{R}^{n_{\boldsymbol{w}}}$ *s.t.* $\Pi_{\boldsymbol{w}}(Q^{\boldsymbol{w}}) \equiv \left(\boldsymbol{\psi}^{\boldsymbol{w}}\right)^\top \boldsymbol{\theta}$ and gradient can be rewritten in the form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})}\left[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\right]\boldsymbol{\theta} = G_{\boldsymbol{w}}\boldsymbol{\theta},$$

where $G_{\boldsymbol{w}}$ is defined in the obvious manner. The final component in an actor-critic algorithm is to define the update procedure for the critic parameters, for which a popular choice is some form of *temporal-difference* learning [160]. The use of temporal-difference learning algorithms is motivated by various results in the literature relating these algorithms to different forms of projections of the (state-action) value function, see *e.g.* [173, 145]. We do not detail an particular instances of actor-critic algorithms here but exact forms of the critic update can be found in *e.g.* [29]. This completes our brief description of actor-critic methods.

3.4 Experiments

In this section we perform various experiments to highlight some of the theoretical work that has been presented in this chapter. In particular, we present some experiments relating to the RTS-inference routine when applied to linear systems, with a possibly non-linear reward structure. Even though it is clear that RTS-inference is more efficient than forward-backward inference in the this model, having a linear instead of quadratic run-time, it is perhaps instructive to illustrate the amount of computational savings that can be made in practice. For this reason we detail a finite horizon experiment where a comparison is made between RTS-inference and forward-backward inference. Additionally, we detail an experiment for the infinite horizon recursion on this model. In the experiments we used linearised versions of the Lotka-Volterra system and N-link rigid manipulator, which we now describe in detail.

Lotka-Volterra System

The Lotka-Volterra equations [177] are a standard method to describe the population dynamics of a group of n_s interacting species of animal. The uncontrolled version of the equations take the form

$$\dot{\boldsymbol{s}} = D(\boldsymbol{s})\big(A\boldsymbol{s} + \boldsymbol{c}\big) + \boldsymbol{\eta},$$

where the $\boldsymbol{s} \in \mathbb{R}_+^{n_s}$, $\boldsymbol{c} \in \mathbb{R}^{n_s}$, $A \in \mathbb{R}^{n_s \times n_s}$, $D(\boldsymbol{s})$ is a diagonal matrix with $\boldsymbol{s}$ running along the diagonal and $\boldsymbol{\eta}$ is a zero mean Gaussian. The vector $\boldsymbol{c}$ describes the growth/death rates of the animals in the absence of interaction of other species. The term $D(\boldsymbol{s})A\boldsymbol{s}$ describes a quadratic interaction between the species, where A is ordinarily assumed to skew-symmetric. The assumption of skew-symmetry of A stems from the observation that if species i is a predator of species j, then species i will benefit from the predation of species j through the quadratic form $a_{ij}x_ix_j$, while the prey is being consumed at the rate $-a_{ij}x_ix_j$. A possible controlled version [70] of these equations is

$$\dot{\boldsymbol{s}} = D(\boldsymbol{s})\big(A\boldsymbol{s} + \boldsymbol{c} + \boldsymbol{f}(\boldsymbol{a})\big) + \boldsymbol{\eta},$$

where $\boldsymbol{a} \in \mathbb{R}^{n_a}$ is the control vector and $\boldsymbol{f} : \mathbb{R}^{n_a} \to \mathbb{R}^{n_s}$ is a vector-valued function of the control. The control can be seen as a deliberate intervention with the environment (on the part of the controller) in order to modulate the populations of the species to some pre-defined levels. Examples include the culling of specific animals, which directly affects only a subset of the species, or the adjustment of a global parameter (such as the temperature of a controlled environment) which affects all of the species.

We shall consider an environment with six species [70] where there are two super-predators (x_1 and x_2), two prey (x_3 and x_6) and two intermediate species (x_4 and x_5). In this example the coefficients of the uncontrolled Lotka-Volterra equations are given by

$$A = \begin{bmatrix} 0 & 0 & 4 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 0 \\ -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & -2 & 0 & 0 & 1 & 4 \\ 0 & -3 & 0 & -1 & 0 & 5 \\ 0 & 0 & 0 & -4 & -5 & 0 \end{bmatrix}, \qquad \boldsymbol{c} = \begin{bmatrix} -13 \\ -9 \\ 4 \\ -10 \\ -11 \\ 17 \end{bmatrix}.$$

In [70] the control is a scalar variable u and $\boldsymbol{f}_i(u) = -(\delta_{i,1} + \delta_{i,2})u$, *i.e.* the controller only has direct control of the two super-predator species. In this example however we will consider a control scenario where $\boldsymbol{a} \in \mathbb{R}^6$ and $\boldsymbol{f}_i(\boldsymbol{a}) = u_i$, which corresponds to having direct control of all six species individually. We consider such a controller for the sake of simplicity as it easily allows for the linearisation of the system by defining the controller $\boldsymbol{a}$ *s.t.*

$$a_i = \frac{\tau_i}{s_i} - A_i\boldsymbol{s} - c_i,$$

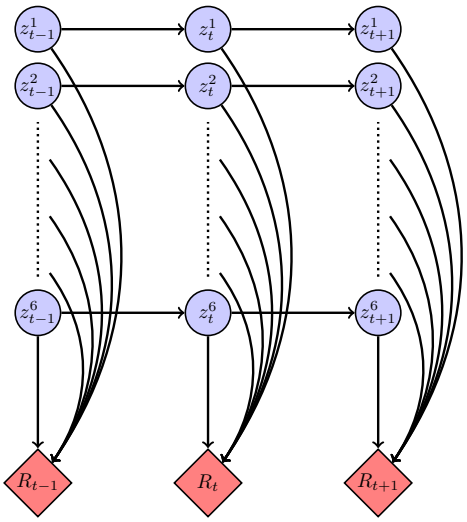


Figure 3.6: A dynamic Bayesian network representation of the transition and reward structure of the Lotka-Volterra system after the system has been linearised through feedback-linearisation.

where A_i denotes the i^{th} row of the matrix A. This results in the linear system dynamics

$$\dot{\boldsymbol{s}} = \boldsymbol{\tau} + \boldsymbol{\eta},$$

where $\boldsymbol{\tau}$ is now the control variable. The transition dynamics and reward structure of the linearised system is given in fig(3.6). It can be seen that the population of each species can be modulated independently while the species are coupled in the objective function through the reward function. In this system we have $n_s = 6$ and $n_a = 6$ and due to the structure of the controller the matrix K is diagonal and there are 13 policy parameters.

In the experiments we set $\Delta_t = 0.1$'s. In the finite horizon experiments we considered a planning horizon of $H = 100$, while in discounted problems we considered a discount factor of $\gamma = 0.95$. The mean of the initial population level for each species was randomly selected from the interval $[0.1, 10]$. The diagonals of K were initialised randomly from the interval $[-2, 0]$, which ensures the spectrum of $F(K)$ lies within in the unit circle for the initial policy parameterisation. The parameters m and π_σ were initialised randomly from the intervals $[-1, 1]$ and $[10, 20]$ respectively. A single reward function was used with $\boldsymbol{y} = (1, 2, 1, 3, 1, 4)^\top$, which is an equilibrium point of the uncontrolled Lotka-Volterra equations for this system [70]. The matrix M was set so that reward depended only upon populations of the species and was independent of the control variable. To construct the covariance matrices of the initial state distribution, transition dynamics and the reward function we used the representation $U^\top DU$, where U is an orthogonal matrix and D is diagonal. We generated the orthogonal matrices by orthonormalising a randomly generated full rank matrix, where the elements of the matrix were sampled from the interval $[0, 1]$. The elements of the diagonal matrices were sampled from the intervals $[0, 0.05]$, $[0, 0.05]$ and $[0, 0.1]$ for the three covariance matrices respectively. The uniform distribution was used for the initialisation of all random parameters.

N-link Rigid Manipulator

The N-link rigid robot arm manipulator is a standard continuous model, consisting of an end effector connected to an N-linked rigid body [90]. A graphical depiction of a 3-link rigid manipulator is given in fig(3.7). A typical continuous control problem for such systems is to apply appropriate torque forces to the joints of the manipulator so as to move the end effector into a desired position. The state of the system is given by $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}} \in \mathbb{R}^N$, where $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ denote the angles, velocities and accelerations of the joints respectively, while the control variables are the torques applied to the joints $\boldsymbol{\tau} \in \mathbb{R}^N$. The nonlinear state equations of the system are given by, see *e.g.* [155],

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\dot{\boldsymbol{q}}, \boldsymbol{q})\dot{\boldsymbol{q}} + g(\boldsymbol{q}) = \boldsymbol{\tau} \tag{3.28}$$

where $M(\boldsymbol{q})$ is the inertia matrix, $C(\dot{\boldsymbol{q}}, \boldsymbol{q})$ denotes the Coriolis and centripetal forces and $g(\boldsymbol{q})$ is the gravitational force. While this system is highly nonlinear it is possible to define an appropriate control function $\hat{\boldsymbol{\tau}}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ that results in linear dynamics in a different state-action space. This process is called *feedback linearisation*, see *e.g.* [90], and in the case of an N-link rigid manipulator recasts the torque action space into the acceleration action space. This means that the state of the system is now given by $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, while the control is $\boldsymbol{a} = \ddot{\boldsymbol{q}}$. Ordinarily in such problems the reward would be a function of the generalised co-ordinates of the end effector, which results in a non-trivial reward function in terms of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$. While this reward function can be modeled as a mixture of Gaussians, see [77], for simplicity we consider the simpler problem where the reward is a function of $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ directly.

In the experiments we considered a 3-link rigid manipulator, which results in a 9-dimensional state-action space and a 22-dimensional policy. In the experiment we discretised the continuous time dynamics into time-steps of $\Delta_t = 0.1$ and considered a finite planning horizon of $H = 100$. The mean of the initial state distribution was set zero. The elements of the policy parameters m and π_σ were initialised randomly from the interval $[-2, 2]$ and $[1, 2]$ respectively. The matrix K was initialised to be zero on inter-link entries, while intra-link entries were initialised using rejection sampling. We sampled the parameters for each link independently from the set $[-400, 40] \times [-50, 10]$ and rejected the sample if the corresponding link was unstable. In the reward function the desired angle of each joint was randomly sampled from the interval $[\pi/4, 3\pi/4]$. The covariance matrices of the initial state distribution and state transition dynamics were set to diagonals, where the diagonal elements were initialised randomly from the interval $[0, 0.05]$. The covariance matrix of the reward function was set to be a diagonal with all entries equal to 0.1.

RTS-Inference Vs. Forward-Backward Inference - Finite Horizon

In this experiment we compared the efficiency of RTS-inference [59] and forward-inference [77] in linear systems with a finite horizon. We performed the experiment on both the Lotka-Volterra system and the 3-link rigid manipulator and repeated the experiment 100 for each system. Each training algorithm

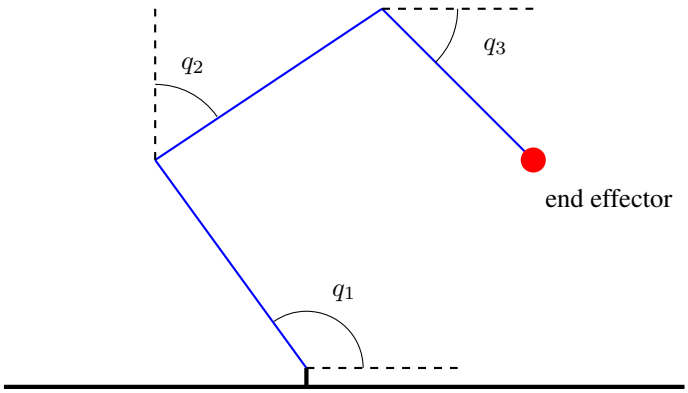


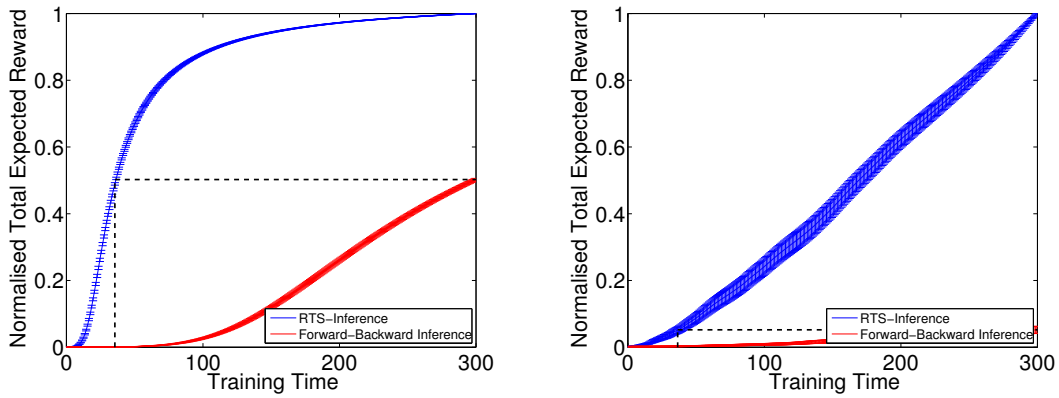
Figure 3.7: A graphical depiction of a 3-link robot manipulator arm, where the angles of the joints are given by q_1, q_2 and q_3 respectively.

was given 300 seconds of training time in both of the experiments. The results are shown in fig(3.8) where the normalised total expected reward is plotted against the training time (in seconds). The plot shows the results for the RTS-inference algorithm (blue) and the forward-backward inference algorithm (red). The dashed lines indicates the performance of forward-backward inference after the total allotted training compared to the RTS-inference algorithm. It can be seen that in both systems the RTS-inference algorithms needs only around $35 - 45$ seconds to obtain the same level of performance as the forward-backward algorithm with 300 seconds of training. Additionally, it can be seen that in comparison to the RTS-inference algorithm the forward-backward algorithm only obtains around 50% and 5% in the Lotka-Volterra and 3-link rigid manipulator systems respectively. Thus, even in these comparatively small experiments the RTS-inference algorithm significantly outperforms the forward-backward algorithm and this difference in performance can be expected to be even more marked in larger-scale problems with longer planning horizons.

RTS-Inference - Infinite Horizon

We also performed the discounted infinite horizon RTS-inference procedure on the Lotka-Volterra system. As the forward-backward algorithm of [77] is applicable only to finite horizons and there is no other model-based procedure that is able to perform exact inference in this framework we made a comparison with the finite horizon RTS-inference procedure, where we used a predetermined planning horizon. We selected a finite horizon of $H = 100$, which was chosen to be sufficiently long so as to consistently obtain reasonable performance in the problems considered.

We performed the experiment 100 times and both algorithms were given 60 seconds training time in each experiment. The results are shown in fig(3.9) where we have plotted the mean and standard error of the infinite recursion (blue) and the finite horizon heuristic (red). It is clear to see that the infinite horizon procedure is consistently able to outperform the finite horizon heuristic. This difference in performance is explained by the number of training iterations the two algorithms were able to perform in the training time allotted. The finite horizon algorithm was able to perform 1039.8 ± 0.7 training



(a) Lotka-Volterra Equations (b) 3-Link Rigid Manipulator

Figure 3.8: Normalised total expected reward plotted against training time (in seconds) for the Lotka-Volterra equations (a) and the controlled pendulum (b). The plot shows the results for RTS-inference (blue) and forward-backward inference [77] (red).

iterations, while the infinite horizon algorithm was able to perform 7986.6 ± 221.8 training iterations. If the two algorithms were plotted in terms of training iterations one would expect the difference to be less marked[6]. The reason the infinite horizon recursion was able to perform more training iterations was due to its run-time properties, which were discussed in detail in section(3.2). In the problem we considered the largest eigenvalue of $F(K)$ was usually sufficiently far away from the boundary of the unit circle which ensured rapid convergence of the infinite horizon recursion. The convergence properties of the current formulation of the infinite horizon recursion has the obvious disadvantage that one could easily construct a control problem where inference would take arbitrarily long by ensuring that the optimal K is arbitrarily close to the boundary of the unit circle. Indeed, we also tried this experiment on the 3-link manipulator but found that with the reward structures that were used the algorithm often tended to the edge of the unit circle and as a result performed poorly. This is obviously an undesirable point of our infinite horizon recursion and it would be desirable to obtain an alternative formulation whose convergence properties don't rely on the stationarity of the occupancy distribution. It would then be a problem dependent issue as to which formulation would be most appropriate.

These issues with the convergence of the occupancy distribution are a general modelling issue with these systems. If it is of no concern that certain dimensions of the state-action distribution diverge then our algorithm is obviously limited by being constrained to the unit circle. On the other-hand if this is a constraint of the system then our formulation is completely general, even though it could have poor convergence in certain cases. In such cases this optimisation problem should be properly treated as a constrained problem, which is an issue that we have not approached in this work.

3.5 Discussion

In this chapter we have presented a new range of model-based inference algorithms for parametric policy search methods, such as steepest gradient ascent and Expectation Maximisation. Typical model-based in-

[6]We have done this but the plot is not shown.

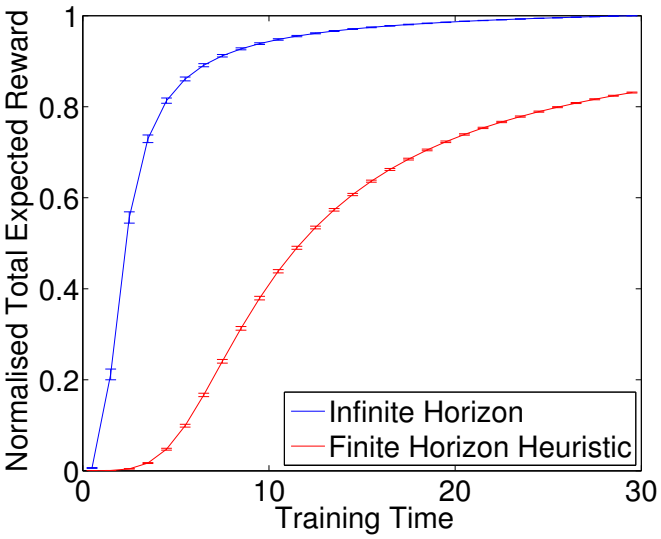


Figure 3.9: Normalised total expected reward plotted against training time (in seconds) for the Lotka-Volterra equations (a) and the controlled pendulum (b). The plot shows the results for the discounted infinite horizon RTS-inference algorithm (blue) and a heuristic of using a prespecified finite planning horizon (red).

ference algorithms in this area use forward-backward inference routines [170, 171, 77, 102, 104], which are analogous to similar forward-backward routines in time-series models. In contrast our approach is analogous to RTS-smoothing [133]. This extension is non-trivial due to the mixture structure of the reward weighted trajectory distribution, as well as the infinite number of components in the mixture distribution when the planning horizon is infinite. An important difference in these two techniques is the contrasting form of the backward recursions (3.3) and (3.7 & 3.12). In particular the different directions of the transition dynamics in the two recursions. This difference is important and it allows for efficient model-based inference in models where inference would be more inefficient through forward-backward techniques. For example, in continuous system it allows for the construction of recursions over the moments of the RTS state-action value functions, as opposed the functions themselves. An example that we have considered in detail are linear systems with a possibly non-linear reward structure. In these models it is only possible to construct a forward-backward inference algorithm for finite planning horizons, in which case it has a run-time that is quadratic in the planning horizon [77]. In contrast, by using RTS-inference it is possible to perform inference in finite horizon problems in linear time, while it is also possible to extend the algorithm to infinite planning horizons with discounted rewards. This last problem is non-trivial and involves calculating the moments of an infinite number of mixture components. Furthermore, we shall see chapter(4) that we will be able to extend the RTS framework to calculate the Hessian in linear time, while the forward-backward algorithm has a cubic complexity.

An additional important property of RTS-inference methods is that the normalisation constant of RTS state-action value functions are directly related to the component weights of the reward weighted trajectory distribution, which are themselves typically easily obtainable from the forward messages. This property means that it is only necessary to approximate the RTS state-action value functions up to

a positive scaling. It is therefore sufficient to approximate the distributions $\{\hat{Q}_\tau^{\text{rts}}(\boldsymbol{z};\boldsymbol{w})\}_{\tau\in\mathbb{N}_H}$, which can be done through numerous approximate inference techniques. To highlight the applicability of such techniques we have briefly considered their application to high-dimensional multi-agent systems where inference in the reward weighted trajectory distribution is intractable. At present we have only detailed the advantages of performing inference on these models using RTS-inference. The exact construction and study of various approximate inference algorithms for these models is an area of future research.

Chapter 4

Parametric Policy Search Methods : Search Direction Analysis

We have seen in chapter(2) that there are various parametric optimisation methods, either gradient-based or bound-based, that can be applied to the MDP framework. Due to the inherent difficulties of complex real-world planning problems, however, these methods typically have to be applied in an approximate, usually stochastic, manner. In most cases of interest, therefore, we are considering a stochastic non-concave optimisation problem and as such are restricted to stochastic versions of these optimisation methods. Almost exclusively in the MDP literature the methods considered are steepest gradient ascent, natural gradient ascent or Expectation Maximisation, with the latter two being the current methods of choice. The Newton method has also been previously considered [19, 120], but it suffers from numerous problems, which shall be discussed in detail in section(4.2), that make its application undesirable. While both natural gradient ascent and Expectation Maximisation have been successfully applied to numerous challenging planning problems, such as Tetris [83] and real-world robotics domains [94], there is currently little understanding in the relationship between the two algorithms. Furthermore, little work has been done on alternative gradient-based algorithms that are applicable to the MDP framework, and in particular that can be applied in a stochastic manner.

We make two novel contributions in this chapter. The first contribution is to provide a novel analysis of the step directions of natural gradient ascent and Expectation Maximisation, where in particular we are able to show that the two algorithms are closely related to a particular form of approximate Newton method. Motivated by this analysis we make our second contribution of the chapter, the consideration of the direct application of this approximate Newton method to the MDP objective, which is analogous to the Gauss-Newton method for nonlinear least squares. During our considerations we shall see that this method has many desirable properties that are absent in the naive application of the Newton method. These properties include guarantees that the approximate Hessian is negative-semidefinite when the policy is $\log$-concave in the control parameters; sparsity properties in the approximate Hessian, absent in the Hessian, that make the matrix inversion more efficient; that the search direction can be evaluated using the same techniques as described in chapter(3), which is not true of the Newton method. The approximate Newton method is also shown to have the desirable property of being invariant to non-

singular linear transformations of the parameter space.

4.1 Search Direction Analysis

In this section we will perform our novel comparison of natural gradient ascent and Expectation Maximisation, where particular focus shall be on the search-direction of two algorithms. In gradient-based algorithms of Markov Decision Processes the update of the policy parameters take the form

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w} + \alpha \mathcal{M}(\boldsymbol{w}) \nabla_{\boldsymbol{w}} U(\boldsymbol{w}), \tag{4.1}$$

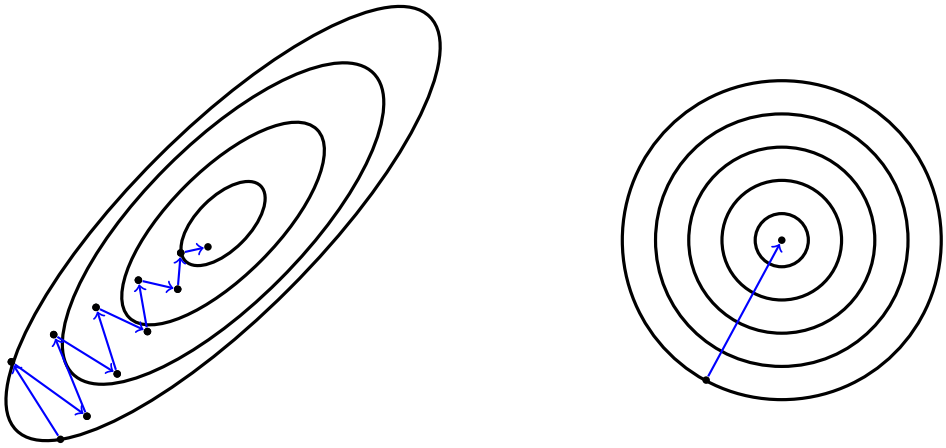
where $\alpha \in \mathbb{R}^+$ is the step-size parameter and $\mathcal{M}(\boldsymbol{w})$ is some preconditioning matrix that possibly depends on $\boldsymbol{w}$. An immediate issue concerning updates of the form (4.1) is in the selection of the 'optimal' choice of the matrix $\mathcal{M}(\boldsymbol{w})$, which clearly depends on the sense in which optimality is defined. There are numerous reasonable properties that are desirable of such an update, including the numerical stability and computational complexity of the parameter update, as well as the rate of convergence of the overall algorithm resulting from these updates. We now discuss these various properties in more detail, before then proceeding to our novel analysis. While the parameter update of Expectation Maximisation does not have the form (4.1) we shall see in section(4.1.2) that its parameter update is closely related to such an update.

One of the most important property of an update of the form (4.1) is that it actually increases the value of the objective function. Parameter updates of the form (4.1) are guaranteed to increase the objective provided that the search direction is an ascent direction of the objective function. It is well-known that when $\mathcal{M}(\boldsymbol{w})$ is positive-definite the corresponding search direction will be an ascent direction. Indeed performing a Taylor expansion of $U(\boldsymbol{w}_k + \boldsymbol{p})$ *w.r.t.* $\boldsymbol{p}$ gives, up to first order, the following inequality

$$U(\boldsymbol{w}_{k+1}) = U(\boldsymbol{w}_k) + \alpha_k \nabla_{\boldsymbol{w}}^{\top} U(\boldsymbol{w}_k) \mathcal{M}(\boldsymbol{w}_k) \nabla_{\boldsymbol{w}} U(\boldsymbol{w}_k) \geq U(\boldsymbol{w}_k),$$

where this inequality follows immediately from the fact that $\mathcal{M}(\boldsymbol{w}_k)$ is positive-definite. In steepest gradient ascent we have $\mathcal{M}(\boldsymbol{w}) = I$, which is trivially positive-definite, so that steepest gradient ascent always gives an ascent direction. In natural gradient ascent we have $\mathcal{M}(\boldsymbol{w}) = G^{-1}(\boldsymbol{w})$, where $G(\boldsymbol{w})$ is the Fisher information matrix. The Fisher information matrix is positive-semidefinite over the entire parameter space, which can be seen because it is the non-negative mixture of positive-semidefinite matrices. Natural gradient ascent is therefore guaranteed to provide an ascent direction, or at least a non-descent direction, over the entire parameter space. By contrast, the Newton method is not guaranteed to provide an ascent direction in non-concave problems. This is because the Hessian will not be positive-definite over the entire parameter space when the objective is non-concave.

Another primary criteria for comparing optimisation algorithms is the rate of convergence, *i.e.* the rate (in terms of number of iterations) that the solution converges to a local optima. A brief introduction of the technical definitions for various rate of convergence are given in appendix(A). Generally speaking



(a) An Ill-Scaled Quadratic Function (b) An Well-Scaled Quadratic Function

Figure 4.1: An example of (a) ill-scaled quadratic function and (b) well-scaled quadratic function. For the ill-scaled function steepest gradient ascent is showing typical *zig-zagging* behaviour, also known as *plateauing* behaviour, where the search direction can be highly skewed by the poor scaling of the quadratic component of the objective. Conversely, the well-scaled quadratic function has the identity matrix as its quadratic component, which leads to the search direction always correctly pointing to the optimum regardless of the initial point.

it is desirable to construct optimisation algorithms that have a quadratic rate of convergence in the vicinity of a local optima, which is generally rapid and only requires a few iterations. Started from a point that is sufficiently close to a local optimum the Newton method has a quadratic rate of convergence. The rate of convergence of steepest gradient ascent is linear, where the rate of convergence is determined by the condition number of the Hessian around a local optimum. See *e.g.* [121] for more details. I am unaware of technical results for the rate of convergence of natural gradient ascent, but it is generally believed to increase performance over steepest gradient ascent. This is has been shown to be the case empirically, both in the MDP framework and other areas of research. Another property that is closely related is computational complexity, *i.e.* the computational cost of performing a single parameter update. When considering the overall speed of an algorithm it is necessary that both the rate of convergence and the computational complexity are taken into account.

An important aspect of any optimisation problem is the issue of scaling. Intuitively a problem is poorly-scaled if changes to the parameters in one direction produce much larger variations in the objective than changes in another direction. A typical example of a poorly-scaled objective is a weighted quadratic function

$$f(\boldsymbol{w}) = \alpha w_1^2 + w_2^2,$$

where $\alpha \gg 1$. The function f is far more sensitive to small changes in w_1 than it is to small changes in w_2. This can be observed visually by observing the behaviour of the contours of a quadratic function as it becomes more poorly-scaled. As the function is quadratic the contours will be ellipses and as it becomes more poorly-scaled the contours will become more elongated. This pronounced elongation can result in some methods *zig-zagging*, also referred to as *plateauing*, in the parameter space and hence having poor

convergence. An example is given in fig(4.1) where an illustration of such *zig-zagging* behaviour can be observed. The robustness of optimisation algorithms to poor scaling can be of significant practical importance, especially in problems where only an approximation of the gradient is possible and line search procedures become sensitive to errors in the function evaluations. Often in these approximate, or stochastic, gradient-based algorithms a step-size sequence is defined prior to training in an off-line manner and in poorly-scaled problems it can be extremely difficult to gauge an appropriate scale for these steps lengths. This is not only an issue of slow convergence but also of overshooting in the parameter space, due to an overly large step in the parameter space, which can easily occur in poorly-scaled problems. Steepest gradient ascent is notoriously sensitive to poor-scaling. This can be seen by the fact the condition number of the Hessian increases as the objective becomes more poorly-scaled, where, as previously mentioned, the rate of convergence of steepest gradient ascent is given by the condition number of the Hessian. Again, more details can be found in [121]. By contrast, the Newton method is scale-invariant [121]. It is well-known that poor-scaling of the objective often occurs in the modeling and optimisation of physical and chemical systems. It is therefore of no surprise that poor scaling of the gradient often occurs in Markov Decision Processes, as well as similar planning models. This is a well known problem in the reinforcement learning and planning communities and various attempts have been made to resolve it, most notably natural gradient ascent and Expectation Maximisation.

Finally, the numerical stability of the algorithm also needs to be taken into consideration. There are two possible sources of instability that need to be considered: i) The stability of the parameter update, which includes issues such as inverting a matrix that is nearly singular; ii) The numerical qualities of the inference routine required for a parameter update, which can include issues such as the variance of the estimand in model-free inference routines.

While all reasonable criteria the rate of convergence is of such importance in an optimisation algorithm that it is a logical starting point in our analysis. For this reason we concern ourselves with relating natural gradient ascent and Expectation Maximisation to the Newton method, which has the highly desirable property of having a quadratic rate of convergence in the vicinity of a local optimum. The Newton method is well-known to suffer from problems that make it either infeasible or unattractive in practice, but in terms of forming a basis for theoretical comparisons it is a logical starting point. We shall discuss some of the issues with the Newton method in more detail in section(4.2) when considering our approximate Newton method. In the Newton method the matrix $\mathcal{M}(\boldsymbol{w})$ is set to the negative inverse Hessian, *i.e.*

$$\mathcal{M}(\boldsymbol{w}) = -\mathcal{H}^{-1}(\boldsymbol{w}), \quad \text{where } \mathcal{H}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^{\top} U(\boldsymbol{w}).$$

Using methods similar to those used to calculate the gradient it can be shown that the Hessian takes the form

$$\mathcal{H}(\boldsymbol{w}) = \mathcal{H}_1(\boldsymbol{w}) + \mathcal{H}_2(\boldsymbol{w}), \tag{4.2}$$

where

$$\mathcal{H}_1(\boldsymbol{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p(\boldsymbol{z}_{1:t};\boldsymbol{w})}\Big[\gamma^{t-1} R(\boldsymbol{z}_t) \nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^{\top} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big], \tag{4.3}$$

$$\mathcal{H}_2(\boldsymbol{w}) = \sum_{t=1}^{\infty} \mathbb{E}_{p(\boldsymbol{z}_{1:t};\boldsymbol{w})}\Big[\gamma^{t-1} R(\boldsymbol{z}_t) \nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^{\top} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big]. \tag{4.4}$$

Similar equations can be obtained for the Hessian of the finite horizon and infinite horizon average reward frameworks. Having discussed some of the main desirable properties of parameter updates of the form (4.1), we now provide provide our novel analysis of the natural gradient ascent and Expectation Maximisation.

4.1.1 Natural Gradient Ascent

In section(2.3) we introduced natural gradient ascent when applied to Markov Decision Processes, where this algorithm can be seen as performing steepest gradient ascent on the parameter manifold defined through the trajectory distribution. In this section we give our novel analysis of natural gradient ascent. In order to do so we use the form of the Fisher information matrix given by

$$G(\boldsymbol{w}) = -\mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^{\top} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big]. \tag{4.5}$$

Similarly, it is helpful to rewrite the matrix (4.4) into the following form

$$\mathcal{H}_2(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^{\top} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big], \tag{4.6}$$

where this form of $\mathcal{H}_2(\boldsymbol{w})$ is obtained through similar manipulations to those used in section(2.2). Comparing the Fisher information matrix (4.5) with the matrix (4.6) it is clear that natural gradient ascent has a relationship with the approximate Newton method that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$, which in terms of (4.1) corresponds to setting $\mathcal{M}(\boldsymbol{w}) = -\mathcal{H}_2^{-1}(\boldsymbol{w})$. In particular it can be seen that the difference between the two methods lies in the non-negative *w.r.t.* which the expectation is taken in (4.5) and (4.6). In the Fisher information matrix the expectation is taken *w.r.t.* to the state-action occupancy marginals of the trajectory distribution, while in $\mathcal{H}_2(\boldsymbol{w})$ the expectation is taken *w.r.t.* the non-negative function, $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$. It also appears that there is a difference in sign, but observing the form of $\mathcal{M}(\boldsymbol{w})$ for each algorithm shows that this is not the case. This is an important distinction as it shows that the $\mathcal{H}_2(\boldsymbol{w})$ accounts for the reward structure of the problem, while the Fisher information matrix does not. It is therefore natural to expect $\mathcal{H}_2(\boldsymbol{w})$ to contain more information about the curvature of the objective function. This observation has not been noted before, but it yet it is clearly important an important result in terms of our understanding of natural gradient techniques in the context of Markov Decision Processes. That this observation has never been noted before probably lies in the fact that the Fisher information matrix is typically written in the form (2.9), in which form its relationship with $\mathcal{H}_2(\boldsymbol{w})$ is not apparent.

Due to the alternative form of the Fisher information matrix, *i.e.*

$$G(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big], \tag{4.7}$$

there is also an apparent relationship between the Fisher information matrix and $\mathcal{H}_1(\boldsymbol{w})$. However, this relationship is more complex. This is because for each $t \in \mathbb{N}$, the terms $\nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})$ and $\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})$ in (4.3) and (4.4), respectively, take the form

$$\nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}) = \sum_{\tau_1=1}^{t} \sum_{\tau_2=1}^{t} \nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}_{\tau_1}|\boldsymbol{s}_{\tau_1};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}_{\tau_2}|\boldsymbol{s}_{\tau_2};\boldsymbol{w}) \tag{4.8}$$

$$+ \text{ terms independent of } \pi,$$

$$\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}) = \sum_{\tau=1}^{t} \nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}_\tau|\boldsymbol{s}_\tau;\boldsymbol{w}) + \text{terms independent of } \pi, \tag{4.9}$$

The double summation in (4.8), as opposed to the single summation in (4.9), complicates the expression for $\mathcal{H}_1(\boldsymbol{w})$. However, taking the terms in (4.8) where $\tau_1 = \tau_2$, *i.e.* the terms

$$\sum_{\tau=1}^{t} \nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}_\tau|\boldsymbol{s}_\tau;\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}_\tau|\boldsymbol{s}_\tau;\boldsymbol{w}),$$

for each $t \in \mathbb{N}$ in (4.3) gives, after the, by now, standard manipulations from section(2.2), the following matrix

$$\mathcal{H}_{11}(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big]. \tag{4.10}$$

The relationship between (4.7) and (4.10) is analogous to the relationship between (4.5) and (4.6). However, while $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$ is a valid preconditioner in a gradient-based optimisation algorithm, where this matrix can be seen to be positive-semidefinite because it is a positive mixture of positive-semidefinite matrices, we don't consider this preconditioner in this work. The reason for this is that the sign of this matrix is incorrect in terms of an approximate Newton method, *i.e.* the $\mathcal{H}_{11}(\boldsymbol{w})$ is positive-semidefinite and not negative-semidefinite. As a result the preconditioner, $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$, bears no immediate relation to the Newton method. While the two forms of the Fisher information matrix (4.5) and (4.7) are equivalent, the same is not true of the matrices $\mathcal{H}_{11}(\boldsymbol{w})$ and $-\mathcal{H}_2(\boldsymbol{w})$. This is because the equivalence between (4.5) and (4.7) relies on the fact that the integrals in (4.5) and (4.7) is taken *w.r.t.* to a distribution, which is not true in the case of $\mathcal{H}_{11}(\boldsymbol{w})$ or $-\mathcal{H}_2(\boldsymbol{w})$. Further analysis of the matrix $\mathcal{H}_{11}(\boldsymbol{w})$ and its relation to $\mathcal{H}_2(\boldsymbol{w})$ is a point of future research, but we do perform a simple preliminary comparison between these two preconditioners in section(4.3).

4.1.2 Expectation Maximisation

In section(2.4) we introduced the application of Expectation Maximisation to the Markov Decision Process framework, as well as other planning frameworks. Since its introduction this algorithm has proven popular and has been the centre of much research in the reinforcement learning and planning communities [40, 171, 170, 94, 93, 77, 57, 56]. In this section we provide our novel analysis of the search

direction of the EM-algorithm when applied to Markov Decision Process, as well as similar planning models. Prior to this, however, we first detail what has been previously noted about the search direction of the EM-algorithm.

In [94] it was noted that the parameter update of steepest gradient ascent and the EM-algorithm can be related through the 'energy' term (2.14). In particular the gradient (2.4) evaluated at $\boldsymbol{w}_k$ can also be written as follows $\nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k} U(\boldsymbol{w}) = \nabla^{10}_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k} \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$, where we use the notation $\nabla^{10}_{\boldsymbol{w}}$ to denote the first derivative *w.r.t.* the first parameter, while the update of the EM-algorithm is given by $\boldsymbol{w}_{k+1} = \operatorname*{argmax}_{\boldsymbol{w}\in\mathcal{W}} \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$. In other words, steepest gradient ascent moves in the direction that most rapidly increases $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ *w.r.t.* the first variable, while the EM-algorithm maximises $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ *w.r.t.* the first variable. While this relationship is true, it is also quite a negative result. It states that in situations where it is not possible to explicitly perform the maximisation over $\boldsymbol{w}$ in (2.14) then the alternative, in terms of the EM-algorithm, is this generalised EM-algorithm, which is equivalent to steepest gradient ascent. Considering that algorithms such as EM are typically considered because of the negative aspects related to steepest gradient ascent this is an undesirable alternative. It is possible to find the optimum of (2.14) numerically, but this is also undesirable as it results in a double-loop algorithm that could be computationally expensive. Finally, this result provides no insight into the behaviour of the EM-algorithm, in terms of the direction of its parameter update, when the maximisation over $\boldsymbol{w}$ in (2.14) can be performed explicitly.

Instead we provide the following result, which shows that the step-direction of the EM-algorithm has an underlying relationship with the Newton method. In particular we show that, under suitable regularity conditions, the direction of the EM-update, *i.e.* $\boldsymbol{w}_{k+1} - \boldsymbol{w}_k$, is the same, up to first order, as the direction of an approximate Newton method that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$.

Theorem 4. *Suppose we are given a Markov Decision Process with objective (2.1) and Markovian trajectory distribution (2.2). Consider the update of the parameter through Expectation Maximisation at the* k^{th} *iteration of the algorithm,* i.e.

$$\boldsymbol{w}_{k+1} = \operatorname*{argmax}_{\boldsymbol{w}\in\mathcal{W}} \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k).$$

Provided that $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ *is twice continuously differentiable in the first parameter we have that*

$$\boldsymbol{w}_{k+1} - \boldsymbol{w}_k = -\mathcal{H}_2^{-1}(\boldsymbol{w}_k) \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k} U(\boldsymbol{w}) + \mathcal{O}(\|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k\|^2). \tag{4.11}$$

Additionally, in the case where the log*-policy is quadratic the relation to the approximate Newton method is exact,* i.e. *the second term on the* r.h.s. *(4.11) is zero.*

Proof. The idea of the proof is simple and only involves performing a Taylor expansion of $\nabla^{10}_{\boldsymbol{w}} \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$. As $\mathcal{Q}$ is assumed to be twice continuously differentiable in the first component this Taylor expansion is possible and gives

$$\nabla^{10}_{\boldsymbol{w}} \mathcal{Q}(\boldsymbol{w}_{k+1}, \boldsymbol{w}_k) = \nabla^{10}_{\boldsymbol{w}} \mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k) + \nabla^{20}_{\boldsymbol{w}} \mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k)(\boldsymbol{w}_{k+1} - \boldsymbol{w}_k) + \mathcal{O}(\|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k\|^2). \tag{4.12}$$

As $\boldsymbol{w}_{k+1} = \underset{\boldsymbol{w}\in\mathcal{W}}{\operatorname{argmax}}\ \mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$ it follows that $\nabla^{10}_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}_{k+1}, \boldsymbol{w}_k) = 0$. This means that, upon ignoring higher order terms in $\boldsymbol{w}_{k+1} - \boldsymbol{w}_k$, the Taylor expansion (4.12) can be rewritten into the form

$$\boldsymbol{w}_{k+1} - \boldsymbol{w}_k = -\nabla^{20}_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k)^{-1}\nabla^{10}_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k). \tag{4.13}$$

The proof is completed by observing that $\nabla^{10}_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k) = \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k}U(\boldsymbol{w})$ and $\nabla^{20}_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}_k, \boldsymbol{w}_k) = \mathcal{H}_2(\boldsymbol{w}_k)$. The second statement follows because in the case where the log-policy is quadratic the higher order terms in the Taylor expansion vanish. □

Theorem 4 gives us a deeper understanding of the application of Expectation Maximisation to MDPs, and similar planning models. The derivation of EM-algorithm in terms of optimising a lower-bound of the log-objective gives little insight into the update direction of the algorithm. This result now shows that the algorithm is, up to first order, taking steps in the direction of a approximate Newton method (that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$) with a constant step-size of one. Furthermore, when the log-policy is quadratic in the policy parameters the relation (4.11) given in theorem 4 is exact, *i.e.*

$$\boldsymbol{w}_{k+1} - \boldsymbol{w}_k = -\mathcal{H}_2^{-1}(\boldsymbol{w}_k)\nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k}U(\boldsymbol{w}).$$

This follows because the higher order terms in the Taylor expansion, *i.e.* $\nabla^{i0}_{\boldsymbol{w}}\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w}_k)$, $\forall i > 2$, are equal to zero in this case. In this case an EM-step is exactly equal to the approximate Newton step that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$.

There is a superficially similar result that was given in terms of the problem of marginal log-maximisation, see lemma 1 of [81]. In [81] the following relation is given

$$\boldsymbol{w}_{k+1} - \boldsymbol{w}_k = -\big(\mathcal{P}(\hat{\boldsymbol{w}}, \hat{\boldsymbol{w}})\big)^{-1}\nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k}\log L(\boldsymbol{w}) + \mathcal{O}(\|\boldsymbol{w}_k - \hat{\boldsymbol{w}}\|^2), \tag{4.14}$$

where $\log L(\boldsymbol{w})$ is the marginal log-likelihood function, $\hat{\boldsymbol{w}}$ is the local maximum of the log-likelihood to which the iterates of the algorithm converge and $\mathcal{P}(\boldsymbol{w}, \boldsymbol{w})$ is the expectation of the complete log-likelihood conditioned on the observed data. The terms $\log L(\boldsymbol{w})$ and $\mathcal{P}(\boldsymbol{w}, \boldsymbol{w})$ play similar roles to terms $U(\boldsymbol{w})$ and $\mathcal{Q}(\boldsymbol{w}, \boldsymbol{w})$ in our problem and while (4.11) and (4.14) are superficially similar there are significant differences. As $\hat{\boldsymbol{w}}$ is unknown in practice the relation (4.14) has no practical use and its purpose in [81] is simply as a motivation for using $\boldsymbol{w}_{k+1} - \boldsymbol{w}_k$ as a search direction in a line search algorithm. Additionally, the relation (4.14) requires $\hat{\boldsymbol{w}}$ to both exist and be an interior point of the parameter space, which is often not the case in the control and planning frameworks.[1] Finally, the distance between the current iterate and $\hat{\boldsymbol{w}}$ can be significant for large parts of the optimisation process, which means the second term of (4.14) will dominate and this relation becomes effectively meaningless.

[1] For example in a discrete state-action MDP with a differentiable parameterisation of the table look-up policy there will generally be no fixed-point of the EM-algorithm. This is because the optimal policy is deterministic and cannot be represented with a set of finite-valued parameters in this parameterisation. Additionally, in a continuous MDP with a Gaussian policy the covariance is often on the boundary of the parameter space.

4.1.3 Summary

In this section we have provided a novel analysis of both natural gradient ascent and Expectation Maximisation when applied to the MDP framework. Previously, while both of these algorithms have proved popular methods for MDP optimisation, there has been little understanding of them in terms of their search-direction in the parameter space or their relation to the Newton method. Firstly, our analysis shows that the Fisher information matrix, which is used in natural gradient ascent, is similar to $\mathcal{H}_2(\boldsymbol{w})$ in (4.2), with the exception that the information about the reward structure of the problem is not contained in the Fisher information matrix, while such information is contained in $\mathcal{H}_2(\boldsymbol{w})$. Additionally we have shown that the step-direction of the EM-algorithm is, up to first order, an approximate Newton method that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$ and employs a constant step-size of one.

4.2 An Approximate Newton Method

A natural follow on from the analysis in section(4.1) is the consideration of using $\mathcal{M}(\boldsymbol{w}) = -\mathcal{H}_2^{-1}(\boldsymbol{w})$ in (4.1), a method we call the *full approximate Newton method* from this point onwards. In this section we show that this method has many desirable properties that make it an attractive alternative to other parametric policy search methods. Additionally, denoting the diagonal matrix formed from the diagonal elements of $\mathcal{H}_2(\boldsymbol{w})$ by $\mathcal{D}_2(\boldsymbol{w})$, we shall also consider the method that uses $\mathcal{M}(\boldsymbol{w}) = -\mathcal{D}_2^{-1}(\boldsymbol{w})$ in (4.1). We call this second method the *diagonal approximate Newton method*.

4.2.1 Properties of the Approximate Newton Methods

Recall that in (4.1) it is necessary that $\mathcal{M}(\boldsymbol{w})$ is positive-definite (in the Newton method this corresponds to requiring the Hessian to be negative-definite) to ensure an increase of the objective. In general the objective (2.1) is not concave, which means that the Hessian will not be negative-definite over the entire parameter space. In such cases the Newton method can actually lower the objective and this is an undesirable aspect of the Newton method.[2] An attractive property of the approximate Hessian, $\mathcal{H}_2(\boldsymbol{w})$, is that it is always negative-definite when the policy is $\log$–concave in the policy parameters. This fact follows from the observation that in such cases $\mathcal{H}_2(\boldsymbol{w})$ is a non-negative mixture of negative-definite matrices, which again is negative-definite [32]. Additionally, the diagonal terms of a negative-definite matrix are negative and so $\mathcal{D}_2(\boldsymbol{w})$ is also negative-definite when the controller is $\log$-concave.

To motivate this result we now briefly consider some widely used policies that are either $\log$-concave or blockwise $\log$-concave. Firstly, consider the Gibb's policy, $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \propto \exp \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{a},\boldsymbol{s})$, where $\boldsymbol{\phi}(\boldsymbol{a},\boldsymbol{s}) \in \mathbb{R}^{n_{\boldsymbol{w}}}$ is a feature vector. This policy is widely used in discrete systems and is $\log$-concave in $\boldsymbol{w}$, which can be seen from the fact that $\log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})$ is the sum of a linear term and a negative *log-sum-exp* term, both of which are concave [32]. In systems with a continuous state-action space a common choice of controller is $\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}_{\text{mean}},\Sigma) = \mathcal{N}(\boldsymbol{a}|K\boldsymbol{\phi}(\boldsymbol{s}) + \boldsymbol{m}, \Sigma(\boldsymbol{s}))$, where $\boldsymbol{w}_{\text{mean}} = \{K, \boldsymbol{m}\}$ and $\boldsymbol{\phi}(\boldsymbol{s}) \in \mathbb{R}^{n_{\boldsymbol{w}}}$ is a feature vector. The notation $\Sigma(\boldsymbol{s})$ is used because there are cases where is it beneficial to have state dependent noise in the controller. This controller is not jointly $\log$-concave in $\boldsymbol{w}_{\text{mean}}$ and

[2] Procedures exist, such as *modification symmetric indefinite factorisation* or *modified spectral decomposition*, see *e.g.* [121], for modifying the Hessian in such a manner that the modified Hessian is negative-semidefinite. However, these modifications can be expensive to perform and can also lose much of the information about the curvature of the objective function.

Σ, but it is blockwise $\log$-concave in $\boldsymbol{w}_{\text{mean}}$ and Σ^{-1}. In terms of $\boldsymbol{w}_{\text{mean}}$ the $\log$-policy is quadratic and the coefficient matrix of the quadratic term is negative-definite. In terms of Σ^{-1} the $\log$-policy consists of a linear term and a $\log$-determinant term, both of which are concave.

Although $\mathcal{H}_2(\boldsymbol{w})$ won't necessarily be negative-definite over the whole parameter space, unless the controller is $\log$-concave in the control parameters, it will negative-definite in a neighbourhood of a local optimum, $\boldsymbol{w}^* \in \mathcal{W}$, provided the Hessian $\mathcal{H}(\boldsymbol{w}^*)$ has its eigenvalues bounded away from zero. This can be seen from the relation

$$\mathcal{H}_2(\boldsymbol{w}) = \mathcal{H}(\boldsymbol{w}) - \mathcal{H}_1(\boldsymbol{w}),$$

where $\mathcal{H}_1(\boldsymbol{w})$ is always positive-semidefinite, as it is the non-negative mixture of outer-product matrices, and the Hessian is negative-definite in a neighbourhood of $\boldsymbol{w}^*$.

Another attractive property of the approximate Newton methods is the ease with which it is possible to extend the evaluation techniques necessary for a parameter update, such as those detailed in chapter(3), to the approximate Newton framework. Comparing the form of $\mathcal{H}_2$ given in (4.6) with the form of the gradient given in (2.4) it is clear that the integrals necessary for a parameter update using an approximate Newton method can be done in an almost identical manner to that of other parametric policy search methods, such as steepest gradient ascent or natural gradient ascent. In particular, gradient evaluation requires calculating the expectation of the derivative of the $\log$-policy *w.r.t.* $p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})$. In terms of inference the only additional calculation necessary to implement either the full or diagonal approximate Newton methods is the calculation of the expectation (*w.r.t.* to the same function) of the Hessian of the $\log$-policy, or its diagonal terms. As an example in algorithm(4.1) we consider the straightforward extension of the recurrent state formulation of gradient evaluation in the average reward framework, see *e.g.* [182], to the approximate Newton method. We use this extension in the Tetris experiment that we consider in section(4.3). Given n_s samples and $n_{\boldsymbol{w}}$ parameters the complexity of these extensions scale as $\mathcal{O}(n_s n_{\boldsymbol{w}})$ for the diagonal approximate Newton method, while it scales as $\mathcal{O}(n_s n_{\boldsymbol{w}}^2)$ for the full approximate Newton method.

While the extension of evaluation techniques to the approximate Newton framework is not difficult, by contrast the Newton method requires the construction of an additional set of inference routines. These additional routines are necessary for the calculation of $\mathcal{H}_1(\boldsymbol{w})$. Note that due to the Markov structure of the trajectory distribution $\mathcal{H}_1(\boldsymbol{w})$ can be written in the equivalent form

$$\mathcal{H}_1(\boldsymbol{w}) = \sum_{t=1}^{H} \sum_{\tau,\tau'=1}^{t} \mathbb{E}_{\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w})} \left[\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^{\top} \log \pi(\boldsymbol{a}'|\boldsymbol{s}';\boldsymbol{w}) \right]$$

where we have used the notation $\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w}) \equiv \tilde{p}(\boldsymbol{z}_\tau = \boldsymbol{z}, \boldsymbol{z}_{\tau'} = \boldsymbol{z}', t;\boldsymbol{w})$. It is clear that the calculation of $\mathcal{H}_1(\boldsymbol{w})$ requires the calculation of marginals of the form $\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w})$, for $\tau \neq \tau$. A sample-based calculation of this matrix is provided in [19], while we provide two novel model-based inference routines for the calculation of $\mathcal{H}_1(\boldsymbol{w})$: The first is a forward-backward routine for discrete systems where it is possible to enumerate over $\mathcal{Z}$; The second is a RTS-inference routine for linear systems. While these inference routines are interesting in themselves the actual form of the routines

Sample a state from the initial state distribution:

$$\boldsymbol{s}_1 \sim p_1(\cdot).$$

for $t = 1, ..., N$, for some $N \in \mathbb{N}$, **do**
Given the current state, sample an action from the policy:

$$\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t; \boldsymbol{w}).$$

if $\boldsymbol{s}_t \neq \boldsymbol{s}^*$ **then**
Update the eligibility traces:

$$\boldsymbol{\Phi}^1 \leftarrow \boldsymbol{\Phi}^1 + \nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t; \boldsymbol{w}) \qquad \boldsymbol{\Phi}^2 \leftarrow \boldsymbol{\Phi}^2 + \nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t; \boldsymbol{w})$$

else
reset the eligibility traces:

$$\boldsymbol{\Phi}^1 = \mathbf{0}, \qquad \boldsymbol{\Phi}^2 = \mathbf{0}.$$

end if

Update the estimates of the gradient and the approximate Hessian:

$$\boldsymbol{\Delta}^1 \leftarrow \boldsymbol{\Delta}^1 + R(\boldsymbol{a}_t, \boldsymbol{s}_t)\boldsymbol{\Phi}^1, \qquad \boldsymbol{\Delta}^2 \leftarrow \boldsymbol{\Delta}^2 + R(\boldsymbol{a}_t, \boldsymbol{s}_t)\boldsymbol{\Phi}^2.$$

Sample state from the transition dynamics:

$$\boldsymbol{s}_{t+1} \sim p(\cdot|\boldsymbol{a}_t, \boldsymbol{s}_t).$$

end for

Return the estimated gradient and approximate Hessian, which up to a positive scaling are given by $\boldsymbol{\Delta}^1$ and $\boldsymbol{\Delta}^2$ respectively.

Algorithm 4.1: Recurrent state sampling algorithm to estimate the search direction of the approximate Newton method when applied to an MDP with an infinite planning horizon with average rewards.

is not important to the discussion and the derivations are provided in appendix(C). While these additional inference routines are efficient (in the sense that they have a computational complexity that is linear in the the planning horizon, assuming the planning horizon is finite) performing the inference is already the most expensive part of policy search algorithms and this additional burden is unattractive. There is also a further drawback in the model-free setting, namely that the variance of sample-based estimates of $\mathcal{H}_1(\boldsymbol{w})$ will generally be larger than either the $\nabla_{\boldsymbol{w}} U(\boldsymbol{w})$ or $\mathcal{H}_2(\boldsymbol{w})$. This is for two reasons. Firstly, the elements in $\mathcal{H}_1(\boldsymbol{w})$ consists of the sum of expectations *w.r.t.* marginal distributions of the reward weighted trajectory distribution, where these marginals span different time-points, *i.e.* marginals of the form $\tilde{p}(\boldsymbol{z}, \boldsymbol{z}', \tau, \tau', t; \boldsymbol{w})$ where $\tau \neq \tau'$. In many systems the variance of these marginals will increase as the time difference, $|\tau - \tau'|$, increases. Additionally, terms of the form $\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w}) \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}'|\boldsymbol{s}'; \boldsymbol{w})$ will often be of a higher order than terms of the form $\nabla_{\boldsymbol{w}}^\top \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w})$ or $\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w})$. For example, in a continuous control problem with a linear-Gaussian policy the term $\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{w}) \nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}'|\boldsymbol{s}'; \boldsymbol{w})$ will contain monomials in the state-

action variables that are of order 4, while in $\nabla_{\boldsymbol{w}}^{\top}\nabla_{\boldsymbol{w}}\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})$ or $\nabla_{\boldsymbol{w}}\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})$ they will be of most order 2. Taking these points into account it is clear that the variance of sample-based estimates of $\mathcal{H}_1(\boldsymbol{w})$ will, in general, be larger than either $\mathcal{H}_2(\boldsymbol{w})$ or $\nabla_{\boldsymbol{w}}U(\boldsymbol{w})$.

An additional issue with the Newton method is the calculation and inversion of the Hessian matrix, which scale as $\mathcal{O}(n_{\boldsymbol{w}}^2)$ and $\mathcal{O}(n_{\boldsymbol{w}}^3)$ respectively in the worst case. In the standard application of the Newton method these operations have to be performed at each iteration, which in large parameter systems becomes prohibitively costly. In general $\mathcal{H}(\boldsymbol{w})$ will be dense and no computational savings will be possible when performing these operations. The same is not true, however, of the full and diagonal approximate Newton methods. Firstly, in the diagonal approximate Newton method the matrix $\mathcal{D}_2(\boldsymbol{w})$ is diagonal, so that the calculation and inversion of this matrix both scale as $\mathcal{O}(n_{\boldsymbol{w}})$. Additionally, there are several sources of sparsity in the matrix $\mathcal{H}_2(\boldsymbol{w})$ that can make the construction and inversion of this matrix more efficient. The reason that $\mathcal{H}(\boldsymbol{w})$ does not exhibit any such sparsity properties is due to the term $\mathcal{H}_1(\boldsymbol{w})$ in (4.2), which consists of the non-negative mixture of outer-product matrices. The vector in these outer-products is the derivative of the $\log$-trajectory distribution and this typically produces a dense matrix.

A first source of sparsity in $\mathcal{H}_2(\boldsymbol{w})$ comes from taking the second derivative of the $\log$-trajectory distribution in (4.4). This property ensures that any (product) sparsity over the control parameters in the trajectory distribution will correspond to sparsity in $\mathcal{H}_2(\boldsymbol{w})$. For example, in a *partially observable Markov Decision Processes* where the policy is modeled through a finite state controller, see *e.g.* [114], there are three functions to be optimised, namely the *initial belief distribution*, the *belief transition dynamics* and the *policy*. When the parameters of these three functions are independent $\mathcal{H}_2(\boldsymbol{w})$ will be block-diagonal (across the parameters of the three functions) and the construction and inversion of $\mathcal{H}_2(\boldsymbol{w})$ can be performed more efficiently by considering each of these block matrices individually. Another source of sparsity can occur in $\mathcal{H}_2(\boldsymbol{w})$ provided that the Hessian of the $\log$-policy has a suitably sparse structure. For example, consider an MDP with a discrete state-action space and the following policy parameterisation

$$\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) = \frac{e^{w_{\boldsymbol{a},\boldsymbol{s}}}}{\sum_{\boldsymbol{a}'\in\mathcal{A}} e^{w_{\boldsymbol{a}',\boldsymbol{s}}}}, \qquad \text{s.t. } w_{\bar{\boldsymbol{a}},\boldsymbol{s}} = 0, \text{ for some } \bar{\boldsymbol{a}} \in \mathcal{A}.$$

In this case the policy has a separate set of parameters for each state in the state space and it is not difficult to show that $\mathcal{H}_2(\boldsymbol{w})$ has a block diagonal structure across the parameters of the various states. In itself this example is not particularly interesting, simply because when such a parameterisation is feasible then it will also be feasible to perform dynamic programming. However, analogous parameterisations can be used in planning models that do not permit dynamic programming solutions, such as partially observable domains. Furthermore, such parameterisations can be used as part of a larger policy model. A simple example would be a high-dimensional discrete system, such as a multi-agent system, where each action

is of the form $\boldsymbol{a} = (\boldsymbol{a}_1, ..., \boldsymbol{a}_N)$ and the overall policy is given by the product of sub-policies, *i.e.*

$$\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) = \prod_{n=1}^{N} \pi^n(\boldsymbol{a}_n|\boldsymbol{s}^n;\boldsymbol{w}),$$

where $\boldsymbol{s}^n$ is some subset of the state variables that is sufficiently small so as to allow the parameterisation to be feasible. In this example $\mathcal{H}_2(\boldsymbol{w})$ has a block-diagonal structure across the sub-policies, $\{\pi^n\}_{n=1}^N$, and then, furthermore, each of these block matrices also has a block-diagonal structure across the states in the conditioning set. A final motivating example is an MDP with a continuous state-action space, where the parameters are a set of $n_{\boldsymbol{a}} \times n_{\boldsymbol{s}}$ matrices, $\boldsymbol{w} = \{K_i\}_{i=1}^N$. For any given state the policy is given by

$$\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) = \mathcal{N}(\boldsymbol{a}|K_i\boldsymbol{s};\sigma^2),$$

where the matrix K_i is selected through a Vornoi tessellation of the state space. In this example $\mathcal{H}_2(\boldsymbol{w})$ has a block diagonal structure across the matrices, $\{K_i\}_{i=1}^N$, again making evaluation and inversion more efficient.

An undesirable aspect of steepest gradient ascent is that its performance is affected by the choice of basis used to represent the parameter space. This choice of representation is essentially arbitrary and that the performance of steepest gradient ascent depends upon it is undesirable. An important and desirable property of the Newton method is that it is invariant to non-singular linear (affine) transformations of the parameter space, see *e.g.* [32]. This means that given a non-singular linear (affine) mapping $\mathcal{T} \in \mathbb{R}^{n_{\boldsymbol{w}} \times n_{\boldsymbol{w}}}$, the Newton update of the objective $\tilde{U}(\boldsymbol{w}) = U(\mathcal{T}\boldsymbol{w})$ is related to the Newton update of the original objective through the same linear (affine) mapping, *i.e.*

$$\boldsymbol{v} + \Delta\boldsymbol{v}_{\text{nt}} = \mathcal{T}\big(\boldsymbol{w} + \Delta\boldsymbol{w}_{\text{nt}}\big),$$

where $\boldsymbol{v} = \mathcal{T}\boldsymbol{w}$ and $\Delta\boldsymbol{v}_{\text{nt}}$ and $\Delta\boldsymbol{w}_{\text{nt}}$ denote the respective Newton steps. In other words running the Newton method on $U(\boldsymbol{w})$ and $\tilde{U}(\mathcal{T}^{-1}\boldsymbol{w})$ will give identical results. An important point to note is that this desirable property is maintained when using $\mathcal{H}_2(\boldsymbol{w})$ in an approximate Newton method, which we prove in the following lemma.

Lemma 3. *Given a discrete Markov Decision Process, where the policy is parameterised by $\boldsymbol{w} \in \mathcal{W}$, the approximate Newton method that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$ is invariant to non-singular linear mappings of the parameter space.*

Proof. We denote an arbitrary non-singular linear mapping of the parameter space by $\mathcal{T} : \mathbb{R}^{n_{\boldsymbol{w}}} \to \mathbb{R}^{n_{\boldsymbol{w}}}$. To show the affine invariance of the approximate Newton method we use the following formulae

$$\nabla_{\boldsymbol{w}}\tilde{f}(\boldsymbol{w}) = \mathcal{T}^\top \nabla_{\boldsymbol{v}} f(\boldsymbol{v}), \qquad \nabla_{\boldsymbol{w}}\nabla_{\boldsymbol{w}}^\top \tilde{f}(\boldsymbol{w}) = \mathcal{T}^\top \nabla_{\boldsymbol{v}}\nabla_{\boldsymbol{v}}^\top f(\boldsymbol{v})\mathcal{T},$$

where f is some twice differentiable function of $\boldsymbol{w}$, $\tilde{f}(\boldsymbol{w}) = f(\mathcal{T}\boldsymbol{w})$ and $\boldsymbol{v} = \mathcal{T}\boldsymbol{w}$. Using these formulae

we have the following two identities

$$\nabla_{\boldsymbol{w}} \log \pi(\boldsymbol{a}|\boldsymbol{s}; \mathcal{T}\boldsymbol{w}) = \mathcal{T}^\top \nabla_{\boldsymbol{v}} \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{v}),$$
$$\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s}; \mathcal{T}\boldsymbol{w}) = \mathcal{T}^\top \nabla_{\boldsymbol{v}} \nabla_{\boldsymbol{v}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{v}) \mathcal{T},$$

which hold for each $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}$. Defining $\tilde{U}(\boldsymbol{w}) = U(\mathcal{T}\boldsymbol{w})$, we have $\nabla_{\boldsymbol{w}} \tilde{U}(\boldsymbol{w}) = \mathcal{T}^\top \nabla_{\boldsymbol{v}} U(\boldsymbol{v})$. Following calculations almost identical to those in section(4.1) it can be shown that $\tilde{\mathcal{H}}_2(\boldsymbol{w})$ takes the form

$$\tilde{\mathcal{H}}_2(\boldsymbol{w}) = \mathbb{E}_{p_\gamma(\boldsymbol{z}; \mathcal{T}\boldsymbol{w}) Q(\boldsymbol{z}; \mathcal{T}\boldsymbol{w})} \Big[\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s}; \mathcal{T}\boldsymbol{w}) \Big],$$

which gives the following

$$\begin{aligned}
\tilde{\mathcal{H}}_2(\boldsymbol{w}) &= \mathbb{E}_{p_\gamma(\boldsymbol{z}; \mathcal{T}\boldsymbol{w}) Q(\boldsymbol{z}; \mathcal{T}\boldsymbol{w})} \Big[\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s}; \mathcal{T}\boldsymbol{w}) \Big], \\
&= \mathbb{E}_{p_\gamma(\boldsymbol{z}; \boldsymbol{v}) Q(\boldsymbol{z}; \boldsymbol{v})} \Big[\mathcal{T}^\top \nabla_{\boldsymbol{v}} \nabla_{\boldsymbol{v}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{v}) \mathcal{T} \Big], \\
&= \mathcal{T}^\top \mathbb{E}_{p_\gamma(\boldsymbol{z}; \boldsymbol{v}) Q(\boldsymbol{z}; \boldsymbol{v})} \Big[\nabla_{\boldsymbol{v}} \nabla_{\boldsymbol{v}}^\top \log \pi(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{v}) \Big] \mathcal{T}, \\
&= \mathcal{T}^\top \mathcal{H}_2(\boldsymbol{v}) \mathcal{T}.
\end{aligned}$$

Using these two expressions we have that the parameter updates, under the approximate Newton method, of the objective functions U and $\tilde{U}$ are related as follows

$$\begin{aligned}
\boldsymbol{v}_{\text{new}} &= \boldsymbol{v} + \alpha \mathcal{H}_2(\boldsymbol{v})^{-1} \nabla_{\boldsymbol{v}} U(\boldsymbol{v}), \\
&= \mathcal{T}\big(\boldsymbol{w} + \alpha \tilde{\mathcal{H}}_2(\boldsymbol{w})^{-1} \nabla_{\boldsymbol{v}} \tilde{U}(\boldsymbol{w})\big),
\end{aligned}$$

where α is some step-size parameter. This shows that the approximate Newton method is affine invariant.

□

The proof to lemma 3 can be changed with little difficulty to show that the diagonal approximate Newton method is invariant to arbitrary (non-zero) rescaling of the parameters along the various dimensions of the parameter space. This is a less general form of invariance than affine invariance, and algorithms with this form of invariance are generally known as covariant. Note that the proof in lemma 3 shows that the affine invariance of the approximate Newton method is independent of the step-sizes used during the optimisation. Additionally, the same argument can also be made to show that natural gradient ascent in affine invariant. Due to some apparently contradictory results in [83, 13], as well as the conflicting use of terminology in [83, 13, 123, 111], it is worthwhile to clarify the distinction between the different types of invariance an optimisation algorithm can possess, as well as providing the exact categorisation of the invariance of the approximate Newton method and natural gradient ascent. Three popular types of invariance that appear in the literature are *covariant algorithms*, *linear (affine) invariant algorithms* and *invariant algorithms*. As we have noted previously the class of linear (affine) invariant algorithms is the class of algorithms that are invariant to non-singular linear (affine) transformations of

the parameter space. Covariant algorithms, see *e.g.* [111], are the subset of linear invariant algorithms that are invariant to non-singular orthogonal linear transformations of the parameter space, *i.e.* mappings where $\mathcal{T}$ is a diagonal matrix with non-zero elements along the diagonal. Invariant algorithms are invariant to arbitrary non-singular transformations of the parameter space. Natural gradient ascent algorithms that use Fisher information matrix as a local metric are invariant, but only up to a infinitesimal step in the parameter space [37]. Additionally, using the same argument as in lemma 3 it can be seen that natural gradient ascent algorithms are also invariant to non-singular linear (affine) transformations of the parameter space, where this invariance is independent of the step-sizes used in the parameter updates. Now in terms of the application of natural gradient ascent methods to Markov Decision Processes it was, incorrectly, claimed in [83] that the algorithm was not covariant, where this claim was based on empirical evaluations. In later papers in this area [123, 13] this error was rectified and the invariance properties of natural gradient ascent were clarified, namely that natural gradient ascent techniques are invariant when using the Fisher information matrix as the local norm on the parameter manifold, although they use the terminology of covariant algorithms instead of invariant algorithms. Also in [123, 13] a possible explanation for the results in [83] were given, where they state that these results maybe due to the necessity to perform infinitesimal updates to maintain the invariance of natural gradient ascent. This explanation was also used in [13] to account for similar anomalies in the results presented in that paper, where the experiments considered were again related to the covariant, and not the invariant, property of natural gradient ascent. We can now see that this claim is in fact false and the covariant property of natural gradient ascent is independent of the step-sizes used in the parameter update. We do not offer a possible reason for the results of [83, 13]. To summarise natural gradient ascent is invariant for steps of infinitesimal size in the parameter space and linear (affine) invariant for steps of arbitrary size. In contrast we have currently only shown that the full approximate Newton method is linear (affine) invariant, while the diagonal approximate Newton method is covariant, both for steps of arbitrary size.

We performed an empirical illustration that the full approximate Newton method is invariant to linear transformations of the parameter space. We considered the simple two state example of [83] as it allows us to plot the trace of the policy during training, since the policy has only two parameters. The policy was trained using both steepest gradient ascent and the full approximate Newton method and in both the original and linearly transformed parameter space. The policy traces of the two algorithms are plotted in fig(4.2). As expected steepest gradient ascent is affected by such mappings, whilst the full approximate Newton method is invariant to them.

It is well-known that the search direction of gradient-based algorithms of the form (4.1) correspond to the direction of steepest ascent *w.r.t.* to the local quadratic norm induced by $\mathcal{M}(\boldsymbol{w})$. In particular, given that $\mathcal{M}(\boldsymbol{w})$ is positive-definite it is possible to define a local quadratic norm, centred at $\boldsymbol{w}$, so that given $\boldsymbol{v} \in \mathcal{W}$, where $\boldsymbol{v} = \boldsymbol{w} + \boldsymbol{p}$, the local norm of $\boldsymbol{v}$ is defined as follows

$$\|\boldsymbol{v}\|_{\mathcal{M}(\boldsymbol{w})} = \big(\boldsymbol{v} - \boldsymbol{w}\big)^\top \mathcal{M}(\boldsymbol{w}) \big(\boldsymbol{v} - \boldsymbol{w}\big) = \boldsymbol{p}^\top \mathcal{M}(\boldsymbol{w}) \boldsymbol{p},$$

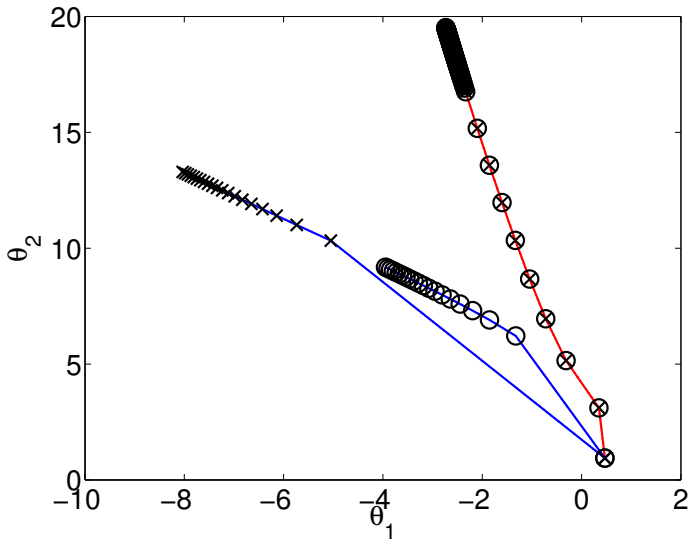


Figure 4.2: An empirical illustration of the affine invariance of the full approximate Newton method, performed on the two state MDP of [83]. The plot shows the trace of the policy during training for the two different parameter spaces, where the results of the latter have been mapped back into the original parameter space for comparison. The plot shows the two steepest gradient ascent traces (blue cross and blue circle) and the two traces of the full approximate Newton method (red cross and red circle).

where this norm is well-defined because the $\mathcal{M}(\boldsymbol{w})$ is positive-definite.[3] If at any given point in the parameter space, $\boldsymbol{w}_k \in \mathcal{W}$, we consider optimising the first order variation of $U(\boldsymbol{w}_k + \boldsymbol{p})$ *w.r.t.* $\boldsymbol{p}$, under the constraint that $\|\boldsymbol{w}_k + \boldsymbol{p}\|_{\mathcal{M}(\boldsymbol{w}_k)}$ is equal to some infinitesimally small constant, then one obtains the following constrained optimisation problem

$$\max_{\substack{\boldsymbol{p}\in\mathbb{R}^{n_{\boldsymbol{w}}} \\ s.t. \|\boldsymbol{w}_k+\boldsymbol{p}\|_{\mathcal{M}(\boldsymbol{w}_k)}=\epsilon}} U(\boldsymbol{w}_k) + \boldsymbol{p}^\top \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k} U(\boldsymbol{w}). \tag{4.15}$$

Solving this constrained optimisation problem through the method of Lagrange multipliers, see *e.g.* [25], then one obtains the solution

$$\boldsymbol{p} = \mathcal{M}^{-1}(\boldsymbol{w}_k) \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k} U(\boldsymbol{w}). \tag{4.16}$$

This shows that gradient-based algorithms of the form (4.1) are moving in the direction of steepest ascent under the quadratic norm induced by $\mathcal{M}(\boldsymbol{w})$, where this matrix is evaluated at the iterates of the given gradient-based algorithm. This viewpoint gives a new perspective to the various gradient-based algorithm. For instance, in steepest gradient ascent, where $\mathcal{M}(\boldsymbol{w}) = I$ for all $\boldsymbol{w} \in \mathcal{W}$, the search direction corresponds to the direction of steepest ascent under the Euclidean norm. Similarly, the search direction of natural gradient ascent corresponds to the steepest ascent direction under the local norm defined on the parameter manifold, where we recall that this norm is defined through the Fisher information matrix. When the objective is concave the Hessian will be negative-definite over the entire

[3] Actually, as $\mathcal{M}(\boldsymbol{w})$ is positive-definite, this norm is defined over the entire parameter space. However, the norm is described as local because the matrix $\mathcal{M}(\boldsymbol{w})$ only provides local information about the objective function.

parameter space and it is possible to define the following norm

$$\|\boldsymbol{w}+\boldsymbol{p}\|_{\mathcal{H}(\boldsymbol{w})} = -\boldsymbol{p}^\top \Big(\nabla_{\boldsymbol{w}}^\top \nabla_{\boldsymbol{w}} U(\boldsymbol{w}) \Big) \boldsymbol{p},$$

which is referred to as the Hessian norm, see *e.g.* [32]. This view of the Newton method moving in the direction of steepest ascent under the Hessian norm helps to explain the good performance of the Newton method in the vicinity of a local optimum, where the objective is approximately quadratic.

Taking these points into account it is natural to ask the question: What is the norm under which the search direction of the approximate Newton method corresponds to the steepest gradient ascent direction? Using the same argument as above the explicit functional form of this norm can be found to be the quadratic norm defined as follows

$$\|\boldsymbol{w}+\boldsymbol{p}\|_{\mathcal{H}_2(\boldsymbol{w})} = \boldsymbol{p}^\top \mathbb{E}_{p_\gamma(\boldsymbol{z};\boldsymbol{w})Q(\boldsymbol{z};\boldsymbol{w})} \Big[\nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \Big] \boldsymbol{p}. \tag{4.17}$$

Considering $\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})$ as a function of $\boldsymbol{w}$, for each $(\boldsymbol{a},\boldsymbol{s}) \in \mathcal{A}\times\mathcal{S}$, the term $\boldsymbol{w}^\top \nabla_{\boldsymbol{w}} \nabla_{\boldsymbol{w}}^\top \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \boldsymbol{w}$ can be considered as the Hessian norm *w.r.t.* $\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})$. From this perspective the norm (4.17) can be seen as the non-negative mixture of Hessian norms, where this mixture is taken *w.r.t.* to $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$. Considering the form of the gradient, which is given by a non-negative mixture of the derivatives of the $\log$-policy, where the mixture is *w.r.t.* $p_\gamma(\cdot;\boldsymbol{w})Q(\cdot;\boldsymbol{w})$, this form of the quadratic norm is intuitive.

Thus far we have been concerned with comparing the approximate Newton methods with the Newton method. It is also important to provide a brief comparison with both natural gradient ascent and Expectation Maximisation. A first point to note is that implementations of these algorithms that only use an actor for the evaluation stage are very similar, with almost identical computational complexity in practice. In terms of natural gradient ascent there are several important distinctions. Firstly, when using the Fisher information matrix as the local norm on the manifold of trajectory distributions, natural gradient ascent is invariant to arbitrary non-singular transformations of the parameter space, while the approximate Newton method is invariant to linear (affine) transformations of the parameter space. Additionally, the Fisher information matrix is always positive-semidefinite, while the approximate Hessian is guaranteed to be negative-semidefinite only when the policy is $\log$-concave. Furthermore, in actor-critic versions of natural gradient ascent it is possible to construct versions of the algorithm where it is unnecessary to perform the inversion of the Fisher information matrix, see *e.g.* [83, 29]. These are all important points but the approximate Newton method has one significant advantage over natural gradient ascent, namely that the approximate Hessian contains information about the reward structure of the problem while the Fisher information matrix does not. Hence the approximate Hessian will contain more information about the curvature of the objective than the Fisher information matrix. In the experiments performed in section(4.3) we find that, especially in continuous systems, this improved estimate of the curvature provides significant benefits to the approximate Newton method, both in terms of the quality of the solution found and in terms of finding a good step-size sequence. A final point concerns the selection of a step-size sequences in a stochastic, model-free, version of the approximate Newton

method. Associated with the Newton method is the 'natural' step-size of one, which comes from the view of the Newton method as optimising a quadratic approximation to the objective function, see *e.g.* [121]. Typical applications of the Newton method use an initial step-size of one and then only consider tuning the step-size if there is not a satisfactory change in the value of the objective function. In view of theorem 4 there is, intuitively at least, a similar 'natural' step-size of one, which (up to first order) corresponds to an EM-step. Empirically, in the experiments considered in section(4.3), this intuition has been found to be useful as an initial gauge for the scale of the step-sizes that will be appropriate for a given problem.

In terms of a comparison between the approximate Newton method and Expectation Maximisation there are several points to note. Firstly, there is an important distinction between the approximate Newton method and Expectation Maximisation in the case where it is not possible to perform the maximisation over $\boldsymbol{w}$ explicitly in (2.14). When this is the case it is necessary, in terms of the EM-algorithm, to consider either steepest gradient ascent or a double-loop type algorithm, both of which are unappealing. In contrast, provided that the policy is $\log$-concave, the approximate Newton method can still be applied as normal. Conversely, the EM-algorithm is more general than the approximate Newton method in that it can be applied to policies that are not $\log$-concave, even if it is necessary to perform a double-loop type procedure to perform the optimisation over $\boldsymbol{w}$ in (2.14). It was noted earlier that in the special case where $\log \pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})$ is quadratic the search direction of the approximate Newton method coincides with the direction of the EM-step. In this special case the EM-algorithm can be seen as a specialisation of the approximate Newton method with a fixed step-size of one. The approximate Newton method is therefore more general in this case and superior performance can be obtained by tuning the step-sizes. In the experiments performed in section(4.3) it was found that tuning the step-size of the approximate Newton method to obtain, sometimes markedly, superior performance was not a difficult problem.

Quasi-Newton Methods

While our focus on the approximate Newton method is primarily as a stochastic optimisation algorithm it is also interesting to consider the situation where it is possible to perform policy evaluation analytically, for example in linear systems. When it is possible to calculate the gradient exactly quasi-Newton methods provide an attractive alternative, see *e.g.* [121] for an overview. Quasi-Newton methods, such as the l-bfgs method, work by constructing an estimate to the product of the negative inverse Hessian with the gradient, with the constraint that the resulting search direction is an ascent direction. While these methods only use the gradient at previous iterates of the algorithm they are able to obtain super-linear convergence in the full case, or linear convergence in the limited memory case. Although typical applications of quasi-Newton methods use only the gradient of previous iterates it is also possible to incorporate knowledge about the Hessian, such as the particular form of certain terms in the Hessian, into these methods. Such an approach has been successfully considered previously in problems of marginal $\log$-likelihood maximisation [106] and non-linear regression [118]. Given our knowledge about the structure

of the Hessian for MDPs such an approach is again possible in this framework. We do not consider this point in depth, but we do provide a simple illustrative example in section(4.3) to highlight the possibility of these methods.

4.2.2 Convergence Analysis

Clearly, as $\mathcal{H}_2(\boldsymbol{w})$ is only an approximation to $\mathcal{H}(\boldsymbol{w})$, the rate of convergence of the Newton method in the vicinity of a local optimum will generally be superior to that of the approximate Newton method. To avoid going into too much technical details we simply give a sketch of the convergence analysis. We denote the mapping of the parameter vector defined through the update equation of the approximate Newton method by $\mathcal{M}(\boldsymbol{w})$, *i.e.*

$$\mathcal{M}(\boldsymbol{w}) = \boldsymbol{w} - \mathcal{H}_2^{-1}(\boldsymbol{w})\nabla_{\boldsymbol{w}} U(\boldsymbol{w}),$$

where we suppose a step-size of one for simplicity. This notation is not to be confused with the notation in (4.1). A standard approach for studying the rate of convergence of an algorithm is to first perform a Taylor expansion of $\mathcal{M}(\boldsymbol{w}_k)$, for a sufficiently large $k \in \mathbb{N}$, around a local optimum of the objective, which we denote by $\boldsymbol{w}^*$. Up to first order in $\|\boldsymbol{w}_k - \boldsymbol{w}^*\|$ this Taylor expansion gives

$$\boldsymbol{w}_{k+1} - \boldsymbol{w}^* = \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w})\big(\boldsymbol{w}_{k+1} - \boldsymbol{w}^*\big).$$

For notational simplicity we denote $\nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w})$ and $\nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}^*}U(\boldsymbol{w})$ respectively by $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ and $\nabla_{\boldsymbol{w}^*}U(\boldsymbol{w}^*)$. Intuitively $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ can be viewed as contraction mapping with a Lipschitz constant corresponding to the largest eigenvalue of $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$. Note that, as we shall see shortly, when $\mathcal{H}(\boldsymbol{w}^*)$ is negative-definite, negative-semidefinite, the eigenvalues of $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ will be contained in $[0, 1)$ and $[0, 1]$ respectively. There will be rapid convergence in the direction of eigenvectors which correspond to eigenvalues that are close to zero, while those corresponding to eigenvalues close to unity will have slow convergence. The explicit form of $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ can be calculated as follows

$$\begin{aligned}\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*) &= I - \Big(\nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}^*}\mathcal{H}_2^{-1}(\boldsymbol{w})\Big)\nabla_{\boldsymbol{w}^*}U(\boldsymbol{w}^*) - \mathcal{H}_2^{-1}(\boldsymbol{w}^*)\Big(\nabla_{\boldsymbol{w}}^{\top}\nabla_{\boldsymbol{w}}U(\boldsymbol{w})\Big)_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}^*}, \\ &= I - \mathcal{H}_2^{-1}(\boldsymbol{w}^*)\mathcal{H}(\boldsymbol{w}^*).\end{aligned}$$

The second line follows from the first as the gradient of the objective is zero at a local maximum. Given the fact that $\mathcal{H}(\boldsymbol{w}) = \mathcal{H}_1(\boldsymbol{w}) + \mathcal{H}_2(\boldsymbol{w}), \forall \boldsymbol{w} \in \mathcal{W}$, this can be simplified further

$$\begin{aligned}\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*) &= \mathcal{H}_2^{-1}(\boldsymbol{w}^*)\Big(\mathcal{H}_2(\boldsymbol{w}^*) - \mathcal{H}(\boldsymbol{w}^*)\Big), \\ &= -\mathcal{H}_2^{-1}(\boldsymbol{w}^*)\mathcal{H}_1(\boldsymbol{w}^*).\end{aligned}$$

It is clear that the rate of convergence is determined by the relative sizes of $\mathcal{H}_1(\boldsymbol{w})$ and $\mathcal{H}_2(\boldsymbol{w})$. When the eigenvalues of $\mathcal{H}_1(\boldsymbol{w})$ are small in comparison to the eigenvalues of $\mathcal{H}_2(\boldsymbol{w})$ convergence will typically

be fast. It is also possible to relate $\mathcal{M}(\boldsymbol{w})$ to the Newton method be writing $\mathcal{M}(\boldsymbol{w})$ in the following form

$$\mathcal{M}(\boldsymbol{w}) = \boldsymbol{w} - \Big(I - \big(-\mathcal{H}_2^{-1}(\boldsymbol{w})\mathcal{H}_1(\boldsymbol{w})\big)\Big)\mathcal{H}^{-1}(\boldsymbol{w})\nabla_{\boldsymbol{w}}U(\boldsymbol{w}).$$

The approximate Newton method can now be seen to take the step-direction of the Newton method, $-\mathcal{H}^{-1}(\boldsymbol{w})\nabla_{\boldsymbol{w}}U(\boldsymbol{w})$, and to modify it by mapping it through the matrix, $I - \big(-\mathcal{H}_2^{-1}(\boldsymbol{w})\mathcal{H}_1(\boldsymbol{w})\big)$. Sufficiently close to $\boldsymbol{w}^*$ we have the approximate relation

$$\mathcal{M}(\boldsymbol{w}) \approx \boldsymbol{w} - \Big(I - \nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)\Big)\mathcal{H}^{-1}(\boldsymbol{w})\nabla_{\boldsymbol{w}}U(\boldsymbol{w}).$$

When the eigenvalues of $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ are close to zero the identity matrix will dominate the term $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ and quasi-Newton type behaviour is possible. Conversely, when the eigenvalues of $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ are close to one the step-sizes will be small and slow convergence will result.

We noted earlier that when $\mathcal{H}(\boldsymbol{w}^*)$ is negative-definite, negative-semidefinite, the matrix $\mathcal{H}_2(\boldsymbol{w}^*)$ is respectively negative-definite or negative-semidefinite. Additionally, the relation $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*) = I - \mathcal{H}_2^{-1}(\boldsymbol{w}^*)\mathcal{H}(\boldsymbol{w}^*)$ can be rewritten in the form $\mathcal{H}(\boldsymbol{w}^*) = \mathcal{H}_2(\boldsymbol{w}^*)\big(I - \nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)\big)$. It now follows that when the Hessian is negative-definite, negative-semidefinite, the eigenvalues of $\nabla_{\boldsymbol{w}^*}\mathcal{M}(\boldsymbol{w}^*)$ are respectively contained in $[0, 1)$ or $[0, 1]$, which validates our earlier claim.

4.2.3 Summary

In this section we have considered the application of the approximate Newton method that uses the approximate Hessian, $\mathcal{H}_2(\boldsymbol{w})$, in place of the Hessian, $\mathcal{H}(\boldsymbol{w})$. We have found that it has many desirable properties that are absent in the naive application of the Newton method. These advantages include guarantees that the $\mathcal{H}_2(\boldsymbol{w})$ will being negative-semidefinite when the policy is $\log$-concave; Sparsity properties of $\mathcal{H}_2(\boldsymbol{w})$, not present in $\mathcal{H}(\boldsymbol{w})$, that make the inversion of the approximate Hessian more efficient; More efficient policy evaluation. Additionally, the approximate Newton method maintains the attractive property of being invariant to linear (affine) transformations of the parameter space. The use of this approximate Hessian comes at the cost of a reduced rate of convergence, where the rate of convergence of the approximate Newton method is anywhere between sub-linear and quadratic.

Considering the attractive properties of the Newton method, in terms of its rate of convergence and scale invariance, it is surprising that an approximate Newton method has not been previously considered for the optimisation of Markov Decision Processes. In many optimisation problems the Hessian often has the form of being the sum of two matrices, where one is an outer-product matrix of first order derivatives and the second matrix consists of second order derivatives. Typically the outer-product matrix is used to approximate the Hessian as it often easier to calculate and it is guaranteed to be positive-semidefinite. This is often called the Gauss-Newton method or the outer-product approximation [30] and an example of its application is in nonlinear least-squares problems. However, we have seen that, due to the temporal structure of the objective for Markov Decision Processes, this term is troublesome and such an approximation has numerous undesirable properties. The most similar approximate Newton method that we have been able to locate in the literature is an approximation to the Hessian of the marginal

log-likelihood function in a maximum likelihood problem [105, 106]. This is one possible suggestion as to why no previous attempts have been made to make efficient approximations to the Hessian in this problem setting.

4.3 Experiments

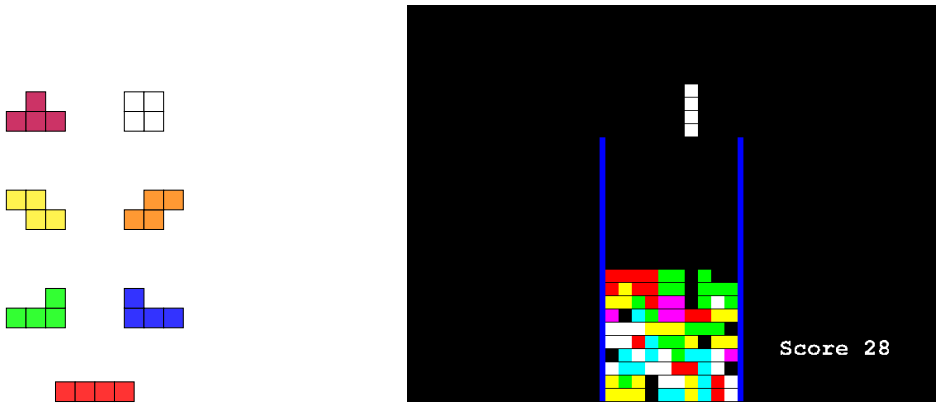
In this section we detail some experiments performed on the approximate Newton methods. Firstly, we demonstrate the approximate Newton methods on the game of tetris, which is a non-trivial high-dimensional discrete system. In continuous system we first demonstrate the approximate Newton method on a high-dimensional linear system. This allows the search directions of the various parametric policy search methods to calculated exactly, thus removing any issues the approximate inference clouding the results, and allows for the comparison of search directions and selection of step-size sequences. Additionally, we provide a simple illustration of using our knowledge of the Hessian to improve the performance of a quasi-Newton method. Finally, we demonstrate the approximate Newton method on a simple two-dimensional non-linear system where a forwarding-sampling is used to evaluate the search directions.

Tetris

We considered the tetris domain, which is a popular computer game designed by Alexey Pajitnov in 1985. See [51] for more details. Firstly, we compared the performance of the full and diagonal approximate Newton methods to other parametric policy search methods. Tetris is typically played on a 20×10 grid, but due to computational costs we considered a 10×10 grid in the experiment. This results in a state space with roughly 7×2^{100} states. We modelled the policy through a Gibb's distribution, where we considered a feature vector with the following features: the heights of each column, the difference in heights between adjacent columns, the maximum height and the number of 'holes'. Under this policy it is not possible to obtain the explicit maximum over $\boldsymbol{w}$ in (2.14) and so a straightforward application of EM is not possible in this problem. We therefore compared the diagonal and full approximate Newton methods with steepest and natural gradient ascent.

We now detail the procedure used for each of the algorithms in this experiment. The same general procedure was used for all the algorithms considered in the experiments. We modelled the environment through with an infinite planning horizon with average rewards. The reward at each time-point is equal to the number of lines deleted. We used a recurrent state formulation [182] of the gradient of the average reward framework to perform the gradient evaluation. We used analogous versions of this recurrent state formulation for natural gradient ascent, the diagonal approximate Newton method and the full approximate Newton method. As in [83] we used the sample trajectories obtained during the gradient evaluation to estimate the Fisher information matrix. Irrespective of the policy a game of tetris is guaranteed to terminate after a finite number of turns, see *e.g.* [27]. As we are considering the average reward framework a new game starts when the previous one terminates. We used the empty board as a recurrent state where, because a new game starts with an empty board, this state is recurrent.[4] During each training iteration the (approximation of the) search direction was obtained by sampling 1000 games, where these games were sampled using the current policy parameters. Given the current approximate search direction the

[4]This is actually an approximation because it doesn't take into account that the state is given by the configuration of the board and the current piece, so this particular 'recurrent state' ignores the current piece. Empirically we found that this approximation gave better results, presumably due to reduced variance in the estimands, and there is no reason to believe that it is unfairly biasing the comparison between the various parametric policy search methods.



(a) Game Pieces (b) Game Board

Figure 4.3: A graphical illustration of the game of tetris with (a) the collection of possible pieces, or tetrozoids, of which there are seven (b) a possible configuration of the board, which in this example is of height 20 and width 10.

following basic line search method was used to obtain a step-size: For every step-size in a given finite set of step-sizes sample a set number of games and then return the step-size with the maximal score over these games. In practice, in order to reduce the susceptibility to random noise, we used the same simulator seed for each possible step-size in the set. To avoid over-fitting a different simulator seed was used during each training iteration. In this line search procedure we sampled 1000 games for each of the possible step-sizes. The same set of step-sizes was used in all of the different training algorithms considered in the experiment, where we used the set

$$\big\{0.1, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0, 128.0\big\}.$$

To reduce the amount of noise in the results we used the same set of simulator seeds in the search direction evaluation for each of the algorithms considered in the experiment. In particular, we generated a $n_{\text{experiments}} \times n_{\text{iterations}}$ matrix of simulator seeds, where $n_{\text{experiments}}$ was the number of repetitions of the experiment and $n_{\text{iterations}}$ was the number of training iterations in each experiment. We then used this one matrix in all of the different training algorithms, where the element in the j^{th} column and i^{th} row corresponds to the simulator seed used in the j^{th} training iteration of the i^{th} experiment. In a similar manner the set of simulator seeds used for the line search procedure was the same for all of the different training algorithms. Finally, to make the line search consistent among all of the different training algorithms the search direction was normlised and the resulting unit vector was the vector used in the line search procedure.

We ran 100 repetitions of the experiment, each consisting of 100 training iterations, and the mean and standard error of the results are given in fig(4.4). It can be seen that the full approximate Newton method outperforms all of the other methods, while the performance of the diagonal approximate Newton method is comparable to natural gradient ascent. We also ran several training runs of the full approximate Newton method on the full-sized 20×10 board and were able to obtain a score in the region of $14,000$ completed lines, which was obtained after roughly 40 training iterations. An approximate dynamic

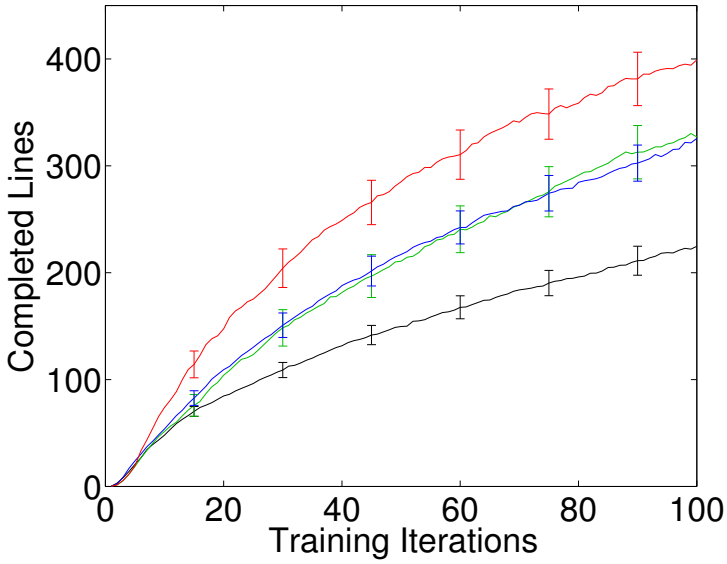


Figure 4.4: Results of the tetris problem for steepest gradient ascent (black), natural gradient ascent (green), the diagonal approximate Newton method (blue) and the approximate Newton method (red).

programming based method has previously been applied to the Tetris domain in [27]. The same set of features were used and a score of roughly $4,500$ completed lines was obtained after around 6 training iterations, after which the solution then deteriorated.

Linear System

In this section we perform various comparative experiments between the full approximate Newton method and other existing parametric policy search algorithms. We consider linear systems because they allow the search directions to be evaluated exactly, using methods described in section(3.2.2), but sufficiently difficult to provide a challenging platform for gradient-based optimisation methods. In all the experiments we shall consider in this section the policy takes the form

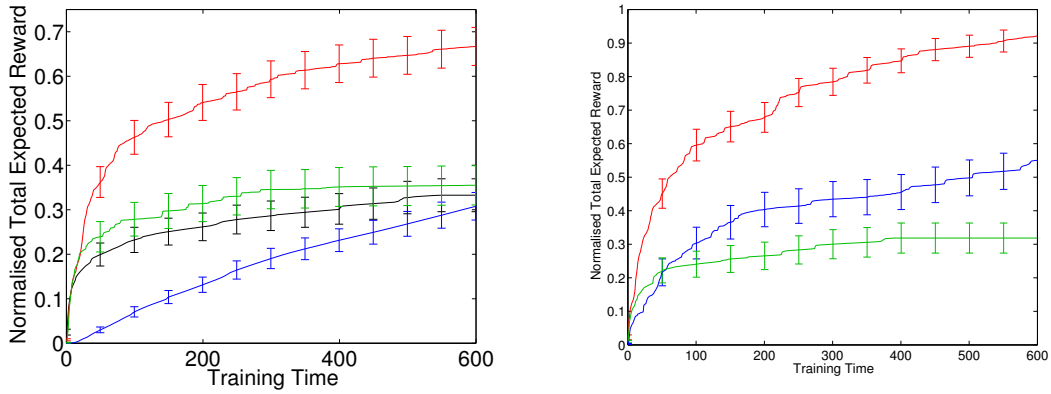
$$\pi(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) = \mathcal{N}(\boldsymbol{a}|K\boldsymbol{s}+\boldsymbol{m},\sigma^2 I),$$

where $\boldsymbol{w} = (K, \boldsymbol{m}, \sigma)$. The calculation of the derivative of the $\log$-policy *w.r.t.* the policy parameters can be performed in a straightforward manner. We consider finite horizon problems in this section. To calculate the search directions of steepest gradient ascent, natural gradient ascent, Expectation Maximisation and the full approximate Newton method it is necessary to calculate the first two moments of the RTS state-action value functions, $\{Q_t^{\text{rts}}\}_{t=1}^H$, which can be done using the methods described in section(3.2.2). To calculate the search directions of the Newton method and the method that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$ it is necessary to use the extensions of the RTS inference routines described in appendix(C). All of the experiments in this section were performed on the 3-link manipulator as detailed in section(3.4). In this system the maximal value of the objective function varied dramatically depending on the random initialisation of the system, where this initialisation was done as described in section(3.4). To account for the variation in the maximal value of the objective function, between the repetitions of the experiment,

the results of each experiment were normalised by the maximal value achieved between the algorithms considered in that experiment. Hence, all results display the normalised total expected reward, *i.e.* the percentage of reward received in comparison to the best results among the algorithms considered in the experiment.

The first experiment on the linear system was performed to compare the search/step-direction of the steepest gradient ascent, natural gradient ascent, Expectation Maximisation and the full approximate Newton method. The experiment was performed on a linear system to remove any issues of approximate inference obscuring the results. We performed the experiment on the 3-link manipulator as detailed in section(3.4), considering a finite horizon of $H = 100$ and using RTS-inference to perform the evaluation of the search direction. We used the `minFunc`[5] optimisation library in all of the gradient-based algorithms. We found that both the line search algorithm and the step-size initialisation had a significant effect on the performance of all the algorithms. We therefore tried various combinations of these settings for each algorithm and selected the one that gave the best performance. We tried bracketing line search algorithms with: *step-size halving*; *quadratic/cubic interpolation from new function values*; *cubic interpolation from new function and gradient values*; *step-size doubling and bisection*; *cubic interpolation/extrapolation with function and gradient values*. We tried the following step-size initialisations: *quadratic initialization using previous function value and new function value/gradient*; *twice the previous step-size*. To handle situations where the initial policy parameterisation was in a 'flat' area of the parameter space far from any optima we set the function and point toleration of `minFunc` to zero for all algorithms. We repeated each experiment 100 times and the results are shown in fig(4.5) where the mean and standard error are plotted for steepest gradient ascent (black), Expectation Maximisation (blue), the full approximate Newton method (red), natural gradient ascent (green). It can be observed that the full approximate Newton method significantly outperforms all of the comparison algorithms. In terms of steepest gradient ascent and natural gradient ascent this superior performance is explained by the superior estimate of the curvature of the objective function provided by the approximate Hessian. The step-direction of Expectation Maximisation is very similar to the search direction of the full approximate Newton method in this problem. In fact over the mean parameters they are the same because the log-policy is quadratic in the mean parameters. The difference in performance between the full approximate Newton method and Expectation Maximisation is explained by the tuning of the step-size in the full approximate Newton method, compared to the constant step-size of one in Expectation Maximisation. To observe the issues of poor scaling to the various optimisation algorithms we observed the number of iterations required by each algorithm. These counts are given in table(4.1) where it can be observed that steepest gradient ascent requires far more iterations than either natural gradient ascent or the full approximate Newton method, both of which require roughly the same amount of iterations. This validates that both natural gradient ascent and the full approximate Newton method are more robust to poor scaling than steepest gradient ascent. Finally, we note that as the gradient can be calculated exactly in this problem it is possible to apply quasi-Newton methods, such as l-bfgs, which perform well

[5] This software library is freely available at `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`.



(a) Line-Search Results I

(b) Line-Search Results II

Figure 4.5: Normalisaed total expected reward plotted against training time (in seconds) for the 3-link rigid manipulator using line search methods. (a) The plot shows the results for steepest gradient ascent (black), Expectation Maximisation (blue), the full approximate Newton method (red) and natural gradient ascent (green). (b) The plot shows the results for the full approximate Newton method (red), the methods that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$ (blue) and the Newton method (green).

in this particular problem. However, the aim of this experiment is to compare the search directions of the current stochastic parametric policy search algorithms which, as mentioned earlier, does not include quasi-Newton methods.

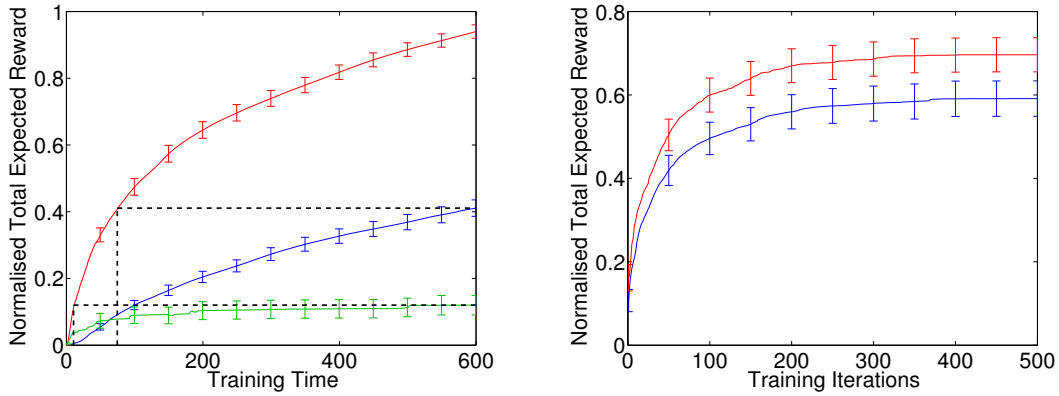
In the second experiment we compared the performance of the full approximate Newton method against the Newton method and the methods that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$. We performed the experiment on the 3-link manipulator as detailed in section(3.4), considering a finite horizon of $H = 100$ and using RTS-inference to perform the evaluation of the search direction. We repeated each experiment 100 times and the results are shown in fig(4.5) where the mean and standard error are plotted for the full approximate Newton method (red), the Newton method (green) and the method that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$ (blue). It can be seen that the Newton method performs poorly in comparison to the full approximate Newton method. This is probably best explained by the fact that the Hessian is not negative-definite over the entire parameter space, which means that it is necessary to correct the Hessian to ensure an ascent direction is obtained. These correction operations can destroy much of the curvature information in the Hessian, which is presumably what has occurred in this instance. This shows the benefit of the guarantee that $\mathcal{H}_2(\boldsymbol{w})$ is negative-definite, when the policy is $\log$-concave, which negates the necessity of correcting this preconditioning matrix. The full approximate Newton method also outperforms the method that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$, although this method obtains superior performance than the Newton method. We also compared the performance of the full approximate Newton method against the performance of the method that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$ in terms of the number of iterations, and found the performance of the two methods to be comparable. Note that in this system the full approximate Newton method requires the calculation of only the first two moments of the RTS state-action value functions, while the method that uses $\mathcal{M}(\boldsymbol{w}) = \mathcal{H}_{11}^{-1}(\boldsymbol{w})$ requires the first four moments of these functions. This additional computational cost explains the superior performance of the full approximate Newton method in this problem. It would be of interest to compare these two methods in a system where this additional com-

putational cost is not present, such as in the Tetris domain. It is also of interest to further understand the relationship between $\mathcal{H}_2(\boldsymbol{w})$ and $\mathcal{H}_{11}(\boldsymbol{w})$. These are both points of future research.

In the third experiment we again considered the 3-link manipulator described in section(3.4), but in this experiment we used a fixed step-size in steepest gradient ascent, natural gradient ascent and the full approximate Newton method. This experiment was performed to obtain a gauge on the difficulty of selecting a step-size sequence in the various methods where the step-size sequence is an open parameter. This is a difficult problem for algorithms such as steepest gradient ascent because the parameter space has a non-trivial number of dimensions and the objective is poorly-scaled. In both steepest gradient ascent and natural gradient ascent we considered the following fixed step-sizes 0.001, 0.01, 1, 10, 20, 30, 100 and 250. We were unable to obtain any reasonable results with steepest gradient ascent with any of these fixed step-sizes, for which reason the results are omitted. In natural gradient ascent we found 30 to be the best step-size of those considered. In the approximate Newton method we considered the following fixed step-sizes 10, 20, 30, 100 and 250 and found that the fixed step-size of 30 gave consistently good results without overstepping in the parameter space. The smaller step-sizes still obtained better results than Expectation Maximisation, but still less than the fixed step-size of 30. The larger step-sizes often found superior results, but would sometimes overstep in the parameter space. For these reasons we used the fixed step-size of 30 in the final experiment. We repeated the experiment 100 times and the results of the experiment are plotted in fig(4.6(a)), where the mean and standard error of the results are plotted. The results show that even though this step-size tuning is crude it is still possible to obtain strong results in comparison to Expectation Maximisation, which doesn't require the selection of a step-size sequence. In the experiment the approximate Newton method only took around 50 seconds to obtain the same performance as 300 seconds of training with Expectation Maximisation. Furthermore Expectation Maximisation was only able to obtain 40% of the performance of the approximate Newton method, while natural gradient ascent was only able to obtain around 15% of the performance. The reason that natural gradient ascent performed so poorly in this problem was because the initial control parameters were typically in a plateau region of the parameter space where the objective was close to zero. To get out of this plateau region on a regular basis and in the given amount of training time would require on overly large step-size. However, once in a high reward part of the parameter space we found that, using natural gradient ascent, these large step-sizes would result in overshooting in the parameter space and poor performance. The step-size of 30 was able to locate areas of high reward in a subset of the problems considered in the experiment, while not suffering from overshooting as much as the larger step-sizes. Clearly, as we have seen in the first experiment, it is possible to improve the performance of natural gradient ascent through tuning of the step-size sequence. However, the point of the experiment was to highlight the robustness of the approximate Newton method to poor scaling, as well as the resulting ease

	Steepest Gradient Ascent	Natural Gradient Ascent	Approximate Newton Method
Iterations	3684 ± 314	203 ± 34	310 ± 40

Table 4.1: Iteration counts of the 3-link manipulator experiment for steepest gradient ascent, natural gradient ascent and the approximate Newton method when using the `MinFunc` optimisation library.



(a) Fixed Step-Size Results

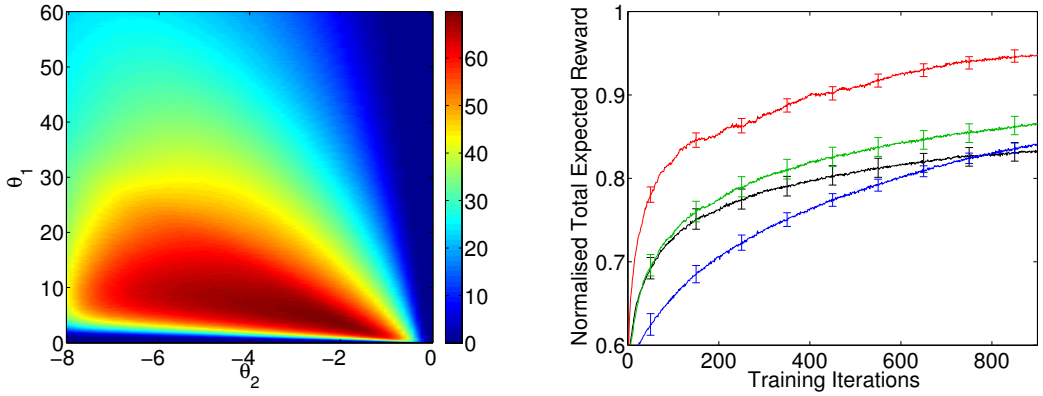
(b) Quasi-Newton Acceleration Experiment

Figure 4.6: (a) Normalised total expected reward plotted against training time (in seconds) for the 3-link rigid manipulator using a predefined step-size sequence. The plot shows the results for steepest gradient ascent (black), Expectation Maximisation (blue), the full approximate Newton method (red), natural gradient ascent (green). (b) The results of the quasi-Newton acceleration experiment. The l-bfgs method that uses with $\mathcal{B}_k^0(\boldsymbol{w}) = \boldsymbol{s}_k^\top \boldsymbol{y}_k / \boldsymbol{y}_k^\top \boldsymbol{y}_k$ is plotted in blue, while the l-bfgs method that uses $\mathcal{B}_k^0(\boldsymbol{w}) = \mathcal{H}_2^{-1}(\boldsymbol{w})$ is plotted in red.

(in comparison to algorithms such as natural gradient ascent) of selecting a good step-size sequence.

We also performed a simple experiment where we used our knowledge that the Hessian takes the form (4.2), and that $\mathcal{H}_2(\boldsymbol{w})$ is negative-semidefinite, in a quasi-Newton method. In particular we considered the *l-bfgs* method, which uses the last m iterates of the algorithm and an initial estimate of the inverse hessian to construct an approximation to the product of the inverse Hessian with the gradient vector. We denote this initial inverse Hessian by $\mathcal{B}_k^0$, where k is the current iteration. Using our knowledge of the Hessian we considered the initial estimate $\mathcal{B}_k^0 = \mathcal{H}_2^{-1}(\boldsymbol{w}_{k-m})$, where we used the approximate Hessian from the initial point for the first m iterations.[6] We performed the experiment on the 3-link manipulator described in section(3.4), considering a finite horizon of $H = 100$. In the experiment we used the last 5 iterates of the training algorithm. For comparison we considered the l-bfgs method where the initial estimate of the inverse Hessian was set to $\mathcal{B}_k^0 = \rho_k I$, where $\rho_k = \boldsymbol{s}_k^\top \boldsymbol{y}_k / \boldsymbol{y}_k^\top \boldsymbol{y}_k$, where $\boldsymbol{s}_k = \boldsymbol{w}_{k+1} - \boldsymbol{w}_k$ and $\boldsymbol{y}_k = \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_{k+1}} U(\boldsymbol{w}) - \nabla_{\boldsymbol{w}|\boldsymbol{w}=\boldsymbol{w}_k} U(\boldsymbol{w})$. This is a common choice in the l-bfgs method, see *e.g.* [121] for more details. We repeated the experiment 100 times and the results are shown in fig(4.6(b)), where the plot shows the mean and standard error of the results. It can be seen that the l-bfgs method that uses $\mathcal{B}_k^0 = \mathcal{H}_2^{-1}(\boldsymbol{w}_{k-m})$ performs consistently better than the l-bfgs method that uses $\mathcal{B}_k^0 = \rho_k I$. We note that this method may not work as well as m becomes larger and $\mathcal{H}_2(\boldsymbol{w}_{k-m})$ contains less information about the curvature of the objective function. This example is mainly meant to illustrate the possibility of using the approximate Hessian, $\mathcal{H}_2(\boldsymbol{w})$, in a quasi-Newton method. In practice it would be preferable to incorporate our knowledge about the structure of the Hessian into the actual update of the inverse Hessian vector product. Such a procedure has been successfully considered previously in problems of marginal log-likelihood maximisation [106] and non-linear least-squares [118, 121, 54].

[6] We considered using the inverse of the approximate Hessian from different iterates, *i.e.* $\mathcal{H}_2^{-1}(\boldsymbol{w}_i)$ for some $i \leq k$, but found that using $\mathcal{H}_2^{-1}(\boldsymbol{w}_{k-m})$ work best in the experiment we considered.



(a) Objective Function (b) Experiment Results

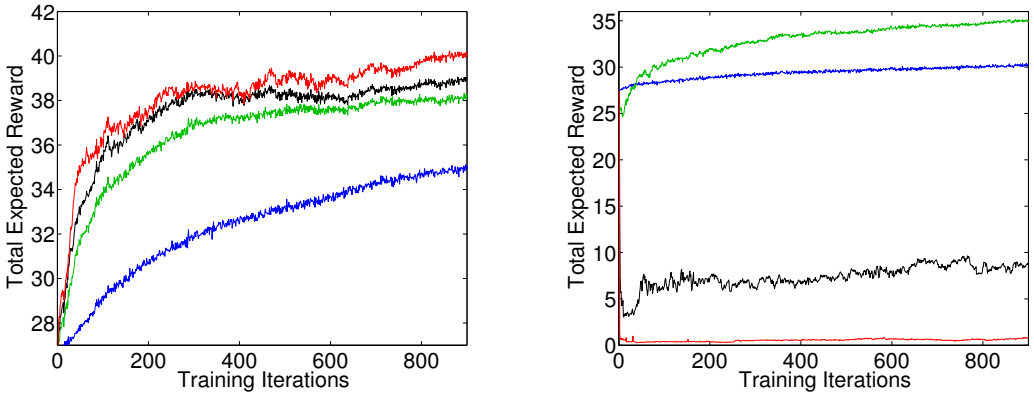
Figure 4.7: (a) A visual illustration of the objective function in the range $\boldsymbol{\theta} \in [0, 60] \times [-8, 0]$. (b) Results of the model-free non-linear experiment, where the plot shows the results for steepest gradient ascent (black), Expectation Maximisation (blue), natural gradient ascent (green) and the approximate Newton method (red). The plot shows the mean and standard error of the results.

4.3.1 Non-Linear System

Finally we performed an experiment on the synthetic two-dimensional non-linear MDP considered in [176]. The state-space of the problem is two-dimensional, $\boldsymbol{s} = (s^1, s^2)$, where s^1 is the agent's position and s^2 is the agent's velocity. The control is one-dimensional and the dynamics of the system is given as follows

$$s^1_{t+1} = s^1_t + \frac{1}{1 + e^{-u_t}} - 0.5 + \kappa,$$
$$s^2_{t+1} = s^2_t - 0.1 s^1_{t+1} + \kappa,$$

where κ is a zero-mean Gaussian random variable with standard deviation $\sigma_\kappa = 0.02$. The agent starts in the state $\boldsymbol{s} = (0, 1)$, with the addition of Gaussian noise with standard deviation 0.001, and the objective is to transport the agent to the state $(0, 0)$. We use the same policy as in [176], which is given by $a_t = (\boldsymbol{w} + \boldsymbol{\epsilon}_t)^\top \boldsymbol{s}_t$, where $\boldsymbol{w}$ are the control parameters and $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{\epsilon}_t; \mathbf{0}, \sigma_\epsilon^2 I)$. The objective function is non-trivial for $\boldsymbol{w} \in [0, 60] \times [-8, 0]$ and for illustration is plotted in fig(4.7(a)). In the experiment the initial control parameters were sampled from the region $\boldsymbol{w}_0 \in [0, 60] \times [-8, 0]$. In all algorithms 50 trajectories were sampled during each training iteration, which where then used to estimate the search direction. We considered a finite planning horizon with a planning horizon of $H = 80$. We repeated the experiment 100 times and the results of the experiment are given in fig(4.5), where the plot shows the results for steepest gradient ascent (black), Expectation Maximisation (blue), natural gradient ascent (green) and the approximate Newton method (red). The plot shows the mean and standard error of the results. The step-size sequences of steepest gradient ascent, natural gradient ascent and the approximate Newton method were all tuned for performance and the results shown were obtained from the best step-size sequence for each algorithm. As in the linear system the approximate Newton method consistently outperforms the other algorithms. It is interesting to observe that the difference in performance between



(a) Approximate Newton (b) Natural Gradient Ascent

Figure 4.8: Results of the approximate Newton method and natural gradient ascent on the synthetic two-dimensional non-linear system for various step-size sequences. In the approximate Newton method we considered step-size sequences of the form $\alpha_k = (1-k/N)\alpha + k/N$, where N is the number of training iterations considered in the experiment and $\alpha \in \mathbb{R}^+$. In natural gradient ascent we considered step-size sequences of the form $\alpha_k = \alpha/\sqrt{k}$ for varying values of $\alpha \in \mathbb{R}^+$. Plot (a) shows the results of the approximate Newton method for $\alpha = 1$ (blue), $\alpha = 6$ (green), $\alpha = 12$ (black) and $\alpha = 24$ (red). Plot (b) shows the results of natural gradient ascent for $\alpha = 0.0001$ (blue), $\alpha = 0.001$ (green), $\alpha = 0.01$ (black) and $\alpha = 0.1$ (red). In the case of natural gradient ascent the values $\alpha = 1$, 2 and 4 were also tried but we found to give similar results to $\alpha = 0.1$ and so are omitted.

the approximate Newton method and the other methods is not of the same magnitude in this system as it is in the linear system. A possible explanation for this point is that this problem is sufficiently small and well-scaled that it is a relatively easy problem in practice. Extending the approximate Newton method to more challenging non-linear systems is a point of future work and this will help clarify this point.

Finally, it is illustrative to detail the contrasting nature of the approximate Newton method and natural gradient ascent to step-size tuning in this problem.[7] Using the intuition that the approximate Newton method has a natural step-size of one, which corresponds to an EM-step in this problem because the log-policy is quadratic in the control parameters, we considered step-size sequences of the form $\alpha_k = (1 - \frac{k}{N})\alpha + \frac{k}{N}$, where N is the total number of training iterations considered and $\alpha \in \mathbb{R}^+$. In the experiment we considered the values $\alpha = 1$, 6, 12 and 24. The intuition used in this selection was that, provided that the steps were not so large as to cause overshooting in the parameter space, larger steps will increase performance. In natural gradient ascent it was necessary to obtain a gauge of a reasonable scale of a good step-size sequence. For this reason in the experiment we considered step-size sequences of the form $\alpha_k = \frac{\alpha}{\sqrt{k}}$ with $\alpha = 0.0001$, 0.001, 0.01, 0.1, 1, 2 and 4. The results of this step-size training are plotted in fig(4.8(a)) for the approximate Newton method and fig(4.8(b)) for natural gradient ascent. It was found that the sequence $\alpha = 24$ gave the best results for the approximate Newton method, while the sequence $\alpha = 0.001$ gave the best results for natural gradient ascent. The results show that the intuition about the approximate Newton method was correct and as a result the step-size sequence was easy to tune, while also giving reasonably consistent results over the sequences considered. These results also demonstrate the ease with which it is possible to obtain superior results than Expectation Maximisation,

[7] The tuning of the step-size sequence in steepest gradient ascent was similar in nature to natural gradient ascent and so is omitted from the discussion.

even though a step-size sequence is required in the approximate Newton method. In contrast a reasonable scale for the step-size sequence in natural gradient ascent was more difficult to find, while the algorithm was also more sensitive to the different step-size sequences considered.

4.4 Discussion

In this chapter we presented a novel analysis for the application of natural gradient ascent and Expectation Maximisation to the MDP objective. In particular we were able to show that both algorithms are closely related to the approximate Newton method that uses $\mathcal{H}_2(\boldsymbol{w})$ in place of $\mathcal{H}(\boldsymbol{w})$. Inspired by this analysis we then considered the direct application of this approximate Newton method to the MDP objective, where we found that this method has many desirable properties that are absent in the naive application of the Newton method. These include a guarantee that $\mathcal{H}_2(\boldsymbol{w})$ is negative-semidefinite provided that the controller is $\log$-concave, sparsity properties in $\mathcal{H}_2(\boldsymbol{w})$ that are not present in $\mathcal{H}(\boldsymbol{w})$ and more efficient evaluation. Initial empirical experiments suggest that the method has both strong performance and is robust to the selection of the step-size sequence. While the initial results are promising it would be ideal to test this method on more realistic real-world problems, such as robotics or large scale discrete MDPs. Additionally, we have only considered our approximate Newton methods in terms of an actor method and in future work we shall consider the actor-critic extension of these algorithm. Finally, it is a point of future work to better understand the relationship between $\mathcal{H}_2(\boldsymbol{w})$ and $\mathcal{H}_{11}(\boldsymbol{w})$, along with a deeper analysis of the Hessian itself.

Chapter 5

Dual Decomposition for Planning with Non-Markovian Policies

5.1 Introduction

It was noted in chapter(1) that one of the main restrictions of dynamic programming is the need for the planning model to have a Markovian policy, *i.e.* the conditioning set of the policy should form a separator set between the current action variable and the trajectory over the preceding time-points. This is an essential requirement in the derivation of dynamic programming and it fails to hold in many models of partially observable environments, such as DEC-MDPs or POMDPs modelled with a BC, MC or a FSC and finite horizon MDPs with a stationary policy. Different optimisation methods, such as the policy-search methods of chapter(2), are required in such models and this is a severe limitation in the applicability of dynamic programming. It is therefore of theoretical and practical interest to extend the applicability of dynamic programming by constructing planning algorithms that have dynamic programming as a core component of the optimisation process, not least because it is a global algorithm. This is the subject of the present chapter, where we focus on the problem of finite horizon MDPs with a stationary policy, detailing possible future directions for other partially observable models in the conclusion.

To approach this problem we use an optimisation technique known as *dual decomposition*, which originates from the convex optimisation literature, see *e.g.* [25, 35], and has been the centre of a recent surge of research in the approximate inference community, see *e.g.* [153, 97]. In these methods the original difficult global optimisation problem is relaxed into a series of easier local optimisations, the solutions of which are used to update the relaxation. An attractive property of these algorithms is that this relaxation provides a convex bound on the original objective function. This bound can be optimised through any number of convex optimisation techniques and this provides these techniques with desirable convergence properties. In the case of a finite horizon MDP with a non-stationary policy dynamic programming splits a difficult optimisation over $\mathcal{O}(A^{SH})$ (deterministic) policies into an easier problem over $\mathcal{O}(ASH)$ policies, with a run-time of $\mathcal{O}(AS^2H)$. When the policy is constrained to be stationary dynamic programming cannot be applied and no such computational saving is possible. We apply dual decomposition to the constrained stationary policy finite horizon MDP in such a manner that dynamic programming is applicable and the original difficult optimisation can be split into a series of easier opti-

misations. This relaxation has a natural interpretation as a non-stationary policy finite horizon MDP with a modified non-stationary reward function, where the modified reward function encourages stationarity in the policy.

The chapter shall be organised as follows: Section(5.2) will detail why Markovian policies are so essential to the derivation of dynamic programming and examine some commonly used partially observable environments where the policy is non-Markovian; Section(5.3) will contain an overview of dual decomposition methods; Section(5.4) will contain our construction of the dual decomposition algorithm for finite horizon MDPs with a stationary policy along with an analysis of the algorithm; Section(5.5) contains empirical experiments of our algorithm, where comparisons shall be made against policy-search methods; Finally in section(5.6) we shall give a discussion of the algorithm along with possible future avenues of research.

5.2 Markovian Policies & Dynamic Programming

A policy is said to be Markovian if the conditioning set of the policy forms a separator set between the current action variable and the trajectory over the preceding time-points. In the case of a MDP (with either a non-stationary policy in a finite planning horizon or a stationary policy in a infinite planning horizon) this property is satisfied because the policy is conditioned on the current state of the environment. Due to the form of the transition dynamics in a MDP the current state of the environment separates the current action from the previous states of the environment. Another model that satisfies this property is the POMDP, where the policy is conditioned on the current belief of the environment. As noted in chapter(1) Markovian policies are important because they allow the construction of a dynamic programming solution to the planning problem, which reduces an optimisation problem over an exponential number of policies[1] to a problem over a linear number of policies. Obviously in many cases of interest this linear complexity is still too costly to be feasible, nevertheless dynamic programming still greatly decreases the complexity of the problem while also providing the basis for many approximate solution techniques.

While a Markovian policy is essential for the construction of a dynamic programming solution there are many models where this property is not satisfied. Some typical examples are stationary policy finite horizon MDPs, transition independent decentralised MDPs and POMDPs with a policy modelled through either a finite state controller, memoryless controller or a blind controller. This list is not meant to be exhaustive but instead an illustration of how non-Markovian policies preclude dynamic programming. We now consider these examples in more detail.

Stationary Policy Finite Horizon Markov Decision Processes

The state of the environment in a finite horizon MDP is given by the current state and the current time-point. The stationary policy finite horizon MDP can therefore be seen as a partially observable problem as the agent is unaware of the current time-point. While a non-stationary policy finite horizon MDP can, in theory, be solved easily through dynamic programming this is not true when the policy is constrained

[1] We are considering the number of deterministic policies and the complexity is *w.r.t.* the conditioning set.

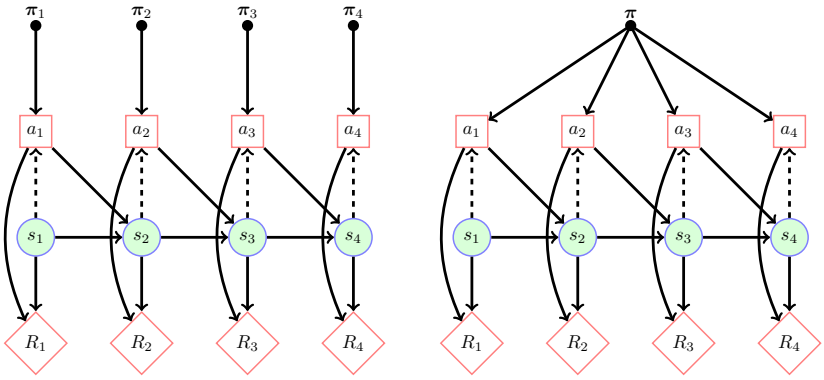


Figure 5.1: (a) An influence diagram representation of an unconstrained finite horizon ($H = 4$) MDP. Rewards depend on the state and action, $R_t(s_t, a_t)$. The policy $p(a_t|s_t, \pi_t)$ determines the decision and the environment is modeled by the transition $p(s_{t+1}|s_t, a_t)$. Based on a history of actions, states and reward, the task is maximize the expected summed rewards with respect to the policy $\boldsymbol{\pi}_{1:H}$. (b) For a stationary policy there is a single policy $\boldsymbol{\pi}$ that determines the actions for all time-points, which add a large clique to the influence diagram.

to be stationary. When the policy is constrained in this manner the finite horizon MDP objective takes the form

$$U(\pi) = \sum_{t=1}^{H} \sum_{a,s} R(a, s) p_t(a, s|\pi). \tag{5.1}$$

This can be seen by comparing the influence diagrams of a finite horizon MDP with a non-stationary policy, fig(5.1a), and with a stationary policy, fig(5.1b). A non-stationary policy means the influence diagram has a chain structure, which is easy to optimise, while a stationary policy causes the influence diagram to lose this chain structure. Indeed the stationary policy couples all time-points together, making the problem of finding the optimal policy π^* much more complex. The exact complexity of this problem class is still unknown: It's known to be P-hard and in NP, but its completeness results are still unknown [117].

Transition Independent Decentralised Markov Decision Processes

Consider a transition independent decentralised Markov Decision Process, with N agents and an infinite planning horizon. In this model the optimisation problem can be written in the form

$$\max_{\pi} \sum_{t=1}^{\infty} \sum_{\boldsymbol{a},\boldsymbol{s}} \gamma^{t-1} R(\boldsymbol{a}, \boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}) p_t(\boldsymbol{s}; \pi), \tag{5.2}$$

$$\textit{s.t. } \pi(\boldsymbol{a}|\boldsymbol{s}) = \prod_{n=1}^{N} \pi^n(a_n|s_n), \quad \forall (\boldsymbol{a}, \boldsymbol{s}) \in \mathcal{A} \times \mathcal{S}. \tag{5.3}$$

While the objective (5.2) has the same form as the objective on an ordinary MDP the constraint on the policy (5.3) means that a direct application of dynamic programming is not possible and will generally result in a policy that violates the conditional independence constraint on the policy. The conditional

independence constraint of the policy means that each agent is only aware of its own representation of the environment, making each agents' policy non-Markovian. It is the decentralised constraint of these models, *i.e.* the conditional independence structure of the policy, that makes the optimisation intractable, where these models are known to be NEXP-complete [23]. Other than direct application of gradient or EM based algorithms these models are typically solved through multi-linear programs [126], which are NP-hard to solve optimally but have been noted to have good initial performance.

Partially Observable Markov Decision Processes with a Finite State Controller

Consider an infinite horizon POMDP with a policy that is modelled with a FSC. For simplicity assume that the initial belief distribution and belief transition dynamics are given and fixed. In this case the optimisation problem is only *w.r.t.* π and takes the form

$$\max_{\pi} \sum_{t=1}^{\infty} \sum_{a,s,b,o} \gamma^{t-1} R(a,s) \pi(a|b,o) p_t(s,b,o;\pi). \tag{5.4}$$

It is clear that the policy $\pi(a|b,o)$ is non-Markovian as it does not depend on the state variable. If dynamic programming were to be applied then each possible combination of the conditioning variables would be considered in turn and the corresponding optimal action selected. As the state variable is not in this conditioning set it would instead be marginalised out and this would cause the reward function to depend on the marginal distribution of the state variable, which is unknown and itself depends on the policy. As a result the application of dynamic programming is not possible in this model. The cases of memoryless and blind controllers are similar with the exception that the actions are respectively based on the observation and the empty set, but the argument still holds and dynamic programming is again not possible. With the exception of blind deterministic controllers these planning problems are NP-hard and very difficult to solve in general, see *e.g.* [175] and references therein.

5.3 Dual Decomposition

In this section we shall a give a brief review of dual decomposition techniques, see *e.g.* [25, 35, 153, 97] for more details. The derivations in [25, 35, 153, 97] differ slightly and we follow the derivation from [97] as this is most similar to our derivation in section(5.4).

Suppose we are given a global objective function that takes the form of a summation over a finite number of *local* functions. The global objective is denoted by E and the local functions are denoted by $\{E_i\}_{i\in\mathcal{I}}$, for some finite index set $\mathcal{I}$. The domain of E is denoted by $\mathcal{X}$, which is generally assumed to be a high-dimensional space that is either continuous or discrete, while the domains of the local functions are subspaces of $\mathcal{X}$ and denoted by $\{\mathcal{X}_i\}_{i\in\mathcal{I}}$. Given a vector $\boldsymbol{x} \in \mathcal{X}$ the notation $\boldsymbol{x}_{|\mathcal{X}_i}$ is used to denote the sub-vector of $\boldsymbol{x}$ whose components are in $\mathcal{X}_i$[2]. In this notation the optimisation problem takes the

[2]We assume that the bases of the subspaces are subsets of the basis of $\mathcal{X}$ so that this definition of $\boldsymbol{x}_{|\mathcal{X}_i}$ corresponds to the projection of $\boldsymbol{x}$ into the subspace $\mathcal{X}_i$.

form

$$\max_{\boldsymbol{x}\in\mathcal{X}} E(\boldsymbol{x}) = \max_{\boldsymbol{x}\in\mathcal{X}} \sum_{i\in\mathcal{I}} E_i(\boldsymbol{x}_{|\mathcal{X}_i}). \tag{5.5}$$

It is assumed that the optimisation of any of these individual local functions is easy in comparison to the optimisation in (5.5). The property that makes the optimisation in (5.5) difficult is that the domains of the local functions will generally overlap, which couples the easier optimisations of the local functions into a difficult global optimisation. For example *maximum a posteriori* estimation in a pairwise Markov random field is NP-hard in general, even through the objective can be written in the form (5.5) and each local function is a function of at most two variables. An exception is when the spaces $\{\mathcal{X}_i\}_{i\in\mathcal{I}}$ are mutually disjoint, in which case (5.5) becomes

$$\max_{\boldsymbol{x}\in\mathcal{X}} E(\boldsymbol{x}) = \sum_{i\in\mathcal{I}} \max_{\boldsymbol{x}_{|\mathcal{X}_i}\in\mathcal{X}_i} E_i(\boldsymbol{x}_{|\mathcal{X}_i}). \tag{5.6}$$

In this case the local functions can be optimised independently, which makes the optimisation easy under the assumption that each of the local functions can be optimised with comparative ease. When (5.5) has the form (5.6) it said to be *separable*. It is the comparative ease of optimising separable objective functions that motivates dual decomposition techniques. In particular these methods relax the original optimisation problem in such a manner that the optimisation consists of solving a series of separable problems, where these separable problems are obtained by iterative tightening of the relaxation.

To obtain this relaxation a set of auxiliary variables $\{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}$ are introduced, where $\hat{\boldsymbol{x}}_i \in \mathcal{X}_i$ for each $i \in \mathcal{I}$, and (5.5) is written in the equivalent form

$$\max_{\boldsymbol{x},\{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}} \sum_{i\in\mathcal{I}} E_i(\hat{\boldsymbol{x}}_i),$$
$$\text{s.t. } \boldsymbol{x} \in \mathcal{X},\ \hat{\boldsymbol{x}}_i \in \mathcal{X}_i \text{ and } \boldsymbol{x}_{|\mathcal{X}_i} = \hat{\boldsymbol{x}}_i, \forall i \in \mathcal{I}.$$

This reformulation almost has the form of a separable problem with the exception of the consistency constraint, where $\boldsymbol{x}_{|\mathcal{X}_i}$ is constrained to agree with $\hat{\boldsymbol{x}}_i$ for each $i \in \mathcal{I}$. This constraint can be removed through *dual decomposition*, also known as *Lagrangian relaxation*, where these constraints are adjoined to the objective through the use of Lagrange multipliers, also known as *dual variables*. This adjoined objective is known as the Lagrangian and, denoting the dual variables by $\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}$, it takes the form

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}, \{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}) &= \sum_{i\in\mathcal{I}} E_i(\hat{\boldsymbol{x}}_i) + \sum_{i\in\mathcal{I}} \boldsymbol{\lambda}_i^\top \left(\hat{\boldsymbol{x}}_i - \boldsymbol{x}_{|\mathcal{X}_i}\right), \\ &= \sum_{i\in\mathcal{I}} \left(E_i(\hat{\boldsymbol{x}}_i) + \boldsymbol{\lambda}_i^\top \hat{\boldsymbol{x}}_i \right) - \boldsymbol{\lambda}^\top \boldsymbol{x}.\end{aligned}$$

where $\boldsymbol{\lambda}$ is a $n_{\boldsymbol{x}}$-component vector whose j^{th} component is given by the summation of the components of $\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}$ that occur in the j^{th} dimension of $\mathcal{X}$. The Lagrangian is separable in $(\boldsymbol{x}, \{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}})$ and it remains to relate the Lagrangian to the original optimisation problem. This is done through the *dual function*, denoted by g, which is a function of the dual variables. Its value is the maximum value of the

Lagrangian over $\left(\boldsymbol{x}, \{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}\right)$, *i.e.*

$$g(\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}) = \max_{\boldsymbol{x},\{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}} \sum_{i\in\mathcal{I}} \left(E_i(\hat{\boldsymbol{x}}_i) + \boldsymbol{\lambda}_i^\top \hat{\boldsymbol{x}}_i \right) + \boldsymbol{\lambda}^\top \boldsymbol{x}. \tag{5.7}$$

The dual function is always convex because it is the point-wise supremum of a family of affine functions in the dual variables, see *e.g.* [32], and when it is unbounded from above it takes the value ∞.

The dual function has the important property that it provides an upper bound on the optimal value of the original objective, see *e.g.* [32], so that

$$g(\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}) \geq \max_{\boldsymbol{x}\in\mathcal{X}} E(\boldsymbol{x}), \tag{5.8}$$

which holds for all values of the dual variables. The dual problem is then to find the tightest upper bound, *i.e.* to solve the minimisation problem $\min_{\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}} g(\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}})$. Given the dual variables, $\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}$, and a feasible primal solution, $\boldsymbol{x}$, the *duality gap* is defined to be the difference $g(\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}) - E(\boldsymbol{x})$. Due to the inequality (5.8) the duality gap is always non-negative and when it is equal to zero the problem (5.5) is said to have *strong duality*. It can be seen from (5.8) that when strong duality holds the global solution to the original optimisation problem has been obtained. When strong duality fails to hold it is necessary to obtain a primal solution from the dual function. There are heuristics to obtain a primal solution in this case, see *e.g.* [32, 153, 97], and we give two methods for obtaining a primal solution from our algorithm in section(5.4.5).

As the dual problem is to minimise the dual function it is possible to remove the term that is linear in $\boldsymbol{x}$ from (5.7). As the Lagrangian is linear *w.r.t.* $\boldsymbol{x}$ it is unbounded from above unless the dual variables satisfy the condition $\boldsymbol{\lambda} = \mathbf{0}$. We denote the set of dual variables where this condition is satisfied by $\Lambda = \{\{\boldsymbol{\lambda}_{i\in\mathcal{I}}\} | \boldsymbol{\lambda} = \mathbf{0}\}$. As the dual problem is to minimise the dual function and this function is infinite outside the set Λ it is sufficient to consider the domain Λ and the dual function takes the final form

$$g(\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}) = \max_{\{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}} \sum_{i\in\mathcal{I}} \left(E_i(\hat{\boldsymbol{x}}_i) + \boldsymbol{\lambda}_i^\top \hat{\boldsymbol{x}}_i \right). \tag{5.9}$$

The maximisation problem over $\{\hat{\boldsymbol{x}}_i\}_{i\in\mathcal{I}}$ in (5.9) is separable and so it is possible to write the dual function in the form $g(\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}) = \sum_{i\in\mathcal{I}} g_i(\boldsymbol{\lambda}_i)$, where

$$g_i(\boldsymbol{\lambda}_i) = \max_{\hat{\boldsymbol{x}}_i} \left(E_i(\hat{\boldsymbol{x}}_i) + \boldsymbol{\lambda}_i^\top \hat{\boldsymbol{x}}_i \right). \tag{5.10}$$

As the dual function is convex, even when (5.5) is not concave, the minimisation of the dual function can be performed by any number of convex optimisation techniques, such as gradient methods (when the dual function is differentiable), sub-gradient methods, projected (sub-)gradient methods or cutting-plane methods, see *e.g.* [25]. In summary the dual decomposition algorithm is a two stage iterative process, where these two stages can be summarised as follows:

Slave Problem Evaluate the dual function *w.r.t.* the current dual variables, $\{\boldsymbol{\lambda}_i\}_{i\in\mathcal{I}}$. This is equivalent

to solving the separable optimisation problem (5.9).

Master Problem Update the dual variables through some convex optimisation technique, such as gradient methods, sub-gradient methods or cutting-plane methods.

As we shall use a projected sub-gradient method in our dual decomposition algorithm we now briefly detail the method, see *e.g.* [34] for more details. To do so it is first necessary to introduce the notion of a sub-gradient, see *e.g.* [33], which is an extension of the notion of a gradient to non-differentiable functions. Given a convex function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ a sub-gradient of f at $\boldsymbol{x}$ is any vector $\boldsymbol{h} \in \mathbb{R}^n$ that satisfies the inequality

$$f(\boldsymbol{x}') - f(\boldsymbol{x}) \geq \boldsymbol{h}^\top \big(\boldsymbol{x}' - \boldsymbol{x}\big),$$

for all $\boldsymbol{x}'$ in the domain of f. This means that $\boldsymbol{h}$ is a sub-gradient of f at $\boldsymbol{x}$ if and only if $(\boldsymbol{h}, -1)$ specifies the supporting hyperplane to the epigraph of f at $(\boldsymbol{x}, f(\boldsymbol{x}))$. A graphical illustration of a sub-gradient of a function is given in fig(5.2). When f is differentiable at $\boldsymbol{x}$ there is a unique sub-gradient and it is equal to the gradient, otherwise there are multiple, perhaps infinitely many, sub-gradients.

In a projected sub-gradient method the dual variables are first updated by taking a step in the direction of a sub-gradient of $g_i(\boldsymbol{\lambda}_i)$, for each $i \in \mathcal{I}$. As this update will generally move the dual variables outside the set Λ it is necessary to project the dual variables back down into this set through some appropriately defined projection operator, denoted by $[\cdot]_\Lambda$. The final form of the update for the dual variables is given by

$$\boldsymbol{\lambda}_i \leftarrow [\boldsymbol{\lambda}_i + \alpha \nabla_{\boldsymbol{\lambda}_i} g_i(\boldsymbol{\lambda}_i)]_\Lambda,$$

where α is a positive step-size parameter and $\nabla_{\boldsymbol{\lambda}_i} g_i(\boldsymbol{\lambda}_i)$ denotes a sub-gradient. All that remains is to calculate the sub-gradient of the dual function. This final step is surprisingly easy and follows from the observation that if the optimum over the primal variables in (5.10) occurs at $\hat{\boldsymbol{x}}_i$ then for any $\boldsymbol{\lambda}_i'$ we have

$$\begin{aligned} g_i(\boldsymbol{\lambda}_i') &\geq \Big(E_i(\hat{\boldsymbol{x}}_i) + \big(\boldsymbol{\lambda}_i'\big)^\top \hat{\boldsymbol{x}}_i\Big), && \text{using (5.10),} \\ &= \Big(E_i(\hat{\boldsymbol{x}}_i) + \boldsymbol{\lambda}_i^\top \hat{\boldsymbol{x}}_i\Big) + \big(\boldsymbol{\lambda}_i' - \boldsymbol{\lambda}_i\big)^\top \hat{\boldsymbol{x}}_i, \\ &= g_i(\boldsymbol{\lambda}_i) + \big(\boldsymbol{\lambda}_i' - \boldsymbol{\lambda}_i\big)^\top \hat{\boldsymbol{x}}_i. \end{aligned}$$

This result states that a sub-gradient of a dual function is given by a optimum over the primal variables in (5.10), *i.e.* the sub-gradient is given by $\hat{\boldsymbol{x}}_i$. This is an important result in computational terms as it states that the sub-gradient of the dual function can be obtained at no computational cost once the dual functions have been evaluated.

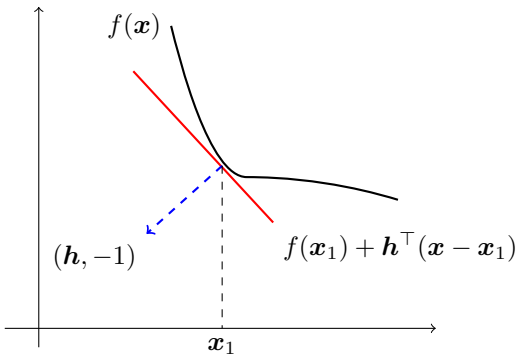


Figure 5.2: A graphical illustration of a sub-gradient of a function. The vector $(\boldsymbol{h}, -1)$ defines the supporting hyperplane to the epigraph of f at $\boldsymbol{x}_1$, given by $f(\boldsymbol{x}_1) + \boldsymbol{h}^\top(\boldsymbol{x} - \boldsymbol{x}_1)$.

5.4 Dual Decomposition of a Stationary Policy Finite Horizon Markov Decision Processes

It was noted in section(5.2) that the stationary policy constraint in a finite horizon MDP results in a highly connected influence diagram and as a result it is not possible to optimise these models through dynamic programming. On the other hand the (unconstrained) non-stationary policy finite horizon MDP is readily solved through dynamic programming, at least in theory. The method of dual decomposition relaxes a difficult global optimisation problem into a series of simpler problems, where these simpler problems are constructed iteratively through the minimisation of an upper bound of the objective function. From this perspective it is natural to apply the method of dual decomposition to stationary policy finite horizon MDPs in such a manner that the slave problems correspond to non-stationary policy finite horizon MDPs. As we shall see these unconstrained MDPs shall be similar to the original MDP model with the exception that the reward function shall take on a modified non-stationary structure. This modified reward function shall be updated during the master problem in such a manner that the policies are encouraged to be consistent over the entire planning horizon.

As noted in section(5.2) the stationary policy finite horizon objective is given by (5.1) so that the optimisation problem takes the form

$$\max_{\pi} U(\pi) = \max_{\pi} \sum_{t=1}^{H} \sum_{s,a} R(s,a) p_t(s,a|\pi). \tag{5.11}$$

As it is the stationarity constraint on the policy that causes the difficulty in the optimisation it is natural to relax this constraint. This is done by introducing a set of auxiliary variables, in this case the non-stationary policies, and rewrite (5.11) into the equivalent form

$$\max_{\pi, \pi_{1:H}} \sum_{t=1}^{H} \sum_{s,a} R(s,a) p_t(s,a|\pi_{1:t}), \tag{5.12}$$

$$\textit{s.t. } \pi^t = \pi, \forall t \in \mathbb{N}_H, \tag{5.13}$$

where $p_t(s, a|\pi_{1:t})$ is the state-action marginal of the non-stationary trajectory distribution. The difficulty now is to adjoin these constraints to the objective in such a manner that the slave problem is easy to solve, *i.e.* that the slave problem corresponds to a non-stationary policy finite horizon MDP. Due to the structure of the objective this is a more difficult problem than in the example considered in section(5.3). Indeed it shall be seen that a simple adjoining of the constraints through the use of Lagrange multipliers is insufficient in this problem.

In (5.11) and (5.12) we have omitted the distribution constraint on the policies, *i.e.* that the policy for each state is a distribution over the action space. In the case of $\pi_{1:H}$ these constraints shall be enforced during the optimisation of the slave problem. Given the distribution constraint on $\pi_{1:H}$ and the consistency constraint (5.13) the distribution constraint on π is redundant. We therefore consider π as unconstrained.

5.4.1 Naive Dual Decomposition

A naive procedure to enforce the consistency constraint between the policies is to construct the Lagrangian in the form

$$\mathcal{L}(\lambda_{1:H}, \pi_{1:H}, \pi) = \sum_{t=1}^{H} \sum_{s,a} \left\{ R(s,a) p_t(s,a|\pi_{1:t}) + \lambda_t(s,a) \left[\pi_t(a|s) - \pi(a|s) \right] \right\},$$

where the Lagrange multipliers are denoted by $\lambda_{1:H}$. However, a dynamic programming solution of the slave problem is not possible in this Lagrangian, which is easiest observed by considering the optimisation of the policy for the final time-point. Considering only the terms of $\mathcal{L}(\lambda_{1:H}, \pi_{1:H}, \pi)$ that depend upon π_H we have that the optimisation problem for this policy takes the form

$$\max_{\pi_H} \sum_{s,a} \left\{ R(s,a) \pi_H(a|s) p_H(s|\pi_{1:H-1}) + \lambda_H(s,a) \pi_H(a|s) \right\}, \tag{5.14}$$

where $p_H(s|\pi_{1:H}) \equiv p(s_H = s|\pi_{1:H})$. It is clear that while the first term in (5.14) depends on the policies of previous time-points the second term does not and therefore the optimisation is heavily dependent on the policies of previous time-points, making dynamic programming impossible. It is also clear that, whilst the constraints are linear in the policies, the marginal $p(s_t, a_t|\pi_{1:t})$ is non-linear and no simple linear program exists to find the optimal policy.

5.4.2 Dynamic Dual Decomposition

It is clear that the immediate application of dual decomposition methods to stationary policy finite horizon MDPs is not appropriate for our original aim, *i.e.* to obtain a simple optimisation algorithm for this model that is able to use dynamic programming as a core part of the optimisation process. The problem lies in adjoining the consistency constraints to the objective (5.12) in such a manner that dynamic programming is applicable. We now consider an alternative expression for these constraints that does result in a set of tractable slave problems.

Denoting the naive constraint functions used in section(5.4.1) by

$$g_t(a, s, \pi, \pi_{1:t}) = \pi_t(a|s) - \pi(a|s),$$

we now consider constraint functions of the form

$$h_t(a, s, \pi, \pi_{1:t}) = g_t(a, s, \pi, \pi_{1:t}) p_t(s|\pi_{1:t-1}).$$

Provided $p_t(s|\pi_{1:t-1}) > 0$, the zeros of the two sets of constraint functions, $g_{1:H}$ and $h_{1:H}$, are equivalent[3]. Adjoining the constraint functions $\boldsymbol{h}_{1:H}$ to the objective function (5.12) gives the Lagrangian

$$\mathcal{L}(\lambda_{1:H}, \pi_{1:H}, \pi) = \sum_{t=1}^{H} \sum_{s,a} \{(R_t(s, a) + \lambda_t(s, a))\, \pi_t(a|s) p_t(s|\pi_{1:t-1}) - \lambda_t(s, a)\pi(a|s) p_t(s|\pi_{1:t-1})\} \quad (5.15)$$

As in section(5.3) the Lagrangian is linear in π and, as π is unconstrained, the dual function is unbounded from above. A bounded dual function is obtained when $\pi_{1:H}$ and $\lambda_{1:H}$ satisfy the constraint

$$\sum_{t=1}^{H} \lambda_t(s, a) p_t(s|\pi_{1:t-1}) = 0, \quad \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \quad (5.16)$$

The dual function now takes the final form

$$g(\lambda_{1:H}) = \max_{\pi_{1:H}} \sum_{t=1}^{H} \sum_{s,a} \left\{ (R_t(s, a) + \lambda_t(s, a)) \pi_t(a|s) p_t(s|\pi_{1:t-1}) \right\}, \quad (5.17)$$

where the maximisation over $\pi_{1:H}$ is constrained through the distributional constraints of a policy, while $\pi_{1:H}, \lambda_{1:H}$ must satisfy (5.16). We have now obtained the dual decomposition of the original constrained MDP.

5.4.3 The Slave Problem

It can be seen from (5.17) that to evaluate the dual function it is necessary to optimise an unconstrained non-stationary policy MDP over $\pi_{1:H}$. Given the current set of dual variables, $\lambda_{1:H}$, this optimisation problem is given by

$$\max_{\pi_{1:H}} U_\lambda(\pi_{1:H}) = \max_{\pi_{1:H}} \sum_{t=1}^{H} \sum_{s,a} R_t^\lambda(a, s) \pi_t(a|s) p_t(s|\pi_{1:t-1}), \quad (5.18)$$

where the modified reward function, which depends on the dual variables, is given by

$$R_t^\lambda(a, s) = R_t(a, s) + \lambda_t(a, s). \quad (5.19)$$

[3] In the case that $p_t(s|\pi_{1:t-1}) = 0$, the policy $\pi_t(\cdot|s)$ is redundant since the state s is not visited.

This slave problem is readily solved in $O\left(AS^2H\right)$ time, using dynamic programming with the modified rewards R_t^λ.

5.4.4 The Master Problem

The master problem consists of updating the dual variables according to the convex optimisation routine that is being used to minimise the dual function. In general the dual function will be differentiable at a given point if the maximisation over the primal variables in (5.9) occurs at a unique point, see proposition 6.1.1 in [25]. In the case of our dual decomposition for stationary policy finite horizon MDPs this corresponds to (5.18) having a unique optimal policy[4]. There will be values of the dual variables where this will not hold and so a sub-gradient method, in particular a projected sub-gradient method, shall be implemented. At the i^{th} iterations we therefore update

$$\lambda_t^i = \lambda_t^{i-1} - \alpha_i \nabla_{\lambda_t} g(\lambda_{1:H}^{i-1}) \tag{5.20}$$

where $\nabla_{\lambda_t} g(\lambda_{1:H}^{i-1})$ is the sub-gradient that is obtained using the procedure detailed in section(5.3) and α_i is the step-size. The sub-gradient contains the positive state occupancy factor and we consider the simplified update equation

$$\lambda_t^i = \lambda_t^{i-1} - \alpha_i \pi_t^{i-1}.$$

Once the dual variables have been updated through this sub-gradient step they need to be projected down on to the feasible set, which is defined through the constraint (5.16). There are different ways to construct this projection, but we restrict our consideration to the projections of the form

$$\lambda_t^i(s,a) \leftarrow \lambda_t^i(s,a) - \sum_{\tau=1}^{H} \rho_\tau(s) \lambda_\tau^i(s,a). \tag{5.21}$$

where we define the state-dependent time distributions

$$\rho_\tau(s) \equiv \frac{p_\tau(s|\pi_{1:\tau-1})}{\sum_{\tau=1}^{H} p_\tau(s|\pi_{1:\tau-1})}. \tag{5.22}$$

One may verify that this setting ensures that λ_t^i satisfy the constraint (5.16).

5.4.5 Algorithm Overview

We now look at two important aspects of the dual decomposition algorithm, obtaining a primal solution from a dual solution and an interpretation of the dual variables.

Obtaining a Primal Solution

A standard issue with dual decomposition algorithms is obtaining a primal solution once the algorithm has terminated. When strong duality holds, *i.e.* the duality gap is zero, then $\pi = \pi_t \ \forall t \in \mathbb{N}_H$ and a

[4]It is simple to determine whether the optimal policy is unique when dynamic programming is being used to perform the optimisation. It is therefore immediately possible to determine if the dual function is differentiable at a given point.

Initialize the Lagrange multipliers $\lambda = 0$.
repeat
Evaluate Dual Function: Solve the finite horizon slave MDP with non-stationary rewards

$$R_t^{\lambda}(s,a) = \lambda_t^i(s,a) + R_t(s,a),$$

using dynamic programming to obtain the optimal non-stationary policies $\pi_{1:H}$.
Sub-Gradient Step: Update the Lagrange multipliers through a gradient step:

$$\lambda_t^{i+1} = \lambda_t^i - \alpha_i \pi_t^i.$$

Projection Step: Project the Lagrange multipliers into the feasible set:

$$\lambda_t^{i+1} \leftarrow \lambda_t^{i+1} - \sum_{\tau=1}^{H} \rho_\tau \lambda_\tau^{i+1}.$$

until $g(\lambda_{1:H}) - U(\pi) < \epsilon$, for some convergence threshold ϵ.
Output a feasible primal solution $\pi(a|s)$.

Algorithm 5.1: Dual Decomposition Dynamic Programming

solution to the primal problem can be obtained from the dual solution. However, this will not generally be the case and we therefore need to specify a way to obtain a primal solution. We considered two approaches; In the first we take the mean of the non-stationary policies

$$\pi(a|s) = \frac{1}{H} \sum_{t=1}^{H} \pi_t(a|s), \tag{5.23}$$

while in the second we optimise the Lagrangian *w.r.t.* π to obtain the primal policy $\pi(a|s) = \delta_{a,a^*(s)}$, where

$$a^*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmin}} \sum_{t=1}^{H} \lambda_t(s,a) p_t(s|\pi_{1:t-1}), \tag{5.24}$$

and the λ_t are taken before projection.

A summary of the complete dynamic dual decomposition procedure is given in algorithm(5.1).

Interpretation of the Dual Variables

To help get an understanding of the dual decomposition algorithm we now have a look at the role of the dual variables. We noted earlier that each of the slave problems is an unconstrained MDP problem with non-stationary rewards given by (5.19). So we can immediately see that the dual variables $\lambda_t(a,s)$ either encourages, or discourages, π_t to perform action a given state s, depending on sign$(\lambda_t(a,s))$. Now consider how the Lagrange multiplier $\lambda_t(s,a)$ gets updated at the i^{th} iteration of the dual decomposition algorithm. Prior to the projected sub-gradient step the update of $\lambda_t^i(s,a)$ takes the form

$$\lambda_t^i(s,a) = \lambda_t^{i-1}(s,a) - \alpha_i \pi_t^i(a|s),$$

where $\pi^i_{1:H}$ denotes the optimal non-stationary policy of the i^{th} round of slave problems. Noting that the optimal policy is deterministic gives

$$\lambda^i_t(s,a) = \begin{cases} \lambda^{i-1}_t(s,a) - \alpha_i, & \text{if } a = \underset{a}{\operatorname{argmax}}\ \pi^i_t(a|s), \\ \lambda^{i-1}_t(s,a), & \text{otherwise.} \end{cases}$$

Once the dual variables are projected down to the space of feasible variables through (5.21) we have

$$\lambda^i_t(s,a) = \begin{cases} \bar{\lambda}^{i-1}_t(s,a) + \alpha_i\big(\sum_{t\in N^i(a,s)} \rho_t - 1\big), & \text{if } a = \underset{a}{\operatorname{argmax}}\ \pi^i_t(a|s) \\ \bar{\lambda}^{i-1}_t(s,a) + \alpha_i \sum_{t\in N^i(a,s)} \rho_t & \text{otherwise} \end{cases}$$

where $N^i(a,s) = \big\{t \in \mathbb{N}_H | \pi^i_t(a|s) = 1\big\}$ is the set of time-points that action a was optimal in state s in the last round of slave problems. The notation $\bar{\lambda}^{i-1}_t$ means $\bar{\lambda}^{i-1}_t = \lambda^{i-1}_t - \big\langle \lambda^{i-1}_t \big\rangle_\rho$, where $\langle\cdot\rangle_\rho$ means taking the average *w.r.t.* to the distribution (5.22). Noting that ρ_t is a distribution over time means that the terms $\sum_{\tau\in N^i(a,s)} \rho_\tau$ and $\big(\sum_{\tau\in N^i(a,s)} \rho_\tau - 1\big)$ are positive and negative respectively. We can now see that the projected sub-gradient step either adds (or subtracts) a positive term to the projection, $\bar{\lambda}$, depending on the optimality of action a (given state s at time t) in the last slave problem. Thus there are two possibilities for the update of the dual variable; Either the action was optimal and a lower non-stationary reward is allocated to this state-action-time triple in the next slave problem, or conversely it was sub-optimal and a higher reward term is attached to this triple. We can now see that the master algorithm tries to readjust the dual variables so that (for each given state) the same action is optimal for all time-points, *i.e.* it encourages the non-stationary policies to take the same form. Additionally, as $|N^i(a,s)| \to H$ then $\sum_{\tau\in N^i_a(s)} \rho_\tau \to 1$, which means that a smaller/larger quantity is subtracted/added to the dual variable $\lambda^i_t(s,a)$ depending of whether or not $t \in N^i(a,s)$. This means that as $|N^i(a,s)| \to H$ the time-points $t \notin N^i(a,s)$ will have a larger positive term added to the reward for this state-action pair, making it more likely that this action will be optimal given this state in the next slave problem. Additionally, those time-points $t \in N^i(a,s)$ will have a smaller term subtracted from their reward, making it more likely that this action will remain optimal in the next slave problem. So we can see that the dual decomposition algorithm automatically weights the rewards according to a 'majority vote'. This type of behaviour is typical of dual decomposition algorithms, and is known as resource allocation via pricing [32].

5.5 Experiments

We made several empirical comparisons to the dual decomposition algorithm. Firstly, in section(5.5.1) we detail a comparison that was made with a well-known heuristic, commonly referred to as the *rolling-horizon policy*, that is often used in finite horizon MDPs with a stationary policy. Also, in section(5.5.2) we detail some comparisons that were performed using several of the policy search algorithms of chapter(2), in particular making comparisons with steepest gradient ascent and Expectation Maximisation.

5.5.1 Rolling-Horizon Comparison

A well-known heuristic that is often applied to finite horizon MDPs with a stationary policy is the so-called *rolling-horizons policy*. In this case dynamic programming is performed on the given finite horizon MDP, but with a non-stationary policy. Using this dynamic programming solution to the non-stationary policy finite horizon MDP a stationary policy is obtained by using the policy of the initial time-point. As this heuristic is commonly used we made a simple comparison between it and our dual decomposition algorithm. The experiment is a simple illustration of the heuristic nature of the rolling-horizon policy in comparison to the principled approach provided by dual decomposition.

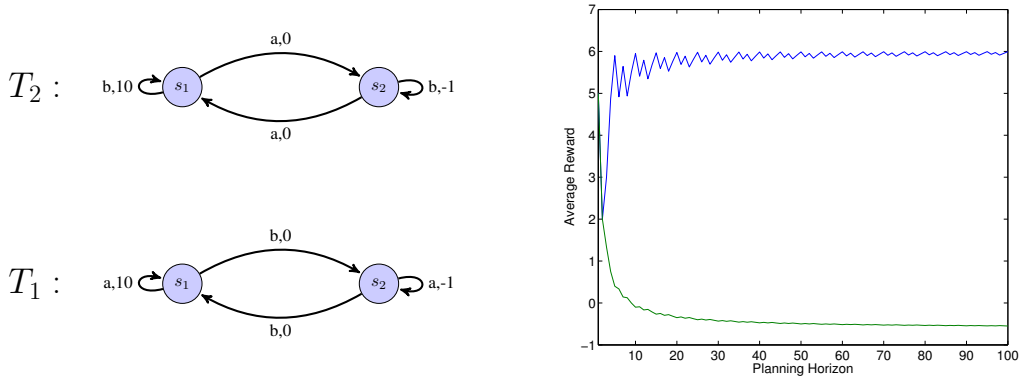
We considered the toy state MDP given fig(5.3(a)), which has periodic transition dynamics that alternate between two different transition matrices, T_1 and T_2, with a given periodicity. Under a non-stationary policy the optimal policy at a given time-point depends on the transition dynamics of that time-point. If the current transition dynamics are given by T_1 then it is optimal to select action 'a' in state s_1 and action 'b' in state s_2, while under transition dynamics T_2 it is optimal to select action 'b' in state s_1 and action 'a' in state s_2. In the experiment the sequence of transition matrices was given by one instance of T_2, which was then followed by four instances of T_1, at which point the sequence would repeat. In this problem the policy obtained from the rolling-horizon heuristic is determined by the transition dynamics of the initial time-point, which in this case is the transition matrix T_2. Considering the sequence of transition matrices, along with the form of the transition and reward matrices, it is to be expected that this rolling-horizon policy will perform poorly. We ran our dual decomposition algorithm and the rolling-horizon heuristic on a sequence of different planning horizons and for each different planning horizon calculated the average reward (per time-step) of the policy obtained from the two algorithms. The results are shown in fig(5.3(b)) where, as expected, the rolling-horizon algorithm performs poorly. In contrast the dual decomposition algorithm was able to obtain the global optimum in all instances, typically after only several iterations of the algorithm. This problem highlights the contrasting natures of the two algorithms: The rolling-horizon algorithm selects the policy solely on the fact that it is the policy of the initial time-point, while our dual decomposition algorithm obtains the policy in a principled manner.

5.5.2 Policy-Search Comparison

We also compared our dual decomposition algorithm against some policy search algorithms from chapter(2), where we considered several benchmark problems, including the *chain* problem [42], the *mountain car* problem [163] and the *puddle world* problem [161]. We now describe the algorithms used in the experiments, and the various parameter settings, before proceeding to the experiments.

Dual Decomposition Dynamic Programming (DD DP).

For the dual decomposition algorithm we used dynamic programming to solve the slave problems, with the overall algorithm is summarised in algorithm(5.1). In the master problem we used a predetermined sequence on step-sizes for the projected sub-gradient step. We experimented with



(a) Periodic Markov Decision Process

(b) Rolling-Horizon Results

Figure 5.3: (a) A graphical illustration of a two state Markov Decision Process with periodic transition dynamics. There are two different transition matrices, denoted by T_1 and T_2, and the transition dynamics of the Markov Decision Process alternates between these two transition matrices with a given periodicity. (b) Results of the rolling-horizon experiment where the plot shows the average reward (per time-point) plotted against the planning horizon, with the dual decomposition algorithm (blue) and the rolling-horizon algorithm (green).

several different step-size sequences and found that sequences of the form

$$\alpha_n = \frac{a}{n^b},$$

gave good results, where we set $a = 1$ and $b = 0.5$ in the chain and mountain car problems, while we set to $a = 0.05$ and $b = 1$ in the puddle world problem.

In the experiments we obtained the primal policy using (5.23). We also ran the experiments with the dual primal policy (5.24) and obtained results that were similar, or even better. We considered the algorithm converged when the duality gap was less than 0.0005.

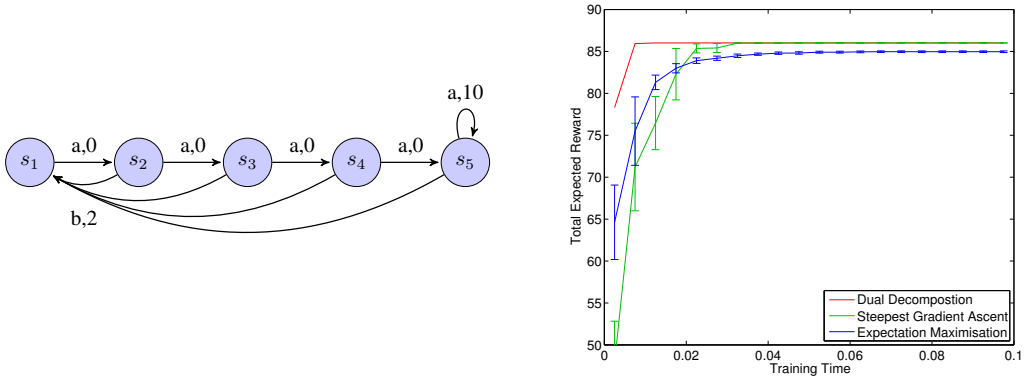
Steepest Gradient Ascent

The first algorithm used in the comparison was steepest gradient ascent, see chapter(2) for details. Steepest gradient ascent was considered because the problems were sufficiently small to allow the gradient to be calculated exactly and we found in chapter(2) that steepest gradient ascent performed well in such situations. We used the `minFunc` optimisation library[5] to perform the line search at each iteration. We set the optimisation method in `minFunc` to *steepest descent*[6]. All other settings of the `minFunc` library were left at the default setting. In the experiments we used a *soft-max* parameterisation of the table look-up policy, where the parameters were randomly initialised from the interval $[-1, 1]$.

Expectation Maximisation

The second algorithm we considered was Expectation Maximisation, see chapter(2) for details. This algorithm was considered as it is generally robust (in relation to other the algorithms from

[5]This software library is freely available at `http://www.di.ens.fr/~mschmidt/Software/minFunc.html`.
[6]We also tried *l-bfgs* but obtained similar results.



(a) Chain Problem

(b) Chain Results

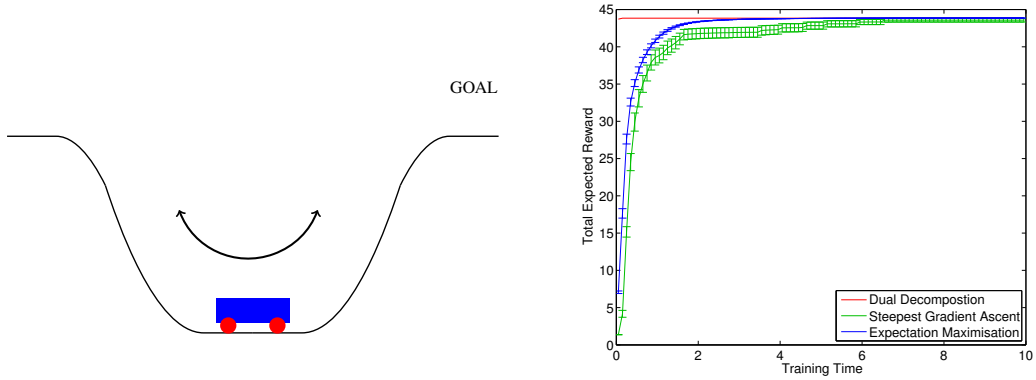
Figure 5.4: (a) A graphical illustration of the chain problem state-action transitions with rewards $R(s, a)$. The initial state is state 1. There are two actions a, b, with each action being flipped with probability 0.2. (b) Results from the chain experiment where the total expected reward is plotted against the run time, in seconds. The plot shows the results of the dual decomposition algorithm (red), steepest gradient ascent (green) and Expectation Maximisation (blue). Steepest gradient ascent and Expectation Maximisation were given 100 different random policy initialisations and the plot shows the mean and standard error of the results.

chapter(2)) to local optima. As with steepest gradient ascent we used a *soft-max* parameterisation of the table look-up policy, where the parameters were randomly initialised from the interval $[-1, 1]$.

Chain Problem

The first experiment we performed was the *chain* problem [42] which has 5 states each having 2 possible actions, as shown in fig(5.4(a)). The initial state is 1 and every action is flipped with 'slip' probability $p_{\text{slip}} = 0.2$, making the environment stochastic. If the agent is in state 5 it receives a reward of 10 for performing action 'a', otherwise it receives a reward of 2 for performing action 'b' regardless of the state. In the experiments we considered a planning horizon of $H = 25$ for which the optimal stationary policy is to travel down the chain towards state 5, which is achieved by always selecting action 'a'. While this is obviously a small problem it is an interesting benchmark because the gradient *w.r.t.* $w_{s_1,a}$ often points strongly in the direction of the sub-optimal behaviour of performing action b in state s_1.

The results of the experiment are shown in fig(5.4(b)) where the total expected reward is plotted against the run time, in seconds. We can see from fig(5.4(b)) that the dual decomposition algorithm outperforms both steepest gradient ascent and Expectation Maximisation. The dual decomposition algorithm converged after 4 iterations and strong duality was found to hold in this problem. Steepest gradient ascent had slower convergence than dual decomposition, but it did consistently converge to the global optimum after around 0.03 seconds. In terms of training iterations the EM-algorithm took 273.82 ± 3.07 iterations, where we counted iterations until the change in the total expected reward was less than 0.001. Steepest gradient ascent took 74.59 ± 3.66 training iterations and 142.15 ± 1.37 function evaluations. It appears that the EM-algorithm suffered from issues of local optima, but this is probably due to the poor convergence of the algorithm.



(a) Mountain Car (b) Mountain Car Results

Figure 5.5: (a) A graphical illustration of the mountain car problem. The agent (driver) starts the problem at the bottom of a valley in a stationary position. The aim of the agent is to get itself to the right most peak of the valley. (b) Results from the mountain car experiment where the total expected reward is plotted against training time, in seconds. The plot shows the results for the dual decomposition algorithm (red), steepest gradient ascent (green) and Expectation Maximisation (blue).

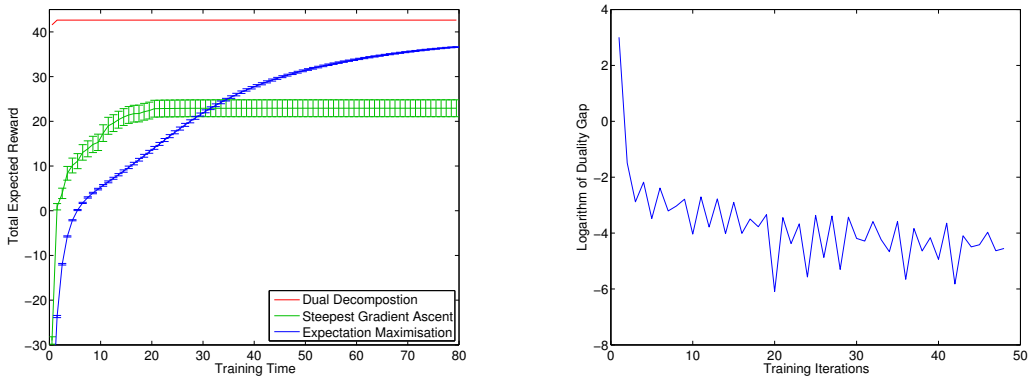
Mountain Car Problem

The second experiment we ran was a discretised version of the *mountain car* problem [163]. In the mountain car problem the agent is driving a car and its state is described by its current position (denoted by x) and its current velocity (denoted by v). The agent has three possible actions which are $\{-1, 0, 1\}$ which correspond to *reversing*, *stopping* and *accelerating*. The continuous dynamics are given by

$$v^{\text{new}} = v + 0.1a - 0.0028\cos(2x - 0.5),$$
$$x^{\text{new}} = x + v^{\text{new}},$$

where the ranges of the state space are restricted to the intervals $x \in [-1, 1]$ and $v \in [-0.5, 0.5]$. At the start of the problem the agent is in a stationary position, *i.e.* $v = 0$, and its position is $x = 0$. The problem is depicted graphically in figure(5.5(a)), where we can see that the agent starts in the bottom of a valley. The aim of the agent is to maneuver itself to the rightmost peak, we therefore set the reward of 1 when the agent is in the rightmost position and 0 otherwise. In the experiment we discretised the position and velocity ranges into bins of width 0.1, which resulted in a total state space of 231 states. We considered a planning horizon of $H = 50$ in the experiment.

The results of the experiment are shown in fig(5.5(b)) where the total expected reward is plotted against the run-time, in seconds, and we have plotted the mean and standard error of the results. Again the dual decomposition algorithm consistently outperforms both of the comparison algorithms. The dual decomposition algorithm converged in 3 iterations and strong duality was again found to hold in this problem. In terms of training iterations the EM-algorithm took 138.68 ± 1.60 iterations, where we counted iterations until the change in the total expected reward was less than 0.001. Steepest gradient ascent took 26.66 ± 0.44 training iterations and 125.11 ± 2.43 function evaluations.



(a) Total Expected Reward

(b) Duality Gap

Figure 5.6: (a) The results from the puddle world experiment where the total expected reward is plotted against training time, in seconds. The plot shows the results for the dual decomposition algorithm (red), steepest gradient ascent (green) and Expectation Maximisation (blue). (b) The duality gap of the dual decomposition algorithm during training, where the plot is given in terms of the training time.

Puddle World

The final experiment we performed was on a discretised version of the *puddle world* problem [161]. In this problem the state space is a continuous 2-dimensional grid $(x, y) \in [0, 1]^2$ that contains two puddles. In the experiment two circular puddles (of radius 0.1) were placed in the state space, where the centres of the puddles were generated uniform randomly over the grid $[0.2, 0.8]^2$. The agent in this problem is a robot that is depicted by a point mass in the state space. The aim of the robot is to navigate itself to a goal region, while avoiding areas of the state space that are covered in puddles. The initial state of the robot was set to the point $(0, 0)$. There are four discrete actions (*up*, *down*, *left* and *right*) that moves the robot 0.1 in that direction. The dynamics where made stochastic by adding the Gaussian noise $\mathcal{N}(0, 0.01)$ to each direction. A reward of 1 is received for all states in the goal region, which is given by those states satisfying $x + y \geq 1.9$. A negative reward of $-40(1 - d)$ is received for all states inside a puddle, where d is the distance from the centre of the puddle. In the experiment we discretised the x and y dimensions into bins of width 0.05, which gave a total of 441 states. In this problem we found the setting the planning horizon to $H = 50$ was sufficient to reach the goal region.

The results of the puddle world experiment are shown in fig(5.6a) where the total expected reward is plotted against the run time, in seconds. The dual decomposition algorithm converged after 49 iterations and strong duality was found upon termination of the algorithm. A plot of the duality gap of the dual decomposition algorithm is given in fig(5.6b). Steepest gradient ascent often got stuck in the local optima that corresponded to avoiding the puddles, neglecting the objective of reaching the goal region. After 80 seconds of training the EM-algorithm generally found superior solutions than steepest gradient ascent, but still inferior to the dual decomposition algorithm. This is due to the poor convergence properties of the EM-algorithm in this problem and performance similar to the dual decomposition algorithm was obtained given enough training time. A possible reason for the slow convergence of EM is that it is necessary to add a positive constant to the reward function to apply the algorithm. As noted in appendix(B)

this construction can have an adverse effect on the rate of convergence of Expectation Maximisation.

Summary

We have performed some initial comparisons with policy search methods on several benchmark problems. In the problems considered we found that the dual decomposition algorithm was consistently able to find the global optimum, typically in comparatively few iterations. In all of the three problem domains considered the initial iteration of the slave problem resulted in a non-stationary policy where there was already a large amount of agreement between the policies of separate time-points. Due to the 'majority vote', or *resource allocation via pricing*, behaviour of dual decomposition algorithms this large amount of agreement between the policies of the different time-points results in rapid convergence.

5.6 Discussion

In this chapter we have considered the problem of extending dynamic programming techniques to finite horizon MDPs with stationary policies. The complexity of this problem class is still an open problem, but a principled application of dynamic programming is non-trivial. We approached the problem using the technique of dual decomposition, which leads a two stage iterative solution method. The first stage consists of solving an unconstrained non-stationary policy finite horizon MDP with a modified reward function. The second stage consists of using the optimal policy obtained in the first stage to update this modified reward function. The algorithm encourages consistency between the non-stationary policies by appropriately modifying the reward function. This is a typical property of dual decomposition techniques and is generally referred to as *resource allocation via pricing*. Experiments indicate that the algorithm has excellent convergence properties, often converging to the optimum within a few iterations. This compared favourably against the policy search algorithms that we used as a comparison, namely steepest gradient ascent and Expectation Maximisation. We also presented an example where the so-called rolling-horizon policy performed very poorly, yet our dual decomposition algorithm was consistently able to locate the global optimum.

At present we have only considered stationary policy finite horizon MDPs with discrete state-action spaces. It would be desirable to extend the ideas of this chapter to more elaborate partially observable environments, such as DEC-MDPs and POMDPs modelled with either a BC, MC or a FSC. In the case of a DEC-MDP, as was observed in section(5.2), the objective can be written in the form of the constrained optimisation problem (5.2 & 5.3). Comparing this constrained objective with the constrained objective of a stationary policy finite horizon MDP (5.12 & 5.13) it can be seen that a similar dual decomposition should be possible. In the case of a POMDP with a FSC policy, for example, it is not immediately obvious that such an extension is possible as the objective (5.4) appears unconstrained. However, writing (5.4) in the equivalent form

$$\max_{\pi,\{\tilde{\pi}\}} \sum_{t=1}^{\infty} \sum_{a,s,b,o} \gamma^{t-1} R(a,s)\tilde{\pi}(a|b,s)p_t(s,b;\pi),$$

$$\textit{s.t. } \pi(a|b,o) = \tilde{\pi}(a|b,s), \quad \forall (a,b,o) \in \mathcal{A}\times\mathcal{B}\times\mathcal{O}, \quad \forall s \in \mathcal{S}_o,$$

where $\mathcal{S}_o$ is the set of states that cause observation o, it is clear that similar methods can now be applied, at least in theory. Such extensions would offer an interesting approach to optimising non-Markovian policies and is a possible area of future research.

Chapter 6

Variational Reinforcement Learning

6.1 Introduction

Thus far we have considered optimal control almost exclusively from the viewpoint of planning, where a model of the environment has been available to perform the necessary inference through either message-passing or sampling-based techniques. In message-passing methods the model is used directly to evaluate quantities of interest, such as the value function or the state-action occupancy distributions. By contrast sampling-based methods typically use the model through the use of a simulator of the environment, which is then used in place of the true environment to obtain samples[1]. In practice, however, it is generally the case that a model of the environment is not available and so a model needs to be constructed. One possible solution to this issue is to construct a model of the environment from any available information, such as samples of the transition dynamics or prior expert knowledge of the system. The disadvantage of this approach is that any errors in the model may have an adverse effect on the performance of any controller obtained from this model. An alternative approach is to construct a distribution over possible models of the environment, which is updated as more information about the system is obtained, and to optimise any controller over the entire distribution of models. By optimising the controller in this fashion some of the uncertainty about the model of the environment will be incorporated into the optimisation process.

This later approach is a form of *Bayesian Reinforcement Learning* (B-RL) and is the subject of the current chapter. Given the close connection between the optimisation of MDPs and maximisation of the marginal $\log$-likelihood of latent time-series models it is natural to extend the Bayesian approaches to these time-series models, see *e.g.* [112, 20, 16], to the B-RL framework. We take this approach in this chapter and in particular we construct an EM-algorithm for this Bayesian framework, where to construct this EM-algorithm we extend the methods of chapter(2) to the B-RL objective function. This objective function incorporates uncertainty in our knowledge of the environment by simultaneously considering the MDP objective for all models, where each of these MDP objectives is weighted by the probability of the model under the distribution over models. As the optimisation of the controller occurs over the entire distribution of transition models this EM-algorithm is intractable and approximate solutions are

[1] While it is possible to directly apply sampling-based methods to a given system (without the construction of a simulator) the number of samples required by these methods is prohibitive in practice. For example, real-world robotic systems are expensive and fragile and excessive use of the system through simulation-based optimisation is not feasible.

required. To overcome this intractability we propose three approximations that are based on approximate inference techniques, including a variational Bayes approximation [142, 21], Expectation Propagation [115] and a stochastic EM-algorithm.

This chapter shall be organised as follows: In section(6.2) we shall introduce some notation and also provide an overview of the standard formulation of Bayesian Reinforcement Learning, which differs slightly from our own; Section(6.3) shall contain the construction of the EM-algorithm for our formalism of Bayesian Reinforcement Learning; As this EM-algorithm will prove intractable we shall develop several approximate algorithms in section(6.4); In section(6.5) we shall provide an empirical evaluation our approximate algorithms and finally in section(6.6) we shall summarise the chapter.

6.2 Bayesian Reinforcement Learning

In the framework of Reinforcement Learning the transition dynamics are unknown and are instead treated as variables, denoted by $\boldsymbol{\theta} = \{\theta^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}}\}_{(\boldsymbol{s}',\boldsymbol{s},\boldsymbol{a})\in\mathcal{S}\times\mathcal{S}\times\mathcal{A}}$, where $\theta^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}} = p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})$. We denote the space of possible transition models by Θ. As the transition dynamics are unknown the only knowledge the agent has of the environment is gained through observed transitions, which are denoted collectively by $\mathcal{D} = \{(\boldsymbol{s}_n,\boldsymbol{a}_n) \to \boldsymbol{s}_{n+1}, n = 1,\ldots,N\}$. A typical approach it is to use a point-based estimate of the transition model, such as the maximum likelihood (ML) estimator, which is then used in any of a number of MDP planning algorithms. However, use of such an estimate fails to take into account any uncertainty in the knowledge of the transition dynamics and can adversely affect the overall policy solution, resulting in myopic behaviour that is unaware of other potentially beneficial parts of the state-action space. An alternative approach is to take a Bayesian perspective, where a distribution over transition models is maintained and the controller is optimised over this distribution. This later approach is a form of Bayesian Reinforcement Learning.

To construct a Bayesian framework for Reinforcement Learning it is necessary to construct a distribution over transition models. As $\boldsymbol{\theta}$ is a set of independent categorical distributions a natural conjugate prior $p(\boldsymbol{\theta})$ is the product of independent Dirichlet distributions, *i.e.*

$$p(\boldsymbol{\theta}) \sim \prod_{\boldsymbol{s},\boldsymbol{a}} \text{Dir}(\theta^{\cdot}_{\boldsymbol{s},\boldsymbol{a}}|\alpha^{\cdot}_{\boldsymbol{s},\boldsymbol{a}}) \tag{6.1}$$

where $\boldsymbol{\alpha}$ are hyper-parameters. Given the set of observed transitions the posterior of $\boldsymbol{\theta}$ is formed from Bayes' rule

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}), \tag{6.2}$$

which, due to the conjugacy of the prior, gives the posterior

$$p(\boldsymbol{\theta}|\mathcal{D}) = \prod_{\boldsymbol{s},\boldsymbol{a}} \text{Dir}(\theta^{\cdot}_{\boldsymbol{s},\boldsymbol{a}}|c^{\cdot}_{\boldsymbol{s},\boldsymbol{a}} + \alpha^{\cdot}_{\boldsymbol{s},\boldsymbol{a}}), \tag{6.3}$$

where c is the count of observed transitions:

$$c^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}} = \sum_{n=1}^{N} \mathbb{I}\left[\boldsymbol{s}_n = \boldsymbol{s}, \boldsymbol{a}_n = \boldsymbol{a}, \boldsymbol{s}_{n+1} = \boldsymbol{s}'\right]. \tag{6.4}$$

The Dirichlet distribution is used primarily because it makes the calculation of the posterior distribution tractable. It is possible to consider other, more expressive, priors over the transition dynamics but we do not do so in this work.

Given the prior over models there are several different ways to formulate a Bayesian Reinforcement Learning framework. We shall consider a different formulation in section(6.3) but there is a standard formulation that it is important to mention, not least because it will highlight the differences with the objective that we consider. The standard framework for B-RL, see *e.g.* [50], simultaneously tackles the problems of learning the environment (in an on-line manner) while also attempting to maximise its total expected reward. This is known generally as the exploration/exploitation dilemma and it requires a trade-off between exploratory behaviour, where previously unseen parts of the state space are explored, and exploitative behaviour that uses the current knowledge of the environment to gain rewards. As such in this standard formulation the agent is given a prior distribution at the initial time-point and this distribution is updated in an on-line manner as the agent transitions through the state space. Following standard notation in the remainder of this section we denote this distribution by $b(\boldsymbol{\theta})$ and refer to it as the *belief*, or *belief-state*[2]. Now the main idea behind the standard formulation of B-RL is to combine the state space of the original MDP with the space of possible beliefs to form a hyper-state space. Uncertainty in the knowledge of the environment is then incorporated into the optimisation by performing planning in this hyper-state space, *i.e.* over the space of states and possible transition dynamics. In particular a belief-augmented MDP (BAMDP) [50] is a MDP with a state space given by $\mathcal{S} \times \Theta$ and an action space that is given by $\mathcal{A}$. Assuming that the reward function of the MDP is known then the reward function of the BAMDP is the same as that of the MDP. Due to Bayes' rule the transition dynamics in the BAMDP are Markov and take the form

$$p(\boldsymbol{s}', b'(\boldsymbol{\theta})|\boldsymbol{s}, \boldsymbol{a}, b(\boldsymbol{\theta})) = \delta_{b'(\boldsymbol{\theta}), b^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}}(\boldsymbol{\theta})} \theta^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}},$$

where $b^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}}(\boldsymbol{\theta})$ denotes the update of the belief obtained from $b(\boldsymbol{\theta})$ when observing the transition $(\boldsymbol{s}, \boldsymbol{a}) \to \boldsymbol{s}'$. Due the Markovian structure of the BAMDP it is possible to obtain the following Bellman equation

$$V^*(\boldsymbol{s}, b(\boldsymbol{\theta})) = \max_{\boldsymbol{a} \in \mathcal{A}} \left\{ R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}, b(\boldsymbol{\theta})) V^*(\boldsymbol{s}', b^{\boldsymbol{s}'}_{\boldsymbol{s},\boldsymbol{a}}(\boldsymbol{\theta})) \right\},$$

which can then be used to solve the B-RL problem. Equivalently, it is possible to consider this formulation of the B-RL problem as a POMDP. In this case the transition dynamics are adjoined to the state of the true underlying MDP and, as these transition dynamics are unknown, this constitutes a POMDP. The

[2]This is in keeping with the POMDP interpretation of the B-RL problem.

construction of this POMDP follows in an analogous manner to that of the BAMDP, see *e.g.* [50, 129]. In this case the B-RL problem can be solved using POMDP solution techniques. Unfortunately, solving a BAMDP is intractable because the number of belief states is infinite, in the case of an infinite planning horizon, or grows exponentially *w.r.t.* the planning horizon in the case of a finite planning horizon.

As solving this standard formulation of B-RL is an intractable problem (for all but the smallest of environments) research in this area has focused on approximate solution techniques. We now give a brief overview of the literature in the area, which can be broadly categorised into four approaches: Direct model-based Bayesian Reinforcement Learning that apply POMDP solution methods to the B-RL POMDP [129]; Sampling approaches that sample MDPs from the posterior (at certain points during the run-time of the system) and then use these MDPs to perform some type of planning, see *e.g.* [158, 10, 48]; *Probably Approximately Correct* (PAC) approaches[3] that use large deviation bounds to construct modified reward functions that obtain a suitable trade-off between exploitative and exploratory behaviour, see *e.g.* [96, 157, 154]; Belief tree search approaches that use tree search algorithms to perform planning directly in the BAMDP, see *e.g.* [11, 46, 47, 74].

Perhaps the most direct approach to B-RL is through the POMDP formulation of the framework, and in particular through the application of POMDP value iteration methods. Such an approach was considered in [129] where point-based value iteration methods from the POMDP literature were used to perform approximate dynamic programming. While this solution is elegant it is also computationally intractable, bacause the complexity of maintaining the point-based estimate of the value function scales exponentially in the number of Bellman updates. To overcome this exponential scaling in complexity another level of approximation is introduced through the use of a projection mapping, which is used to project the value function after each of the Bellman updates.

In the sampling-based approach of [158] a posterior distribution over the transition dynamics is maintained throughout the course of the interaction with the system. At certain intervals, determined by some predefined switching criterion, a MDP is sampled from this posterior distribution. This MDP is then solved using a standard MDP planning algorithm, such as dynamic programming, and the agent then acts greedily according to the policy obtained from this planning algorithm. The agent acts in this manner until the switching criterion are again met and a new MDP is sampled from the posterior. This process is repeated until the system terminates. This algorithm is based on Thompson sampling [165], which, while intuitive, is a heuristic. Extensions of this algorithm have been considered in [10, 48], where now multiple MDPs are sampled and these multiple samples are used in a specially constructed dynamic programming algorithms. These works also also provide theoretical analysis for their algorithms, which was lacking in [158].

Two of the first instances of large deviation bounds being used to obtain PAC performance guarantees in Reinforcement Learning literature are the so called R_{max} and E^3 algorithms, see [36] and [88] respectively. Such approaches have since been extended to B-RL, see *e.g.* [96, 157, 154, 9]. From the algorithmic perspective these algorithms work by applying standard planning algorithms to a MDP

[3] See *e.g.* [8] for a brief introduction into the theory of probably approximately correct algorithms.

whose transition dynamics are set to the ML estimate, while the reward function is a modified version of the original reward function. This modified reward function is constructed in such a manner that a mixture of exploratory and exploitative behaviour is achieved. This modified reward function is typically obtained by adjoining a non-negative term to the reward of each state-action pair, where this term is usually a decreasing function in the number of observations of the state-action pair. While these techniques provide *probably approximate correct* performance guarantees, the bounds they provide are generally so loose in practice that the parameter settings obtained from the bound perform poorly and heuristic settings of these parameters are used instead.

Tree search algorithms, such as the *upper confidence bounds on trees* (UCT) [95] and *sparse sampling* [87], have recently emerged as a powerful set of methods for optimising MDPs. Instead of planning over the entire MDP, like standard planning algorithms such as dynamic programming, these methods plan from the current state. They work by expanding a search tree, with its root based at the current state, to the leaves of the tree, which may correspond to terminal states of the MDP, and propagating the value of the leaves back up the tree. The optimal action for the current state then corresponds to the optimal branch of the tree. As full expansion of the tree is generally too expensive to perform in practice these methods use various techniques to prune the tree, such as branch and bound techniques. These tree search methods have also been applied to the B-RL framework and they have the attractive property that they can be directly applied to the BAMDP, see *e.g.* [11, 46, 47], thus directly tackling the exploration/exploitation dilemma.

6.3 Variational Reinforcement Learning

In the BAMDP approach to B-RL the policy is dependent on the current belief, where the belief is considered as a variable that is updated on-line according to Bayes' rule. While in theory this approach may be optimal it is also highly intractable, with a complexity that scales exponentially in the planning horizon. Additionally, the BAMDP approach tackles a different problem than the one that is of interest to us and that was introduced in section(6.1). In particular the BAMDP approach considers the problem "What is the optimal control given the current, and possible future, knowledge of the environment?", which is another way of phrasing the exploration/exploitation dilemma. In contrast we are interested in the alternative problem "Given a distribution over models, which is constructed using previously observed transitions, how do you optimise a controller over this distribution?". As such in our framework we hold the belief fixed in the objective function and consider policies that are conditioned only upon the state. To incorporate the uncertainty of the transition dynamics into the agent's behaviour we optimise the total expected reward given the environmental data

$$U(\boldsymbol{w}|\mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}\Big[U(\boldsymbol{w}|\boldsymbol{\theta})\Big], \tag{6.5}$$

where $U(\boldsymbol{w}|\boldsymbol{\theta})$ is the total expected reward given the transition dynamics, $\boldsymbol{\theta}$, and policy parameters, $\boldsymbol{w}$. While less general than the standard Bayesian Reinforcement Learning paradigm, this problem is still intractable. For instance, it is not possible to apply dynamic programming because it is not possible to

simultaneously optimise the policy over all the possible transition dynamics of the MDP. In this chapter we shall tackle (our version) of Bayesian Reinforcement Learning through parametric policy search algorithms, where we shall construct various approximations to deal with the intractabilities that result in taking a Bayesian approach. It is possible to apply our framework to the on-line setting described in section(6.2), where, for instance, this could be done by choosing to optimise the policy at certain intervals during the run-time of the system.

Considering the close relation between the optimisation of $U(\boldsymbol{w}|\boldsymbol{\theta})$, for any given $\boldsymbol{\theta} \in \Theta$, and the maximisation of the marginal $\log$-likelihood of latent variable time-series models it is natural to consider extending Bayesian approaches to these time-series models, see *e.g.* [112, 20, 16], to the objective function (6.5). A typical approach in these methods is to construct a lower-bound on the marginal $\log$-likelihood, which is then optimised through EM. As the inference in the E-step is generally intractable, approximations, such as variational Bayes, are then used to perform approximate inference. In this section we do likewise and extend the derivation of chapter(2) to the Bayesian objective (6.5), where we shall detail some approximate inference algorithms in section(6.4). It is also possible to use the gradient-based techniques (described in depth in chapter(2)) but we do not take that approach here[4]. We shall, however, briefly detail the calculation of the gradient of (6.5) at the end of this section and discuss this approach further when we discuss the stochastic EM-algorithm in section(6.4.3).

The derivation of the EM-algorithm is similar to the MDP derivation given in chapter(2). Consider the following unnormalised distribution defined over state-action paths and times $t \in \mathbb{N}_H$,

$$\tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w}) = R(\boldsymbol{z}_t)p(\boldsymbol{z}_{1:t}|\boldsymbol{\theta}, \boldsymbol{w}) \tag{6.6}$$

where $p(\boldsymbol{z}_{1:t}|\boldsymbol{\theta}, \boldsymbol{w})$ is the trajectory distribution, up to time t, given the transition dynamics, $\boldsymbol{\theta}$. We now define a joint distribution over state-action trajectories, time-points and transition dynamics as follows

$$\hat{p}(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D}) = \frac{\tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})p(\boldsymbol{\theta}|\mathcal{D})}{U(\boldsymbol{w}|\mathcal{D})} \tag{6.7}$$

As in chapter(2) this distribution is properly normalised, which can be verified through use of (1.1) and (6.5). To obtain a lower-bound on the $\log$-objective we take the Kullback-Leibler divergence between a variational distribution $q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})$ and (6.7), which gives

$$\log U(\boldsymbol{w}|\mathcal{D}) \geq \mathcal{H}_{\text{entropy}}(q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})) + \mathbb{E}_{q(\boldsymbol{\theta})}\Big[\log p(\boldsymbol{\theta}|\mathcal{D})\Big] + \mathbb{E}_{q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})}\Big[\log \tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})\Big]. \tag{6.8}$$

As in chapter(2) an EM-algorithm for optimising $U(\boldsymbol{w}|\mathcal{D})$ is obtained from this lower-bound by coordinate-wise maximisation *w.r.t.* the variational distribution and the policy. This is summarised as follows:

E-step For fixed $\boldsymbol{w}_k$ find the best $q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})$ that maximises the *r.h.s.* of (6.8). For no constraint on $q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})$, this gives $q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}) = \hat{p}(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}|\boldsymbol{w}_k, \mathcal{D})$.

[4]This is for historical reasons. The theoretical content of this chapter precedes the work of preceding chapters. Given the strong performance of the approximate Newton method, for example, such alternatives could be preferable in practice.

M-step For fixed $q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})$ find the best $\boldsymbol{w}$ that maximises the *r.h.s.* of (6.8). This is equivalent to maximising the following 'energy' term *w.r.t.* $\boldsymbol{w}$,

$$\boldsymbol{w}_{k+1} = \underset{\boldsymbol{w} \in \mathcal{W}}{\operatorname{argmax}} \, \mathbb{E}_{q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})} \Big[\log \tilde{p}(\boldsymbol{z}_{1:t}, t | \boldsymbol{\theta}, \boldsymbol{w}) \Big].$$

As the policy is independent of the transition dynamics the update of the policy takes the simple form

$$\boldsymbol{w}_{k+1} = \underset{\boldsymbol{w} \in \mathcal{W}}{\operatorname{argmax}} \sum_{t=1}^{H} \sum_{\tau=1}^{t} \mathbb{E}_{q(\boldsymbol{z}, \tau, t)} \Big[\log p(\boldsymbol{a} | \boldsymbol{s}; \boldsymbol{w}) \Big]. \tag{6.9}$$

Calculating the policy update is now a matter of calculating the state-action marginals of the q-distribution from the previous E-step. If no functional restriction is placed on the q-distribution then it will take the form of (6.7), where $\boldsymbol{w}$ are the policy parameters of the previous M-step. However, examining the form of (6.7), the exact state-action marginals of this distribution are computationally intractable. This can be understood by first carrying out the integral over $\boldsymbol{\theta}$, which has the effect of coupling together all time slices of the path distribution $\hat{p}(\boldsymbol{z}_{1:t}, t; \boldsymbol{w})$.

It is clear that the gradient-based methods of chapter(2) can also be extended to the Bayesian Reinforcement Learning framework. For example, parameterising the policy *w.r.t.* the parameter $\boldsymbol{w} \in \mathcal{W}$ *s.t.* the objective is differentiable *w.r.t.* $\boldsymbol{w}$, then the gradient of (6.5) can be written in the form

$$\nabla_{\boldsymbol{w}} U(\boldsymbol{w} | \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\theta} | \mathcal{D})} \Big[\nabla_{\boldsymbol{w}} U(\boldsymbol{w} | \boldsymbol{\theta}) \Big]. \tag{6.10}$$

Hence the gradient $\nabla_{\boldsymbol{w}} U(\boldsymbol{w} | \mathcal{D})$ is equal to the expectation of model-based derivatives, $\nabla_{\boldsymbol{w}} U(\boldsymbol{w} | \boldsymbol{\theta})$, $\boldsymbol{\theta} \in \Theta$, where this average is taken over the posterior over transition models. As the policy does not depend on the transition dynamics the gradient $\nabla_{\boldsymbol{w}} U(\boldsymbol{w} | \boldsymbol{\theta})$, $\boldsymbol{\theta} \in \Theta$, can be calculated through direct application of the methods in chapter(2). As with the EM-algorithm the expectation over Θ is intractable and cannot be performed analytically.

6.4 Approximate Variational Reinforcement Learning

In this section we shall discuss three approaches for dealing with the intractability of calculating the state-action marginals of the variational distribution, which are required in the variational Reinforcement Learning algorithm. In particular we shall consider a variational Bayes approximation, an approximation based on Expectation Propagation and a simple stochastic EM-algorithm. In the variational Bayes approach a restricted functional form of the variational distribution is considered in the E-step. This restricted functional form is selected in such a manner that inference in the approximate variational distribution is tractable. In Expectation Propagation the marginals of the variational distribution are approximated directly using Expectation Propagation in an approximate message-passing scheme. In the stochastic EM-algorithm the conditional independence structure of the variational distribution is used to construct a simple sampling scheme.

Input: policy parameters $\boldsymbol{w}$, reward r, prior $\boldsymbol{\alpha}$ and transition counts c.
repeat
  For fixed policy parameters $\boldsymbol{w}$
  repeat
    Calculate the q-marginals (6.15) and (6.17).
  until Convergence of the marginals.
  Update the policy parameters according to (6.9).
until Convergence of the policy.

Algorithm 6.1: VB EM-Algorithm

6.4.1 Variational Bayes

In this section we define our variational Bayes approximation to the variational distribution, while detailing the corresponding variational Bayes EM-algorithm. Variational Bayes (VB), see *e.g.* [142, 21, 15, 12], is a general technique for performing approximate inference that stems from the statistical mechanics literature and which is commonly used in the EM-algorithm when the E-step is intractable. The technique is very general, often simple to calculate and usually provides an intuitive functional form for the approximate posterior, or variational distribution. The technique stems from the observation that the Kullback-Leibler divergence, $\mathrm{KL}(q||p)$, is non-negative regardless of the structure of q. Using this observation it is possible to retain the lower-bound in the EM-algorithm, whilst restricting the functional form of the variational distribution in such a manner that the inference in the E-step becomes tractable. The $\log$-objective is no longer guaranteed to increase during each iteration, but the lower-bound is still guaranteed to increase.

In the problem that we are considering the computational intractability occurs because of the coupling of the transition dynamics and the reward weighted state-action trajectories. Therefore a suitable restriction on the functional form of the variational distribution is the factorised approximation:

$$q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}) = q(\boldsymbol{z}_{1:t}, t)q(\boldsymbol{\theta}). \tag{6.11}$$

This approximation maintains the lower-bound in (6.8), which now takes the form

$$\begin{aligned}\log U(\boldsymbol{w}|\mathcal{D}) \geq \mathcal{H}_{\text{entropy}}(q(\boldsymbol{z}_{1:t}, t)) + \mathcal{H}_{\text{entropy}}(q(\boldsymbol{\theta})) \\ + \mathbb{E}_{q(\boldsymbol{\theta})}\Big[\log p(\boldsymbol{\theta}|\mathcal{D})\Big] + \mathbb{E}_{q(\boldsymbol{\theta})q(\boldsymbol{z}_{1:t},t)}\Big[\log \tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})\Big].\end{aligned} \tag{6.12}$$

Under the variational Bayes approximation the M-step is the same as in the original EM-algorithm, while the E-step consists of iteratively updating the approximate variational distribution until convergence. The E-step is obtained by maximising (6.12) *w.r.t.* the distributions $q(\boldsymbol{z}_{1:t}, t)$ and $q(\boldsymbol{\theta})$. Taking the functional derivative of (6.12) with respect to $q(\boldsymbol{z}_{1:t}, t)$ and $q(\boldsymbol{\theta})$, whilst holding the other fixed, gives the following update equations:

$$q(\boldsymbol{z}_{1:t}, t) \propto e^{\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})\right]}, \tag{6.13}$$

$$q(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\mathcal{D}) e^{\mathbb{E}_{q(\boldsymbol{z}_{1:t},t)}\left[\log \tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})\right]}. \tag{6.14}$$

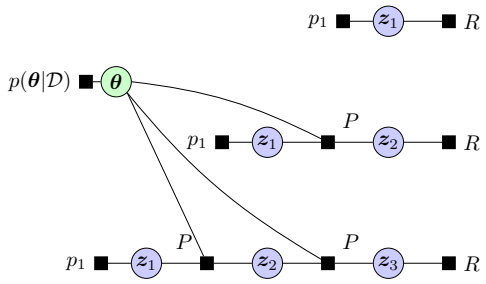


Figure 6.1: A factor graph representation of $q(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta})$ for state-action transition factors P and reward factors R, for a $H = 3$ horizon. The square nodes represent the various factors (functions) of the distribution and the circle nodes represent the variables. The initial time has no transition. The t^{th} chain is the t^{th} row of this diagram for fixed θ.

As (6.13) and (6.14) are coupled these equations need to be iterated until convergence.

Expansion of the term $\log \tilde{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})$ in (6.13) shows that $q(\boldsymbol{z}_{1:t}, t)$ can be written in the form

$$q(\boldsymbol{z}_{1:t}, t) \propto R(\boldsymbol{z}_t)\pi(\boldsymbol{a}_t|\boldsymbol{s}_t; \boldsymbol{w})\left\{\prod_{\tau=1}^{t-1} e^{\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \theta_{\boldsymbol{s}_{\tau+1}, \boldsymbol{s}_\tau, \boldsymbol{a}_\tau}\right]} \pi(\boldsymbol{a}_\tau|\boldsymbol{s}_\tau; \boldsymbol{w})\right\} p(\boldsymbol{s}_1). \tag{6.15}$$

This has the same form as the reward weighted trajectory distribution for a MDP with the transition dynamics, $\boldsymbol{\theta}$, replaced with unnormalised transitions

$$\tilde{\theta}(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a}) \equiv e^{\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \theta_{\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a}}\right]}. \tag{6.16}$$

The averages of $\log \theta$ in the exponent can be computed using digamma functions. Given $q(\boldsymbol{\theta})$, the marginals $q(\boldsymbol{z}, \tau, t)$ can then be calculated using the message-passing techniques detailed in chapter(2). A similar calculation for the parameters of the transition dynamics gives the update equation

$$q(\boldsymbol{\theta}) \propto p(\boldsymbol{\theta}|\boldsymbol{\alpha}, \mathcal{D}) e^{\sum_{t=1}^{H}\sum_{\tau=1}^{t-1} \mathbb{E}_{q(\boldsymbol{z}_{1:t}, t)}\left[\log \theta_{\boldsymbol{s}_{\tau+1}, \boldsymbol{s}_\tau, \boldsymbol{a}_\tau}\right]}$$

The summation of the states and actions in the exponent means that we may write $q(\boldsymbol{\theta})$ in the form

$$q(\boldsymbol{\theta}) = \prod_{\boldsymbol{s}, \boldsymbol{a}} \mathrm{Dir}\left(\theta_{\boldsymbol{s}, \boldsymbol{a}}^{\cdot} | \alpha_{\boldsymbol{s}, \boldsymbol{a}}^{\cdot} + c_{\boldsymbol{s}, \boldsymbol{a}}^{\cdot} + \tilde{r}_{\boldsymbol{s}, \boldsymbol{a}}^{\cdot}\right), \tag{6.17}$$

where

$$\tilde{r}_{\boldsymbol{s}, \boldsymbol{a}}^{\boldsymbol{s}'} = \sum_{t=1}^{H}\sum_{\tau=1}^{t-1} q(\boldsymbol{s}_{\tau+1} = \boldsymbol{s}', \boldsymbol{s}_\tau = \boldsymbol{s}, \boldsymbol{a}_\tau = \boldsymbol{a}, t). \tag{6.18}$$

Equation (6.17) has an intuitive interpretation: for each triple $(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a})$ we have the prior $\alpha_{\boldsymbol{s}, \boldsymbol{a}}^{\boldsymbol{s}'}$ term and the observed counts $c_{\boldsymbol{s}, \boldsymbol{a}}^{\boldsymbol{s}'}$ which deal with the posterior of the transitions. The term $\tilde{r}_{\boldsymbol{s}, \boldsymbol{a}}^{\boldsymbol{s}'}$ encodes an approximate expected reward obtained from starting in state $\boldsymbol{s}$, taking action $\boldsymbol{a}$, entering state $\boldsymbol{s}'$ and then following π afterwards. The posterior $q(\boldsymbol{\theta})$ is therefore a standard Dirichlet posterior on transitions but biased towards transitions that are likely to lead to higher expected reward. Under the approximation (6.11) the E-step consists of calculating the distributions (6.15) and (6.17). As these distributions

are coupled we need to iterate them until convergence. A summary of VB EM-algorithm is given in algorithm(6.1).

6.4.2 Expectation Propagation

In order to implement the Variational Reinforcement Learning approach of section(6.3) we require the state-action marginals of the intractable distribution $\hat{p}(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D})$. As an alternative to the variational Bayes factorised approach we here consider an approximate message passing (AMP) approach that approximates the required marginals directly.

The graphical structure of $\hat{p}(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D})$ is loopy but sparse, see fig(6.1) for a factor representation, so that a sum-product algorithm [100] may provide reasonable approximate marginals. The messages for the factor graph version of the sum-product algorithm take the following form.

$$\mu_{\boldsymbol{x}\to f}(\boldsymbol{x}) = \prod_{h\in n(\boldsymbol{x})\setminus\{f\}} \mu_{h\to\boldsymbol{x}}(\boldsymbol{x}) \tag{6.19}$$

$$\mu_{f\to\boldsymbol{x}} = \sum_{\sim\{\boldsymbol{x}\}} f(\boldsymbol{X}) \prod_{\boldsymbol{y}\in n(f)\setminus\{\boldsymbol{x}\}} \mu_{\boldsymbol{y}\to f}(\boldsymbol{y}) \tag{6.20}$$

where $\sum_{\sim\{\boldsymbol{x}\}}$ means the sum over all variables except $\boldsymbol{x}$, $n(\cdot)$ is the set of neighbouring nodes and $\boldsymbol{X}$ are the variables of the factor f. At convergence the singleton marginals are approximated by

$$p(\boldsymbol{x}) = \prod_{f\in F_{\boldsymbol{x}}} \mu_{f\to\boldsymbol{x}}(\boldsymbol{x}) \tag{6.21}$$

where $F_{\boldsymbol{x}}$ means the set of functions in the factor graph that depend on $\boldsymbol{x}$. Given that when $\boldsymbol{\theta}$ is held fixed the components of $\hat{p}(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D})$ are chain structured there is a natural message-passing scheme that alternates between the following two step: Firstly, holding the messages $\{\mu^{t,\tau}_{\boldsymbol{\theta}\to T}\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$ fixed, perform message-passing along the components of $\hat{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})$ using the messages $\{\mu^{t,\tau}_{\boldsymbol{\theta}\to T}\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$ in place of the transition dynamics; Secondly, using the messages calculated in the first stage of the message-passing scheme, $\{\mu^{t,\tau}_{T\to\boldsymbol{\theta}}\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$, update the messages $\{\mu^{t,\tau}_{\boldsymbol{\theta}\to T}\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$.

Under the current formulation this message-passing procedure is intractable, which is because of the intractabilities of the messages $\{\mu^{t,\tau}_{\boldsymbol{\theta}\to T}\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$. To see this observe that, using (6.19), we have that $\mu_{\boldsymbol{\theta}\to T}(\boldsymbol{\theta})$ takes the form,

$$\mu_{\boldsymbol{\theta}\to T}(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D}) \prod_{T'\neq T} \mu_{T'\to\boldsymbol{\theta}}(\boldsymbol{\theta}), \tag{6.22}$$

where we drop time indices for notational simplicity and $\mu_{T'\to\boldsymbol{\theta}}(\boldsymbol{\theta})$ is given by

$$\mu_{T'\to\boldsymbol{\theta}}(\boldsymbol{\theta}) = \sum_{\mathsf{s}',\mathsf{a},\mathsf{s}} \mu_{\boldsymbol{z}\to T'}(\mathsf{s},\mathsf{a})\mu_{s'\to T'}(\mathsf{s}')\theta^{\mathsf{s}'}_{\mathsf{s},\mathsf{a}}. \tag{6.23}$$

In order to maintain the tractability of this message-passing scheme it is necessary that the messages $\{\mu^{t,\tau}_{\boldsymbol{\theta}\to T}(\boldsymbol{\theta})\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$ be a product of independent Dirichlet's. However, it can be seen from (6.22) and (6.23) that $\mu_{\boldsymbol{\theta}\to T}(\boldsymbol{\theta})$ is a mixture of Dirichlet's, where the number of mixtures is exponential in the

repeat
  for $t = 1$ **to** H **do**
    Perform message-passing along the t^{th} component of the reward weighted trajectory distribution, $q(\boldsymbol{z}_{1:t}, t)$, holding all the messages $\mu_{\boldsymbol{\theta}\to T}(\boldsymbol{\theta})$ fixed.
  end for
  Perform Expectation-Propagation to obtain $q(\boldsymbol{\theta})$, and then update the messages $\{\mu^{t,\tau}_{T\to\boldsymbol{\theta}}(\boldsymbol{\theta})\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$.
until Convergence of the state-action marginals.

Algorithm 6.2: Approximate Message-Passing Schedule

planning horizon H. This makes messages of the form

$$\mu_{T\to s'}(\mathsf{s}') = \int d\boldsymbol{\theta} \sum_{\mathsf{s},\mathsf{a}} \mu_{z\to T}(\mathsf{s},\mathsf{a})\mu_{\boldsymbol{\theta}\to T}(\boldsymbol{\theta})\theta^{\mathsf{s}'}_{\mathsf{a},\mathsf{s}}.$$

computationally intractable. Following the general approach outlined in [115] to make a tractable approximate implantation we therefore project the marginal $\tilde{q}(\boldsymbol{\theta})$ to a product of independent Dirichlet's by moment matching. Given the projection $\tilde{q}(\boldsymbol{\theta})$ it is then necessary to obtain the messages $\{\mu^{t,\tau}_{\boldsymbol{\theta}\to T}\}_{t\in\mathbb{N}_H,\tau\in\mathbb{N}_t}$. One possibility is to use (6.20) and (6.21) to obtain the approximate message

$$\tilde{\mu}_{T\to\boldsymbol{\theta}}(\boldsymbol{\theta}) = \frac{\tilde{q}(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D})\prod_{T'\neq T}\tilde{\mu}_{T'\to\boldsymbol{\theta}}(\boldsymbol{\theta})}. \tag{6.24}$$

In practice (6.24) can lead to improper distributions and so in the experiments we set these messages to the mean of $\tilde{q}(\boldsymbol{\theta})$. The message passing schedule used in the experiments is outlined in algorithm(6.2).

6.4.3 Stochastic Expectation Maximisation

Thus far we have considered deterministic approximations to the variational distribution and in this section we introduce a simple stochastic approximation. As the policy update takes the simple form (6.9) it can written in the following equivalent form

$$\begin{aligned}\boldsymbol{w}_{k+1} &= \underset{\boldsymbol{w}\in\mathcal{W}}{\operatorname{argmax}} \sum_{t=1}^{H}\sum_{\tau=1}^{t} \mathbb{E}_{\hat{p}(\boldsymbol{z},\tau,t|\boldsymbol{w}_k,\mathcal{D})}\Big[\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big], \\ &= \underset{\boldsymbol{w}\in\mathcal{W}}{\operatorname{argmax}} \int d\boldsymbol{\theta} \sum_{t=1}^{H}\sum_{\tau=1}^{t} \mathbb{E}_{\hat{p}(\boldsymbol{z},\tau,t,\boldsymbol{\theta}|\boldsymbol{w}_k,\mathcal{D})}\Big[\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big].\end{aligned}$$

For each $\boldsymbol{\theta}\in\Theta$ the expectation *w.r.t.* $\hat{p}(\boldsymbol{z},\tau,t|\boldsymbol{\theta},\boldsymbol{w}_k)$ can be calculated using standard methods from chapter(2). Given this simple structure in the policy update a naive stochastic approximation is given by

$$\boldsymbol{w}_{k+1} = \underset{\boldsymbol{w}\in\mathcal{W}}{\operatorname{argmax}} \sum_{i=1}^{I}\sum_{t=1}^{H}\sum_{\tau=1}^{t} \mathbb{E}_{\hat{p}(\boldsymbol{z},\tau,t|\boldsymbol{\theta}_i,\boldsymbol{w}_k)}\Big[\log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w})\Big].$$

where $\{\boldsymbol{\theta}_i\}_{i=1}^{I}$ are a set of sample transition parameters that are sampled from $p(\boldsymbol{\theta}|\mathcal{D})$. This sampling scheme is simple but it will, in general, be expensive as it makes inefficient use of the samples. Given the close similarity between the gradient of the Bayesian objective (6.10) and the gradient of the MDP objective it is possible to consider more sophisticated sampling-based methods, such as extensions of

the Bayesian gradient methods [65] or actor-critic methods [98, 99], but we do not do so in this work. Additionally, it will generally be the case that areas of high probability under $p(\boldsymbol{\theta}|\mathcal{D})$ will correspond to areas of low probability under $\hat{p}(\boldsymbol{\theta}|\boldsymbol{w}_k, \mathcal{D})$. As a result this simple Monte-Carlo method will be inefficient, with many of the samples providing a negligible contribution to the policy update. Therefore it will be preferable to consider more sophisticated sampling methods that are designed to obtain samples from areas of high probability under $\hat{p}(\boldsymbol{\theta}|\boldsymbol{w}_k, \mathcal{D})$, such as Gibbs sampling or MCMC methods.

6.5 Experiments

In this section we perform some empirical evaluations of the theory presented in this chapter. In the section(6.5.1) we justify the use of the objective function (6.5) and illustrate that it indeed incorporates some of the uncertainty of our knowledge of the environment into the optimisation process. In section(6.5.2) we compare our three approximation algorithms to the EM-algorithm that uses the maximum likelihood estimate of the transition dynamics.

6.5.1 Incorporation of Uncertainty

The first experiment is designed to demonstrate that the objective function (6.5) indeed incorporates uncertainty in the knowledge of the environment into the policy optimisation process. The experiment is performed on a problem small enough that for short horizons the objective function (6.5) and the EM update (6.9) can be calculated exactly. This allows for characteristics of the objective function to be gleaned without the complicating issue of approximations.

The experiment was performed on a toy two-state problem, with the transition and reward matrices given in fig(6.2(a)). The horizon was set to $H = 5$ and the initial state is 1. The aim of the experiment is to compare the average total expected utility of the policies obtained from the Bayesian and point-based objective functions. The average is taken over the true transition model, θ_{true}, and we compare these averages for increasing numbers of observed transitions, N. We set the distribution over the true transition model to be uniform. Writing the quantities of interest down algebraically we have for the Bayesian objective function

$$\mathbb{E}_{p(\theta_{\text{true}})}[\mathbb{E}_{p(\mathcal{D}|\theta_{\text{true}},N)}[U(\hat{\pi}^{\mathcal{D}}|\theta_{\text{true}})]] = \int d\theta_{\text{true}} d\mathcal{D} U(\hat{\pi}^{\mathcal{D}}|\theta_{\text{true}})) p(\mathcal{D}|\theta_{\text{true}}, N) p(\theta_{\text{true}}) \tag{6.25}$$

where $\hat{\pi}^{\mathcal{D}}$ is the optimal policy of the Bayesian objective function. For the ML objective function we have

$$\mathbb{E}_{p(\theta_{\text{true}})}[\mathbb{E}_{p(\hat{\pi}^{\text{ML}}|\theta_{\text{true}},N)}[U(\hat{\pi}^{\text{ML}}|\theta_{\text{true}})]] = \int d\theta_{\text{true}} d\hat{\pi}^{\text{ML}} U(\hat{\pi}^{\text{ML}}|\theta_{\text{true}}) p(\hat{\pi}^{\text{ML}}|\theta_{\text{true}}, N) p(\theta_{\text{true}}) \tag{6.26}$$

where similarly $\hat{\pi}^{\text{ML}}$ is the optimal policy of the ML objective function.

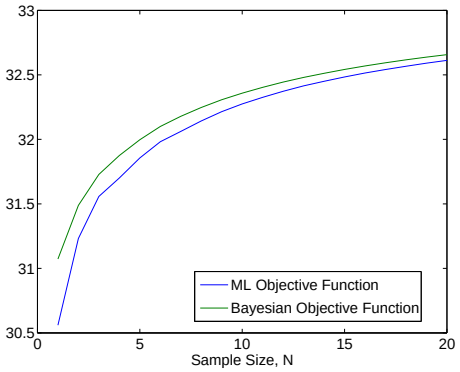
As we can calculate the objective function $U(\pi|\mathcal{D})$ exactly, we can also calculate (6.25) for reasonable values of N. It remains to calculate (6.26), where the difficult term is the probability distribution over the optimal policy, which we now detail.

The settings of the reward matrix and the horizon are such that, given (θ_1, θ_2) are known, the optimal

$$T_i = \begin{bmatrix} \theta_i & 1-\theta_i \\ 1-\theta_i & \theta_i \end{bmatrix}$$

$$R = \begin{bmatrix} 4 & 10 \\ 1 & 1 \end{bmatrix}$$

(a) Transition Dynamics and Reward Function



(b) Incorporation of Uncertainty Experiment

Figure 6.2: (a) The transition and reward matrices for the two-state toy problem. T_i represents the transition matrix from state s_i, where the columns correspond to actions and the rows correspond to the next state. The reward matrix R is defined so that the actions run along the rows and the states run along the columns. (b) The average total expected reward of the policies obtained from the Bayesian objective function, $U(\pi|\mathcal{D})$, and the maximum likelihood objective function, $U(\pi|\boldsymbol{\theta}_{\text{ML}})$. The sample size is plotted against the average total expected reward.

action in state s_2 is a_1 for all values of θ_2. This means that when the transition dynamics are known the optimal policy can be given by a single parameter, $\hat{\pi}_{s_1,a_1}$. In the experiment we set $\theta_1 = \theta_2 = \theta$, so that $\hat{\pi}_{s_1,a_1} = 1$ when $\theta < \hat{\theta}$, and $\hat{\pi}_{s_1,a_1} = 0$ otherwise, where $\hat{\theta} = 0.7021$. The fact that we know the point, $\hat{\theta}$, at which the optimal policy of the MDP changes means that we can form a distribution of $\hat{\pi}^{\text{ML}}_{s_1,a_1}$. Given the sample size and the true value of the transition parameter we have the distribution

$$p(\hat{\pi}^{\text{ML}}_{s_1,a_1} = 1|N, \theta_{\text{true}}) = \sum_{\{n \leq N | n/N < \hat{\theta}\}} B_{N,\theta_{\text{true}}}(n)$$

where $B_{N,\theta_{\text{true}}}$ is the density function of the Binomial distribution with parameters $(N, \theta_{\text{true}})$. Having obtained the distribution over the optimal policy it is now possible to calculate (6.26).

We calculated (6.25) and (6.26) for increasing values of the N, the results of which are shown in fig(6.2(b)). It can be observed that the Bayesian objective function consistently outperforms the point-based objective function. We expect a more dramatic difference in larger problems for which the amount of uncertainty in the transition parameters is greater.

6.5.2 On-Line Learning

In this section we apply our algorithms to the on-line learning framework, where the agent begins with no knowledge of the environment and must optimise its behaviour in an on-line manner as more information about the environment is gained. In the experiment the agent starts with a random policy and performs a certain number of policy updates at certain intervals during the run-time of the system. We ran two experiments where in the first a single policy update was performed after each step in the environment, while in the second a single policy update was performed after every 25 steps in the environment. Due to the prohibitive computational cost of the approximate message-passing algorithm this algorithm was considered only in the later of these two experiments. The policy of the initial time-point was initialised

randomly from a uniform distribution. We considered the chain problem [42] in the experiment, the details of which can be found in the preceding chapter.

In the experiments we considered a total of 1000 time-points and the observation counts were updated after each observed transition. The experiments were repeated 500 times and the results shown display the mean and standard error of the algorithms. Where appropriate the hyper-parameters of the prior were set to $\boldsymbol{\alpha} = 0.1$. We considered two types of prior, *uniform* and *structural*. In the *uniform* prior all the hyper-parameters of the Dirichlet were set to the same value. In the *structural* prior the structure of the transition matrix was assumed to be known, *i.e.* it was known which elements were zero or non-zero, so that the hyper-parameters of the impossible transitions were set to zero. In the case of point-based estimates of the transition matrix this structural information was used when no transitions had been observed for a given state-action pair. In this case the next state probability was set to a uniform distribution over the possible states in the next time-point.

We performed the experiments with the following algorithms:

ML-DP After every n steps in the environment the policy is obtained by performing a single iteration of Expectation Maximisation with the ML estimate of the transition matrix.

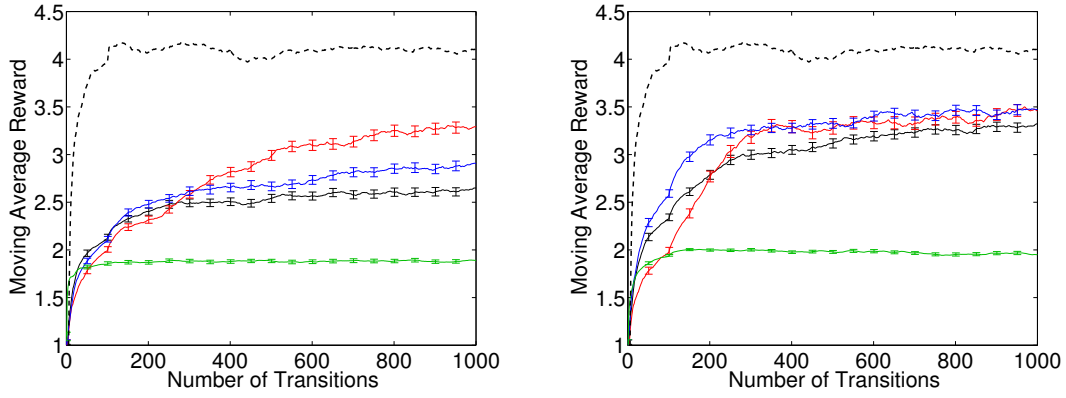
VB-EM After every n steps in the environment the policy is updated using the variational Bayes approach described in section(6.4.1). During each policy update 15 VB iterations were performed.

Heuristic VB-EM Considering the form of the VB variational distribution over $\boldsymbol{\theta}$, given by (6.17 & 6.18), we considered the following heuristic: At any given VB iteration take the mean of the current approximation to $q(\boldsymbol{\theta})$ and use this point estimate as a model to calculate the state-action marginals in (6.18), using the methods of chapter(2); Using these marginals update the approximation to $q(\boldsymbol{\theta})$ using (6.17);

AMP-EM After every n steps in the environment the policy is updated using the approximate message-passing approach described in section(6.4.2). During each policy update we performed 10 repetitions of the message-passing schedule algorithm(6.2).

S-EM After every n steps in the environment the policy is updated using the stochastic EM-algorithm described in section(6.4.3). We took 100 samples in each training iteration.

The results of the two experiments are shown in fig(6.3) and fig(6.4), where we show the mean and standard error of the results. In the first experiment, where a policy update was performed after every step, the stochastic EM-algorithm obtained the best performance of the three Bayesian algorithms presented in this chapter. In the second experiment, where a policy update was performed after every 25 steps in the environment, the approximate message-passing algorithm obtained the best performance. In the first experiment all of the algorithms obtain better performance under the structural prior than the uniform prior. In the second experiment this difference is less apparent. In the maximum likelihood EM-algorithm this is because the point estimate of the transition matrix contains more information about the structure of the problem in state-action pairs that have not been observed. In the Bayesian



(a) Uniform Prior (b) Structural Prior

Figure 6.3: The results of the variational Reinforcement Learning algorithms applied to the chain problem when a policy update is performed after every step in the environment. The plot shows the mean and standard error of the running average per time point, where the running average uses the last 100 steps in the environment. The plot shows the results for ML EM-algorithm (black), variational Bayes EM-algorithm (green), Heuristic variational Bayes EM-algorithm (red), the approximate message-passing EM-algorithm (purple) and the stochastic EM-algorithm (blue). The black dashed line shows that results of using the optimal policy for the chain problem throughout the course of the interaction with the environment. This line was obtained by applying this policy to the random streams used in the experiment.

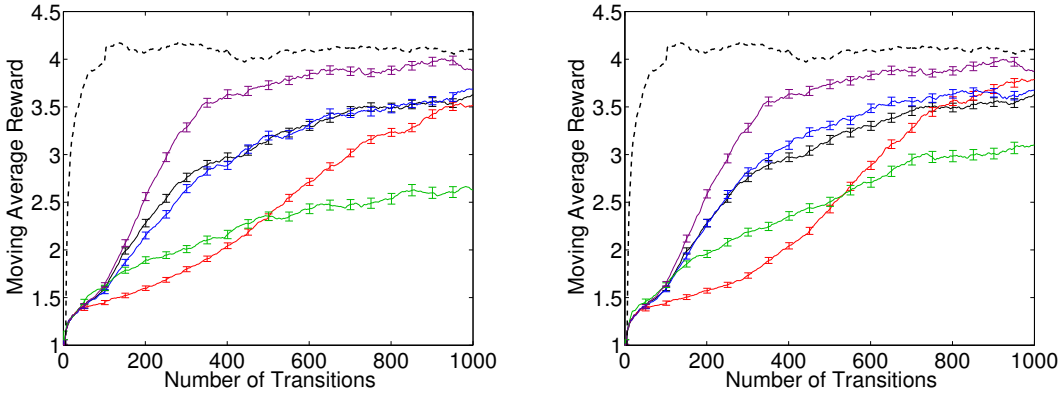
methods this is explained by the fact that a larger proportion of the mass in the posterior is placed on models where transitioning towards the end of the chain is optimal. As a result more 'exploratory' behaviour is obtained and the algorithms are able to find the global optimum more consistently. With the exception of variational Bayes EM-algorithm all of the Bayesian methods outperformed the point-based maximum likelihood EM-algorithm. This highlights the advantages of our Bayesian perspective to the Reinforcement Learning problem. We now consider the results in greater detail.

The VB EM-algorithm performed poorly, obtaining rewards that were substantially lower than using Expectation Maximisation with the maximum likelihood estimate of the transition matrix. There are two possible reasons for this poor performance. The first is that the factorisation assumption (6.11) is too strong and much of the useful information in the variational distribution is lost by this approximation. Another issue is the form of the trans-dimensional distribution $q(\boldsymbol{z}_{1:t}, t)$ that is obtained from the VB approximation. It was observed that (6.15) has the form of a reward weighted trajectory distribution of an MDP, with the exception that the transition dynamics are replaced by terms of the form

$$\tilde{\theta}(\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a}) \equiv e^{\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \theta_{\boldsymbol{s}', \boldsymbol{s}, \boldsymbol{a}}\right]}.$$

As these transition dynamics are geometric means they are subnormalised, *i.e.* their sum is less than or equal to 1, see *e.g.* [112]. While this sub-normality doesn't effect the forward-backward algorithm for HMMs it does effect the inference for the trans-dimensional distribution (6.15). In the case of a HMM it is possible to normalise the forward messages and, as the HMM is chain structured, this normalisation factor cancels out when combining the messages to obtain the marginals of the posterior[5]. In the case of

[5]Recall that in the forward-backward algorithm for a HMM the forward message, say $\alpha(h)$, and backward message, say $\beta(h)$, are combined to give the marginal $p(h) \propto \alpha(h)\beta(h)$. This means that the using the normalised forward message, $\tilde{\alpha}(h) =$



(a) Uniform Prior (b) Structural Prior

Figure 6.4: The results of the variational Reinforcement Learning algorithms applied to the chain problem when a policy update is performed after every 25 steps in the environment. The plot shows the mean and standard error of the running average per time point, where the running average uses the last 100 steps in the environment. The plot shows the results for ML EM-algorithm (black), variational Bayes EM-algorithm (green), Heuristic variational Bayes EM-algorithm (red), the approximate message-passing EM-algorithm (purple) and the stochastic EM-algorithm (blue). The black dashed line shows that results of using the optimal policy for the chain problem throughout the course of the interaction with the environment. This line was obtained by applying this policy to the random streams used in the experiment.

the trans-dimensional distribution (6.15) each component of the mixture is chain structured, but they are all of different lengths. The differing lengths of the chains means that the weights of the mixture components are effected to varying degrees by the issue of subnormalisation, and this issue cannot be removed in the same manner as in the HMM. To obtain a better gauge on the cause for this poor performance we also considered a heuristic version of the VB EM-algorithm, where the E-step consisted of iterating equations (6.17 & 6.18) and using the mean of $q(\boldsymbol{\theta})$ as a point-estimate of the transition dynamics when evaluating (6.18). In this heuristic the form of the posterior over Θ is the same as that in (6.17) but there are no longer any issues with subnormalisation. It can be seen in fig(6.3) that this heuristic can perform substantially better than the actual VB EM-algorithm. Additionally, in the first experiment this heuristic obtains superior results to the ML EM-algorithm with both the uniform and structural prior. These results suggest that the main issue with the VB EM-algorithm lies in the issues of subnormalisation and not so much in the severity of the factorisation assumption (6.11).

The approximate message-passing algorithm performed well in the experiments and obtained consistent performance over the two forms of prior considered. In the experiments it was observed that the approximate distribution over $\boldsymbol{\theta}$ weighted each transition in terms of the amount of reward that could be 'expected' from the transition, where this expectation was approximated through message-passing. Given the form of the distribution $q(\boldsymbol{\theta})$ under the VB approximation this behaviour is unsurprising and the AMP EM-algorithm can be seen to automate its exploration by adjusting the 'transition matrix' (*i.e.* the approximate transition matrices in the message-passing) to make transitions to possibly fruitful parts of the state-action space more likely under the approximation. When only a few transitions have been ob-

$Z^{-1}\alpha(h)$, where Z^{-1} is the normalisation constant, in place of the original forward message leaves the update of the marginal unaffected.

served for a given state-action pair the corresponding part of the 'transition matrix' is dominated by this exploratory weighting. As more transitions are observed the data starts to dominate the corresponding part of the approximate transition dynamics. As an example of this type of behaviour consider the chain problem with a uniform prior, where no transitions have been observed for any state other than the initial state. In this situation the approximate transition dynamics for the unobserved states would be heavily weighted to transition to the end of the chain where the reward is highest. These skewed transition dynamics then encourage the agent to travel to these 'fruitful' states and exploratory behaviour is obtained. While the AMP EM-algorithm performed well in the experiments it was the most computationally expensive of all the algorithms, requiring approximate inference to be performed in order to perform a policy update. For this reason we considered this algorithm only in the second experiment, where the algorithm was considered too computationally expensive to be considered in the first experiment. The average computational cost of performing a run of each experiment for all of the various algorithms is given in table(6.1).

The stochastic EM-algorithm outperforms the ML EM-algorithm under both priors in the first experiment, while it outperforms the ML EM-algorithm in the last 500 iterations of the second experiment under the structural prior. While all algorithms obtained superior performance under the structural prior in the first experiment the comparative performance of stochastic EM-algorithm in relation to the other algorithms is markedly improved under this prior. This is due to the smaller amount of volume under the posterior using the structural prior, which means that this stochastic approximation to the posterior is superior when using the same amount of samples.

6.6 Discussion

Given the close connection between the maximisation of the marginal $\log$-likelihood of latent variable time-series models and the optimisation of Markov Decision Processes it is natural to extend the Bayesian techniques for these time-series models to the Reinforcement Learning framework. This is the approach we have taken in our formulation of Bayesian Reinforcement Learning, giving some groundwork theory to this approach. The advantage of this approach is that it allows methods in approximate inference to be exploited to help overcome difficulties associated with Bayesian Reinforcement Learning. An exact implementation of such a Bayesian formulation of Reinforcement Learning is formally intractable and we considered three approximate solutions, one based on variational Bayes, another on Expectation Propagation and the third based on a naive sampling method. We have shown the benefits of using the Bayesian objective (6.5) in place of a point-based objective by considering a toy MDP problem. Initial empirical results suggest that the latter two approaches are generally to be preferred, although more extensive empirical evaluations are necessary in terms of extending these algorithms to large-scale problems. There are various avenues of possible future research.

The variational Bayes approximation offers a mathematically elegant solution. However, the mixture structure of the reward weighted trajectory distribution leads to issues of subnormalisation affecting the approximation. A possible solution to this issue is to consider the risk-sensitive MDP objective function given in (1.5), or some similar objective that has a product (as opposed to additive) structure over

	Maximum Likelihood	Variational Bayes	Approximate Message-Passing	Stochastic
Run-time (seconds)	0.25	0.90	175	9.25

Table 6.1: Run-time of the second on-line learning experiment for the Maximum Likelihood EM-algorithm, Variational Bayes EM-algorithm, Approximate Message-Passing EM-algorithm and Stochastic EM-algorithm.

the rewards of the various time-points. As noted in chapter(2) constructing an EM-algorithm for this objective would result in a variational distribution that has the structure of a single chain, similar to the Hidden Markov Model. The advantage of such an approach would be that subnormalisation would no longer be an issue in the variational Bayes approximation. Considering the performance of the heuristic variational Bayes approximation there is good reason to believe that this approximation will perform well once the issue of subnormalisation has been removed. This advantage comes at the cost of tuning the risk-sensitive parameter, which can be non-trivial in practice, but the possible advantage of this approach make it an attractive alternative.

There are various possible avenues of future research in terms of sampling methods. The stochastic EM-algorithm from section(6.4.3) is inefficient in terms of the number of samples required, where sampled MDP parameters generally have high probability under $p(\boldsymbol{\theta}|\mathcal{D})$ but low probability under $\hat{p}(\boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D})$. One possible solution is to construct a Gibbs sampler for sampling from $\hat{p}(\boldsymbol{z}_{1:t}, t, \boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D})$, where the sampler alternates between sampling over reward weighted trajectories and transition matrices, *i.e.*

$$\boldsymbol{\theta} \mid \boldsymbol{z}_{1:t}, t, \mathcal{D} \sim \hat{p}(\boldsymbol{\theta}|\boldsymbol{z}_{1:t}, t, \boldsymbol{w}, \mathcal{D}), \qquad \boldsymbol{z}_{1:t}, t \mid \boldsymbol{\theta} \sim \hat{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w}).$$

Obtaining samples from $\hat{p}(\boldsymbol{\theta}|\boldsymbol{z}_{1:t}, t, \boldsymbol{w}, \mathcal{D}) \propto \hat{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w})p(\boldsymbol{\theta}|\mathcal{D})$ is simple when using a conjugate prior, which in this case amounts to sampling from a Dirichlet distribution. One possible routine to obtain a sample from $\hat{p}(\boldsymbol{z}_{1:t}, t|\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D})$ would be to first sample the component of the mixture distribution, $t \sim \hat{p}(t|\boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D})$, and then to sample a trajectory from the distribution corresponding to that mixture component, $\boldsymbol{z}_{1:t} \sim \hat{p}(\boldsymbol{z}_{1:t}|t, \boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D})$. It is possible to use a forward-recursion backward-sample routine, see *e.g.* [148], to sample from $\hat{p}(\boldsymbol{z}_{1:t}|t, \boldsymbol{\theta}, \boldsymbol{w}, \mathcal{D})$. Another possibility is to construct a Monte-Carlo Markov Chain sampler for obtaining samples from $\hat{p}(\boldsymbol{\theta}|\boldsymbol{w}, \mathcal{D})$.

To date we have only considered discrete state-action spaces with a table-look parameterisation of the transition dynamics. It is of interest to extend the work in this chapter to a wider range of problems, including alternative, perhaps more structural, parameterisations of the transition dynamics in discrete state-action problems, as well as problems with a continuous state-action space. One of the strengths of our Bayesian approach is that such extensions pose little problems theoretically. Some work has been previously done in this area, see *e.g.* [132, 43], and strong results have been obtained. In [132, 43] Gaussian processes are employed to model the uncertainty in the transition dynamics and to overcome the intractabilities of the Bayesian Reinforcement Learning framework they use a form of assumed density filtering. Alternative approaches in this area, similar to those discussed in this chapter, could be an interesting point of future work.

Conclusion

A large portion of the work in this thesis has focused on policy search methods, such as steepest gradient ascent, natural gradient ascent and Expectation Maximisation. It has previously been noted that the policy evaluation stage of these algorithms can be seen to be equivalent to performing inference in a latent variable time-series model, where we refer to this distribution as the reward weighted trajectory distribution. The structure of this distribution is markedly different from the structure that typically occurs in latent variable time-series models, having a mixture structure over the planning horizon. This necessarily makes the form of the inference different from typical time-series inference and thus requires the construction of novel inference algorithms. To date all model-based inference algorithms in this area can be seen as a type of forward-backward algorithm. In contrast I have introduced a novel model-based inference algorithm that more closely resembles Rauch-Tung-Striebel type inference algorithms. Due to this mixture structure of the reward weighted distribution, as well as the possibly infinite planning horizon, this reformulation of the inference is non-trivial and has significant consequences. An example we have considered in depth is linear systems with a non-linear reward structure. A forward-backward algorithm has previously been constructed for this model, but it is only applicable to finite planning horizons and has a runtime that scales quadratically in the planning horizon. In contrast the RTS style algorithm that I have presented has a runtime that is linear in the planning horizon and is also applicable to infinite planning horizons with discounted rewards. A current issue with the infinite horizon recursion for these liner systems is the necessity that the eigenvalues of the state-action transition matrix lie within the unit circle. This is a non-trivial constraint and correctly handling it has thus far proved problematic on higher dimensional systems. Further, I have highlighted how this novel reformulation of the policy evaluation problem can be beneficial in other models, specifically high-dimensional discrete problems that have a sufficiently sparse underlying structure that approximate inference techniques can be applied. Thus far I have only highlighted the possible advantages of this RTS approach and the actual construction and evaluation of such inference procedures is a point of future work.

A second contribution to the area of policy search methods is the provision of a unifying perspective of both natural gradient ascent and Expectation Maximisation. While both of these methods have been popular in the area of policy search methods there has previously been little understanding about the relation between the two algorithms. The novel analysis provided greatly clarifies the relation between the two algorithms by relating them both to a particular form of approximate Newton method. Motivated by this analysis a natural consideration is the direct application of this approximate Newton method

to the optimisation of the MDP objective. Such considerations have been made and it has been found that it has many desirable properties that are absent in the naive application of the Newton method. Initial empirical results show that the method has strong performance in comparison to natural gradient ascent and Expectation Maximisation, especially in continuous systems. It is desirable to apply this approximate Newton method to larger scale problems in order to better gauge the scalability of the algorithm. Additionally, thus far we have considered this approximate Newton method only in terms of an actor method and it is a point of future work to consider the extension to actor-critic methods,

Viewing policy evaluation as probabilistic inference easily allows for the extension of these techniques to the Bayesian framework, where a distribution is placed over the model parameters of the Markov Decision Process. In this case the policy evaluation stage of policy search methods corresponds to inference in a Bayesian time-series model where both the trajectory and model parameters are weighted by the total expected reward of the trajectory under the current model. A Bayesian perspective to Reinforcement Learning then corresponds to likelihood maximisation of this Bayesian latent variable time-series model. This is an approach we have taken in chapter(6) and, as the inference is intractable in this Bayesian framework, we have presented several approximate inference routines for this problem, including a variational Bayes approximation, an approximation based on Expectation Propagation and a stochastic approximation. Initial experiments suggest that there are significant possible benefits to the Bayesian approach to Reinforcement Learning and, other than more extensive experiments, there are several possible extensions to this work. These include the consideration of the risk-sensitive objective function (1.5), which would result is a chain structured reward weighted trajectory distribution that is more amenable to the construction of approximate inference routines. Additionally, thus far we have only considered discrete state-action spaces and it would be interesting to extend these approaches to continuous domains. An advantage of the current approach is that such an extension is, theoretically at least, relatively straightforward.

The final significant contribution of this thesis is the novel application of dual decomposition techniques to a particular sub-family of Markov Decision Processes, namely finite horizon Markov Decision Processes with a stationary policy. This problem class is non-trivial because the policy is non-Markovian, due to the stationarity constraint, and this invalidates the application of dynamic programming. By relaxing this stationarity constraint through dual decomposition techniques it is possible to obtain a convex upper bound on the objective function. This bound can be minimised through various convex optimisation techniques and this results in a two stage iterative process. This algorithm has an intuitive interpretation where the inner loop of the algorithm consists of optimising a finite horizon Markov Decision Process with non-stationary policy and reward function, while the outer loop updates the non-stationary reward function to encourage consistency between the policies of different time-points. Empirical results suggest that the dual decomposition algorithm is well suited to this problem class, consistently obtaining the global optimum in relatively few iterations. It is of interest to see if similar performance can be obtained in other planning models with non-Markovian policies, such as a decentralised transition independent Markov Decision Processes.

Bibliography

[1] D. L. Alspach and H. W. Sorenson. Nonlinear Bayesian Estimation Using Gaussian Sum Approximations. *IEEE Transactions on Automatic Control*, 17(4):439–448, 1972.

[2] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall, first edition, 1999.

[3] S. Amari. Neural Learning in Structured Parameter Spaces - Natural Riemannian Gradient. *NIPS*, 9:127–133, 1997.

[4] S. Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10:251–276, 1998.

[5] S. Amari, A. Cichocki, and H. Yang. A New Learning Algorithm for Blind Signal Separation. *NIPS*, 8:757–763, 1996.

[6] S. Amari, K. Kurata, and H. Nagaoka. Information Geometry of Boltzmann Machines. *IEEE Transactions on Neural Networks*, 3(2):260–271, 1992.

[7] C. Amato, D. Bernstein, and S. Zilberstein. Solving POMDPs Using Quadratically Constrain Linear Programs. *IJCAI*, 20:673–680, 2007.

[8] M. Anthony and N. Biggs. *Computational Learning Theory*. Cambridge University Press, second edition, 1997.

[9] M. Araya, O. Buffet, and V. Thomas. Near-Optimal BRL using Optimistic Local Transitions. *ICML*, 29:97–104, 2012.

[10] Asmuth, J. and Li, L. and Littman, M. and Nouri, A. and Wingate, D. A Bayesian Sampling Approach to Exploration in Reinforcement Learning. *UAI*, 25:19–26, 2009.

[11] Asmuth, J. and Littman, M. Learning is Planning: Near Bayes-Optimal Reinforcement Learning via Monte-Carlo Tree Search. *UAI*, 27:19–26, 2011.

[12] H. Attias. A Variational Bayesian Framework for Graphical Models. *NIPS*, 12:209–215, 2000.

[13] J. Bagnell and J. Schneider. Covariant Policy Search. *IJCAI*, 18:1019–1024, 2003.

[14] D. Barber. Expectation Correction for Smoothing in Switching Linear Gaussian State Space Models. *Journal of Machine Learning Research*, 7:2515–2540, 2006.

[15] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2011.

[16] D. Barber and S. Chiappa. Unified Inference for Variational Bayesian Linear Gaussian State-Space Models. *NIPS*, 19:81–88, 2007.

[17] D. Barber and W. Wiegerinck. Tractable Variational Structures for Approximating Graphical Models. In *NIPS*, volume 12, pages 183–189, 1998.

[18] N. Bäuerle and U. Rieder. *Markov Decision Processes with Applications to Finance*. Universitext, 2011.

[19] J Baxter and P. Bartlett. Infinite Horizon Policy Gradient Estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[20] M. Beal. *Variational Algorithms for Approximate Bayesian Inference*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

[21] M. J. Beal and Z. Ghahramani. The Variational Bayesian EM Algorithm for Incomplete Data: with Application to Scoring Graphical Model Structures. In *Bayesian Statistics*, volume 7, pages 453–464. Oxford University Press, 2003.

[22] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[23] D Bernstein, R Givan, N. Immerman, and S. Zilberstein. The Complexity of Decenteralised Control of Markov Decision Processes. *MOR*, 27:819–840, 2002.

[24] D. P. Bertsekas. *Dynamic Programming and Stochastic Control*. Academic Press, 1976.

[25] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

[26] D. P. Bertsekas. Approximate Policy Iteration: A Survey and Some New Methods. Research report, Massachusetts Institute of Technology, 2010.

[27] D. P. Bertsekas and S. Ioffe. Temporal Differences-Based Policy Iteration and Applications in Neuro-Dynamic Programming. Research Report LIDS-P-2349, Massachusetts Institute of Technology, 1996.

[28] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee. Incremental Natural Actor-Critic Algorithms. *NIPS*, 20:105–112, 2008.

[29] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and L. Mark. Natural Actor-Critic Algorithms. *Automatica*, 45:2471–2482, 2009.

[30] C. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, first edition, 1995.

[31] V. Borkar and S. Meyn. Risk-Sensitive Optimal Control for Markov Decision Processes with Monotone Cost. *Mathematics of Operations Research*, 27(1):192–209, 2002.

[32] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[33] S. Boyd and L. Vandenberghe. Sub-Gradients. Lecture notes, Stanford University, 2008.

[34] S. Boyd, L. Xiao, and A Mutapcic. Sub-Gradient Methods. Lecture notes, Stanford University, 2003.

[35] S. Boyd, L. Xiao, A. Mutapcic, and J. Mattingley. Notes on Decomposition Methods. Lecture notes, Stanford University, 2008.

[36] R. Brafman and Tennenholtz. R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[37] A. Coolen, R. Kuehn, and P. Sollich. *Theory of Neural Information Processing Systems*. Oxford University Press, 2005.

[38] W. Davidon. Variable Metric Method for Minimisation. *SIAM Journal on Optimisation*, 1:1–17, 1991.

[39] P. Dayan. Reinforcement Comparison. *Proceedings of the 1990 Connectionist Models Summer School*, 1990.

[40] P. Dayan and G. E. Hinton. Using Expectation-Maximization for Reinforcement Learning. *Neural Computation*, 9:271–278, 1997.

[41] D. de Farias and B. Van Roy. The Linear Programming Approach to Approximate Dynamic Programming. *Operations Research*, 51(6):850–865, 2003.

[42] R. Dearden, N. Friedman, and S. Russell. Bayesian Q learning. *AAAI*, 15:761–768, 1998.

[43] M. Deisenroth and C. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. *ICML*, 28:465–472, 2011.

[44] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[45] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[46] C. Dimitrakakis. Tree Exploration for Bayesian RL Exploration. *CIMCA/IAWTIC/ISE*, pages 1029–1034, 2008.

[47] C. Dimitrakakis. Complexity of Stochastic Branch and Bound Methods for Belief Tree Search in Bayesian Reinforcement Learning. *CoRR*, abs/0912.5029, 2009.

[48] C. Dimitrakakis. Robust Bayesian Reinforcement Learning Through Tight Lower Bounds. *CoRR*, abs/1106.3651, 2011.

[49] A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[50] M. Duff. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2002.

[51] C. Fahey. Tetris AI, Computers Play Tetris `http://colinfahey.com/tetris/tetris_en.html`, 2003.

[52] W. Fleming and H. Soner. *Controlled Markov Processes and Viscosity Solutions.* Springer Velag, 1992.

[53] R. Fletcher. A New Approach to Variable Metric Algorithms. *The Computer Journal*, 3(13):317–322, 1970.

[54] R. Fletcher. *Practical Methods of Optimisation*. John Wiley & Sons, 1987.

[55] R. Fletcher and C. Reeves. Function Minimisation by Conjugate Gradients. *The Computer Journal*, 7:149–154, 1964.

[56] T. Furmston and D. Barber. Solving deterministic policy (PO)MPDs using Expectation-Maximisation and Antifreeze. *European Conference on Machine Learning (ECML)*, 1:50–65, 2009. Workshop on Learning and data Mining for Robotics.

[57] T. Furmston and D. Barber. Variational Methods for Reinforcement Learning. *AISTATS*, 9:241–248, 2010.

[58] T. Furmston and D. Barber. Variational Methods for Reinforcement Learning. *AISTATS*, 9(13):241–248, 2010.

[59] T. Furmston and D. Barber. Efficient Inference for Markov Control Problems. *UAI*, 27:221–229, 2011.

[60] S. Gelly and D. Silver. Achieving Master Level Play in 9 x 9 Computer Go. *AAAI*, 23:1537–1540, 2008.

[61] S. Gelly and D. Silver. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artificial Intelligence*, 175(11):1856–1875, 2011.

[62] A. Gelman, J. Carlin, H. Stern, and D. Rubin. *Bayesian Data Analysis*. Chapman & Hall, 2004.

[63] Z. Ghahramani and G. E. Hinton. Variational Learning for Switching State-Space Models. *Neural Computation*, 12:831–864, 2000.

[64] M Ghavamzadeh and Y. Engel. Bayesian Actor-Critic Algorithms. *ICML*, 24:297–304, 2007.

[65] M. Ghavamzadeh and Y. Engel. Bayesian Policy Gradient Algorithms. *NIPS*, 19:457–464, 2007.

[66] P. W. Glynn. Stochastic Approximation for Monte-Carlo Optimisation. *Proceedings of the 1986 ACM Winter Simulation Conference*, 18:356–365, 1986.

[67] P. W. Glynn. Likelihood Ratio Gradient Estimation for Stochastic Systems. *Communications of the ACM*, 33:97–84, 1990.

[68] E. Greensmith, P. Bartlett, and J. Baxter. Variance Reduction Techniques For Gradient Based Estimates in Reinforcement Learning. *Journal of Machine Learning Research*, 5:1471–1530, 2004.

[69] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, third edition, 2001.

[70] F. Grognard and J. Gouze. Positive Control of Lotka-Volterra Systems. *Proceedings of the 16th IFAC World Congress*, 16(1), 2005.

[71] C. Guestrin, D. Koller, P. Parr, and S. Venkataraman. Efficient Solution Algorithms for Factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468, 2003.

[72] C. Guestrin, D. Koller, and R. Parr. Max-norm Projections for Factored MDPs. *IJCAI*, 1:673–680, 2001.

[73] C. Guestrin, D. Koller, and R. Parr. Multiagent Planning with Factored MDPs. *NIPS*, 15:1523–1530, 2001.

[74] A. Guez, D. Silver, and P. Dayan. Efficient Bayes-Adaptive Reinforcement Learning using Sample-Based Search. *NIPS*, 26, 2012.

[75] N. Hastings and D. Sadjadi. Markov Programming with Policy Constraints. *European Journal of Operations Research*, 3:253–255, 1979.

[76] Kappen H.J. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment*, page P11011, 2005.

[77] M. Hoffman, N. de Freitas, A. Doucet, and J. Peters. An Expectation Maximization Algorithm for Continuous Markov Decision Processes with Arbitrary Rewards. *AISTATS*, 12(5):232–239, 2009.

[78] M. Hoffman, H. Kueck, N. de Freitas, and A. Doucet. New Inference Strategies for Solving Markov Decision Processes Using Reversible Jump MCMC. *UAI*, 25:223–231, 2011.

[79] Hoffman, M. and Doucet, A. and De Freitas, N. and Jasra, A. Bayesian Policy Learning with Trans-Dimensional MCMC. *NIPS*, 20:665–672, 2008.

[80] R. A. Howard. *Dynamic Programming and Markov Processes*. M.I.T. Press, 1960.

[81] M. Jamshidian and R. Jennrich. Conjugate Gradient Acceleration of the EM Algorithm. *Journal of the American Statistical Association*, 88(421):221–228, 1993.

[82] L. Kaelbling, M. Littman, and A. Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101:99–134, 1998.

[83] S. Kakade. A Natural Policy Gradient. *NIPS*, 14:1531–1538, 2002.

[84] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[85] H. Kappen and W. Wiegerinck. Novel Iteration Schemes for the Cluster Variation Method. *NIPS*, 15:415–422, 2001.

[86] N. Karmarkar. A New Polynomial Time Algorithm for Linear Programming. *Combinatorica*, 4(4):373–395, 1984.

[87] M. Kearns, Y. Mansour, and A. Ng. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *IJCAI*, 16:1324–1331, 1999.

[88] M. Kearns and S. Singh. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49:209–232, 2002.

[89] L. G. Khachian. A Polynomial Algorithm for Linear Programming. *Soviet Math Dokl.*, 20:191–194, 1984.

[90] H. Khalil. *Nonlinear Systems*. Prentice Hall, 2001.

[91] H. Kimura, K. Miyazaki, and S. Kobayashi. Reinforcement Learning in POMDP's with Function Approximation. *ICML*, 14:152–160, 1997.

[92] H. Kimura, M. Yamamura, and S. Kobayashi. Reinforcement Learning by Stochastic Hill Climbing on Discounted Reward. *ICML*, 12:295–303, 1995.

[93] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *NIPS*, 21:849–856, 2009.

[94] J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, 84(1-2):171–203, 2011.

[95] L. Kocsis and C. Szepesvári. Bandit Based Monte-Carlo Planning. *European Conference on Machine Learning (ECML)*, 17:282–293, 2006.

[96] Z. Kolter and A. Ng. Near Bayesian Exploration in Polynomial Time. *ICML*, 26:513–520, 2009.

[97] N. Komodakis, N. Paragios, and G. Tziritas. MRF Energy Minimization and Beyond via Dual Decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):531–552, 2011.

[98] V. Konda and J. Tsitsiklis. Actor-critic algorithms. *NIPS*, 13:1008–1014, 1999.

[99] V. R. Konda and J. N. Tsitsiklis. On Actor-Critic Algorithms. *SIAM J. Control Optim.*, 42(4):1143–1166, 2003.

[100] F. R. Kschischang, B. J. Frey, and H-A. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[101] A. Kumar and S. Zilberstein. Message-Passing Algorithms for Large Structured Decentralized POMDPs. *AAMAS*, 10:1087–1088, 2011.

[102] A. Kumar, S. Zilberstein, and M. Toussaint. Scalable Multiagent Planning Using Probabilistic Inference. *IJCAI*, 22:2140–2146, 2011.

[103] H. Kushner and G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer, 2003.

[104] T. Lang and M. Toussaint. Probabilistic Backward and Forward Reasoning in Stochastic Relational Worlds. *ICML*, 27:583–590, 2010.

[105] K. Lange. A Gradient Algorithm Locally Equivalent to the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(2):425–437, 1995.

[106] K. Lange. A Quasi-Newton Acceleration of the EM Algorithm. *Statistica Sinica*, 5:1–18, 1995.

[107] R. Little and D. Rubin. *Statistical Analysis with Missing Data*. Wiley-Blackwell, 2002.

[108] M. Littman, T. Dean, and L. P. Kaelbling. On the Complexity of Solving Markov Decision Problems. *UAI*, 11:394–402, 1995.

[109] M. Littman, R. Sutton, and S. Singh. Predictive Representations of State. *NIPS*, 15:1555–1561, 2001.

[110] Littman, M. Memoryless Policies: Theoretical Limitations and Practical Results. *Simulation of Adaptive Behaviour*, 3:238–245, 1994.

[111] D. MacKay. Maximum Likelihood and Covariant Algorithms for Independent Component Analysis. Research report, University of Cambridge, 1996.

[112] D. J. C. MacKay. Ensemble learning for hidden Markov models. `http://www.inference.phy.cam.ac.uk/mackay/abstracts/ensemblePaper.html`, 1997.

[113] P. Marbach and J. Tsitsiklis. Simulation-Based Optimisation of Markov Reward Processes. *IEEE Transactions on Automatic Control*, 46(2):191–209, 2001.

[114] N. Meuleau, L. Peshkin, K Kim, and L. Kaelbling. Learning Finite-State Controllers for Partially Observable Environments. *UAI*, 15:427–436, 1999.

[115] Minka, T. P. Expectation Propagation for Approximate Bayesian Inference. *UAI*, 17:362–369, 2001.

[116] A. Miyamae, Y. Nagata, I. Ono, and S. Kobayashi. Natural Policy Gradient Methods with Parameter-Based Exploration for Control Tasks. *NIPS*, 23:1660–1668, 2010.

[117] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Encyclopaedia of Complexity Results for Finite-Horizon Markov Decision Process Problems. Research Report TR 273-97, University of Kentucky, 1997.

[118] L. Nazareth. Some Recent Approaches to Solving Large Residual Nonlinear Least Squares Problems. *SIAM Review*, 22(1):1–11, 1980.

[119] R. Neal and G. Hinton. A View of the EM Algorithm That Justifies Incremental, Sparse and Other Variants. *Learning in Graphical Models*, pages 355–368, 1999.

[120] A Ngo, Y. Hwanjo, and C. TaeChoong. Hessian Matrix Distribution for Bayesian Policy Gradient Reinforcment Learning. *Information Sciences*, 181:1671–1685, 2011.

[121] J. Nocedal and S. Wright. *Numerical Optimisation*. Springer, 2006.

[122] B. K. Oksendal. *Stochastic Differential Equations: An Introduction with Applications*. Springer, 2003.

[123] J. Peters and S. Schaal. Natural Actor-Critic. *Neurocomputing*, 71(7-9):1180–1190.

[124] J. Peters and S. Schaal. Policy Gradient Methods for Robotics. *IROS*, 21:2219–2225, 2006.

[125] K. B. Petersen and O. Winther. The EM-algorithm in Independent Component Analysis. *ICASSP*, 5:169–172, 2005.

[126] M. Petrik and S. Zilberstein. A Bilinear Programming Approach for Multiagent Planning. *Journal of Artificial Intelligence Research*, 35:235–274, 2009.

[127] L. Pontryagin, V. Boltyanskii, R. Gamkrelidze, and E. Mischenko. *The Mathematical Theory of Optimal Processes*. Wiley, 1962.

[128] P. Poupart and C. Boutilier. Bounded Finite State Controllers. *NIPS*, 16, 2004.

[129] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An Analytic Solution to Discrete Bayesian Reinforcement Learning. *ICML*, 23:697–704, 2006.

[130] M. H. Protter and C. B. Morrey. *A First Course in Real Analysis*. Springer, second edition, 1997.

[131] M. L. Puterman and M. C. Shin. Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*, 24(11):1127–1137, 1978.

[132] C. Rasmussen and M. Deisenroth. Probabilistic inference for fast learning in control. In S. Girgin, M. Loth, R. Munos, P. Preux, and D. Ryabko, editors, *Recent Advances in Reinforcement Learning*, pages 229–242, 2008.

[133] H.E. Rauch, F. Tung, and C. T. Striebel. Maximum Likelihood Estimates of Linear Dynamic Systems. *AIAA*, 3(8):1445–1450, 1965.

[134] M. I. Reiman and A. Weiss. Sensitivity Analysis via Likelihood Ratios.

[135] M. I. Reiman and A. Weiss. Sensitivity Analysis for Simulations via Likelihood Ratios. *Operations Research*, 37(5):830–844, 1986.

[136] Richter, S. and Aberdeen, D. and Yu, J. Natural Actor-Critic for Road Traffic Optimisation. *NIPS*, 19:1169–1176, 2007.

[137] R. Y. Rubinstein. How to Optimise Complex Stochastic Systems From a Single Sample Path by the Score Function Method. *Annsls of Operations Research*, 27:175–211, 1991.

[138] T. Rückstieb, M. Felder, and J. Schmidhuber. State-Dependent Exploration for Policy Gradient Methods. *European Conference on Machine Learning (ECML)*, 2:234–249, 2008.

[139] G. Rummery and M. Niranjan. On-Line Q-Learning Using Connectionist Systems. Technical report, Cambridge University, 1994.

[140] R. Salakhutdinov, S. Roweis, and Z. Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. *ICML*, (20):672–679, 2003.

[141] Salakhutdinov, R. and Roweis, S. and Ghahramani, Z. On the Convergence of Bound Optimization Algorithms. *UAI*, 19:509–516, 2003.

[142] L. Saul, T. Jaakkola, and M. Jordan. Mean Field Theory for Sigmoid Belief Networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[143] L. Saul and M. Jordan. Exploiting Tractable Substructures in Intractable Networks. volume 9, pages 486–492, 1995.

[144] J. Schaeffer, M. Hlynka, , and V. Jussila. Temporal Difference Learning Applied to a High-Performance Game-Playing Program. *IJCAI*, 17:529–534, 2001.

[145] R. Schoknecht. Optimality of Reinforcement Learning Algorithms with Linear Function Approximation. *NIPS*, 15:1555–1562, 2002.

[146] N. Schraudolph and T. Graepel. Combining Conjugate Direction Methods with Stochastic Approximation of Gradients. *ICONIP*, 9:853–856, 2002.

[147] N. Schraudolph, J. Yu, and S. Gunter. A Stochastic Quasi-Newton Method for Online Convex Optimization. *AISTATS*, 11:433–440, 2007.

[148] S. Scott. Bayesian Estimation of Hidden Markov Chains: A Stochastic Implementation. *Journal of the American Statistical Association*, 97:337–351, 2002.

[149] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Policy Gradients with Parameter-Based Exploration for Control. *ICANN*, 1:387–396, 2008.

[150] Y. Serin and V. Kulkarni. Markov Decision Processes Under Observability Constraints. *Mathematical Methods of Operational Research*, 61:311–328, 2005.

[151] R.D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36:589–604, 1988.

[152] D. Silver, R. Sutton, and M. Müller. Temporal-Difference Search in Computer Go. *Machine Learning*, 87(2):183–219, 2012.

[153] D. Sontag, A. Globerson, and T. Jaakkola. Introduction for Dual Decomposition for Inference. In S. Sra, S. Nowozin, and S. Wright, editors, *Optimisation for Machine Learning*. MIT Press, 2010.

[154] J. Sorg, S. Singh, and R. Lewis. Variance-Based Rewards for Approximate Bayesian Reinforcement Learning. *UAI*, 26:564–571, 2010.

[155] M. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modelling and Control*. John Wiley & Sons, 2005.

[156] R. Stengel. *Optimal Control and Estimation*. Dover, 1993.

[157] A. Strehl and M. Littman. A Theoretical Analysis of Model-Based Interval Estimation. *ICML*, 22:857–864, 2005.

[158] M. Strens. A Bayesian Framework for Reinforcement Learning. *ICML*, 17:943–950, 2000.

[159] H. Sussmann and J. Willems. 300 Years of Optimal Control: From The Brachstochrone to the Maximum Principle. *Control Systems Magazine IEEE*, 17(3):32–44, 1997.

[160] R. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988.

[161] R. Sutton. Generalization in Reinforcment Learning: Successful Examples Using Sparse Coarse Coding. *NIPS*, 9:1038–1044, 1995.

[162] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *NIPS*, 13:1057–1063, 2000.

[163] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[164] G. Tesauro. TD-Gammon, A Self-Teaching Backgammon Program Achieves Master-Level Play. *Neural Computation*, 6:215–219, 1994.

[165] W. Thompson. On the Likelihood that one Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3-4):285–294, 1933.

[166] E. Todorov and W. Li. A Generalised Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Non-Linear Stochastic Systems. *Proceedings of the American Control Conference*, pages 300–306, 2005.

[167] M. Toussaint. Pros and Cons of truncated Gaussian EP in the context of Approximate Inference Control. *NIPS - Workshop on Probabilistic Approaches for Robotics and Control.*, 21, 2009.

[168] M. Toussaint. Robot Trajectory Optimisation Using Approximate Inference. *ICML*, (26):132–139, 2009.

[169] M. Toussaint, L Charlin, and P. Poupart. Hierarchical POMDP Controller Optimization by Likelihood Maximization. *UAI*, 24:562–570, 2008.

[170] M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (PO)MDPs. Research Report EDI-INF-RR-0934, University of Edinburgh, School of Informatics, 2006.

[171] M. Toussaint, A. Storkey, and S. Harmeling. *Bayesian Time Series Models*, chapter Expectation-Maximization methods for solving (PO)MDPs and optimal control problems. Cambridge University Press, 2011. In press. See `userpage.fu-berlin.de/~mtoussai`.

[172] P. Tseng. Solving H-Horizon, Stationary Markov Decision Problems in Time Proportional to log(H) . *Operations Research Letters*, 5(9):287–297, 1993.

[173] J. Tsitsiklis and B. Van Roy. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[174] J. Veness, D. Silver, A. Blair, , and W. Uther. Bootstrapping from Game Tree Search. *NIPS*, 19:1937–1945, 2009.

[175] N. Vlassis, M. Littman, and D. Barber. On the Computational Complexity of Stochastic Controller Optimization in POMDPs. *CoRR*, abs/1107.3090, 2011.

[176] N. Vlassis, M Toussaint, G. Kontes, and S. Piperidis. Learning Model-Free Robot Control by a Monte Carlo EM Algorithm. *Autonomous Robots*, 27(2):123–130, 2009.

[177] V. Volterra. *Variations and Fluctuations of the Number of Individuals in Animal Species Living Together*. McGraw-Hill, 1931.

[178] M. J. Wainwright and M. I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[179] C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8:279–292, 1992.

[180] L. Weaver and N. Tao. The Optimal Reward Baseline for Gradient Based Reinforcement Learning. *UAI*, 17:538–545, 2001.

[181] C. White and D. White. Markov decision processes. *European Journal of Operational Research*, 39:1–16, 1989.

[182] R. Williams. Simple Statistical Gradient Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

Appendix A

Rates of Convergence

The rate of convergence of a sequence is concerned with the speed, in terms of the number of iterations, at which a fixed point is reached. In this appendix we give a brief introduction to the various rates of convergence that we will consider in this work.

Let $\{\boldsymbol{x}_k\}_{k\in\mathbb{N}}$ be a sequence of iterates in $\mathbb{R}^n$ that converge to a point $\boldsymbol{x}^*$. This sequence is said to converge *linearly* if $\exists r \in (0,1)$ *s.t.*

$$\lim_{k\to\infty} \frac{\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}_k - \boldsymbol{x}^*\|} = r, \tag{A.1}$$

for all sufficiently large k. In more descriptive terms (for sufficiently large k) the distance from the fixed point decreases by at least a constant, multiplicative, factor r, which is known as the rate of convergence. Given two linearly convergent sequences, then the sequence with the lower rate of convergence will show the faster convergence (for sufficiently large k). If the limit (A.1) exists and $r = 0$ then the sequence is said to have *super-linear* convergence, whereas in the case $r = 1$ the sequence is said to converge *sub-linearly*.

A sequence is said to have a *quadratic* rate of convergence if

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq M\|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2,$$

for all sufficiently large k and some positive constant M. Note that in quadratic convergence M is only restricted to be positive and not necessarily less than 1. The speed of convergence depends on the constants r and M, which can be problem dependent as well as algorithm dependent. However, regardless of these constants, a sequence with quadratic convergence will always eventually converge faster than a linearly convergent sequence.

To illustrate these definitions we consider the sequences

$$s_k = (0.5)^k, \qquad t_k = k^{-k}, \qquad u_k = \frac{1}{k}, \qquad v_k = (0.5)^{2^k},$$

which respectively have linear, super-linear, sub-linear and quadratic rates of convergence. A plot showing the iterates of these sequences is shown in fig(A.1).

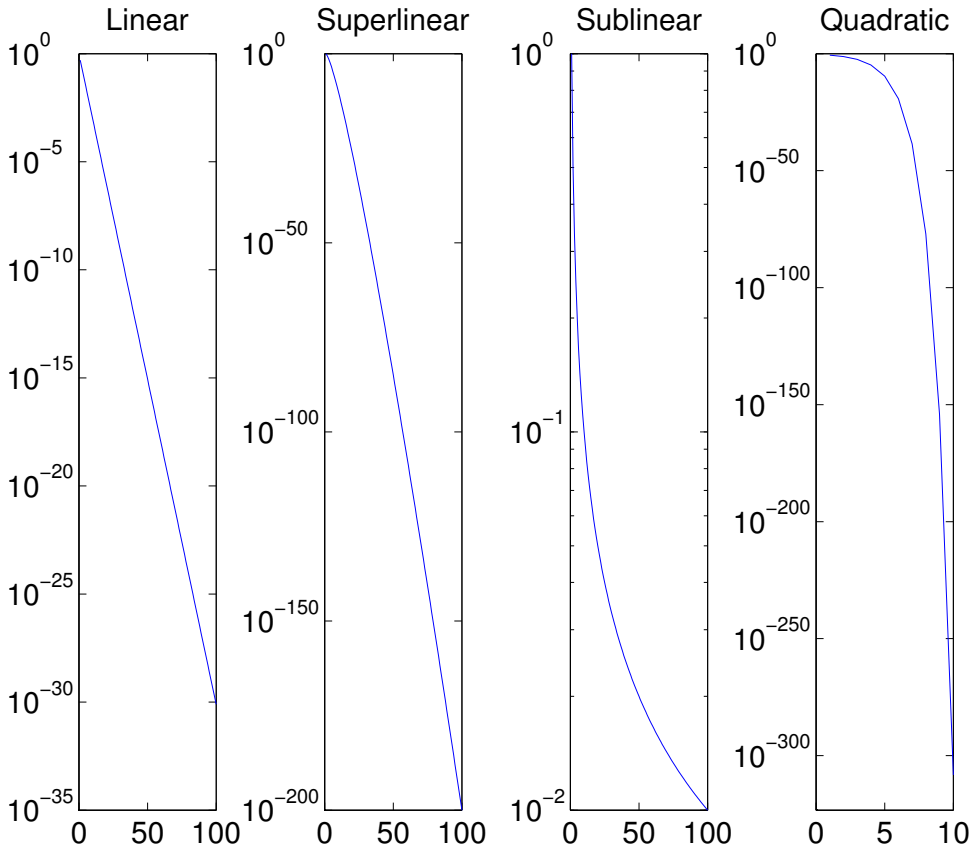


Figure A.1: A graphical illustration of the differing behaviour of linear, super-linear, sub-linear and quadratic convergence.

Appendix B

An Analysis for the Application of Expectation Maximisation to Markov Decision Processes

In addition to the numerous desirable properties of the EM-algorithm there is also a detailed analysis of its convergence properties, see *e.g.* [44, 107, 140, 141]. These analyses typically involve studying the eigenvalues of the Jacobian of the EM-operator around a local optimum of the objective function, with fast, super-linear, convergence as the eigenvalues tend to zero and slow, sub-linear, convergence as the eigenvalues tend to one. While it is possible to calculate the Jacobian of the EM-operator in the case of a MDP with a discrete state-action space it is difficult to analyse the behaviour of the eigenvalues in terms of certain aspects of the MDP, such as the planning horizon or the reward structure. An alternative approach, which we take in this appendix, is to directly analyse the policy updates in terms of these given aspects of the MDP. We do not give any formal rates of convergence, but instead aim to give an intuitive understanding about the convergence behaviour for this particular application of the EM-algorithm. We now give the following lemma, which details the behaviour of the policy update in terms of the reward structure of the Markov Decision Process.

Lemma 4. *Suppose we are given an infinite horizon Markov Decision Process in the discounted rewards framework. Considering this Markov Decision Process separately under the reward functions R_1 and $R_1 + R_2$, then given a policy, π, s.t.*

$$\frac{\pi(a|s)Q_{\pi}^{R_1}(a,s)}{\sum_{a' \in \mathcal{A}} \pi(a'|s)Q_{\pi}^{R_1}(a',s)} \geq \frac{\pi(a|s)Q_{\pi}^{R_2}(a,s)}{\sum_{a' \in \mathcal{A}} \pi(a'|s)Q_{\pi}^{R_2}(a',s)}, \tag{B.1}$$

where the notation $Q_{\pi}^{R}(a,s)$ is used to denote the state-action value function that corresponds to the MDP with reward function R and policy π, then

$$\pi^{R_1}(a|s) \geq \pi^{R_1+R_2}(a|s), \tag{B.2}$$

where π^{R_1} and $\pi^{R_1+R_2}$ denote the policies obtained through respectively applying the EM-algorithm to the given Markov Decision Process with reward functions R_1 and $R_1 + R_2$ and policy π. Additionally,

if the inequality in (B.1) is in the other direction then

$$\pi^{R_1}(a|s) \leq \pi^{R_1+R_2}(a|s).$$

Proof. To prove the lemma we use the fact in the EM-algorithm the policy update in a MDP, with reward function R and policy π, takes the form

$$\pi^{\text{new}}(a|s) = \frac{\pi(a|s)Q_\pi^R(a,s)}{\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^R(a',s)}.$$

This means that the inequality (B.2) has the equivalent form

$$\frac{\pi(a|s)Q_\pi^{R_1}(a,s)}{\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_1}(a',s)} \geq \frac{\pi(a|s)Q_\pi^{R_1+R_2}(a,s)}{\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_1+R_2}(a',s)},$$

and to prove the lemma it is sufficient to prove that the quantity

$$\chi = \pi(a|s)Q_\pi^{R_1}(a,s)\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_1+R_2}(a',s) - \pi(a|s)Q_\pi^{R_1+R_2}(a,s)\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_1}(a',s),$$

is non-negative. As the MDP objective is linear *w.r.t.* the reward the state-action value function, $Q_\pi^{R_1+R_2}(a,s)$, can be written in the equivalent form

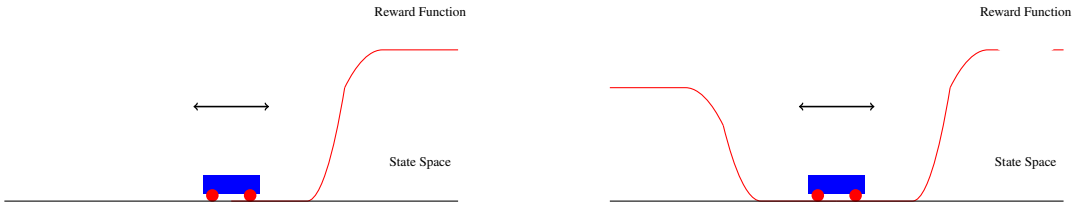
$$Q_\pi^{R_1+R_2}(a,s) = Q_\pi^{R_1}(a,s) + Q_\pi^{R_2}(a,s).$$

This means that χ takes the simpler form

$$\begin{aligned}\chi &= \pi(a|s)Q_\pi^{R_1}(a,s)\sum_{a'\in\mathcal{A}}\pi(a'|s)\Big(Q_\pi^{R_1}(a',s)+Q_\pi^{R_2}(a',s)\Big) \\ &\qquad - \pi(a|s)\Big(Q_\pi^{R_1}(a,s)+Q_\pi^{R_2}(a,s)\Big)\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_1}(a',s), \\ &= \pi(a|s)Q_\pi^{R_1}(a,s)\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_2}(a',s) - \pi(a|s)Q_\pi^{R_2}(a,s)\sum_{a'\in\mathcal{A}}\pi(a'|s)Q_\pi^{R_1}(a',s).\end{aligned}$$

It is clear, under the assumption (B.1), that χ is non-negative and this completes the proof. The reverse inequality follows similarly. □

We now give an illustrative example to highlight the effect that lemma 4 can have on the rate of convergence of the EM-algorithm. Consider the Markov Decision Process depicted figuratively in fig(B.1)(a), which is a one-dimensional problem that we consider to be discretised to allow our analysis to apply. The state space is illustrated by the black line, while the reward function is depicted by the red line and is dependent only upon the state. The agent can move left or right and the optimal policy is always to move to the right. In fig(B.1)(b) a second reward term is added to the reward function of the original MDP, giving a positive reward for being in the *l.h.s.* of the state space. Provided the second reward term is sufficiently small the optimal policy of the initial state is still to travel to the right of the



(a) Unimodal Reward (b) Multimodal Reward

Figure B.1: An illustrative example of the convergence properties of the EM-algorithm applied to discrete Markov Decision Processes. The state space is depicted by the black line and is a one-dimensional line, which we consider to be discretised so that our analysis applies. The reward function is dependent only upon the state and is depicted by the red line. The initial state is in the middle of the state space, which is depicted graphically in the figures. Figure (a) illustrates a unimodal reward function, while figure (b) considers a multimodal reward function.

MDP but, depending on the policy, the convergence of the EM-algorithm will be effected. If the total expected reward *w.r.t.* this second reward term, weighted by the current policy, is larger for the action of moving left than it is for moving right then (B.1) will be satisfied and the convergence of the policy will be slower. An empirical illustration of this behaviour is given in fig(B.2). We can now see how, in this instance, the multimodal reward function effects the rate of convergence of the EM-algorithm. Given the form of the policy update in the EM-algorithm, where the policy update takes the form of a normalised product of the current policy with the total expected reward of the state-action pair, this behaviour is unsurprising. As the difference in the total expected reward of two actions decreases, as it has done in the present example, the convergence becomes slower. Another example that clearly illustrates this behaviour is the chain problem, which we have considered repeatedly in this work. In this case the difference in the total expected reward of actions 'a' and 'b' in state 1 is generally small, even when the policy is close to optimal, and as a result the convergence on the policy in this state is slow.

Lemma 4 highlights an additional property that is peculiar to the EM-algorithm. Suppose we are given an infinite horizon MDP with discounted rewards and reward function, R. Consider a second MDP that is identical to the first with the exception that the reward function takes the form $R_{\text{new}}(a, s) = R(a, s) + c$, where $c \in \mathbb{R}^+$. The objective function of this second MDP takes the form

$$U_{\text{new}}(\boldsymbol{w}) = U(\boldsymbol{w}) + \frac{c}{1-\gamma}, \tag{B.3}$$

where $U(\boldsymbol{w})$ is the objective function of the original MDP. It is simple to see from (B.3) that the curvature properties of $U_{\text{new}}(\boldsymbol{w})$ and $U(\boldsymbol{w})$, such as the gradient and Hessian, are identical. Any gradient-based algorithm is therefore invariant to the addition of a positive constant to the reward function. However, the same is not true of the EM-algorithm. Making the identifications $R_1 \equiv R$ and $R_2 \equiv c$ then (B.1) becomes

$$\frac{Q_\pi^{R_1}(a, s)}{\sum_{a' \in \mathcal{A}} \pi(a'|s) Q_\pi^{R_1}(a', s)} \geq 1.$$

This inequality states that when $Q_\pi^{R_1}(a, s) \geq \mathbb{E}_{\pi(\cdot|s)}[Q_\pi^{R_1}(\cdot, s)]$, then $\pi^{R_1+R_2}(a|s) \leq \pi^{R_1}(a|s)$. Con-

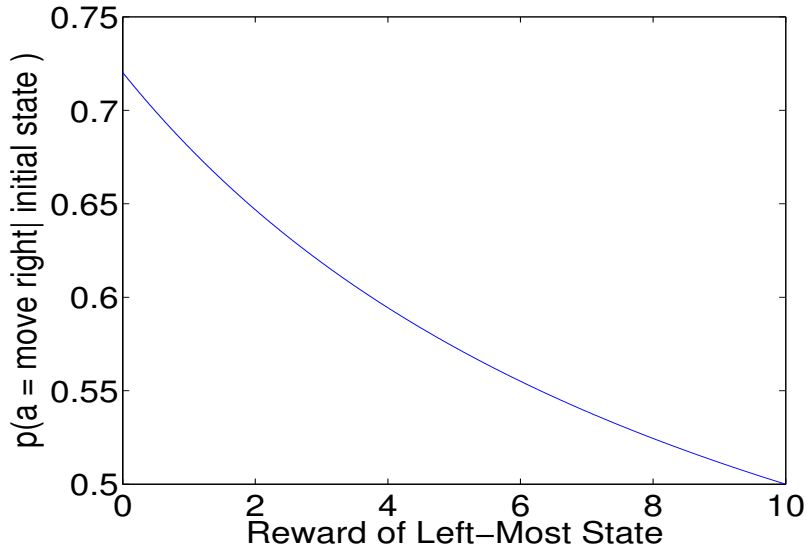


Figure B.2: An illustration of the behaviour of the policy update for the initial state of the MDP depicted in fig(B.1). The initial policy for all states of the MDP is uniform over the two possible actions. The reward of the right-most state is set to 10, while the reward of the left-most state is altered across the various MDPs considered in the experiment. In the plot the magnitude of the second reward term is plotted against the probability of moving right under the new policy, where the new policy is obtained by performing a single policy update using EM. As can be observed the probability of moving right decreases as the magnitude of the second reward term increases, where a fixed-point is reached when the magnitude of the two reward terms is equal.

versely, when $Q_\pi^{R_1}(a,s) \leq \mathbb{E}_{\pi(\cdot|s)}[Q_\pi^{R_1}(\cdot,s)]$, then $\pi^{R_1+R_2}(a|s) \geq \pi^{R_1}(a|s)$. This is a peculiar result and is a direct consequence of the *averaging* behaviour of the policy update in the EM-algorithm. While this result may seem like a pathological case this is not actually true. Recall that to apply the EM-algorithm to MDPs it is necessary that the reward is non-negative, and when this condition is not satisfied it is still possible to apply the EM-algorithm by adding a sufficiently large positive constant to the reward function *s.t.* this condition becomes satisfied under this new reward function. This is possible due to the linearity of the objective *w.r.t.* the reward function and the fact that the reward function is bounded. It can now be seen that such a construction can actually have an effect on the convergence of the algorithm in certain areas of the state-action space.

The example considered in lemma 4 is just one possible such result and similar results can be obtained in different formulations of the MDP problem, such as finite horizons, and different discrete planning models, such as the POMDP with a policy modelled with a FSC. We don't detail these results here as they run along the same lines as lemma 4 and give little additional insight. The update of the control parameters in continuous system is more complicated and obtaining such an intuitive understanding of the convergence analysis of the EM-algorithm is necessarily more complicated and we do not give such an analysis here.

Appendix C

Newton Inference Recursions

In this appendix we detail two novel inference routines for the calculation of $\mathcal{H}_1(\boldsymbol{w})$, which forms part of the Hessian. For ease of reference we now restate the form of $\mathcal{H}_1(\boldsymbol{w})$, namely

$$\mathcal{H}_1(\boldsymbol{w}) = \mathbb{E}_{\tilde{p}(\boldsymbol{z}_{1:t},t;\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{z}_{1:t};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^T \log p(\boldsymbol{z}_{1:t};\boldsymbol{w})\Big],$$

which due to the Markovian structure of the transition dynamics can be written in the equivalent form

$$\mathcal{H}_1(\boldsymbol{w}) = \sum_{t=1}^{H} \sum_{\tau,\tau'=1}^{t} \mathbb{E}_{\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w})}\Big[\nabla_{\boldsymbol{w}} \log p(\boldsymbol{a}|\boldsymbol{s};\boldsymbol{w}) \nabla_{\boldsymbol{w}}^T \log p(\boldsymbol{a}'|\boldsymbol{s}';\boldsymbol{w})\Big].$$

It can now be seen that in order to calculate $\mathcal{H}_1(\boldsymbol{w})$ it is necessary to calculate marginals of the reward weight trajectory distribution of the form $\tilde{p}(\boldsymbol{z}, \boldsymbol{z}', \tau, \tau', t; \boldsymbol{w})$, where $t \in \mathbb{N}_H$ and $\tau, \tau' \in \mathbb{N}_t$ *s.t.* $\tau \neq \tau'$. Individually calculating all the marginals necessary for the construction of $\mathcal{H}_1(\boldsymbol{w})$ would have a run-time that was cubic in the planning horizon. Instead, using methods similar to those detail in section(3.1), it is possible to calculate the summation of these marginals with a run-time that is linear in the planning horizon. We now provide two novel model-based inference routines for the calculation of $\mathcal{H}_1(\boldsymbol{w})$: The first is a forward-backward routine for discrete systems where it is possible to enumerate over the state-action space; The second is a RTS-inference routine for linear systems with a possibly non-linear reward structure. A sample-based procedure for the calculation of this matrix is given in [19].

Forward-Backward Inference

In discrete systems where it is feasible to enumerate over $\mathcal{Z}$ it is possible to obtain a linear time forward-backward inference routine to calculate the statistics required for $\mathcal{H}_1(\boldsymbol{w})$. The calculation of $\mathcal{H}_1(\boldsymbol{w})$ requires the calculation of

$$\sum_{t=1}^{H} \sum_{\tau,\tau'=1}^{t} \tilde{p}(\boldsymbol{z}, \boldsymbol{z}', \tau, \tau', t; \boldsymbol{w}),$$

which can be written in the equivalent form

$$\sum_{\tau=1}^{H}\sum_{\tau'=1}^{\tau-1}\sum_{t=\tau}^{H}\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w})+\sum_{\tau=1}^{H}\sum_{t=\tau}^{H}\tilde{p}(\boldsymbol{z},\tau,t;\boldsymbol{w}).$$

The second term is the same as that required for first order methods, which we already know how to calculate, see *e.g.* [171, 59], and so we concentrate on the calculation of the first term. As with first order methods the summation over t corresponds to the summation over future rewards and can be summarised in terms of the state-action value function as follows

$$\sum_{\tau=1}^{H}\sum_{\tau'=1}^{\tau-1}\sum_{t=\tau}^{H}\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w})=\sum_{\tau=1}^{H}\sum_{\tau'=1}^{\tau-1}p(\boldsymbol{z},\boldsymbol{z}',\tau,\tau';\boldsymbol{w})Q_{\tau}(\boldsymbol{z};\boldsymbol{w}).$$

As we have assumed that it is possible to enumerate over $\mathcal{Z}$ the calculation of the state-action value functions can be performed exactly and in linear time. Note also that the $Q_{\tau}(\boldsymbol{z};\boldsymbol{w})$ is independent of τ' so that it can be pulled through the summation over τ' to give

$$\sum_{\tau=1}^{H}\sum_{\tau'=1}^{\tau-1}\sum_{t=\tau}^{H}\tilde{p}(\boldsymbol{z},\boldsymbol{z}',\tau,\tau',t;\boldsymbol{w})=\sum_{\tau=1}^{H}Q_{\tau}(\boldsymbol{z};\boldsymbol{w})\sum_{\tau'=1}^{\tau-1}p(\boldsymbol{z},\boldsymbol{z}',\tau,\tau';\boldsymbol{w}).$$

Now to obtain a linear time formulation for the calculation of $\mathcal{H}_1(\boldsymbol{w})$ we derive a recursive equation for the calculation of the terms

$$\Lambda_{\tau}(\boldsymbol{z},\boldsymbol{z}';\boldsymbol{w})=\sum_{\tau'=1}^{\tau-1}p(\boldsymbol{z},\boldsymbol{z}',\tau,\tau';\boldsymbol{w}),\qquad\qquad\tau\in\{2,...,H\}.$$

The first term in the recursion, corresponding to $\tau=2$, is equal to $p(\boldsymbol{z}_1,\boldsymbol{z}_2;\boldsymbol{w})$ and is easily obtained. If we now assume that Λ_{τ} has already been calculated then it is possible to obtain $\Lambda_{\tau+1}$ as follows

$$\begin{aligned}\Lambda_{\tau+1}(\boldsymbol{z},\boldsymbol{z}';\boldsymbol{w})&=\sum_{\tau'=1}^{\tau-1}p(\boldsymbol{z},\boldsymbol{z}',\tau+1,\tau';\boldsymbol{w})+p(\boldsymbol{z},\boldsymbol{z}',\tau+1,\tau;\boldsymbol{w}),\\&=\sum_{\boldsymbol{z}''\in\mathcal{Z}}P(\boldsymbol{z}|\boldsymbol{z}'';\boldsymbol{w})\sum_{\tau'=1}^{\tau-1}p(\boldsymbol{z}'',\boldsymbol{z}',\tau,\tau';\boldsymbol{w})+p(\boldsymbol{z},\boldsymbol{z}',\tau+1,\tau;\boldsymbol{w}),\\&=\sum_{\boldsymbol{z}''\in\mathcal{Z}}P(\boldsymbol{z}|\boldsymbol{z}'';\boldsymbol{w})\Lambda_{\tau}(\boldsymbol{z}'',\boldsymbol{z}';\boldsymbol{w})+p(\boldsymbol{z},\boldsymbol{z}',\tau+1,\tau;\boldsymbol{w}).\end{aligned}$$

The last term in this recursion is again easily obtained as it is simply the state-action marginal for adjacent time-points. It is now straightforward to see that all these operations can be performed in linear time *w.r.t.* the planning horizon. We omit the details here but an extension to an infinite planning horizon with discounted rewards is not difficult using methods similar to those in [171].

RTS-Inference

We shall now detail how to calculate $\mathcal{H}_1(\boldsymbol{w})$ in the case of a linear system with an arbitrary rewards. See either [77] or [59] for a specification of these models. In these systems the Hessian is of a similar size to

the matrix that is inverted during each iteration of Expectation Maximisation or natural gradient ascent, therefore it is reasonable to expect the Newton method to be feasible in such systems. While a forward-backward procedure such as [77] can easily be extended to Newton's method the run-time would be cubic in the planning horizon. Instead we derive our inference algorithm in terms of the RTS-inference paradigm [59] which has a linear run-time *w.r.t.* the planning horizon. In the following the state-action value functions will take the Rauch Tung Striebel form given in chapter(2) as opposed to their standard form. Using the usual manipulations of the RTS-inference routine the terms in $\mathcal{H}_1(\boldsymbol{w})$ can be written in the form of a summation of the cross-moments of the reward-weighted trajectory distribution. In particular for each $\tau' \in \mathbb{N}_H$ it is necessary to calculate the terms

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_{\tau'}\right], \qquad i,j \in \mathbb{N}_{n_z},$$

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_{\tau'} \boldsymbol{z}^k_{\tau'}\right], \qquad i,j,k \in \mathbb{N}_{n_z},$$

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_\tau \boldsymbol{z}^k_{\tau'}\right], \qquad i,j,k \in \mathbb{N}_{n_z},$$

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_\tau \boldsymbol{z}^k_{\tau'} \boldsymbol{z}^l_{\tau'}\right], \qquad i,j,k,l \in \mathbb{N}_{n_z}.$$

Where therefore detail the efficient calculation of these summation of cross-moments. In particular we are interested calculating these quantities with a computational complexity that is linear in the planning horizon. Extensions to an infinite planning horizon with discounted rewards can be obtained using the methods of section(3.2.2), but we omit the details here.

Using standard formulae for the conditional distribution of a multivariate Gaussian we have that for any $\tau, \tau' \in \mathbb{N}_H$, *s.t.* $\tau < \tau'$,

$$\boldsymbol{z}_\tau = \overleftarrow{G}_{\tau\tau'} \boldsymbol{z}_{\tau'} + \overleftarrow{\boldsymbol{m}}_{\tau\tau'} + \overleftarrow{\boldsymbol{\eta}}_{\tau\tau'}, \tag{C.1}$$

where

$$\overleftarrow{G}_{\tau\tau'} = \Sigma_{\tau\tau'}\Sigma^{-1}_{\tau'\tau'}, \qquad \overleftarrow{\boldsymbol{m}}_{\tau\tau'} = \boldsymbol{\mu}_\tau - \Sigma_{\tau\tau'}\Sigma^{-1}_{\tau'\tau'}\boldsymbol{\mu}_{\tau'}, \tag{C.2}$$

and $\overleftarrow{\boldsymbol{\eta}}_{\tau\tau'}$ is a zero-mean Gaussian distributed random variable with covariance

$$\overleftarrow{\Sigma}_{\tau\tau'} = \Sigma_{\tau\tau} - \Sigma_{\tau\tau'}\Sigma^{-1}_{\tau'\tau'}\Sigma_{\tau'\tau}.$$

For each $\tau' \in \mathbb{N}_H$ the first two cross-moment equations are linear $\boldsymbol{z}_\tau$, where $\tau < \tau'$, and so these two terms can be obtained easily as follows. Due to (C.1) the first term takes the form

$$\begin{aligned}\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_{\tau'}\right] &= \sum_{\tau=1}^{\tau'-1} \mathbb{E}_{Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\left(\overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{z}_{\tau'} + \overleftarrow{\boldsymbol{m}}^i_{\tau\tau'}\right)\boldsymbol{z}^j_{\tau'}\right], \\ &= \sum_{\tau=1}^{\tau'-1}\left\{\overleftarrow{G}_{\tau\tau'}(i,:)\Sigma_{Q_{\tau'}}(:,j) + \overleftarrow{\boldsymbol{m}}^i_{\tau\tau'}\boldsymbol{\mu}^j_{Q_{\tau'}}\right\}.\end{aligned}$$

The notation $A(i,:)$ is used to denote the i^{th} row of the matrix A, with similar notation for the columns of A. We use the notation from chapter(2) to denote the first two moments of the RTS state-action value functions, *i.e.* $\{\boldsymbol{\mu}_{Q_{\tau'}}\}_{\tau' \in \mathbb{N}_H}$ and $\{\Sigma_{Q_{\tau'}}\}_{\tau' \in \mathbb{N}_H}$. Due to the form of $\overleftarrow{G}_{\tau\tau'}$ and $\overleftarrow{\boldsymbol{m}}_{\tau\tau'}$ in (C.2) this term can rewritten into the form

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_{\tau'}\right] = \left\{\sum_{\tau=1}^{\tau'-1} \Sigma_{\tau\tau'}(i,:)\right\}\Sigma^{-1}_{\tau'\tau'}\left\{\Sigma_{Q_{\tau'}}(:,j) - \boldsymbol{\mu}_{\tau'}\boldsymbol{\mu}^j_{Q_{\tau'}}\right\} + \left\{\sum_{\tau=1}^{\tau'-1} \boldsymbol{\mu}^i_\tau\right\}\boldsymbol{\mu}^j_{Q_{\tau'}}.$$

A similar calculation can be performed for the second term and gives

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q^{\text{rts}}_{\tau'}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}^i_\tau \boldsymbol{z}^j_{\tau'} \boldsymbol{z}^k_{\tau'}\right] = \left\{\sum_{\tau=1}^{\tau'-1} \Sigma_{\tau\tau'}(i,:)\right\}\Sigma^{-1}_{\tau'\tau'}\left\{\chi_{Q_{\tau'}}(:,k,l) - \boldsymbol{\mu}_{\tau'}\Sigma_{Q_{\tau'}}(k,l)\right\} + \left\{\sum_{\tau=1}^{\tau'-1} \boldsymbol{\mu}^i_\tau\right\}\Sigma_{Q_{\tau'}}(k,l),$$

where $\{\chi_{Q_{\tau'}}\}_{\tau' \in \mathbb{N}_H}$ is used to denote the third moments of the RTS state-action value functions. It can now be seen that the efficient calculation of the first two cross-moment equations, for each $\tau' \in \mathbb{N}_H$, requires a recursive calculation of $\sum_{\tau=1}^{\tau'-1} \boldsymbol{\mu}_\tau$ and $\sum_{\tau=1}^{\tau'-1} \Sigma_{\tau\tau'}$. The first recursion is trivial and we shall detail the second recursion shortly, but first we consider the calculation of the remaining two cross-moment equations.

The calculation of the remaining two terms is more protracted due to the quadratic term in $\boldsymbol{z}_\tau$, for each $\tau < \tau'$. To obtain a linear run-time formulation for these terms it is necessary to consider terms of the form

$$\begin{aligned}\boldsymbol{z}^i_\tau \boldsymbol{z}^j_\tau = \overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{z}_{\tau'}\overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{z}_{\tau'} + \overleftarrow{\boldsymbol{m}}^i_{\tau\tau'}\overleftarrow{\boldsymbol{m}}^j_{\tau\tau'} + \overleftarrow{\boldsymbol{\eta}}^i_{\tau\tau'}\overleftarrow{\boldsymbol{\eta}}^j_{\tau\tau'} \\ + \overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{z}_{\tau'}\overleftarrow{\boldsymbol{m}}^j_{\tau\tau'} + \overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{z}_{\tau'}\overleftarrow{\boldsymbol{m}}^i_{\tau\tau'},\end{aligned} \tag{C.3}$$

where we have neglected terms that are linear in $\overleftarrow{\boldsymbol{\eta}}_{\tau\tau'}$ as this is a zero-mean random variable and these terms are zero in expectation. Now the first term in (C.3) can be written in the form

$$\begin{aligned}\overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{z}_{\tau'}\overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{z}_{\tau'} &= \sum_{s,t}\Sigma_{\tau\tau'}(i,s)\Sigma_{\tau\tau'}(j,t)\sum_{k,l}\Sigma^{-1}_{\tau'\tau'}(s,k)\Sigma^{-1}_{\tau'\tau'}(t,l)\boldsymbol{z}^k_{\tau'}\boldsymbol{z}^l_{\tau'}, \\ &= \tilde{\Sigma}_{\tau\tau'}((i\circ j),:)\tilde{\Sigma}^{-1}_{\tau'\tau'}\tilde{\boldsymbol{z}}_{\tau'},\end{aligned}$$

where we have defined the matrices, $\tilde{\Sigma}_{\tau\tau'}, \tilde{\Sigma}^{-1}_{\tau'\tau'} \in \mathbb{R}^{n_z^2} \times \mathbb{R}^{n_z^2}$, as follows

$$\tilde{\Sigma}_{\tau\tau'}\big(i\circ j, s\circ t\big) = \Sigma_{\tau\tau'}(i,s)\Sigma_{\tau\tau'}(j,t), \qquad \tilde{\Sigma}^{-1}_{\tau'\tau'}\big(i\circ j, s\circ t\big) = \Sigma^{-1}_{\tau'\tau'}(i,s)\Sigma^{-1}_{\tau'\tau'}(j,t).$$

The second term in (C.3) takes the form

$$\overleftarrow{\boldsymbol{m}}^i_{\tau\tau'}\overleftarrow{\boldsymbol{m}}^j_{\tau\tau'} = \boldsymbol{\mu}^i_\tau\boldsymbol{\mu}^j_\tau - \boldsymbol{\mu}^i_\tau\overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{\mu}_{\tau'} - \boldsymbol{\mu}^j_\tau\overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{\mu}_{\tau'} + \overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{\mu}_{\tau'}\overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{\mu}_{\tau'}. \quad \text{(C.4)}$$

To write this quantity in terms of the mean, covariance and cross-covariance of the marginals of the trajectory distribution we introduce the matrix, $\Xi_{\tau\tau'} \in \mathbb{R}^{n_z^2} \times \mathbb{R}^{n_z}$, which is defined as follows

$$\Xi_{\tau\tau'}(i \circ j, k) = \boldsymbol{\mu}^i_\tau\Sigma_{\tau\tau'}(j,k). \quad \text{(C.5)}$$

Under this notation the second and third terms of (C.4) can be written, respectively, as

$$\Xi_{\tau\tau'}(i \circ j,:)\Sigma^{-1}_{\tau'\tau'}\boldsymbol{\mu}_{\tau'}, \qquad\qquad \Xi_{\tau\tau'}(j \circ i,:)\Sigma^{-1}_{\tau'\tau'}\boldsymbol{\mu}_{\tau'},$$

so that (C.4) now takes the form

$$\overleftarrow{\boldsymbol{m}}^i_{\tau\tau'}\overleftarrow{\boldsymbol{m}}^j_{\tau\tau'} = \boldsymbol{\mu}^i_\tau\boldsymbol{\mu}^j_\tau - \Xi_{\tau\tau'}(i \circ j,:)\Sigma^{-1}_{\tau'\tau'}\boldsymbol{\mu}_{\tau'} - \Xi_{\tau\tau'}(j \circ i,:)\Sigma^{-1}_{\tau'\tau'}\boldsymbol{\mu}_{\tau'} + \overleftarrow{G}_{\tau\tau'}(i,:)\boldsymbol{\mu}_{\tau'}\overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{\mu}_{\tau'}.$$

The last term in (C.4) can be calculated in the same manner as the first term in (C.3). The expectation of the third term in (C.3) can be written in the form

$$\begin{aligned}\mathbb{E}_{\mathcal{N}(0,\overleftarrow{\Sigma}_{\tau\tau'})}\left[\overleftarrow{\boldsymbol{\eta}}^i_{\tau\tau'}\overleftarrow{\boldsymbol{\eta}}^j_{\tau\tau'}\right] &= \Sigma_{\tau\tau}(i,j) - \sum_{s,t}\Sigma_{\tau\tau'}(i,s)\Sigma_{\tau\tau'}(j,t)\Sigma^{-1}_{\tau'\tau'}(s,t),\\ &= \Sigma_{\tau\tau}(i,j) - \tilde{\Sigma}_{\tau\tau'}(i \circ j,:)\tilde{\mu}_{\Sigma^{-1}_{\tau'\tau'}},\end{aligned}$$

where the vector $\tilde{\mu}_{\Sigma^{-1}_{\tau'\tau'}} \in \mathbb{R}^{n_z^2}$ is defined as follows

$$\tilde{\mu}_{\Sigma^{-1}_{\tau'\tau'}}(i \circ j) = \Sigma^{-1}_{\tau'\tau'}(i,j).$$

The final terms in (C.3) which can be written in the form

$$\begin{aligned}\overleftarrow{\boldsymbol{m}}^i_{\tau\tau'}\overleftarrow{G}_{\tau\tau'}(j,:)\boldsymbol{z}_{\tau'} &= \boldsymbol{\mu}^i_\tau\sum_s\overleftarrow{G}_{\tau\tau'}(j,s)\boldsymbol{z}^s_{\tau'} - \sum_{s,t}\overleftarrow{G}_{\tau\tau'}(i,s)\overleftarrow{G}_{\tau\tau'}(j,t)\boldsymbol{z}^s_{\tau'}\boldsymbol{\mu}^t_{\tau'}\\ &= \Xi_{\tau\tau'}(i \circ j,:)\Sigma^{-1}_{\tau'\tau'}\boldsymbol{z}_{\tau'} - \tilde{\Sigma}_{\tau\tau'}((i \circ j),:)\tilde{\Sigma}^{-1}_{\tau'\tau'}\tilde{\mu}_{z_{\tau'}\mu_{\tau'}}\end{aligned}$$

where the vector $\tilde{\mu}_{z_{\tau'}\mu_{\tau'}} \in \mathbb{R}^{n_z^2}$ is defined as

$$\tilde{\mu}_{z_{\tau'}\mu_{\tau'}} = \boldsymbol{z}_{\tau'}(i)\mu_{\tau'}(j).$$

Collecting all of there terms together gives

$$\boldsymbol{z}_\tau^i \boldsymbol{z}_\tau^j = \tilde{\Sigma}_{\tau\tau'}(i \circ j, :)\left\{\tilde{\Sigma}_{\tau'\tau'}^{-1}\left(\tilde{\boldsymbol{z}}_{\tau'} + \tilde{\boldsymbol{\mu}}_{\tau'}\right) - \tilde{\mu}_{\Sigma_{\tau'\tau'}^{-1}}\right\} + \left\{\Xi_{\tau\tau'}(i \circ j, :) + \Xi_{\tau\tau'}(j \circ i, :)\right\}\Sigma_{\tau'\tau'}^{-1}\left\{\boldsymbol{z}_{\tau'} - \boldsymbol{\mu}_{\tau'}\right\} + \boldsymbol{\mu}_\tau^i \boldsymbol{\mu}_\tau^j + \Sigma_{\tau\tau}(i,j) - \tilde{\Sigma}_{\tau\tau'}(i \circ j, :)\tilde{\Sigma}_{\tau'\tau'}^{-1}\tilde{\mu}_{z\mu} - \tilde{\Sigma}_{\tau\tau'}(j \circ i, :)\tilde{\Sigma}_{\tau'\tau'}^{-1}\tilde{\mu}_{\mu z},$$

which can be used directly to calculate the remaining two terms

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q_{\tau'}^{\text{rts}}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}_\tau^i \boldsymbol{z}_\tau^j \boldsymbol{z}_{\tau'}^k\right], \qquad i,j,k \in \mathbb{N}_{n_z},$$

$$\sum_{\tau=1}^{\tau'-1} \mathbb{E}_{\overleftarrow{p}(\boldsymbol{z}_\tau|\boldsymbol{z}_{\tau'})Q_{\tau'}^{\text{rts}}(\boldsymbol{z}_{\tau'};\boldsymbol{w})}\left[\boldsymbol{z}_\tau^i \boldsymbol{z}_\tau^j \boldsymbol{z}_{\tau'}^k \boldsymbol{z}_{\tau'}^l\right], \qquad i,j,k,l \in \mathbb{N}_{n_z}.$$

To complete derivation it is necessary to derive recursive equations for the calculation of

$$\Lambda_\tau = \sum_{\tau'=1}^{\tau-1} \Sigma_{\tau'\tau}, \qquad \Xi_\tau = \sum_{\tau'=1}^{\tau-1} \Xi_{\tau'\tau}.$$

Both of these recursions can be calculated efficiently using the structure of the cross-covariance matrix, where for $\tau' > \tau$ we have $\Sigma_{\tau\tau'+1} = \Sigma_{\tau\tau'}F^T$. The first term in the first recursion takes the form $\Lambda_2 = \Sigma_{12}$, which is immediately available from the system dynamics. Given Λ_τ, for some $\tau \in \{2, ..., H-1\}$, then the next term in the recursion is given by

$$\Lambda_{\tau+1} = \sum_{\tau'=1}^{\tau} \Sigma_{\tau'\tau+1} = \left\{\sum_{\tau'=1}^{\tau-1} \Sigma_{\tau'\tau} + \Sigma_{\tau\tau}\right\}F^T = \left\{\Lambda_\tau + \Sigma_{\tau\tau}\right\}F^T.$$

In the next recursion we have a sum of outer products between the mean vectors and the cross-covariance matrices of the state-action occupancy marginals of the trajectory distribution,

$$\Xi_\tau = \sum_{\tau'=1}^{\tau-1} \Xi_{\tau'\tau},$$

where $\Xi_{\tau'\tau}$ is defined as in (C.5). The first term in the recursion is given by Ξ_{12} and is available from the occupancy marginals of the trajectory distribution. Given Ξ_τ the next term in the recursion is given by

$$\begin{aligned}\Xi_{\tau+1} &= \sum_{\tau'=1}^{\tau} \Xi_{\tau'\tau+1} = \sum_{\tau'=1}^{\tau-1} \Xi_{\tau'\tau+1} + \Xi_{\tau\tau+1}, \\ &= \left\{\sum_{\tau'=1}^{\tau-1} \Xi_{\tau'\tau} + \Xi_{\tau\tau}\right\}F^T = \left\{\Xi_\tau + \Xi_{\tau\tau}\right\}F^T.\end{aligned}$$

It can be seen from (C.5) that the terms of the form $\{\Xi_{\tau\tau}\}_{\tau=1}^H$ can be obtained from the state-action marginals of the trajectory distribution.

Appendix D

Expectation Maximisation with Deterministic Policies

It is well-known that there are many instances where the EM-algorithm can exhibit extremely slow, sub-linear, rate of convergence. Various acceleration methods have been introduced in the EM literature, including hybrid algorithms that switch to an alternative optimisation algorithm at a given point [140] and Aitken acceleration methods [44]. Similarly, slow convergence has been noted in the case of MDPs and one attempt to increase the rate of convergence (for this particular application of the EM-algorithm) has been made in [170]. The authors of that method exploit the fact that, in the case of MDPs, it is sufficient to search over the space of deterministic policies to obtain global optimality of the MDP objective. Unfortunately this algorithm, which is referred to as *greedy EM* in [170], is not a true EM-algorithm but instead a reformulation of policy iteration. Using the same intuition, but in a more formally correct manner, we now construct an actual EM-algorithm for the restricted search space of deterministic policies.

Expectation Maximisation with Deterministic Policies

To obtain such an algorithm we restrict the policy space to deterministic policies, where $\pi(a|s) = \delta(a, a^*(s))$, and run through the same procedure as in section(4.1.2). The function $a^*(\cdot) : \mathcal{S} \to \mathcal{A}$ maps states to actions and we need to find the mapping that optimises the energy. Expressed in this form the energy becomes

$$E(a^*) = \sum_{t=1}^{H} \sum_{\tau=1}^{t-1} \sum_{s_{\tau+1}, s_\tau \in \mathcal{S}} q(s_{\tau+1}, s_\tau, t) \log p(s_{\tau+1}|s_\tau, a^*(s_\tau)) + \sum_{t=1}^{H} \sum_{s_t \in \mathcal{S}} q(s_t, t) \log R(s_t, a^*(s_t)). \tag{D.1}$$

In contrast to the non-deterministic energy (2.13) we now have an additional term from the reward function. Note that this shows the non-commutative nature of taking the deterministic policy limit and optimising the additional term from the utility cannot be obtained from taking the deterministic limit of (2.13). Our procedure then results in an EM-algorithm in the deterministic case, as opposed to the policy iteration approach of [170]. In our EM approach, for each state, s, we now determine the action, a, that

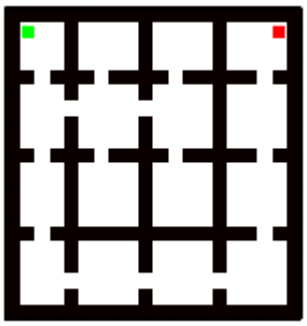


Figure D.1: Maze considered in the MDP experiments. The walls are black, with initial state in the top left corner (green), the goal state in the top right corner (red) and the rest of the maze in white. There are in total 240 states.

maximizes the energy, equation (D.1). Since transition probabilities are stationary, this corresponds to finding for each state, s, the action, a, that maximises

$$\sum_{s' \in \mathcal{S}} \left\{ \sum_{t=1}^{H} \sum_{\tau=1}^{t-1} q(s_{\tau+1} = s', s_\tau = s, t) \log p(s'|s, a) + \left\{ \sum_{t=1}^{H} q(s_t = s, t) \right\} \log R(s, a). \right.$$

The E-step for the q-distribution is as before, expect that we also require the two-time marginals $q(s_{\tau+1} = s', s_\tau = s, t)$.

Experiment

We compare the convergence of the deterministic EM-algorithm with the standard EM-algorithm on solving the problem in fig(D.1) with $\gamma = 1$ and horizon $T = 100$. As can be seen in fig(D.2) our algorithm converges to the optimal policy after the first M-step, whereas the stochastic policy EM-algorithm has slower convergence. For comparisons we also ran policy iteration on this problem and noted that it too converged after the first policy update.

Freezing of Expectation Maximisation

The M-step updates in the EM-algorithm characteristically freeze, in a deterministic or near-deterministic observation distribution, leading to extremely small increases in the log-likelihood. This problem occurs in our EM approach when the transitions and the policy are both deterministic, or close to deterministic. In this case all, or most of, the weight of the q-distribution is put onto the single state-action trajectory that is dictated by the policy and the transition, and the M-step performs the trivial update $\pi_{\text{new}} = \pi_{\text{old}}$. To counter this problem it is possible to add 'anti-freeze' to the environment, rendering it non-deterministic, and then solve the MDP in this new environment. For each state we define the new transition $p_\epsilon(s'|s, a)$ as a convex combination of the transition with a distribution

$$p_\epsilon(s'|s, a) = (1 - \epsilon)p(s'|s, a) + \epsilon \Gamma_s(s')$$

where $\epsilon \in [0, 1)$ and $\Gamma_s(s')$ is an arbitrary probability distribution and then solve the MDP $\langle \mathcal{S}, \mathcal{A}, R, p_\epsilon \rangle$. The idea behind this is encourage 'exploration' during the E-step and therefore enable the algorithm to escape local minima, similar to ϵ-greedy policies used in various Monte-Carlo solution methods to MDPs

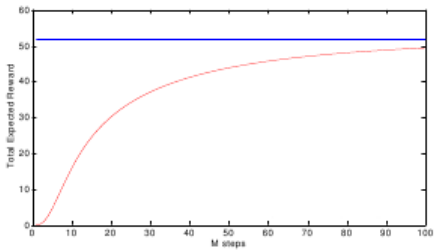


Figure D.2: The deterministic policy EM-algorithm (blue) compared with the standard EM-algorithm (red) on the maze problem in fig(D.1). The discount factor was set to $\gamma = 1$ and the horizon was set to $H = 100$. The algorithms each performed 100 M-steps and were initialized with the same uniform policy. The plot shows the results of the deterministic policy EM-algorithm (blue) and the standard EM-algorithm (red).

[163]. We explain below how the above technique can be both theoretically and practically justified.

Anti-Freeze for Expectation Maximisation

Standard EM-learning

To explain the general problem of freezing in EM and a possible solution, consider a distribution of the form

$$p(v|\theta) = \sum_h p(v|h,\theta)p(h),$$

for which our task is to find the θ that maximises $p(v|\theta)$, given an observed value for v. Treating h as a hidden variable, we may apply the EM-algorithm for which the E-step is

$$q(h|\theta_{\text{old}}) \propto p(v|h,\theta_{\text{old}})p(h),$$

and the M-step sets

$$\theta_{\text{new}} = \operatorname*{argmax}_\theta \mathbb{E}_{p(h|\theta_{\text{old}})}\Big[\log p(v,h|\theta)\Big] = \operatorname*{argmax}_\theta \mathbb{E}_{p(h|\theta_{\text{old}})}\Big[\log p(v|h,\theta)\Big],$$

since $p(h)$ is independent of θ. For a deterministic observation distribution, $p(v|h) = \delta(v, f(h|\theta))$, for some function $f(h|\theta)$ with parameters θ, we have

$$p(h|\theta_{\text{old}}) \propto \delta(v, f(h|\theta))p(h),$$

so that the M-step sets

$$\theta_{\text{new}} = \operatorname*{argmax}_\theta \mathbb{E}_{p(h|\theta_{\text{old}})}\Big[\log \delta(v, f(h|\theta))\Big].$$

Since $p(h|\theta_{\text{old}})$ is zero everywhere except that h for which $v = f(h|\theta)$, then the energy is negative infinity for $\theta \neq \theta_{\text{old}}$ and zero when $\theta = \theta_{\text{old}}$. Hence zero is the optimum of the energy, corresponding to a frozen update. This situation occurs in practice, and has been noted, in particular, in the context of Independent Component Analysis [125] although, as explained here, the phenomenon is quite general. One can attempt to heal this behaviour by deriving an EM-algorithm for the distribution

$$p_\epsilon(v|h,\theta) = (1-\epsilon)p(v|h,\theta) + \epsilon n(h), \qquad 0 < \epsilon < 1,$$

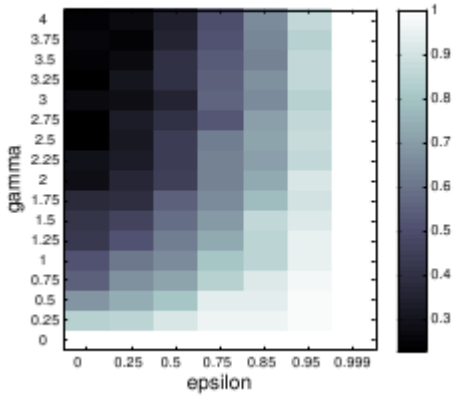


Figure D.3: For each γ, ϵ pair we ran 500 experiments and plot in the figure the fraction of times the correct optimum value is returned by the EM procedure. As γ increases the distribution $p(v|\theta)$ tends to a deterministic distribution and EM optimisation of θ fails. This corresponds to the case $\epsilon = 0$. As we increase noise more noise is included in the process and the EM-algorithm succeeds. Note that for a truly deterministic environment $\gamma = \infty$ a value of $\epsilon < 1$ still suffices, see fig(D.4).

where $n(h)$ is an arbitrary 'anti-freeze' distribution on the hidden variable h. The original deterministic model corresponds to $p_0(v|h, \theta)$. Hence

$$p_\epsilon(v|\theta) \equiv \sum_h p(v|h, \theta)p(h) = (1 - \epsilon)p(v|\theta) + \text{const.}$$

so that applying 'anti-freeze' idea preserves the optima of $p_\epsilon(v|\theta)$ at the same locations as those of $p_\epsilon(v|\theta)$. An EM-algorithm for $p_\epsilon(v|\theta)$, $0 < \epsilon < 1$ satisfies

$$p_\epsilon(v|\theta_{\text{new}}) - p_\epsilon(v|\theta_{\text{old}}) = (1 - \epsilon)[p_0(v|\theta_{\text{new}}) - p_0(v|\theta_{\text{old}})] > 0$$

which implies $p_0(v|\theta_{\text{new}}) - p_0(v|\theta_{\text{old}}) > 0$. Hence the EM-algorithm for the non-deterministic case, $0 < \epsilon < 1$, is guaranteed to increase the likelihood under the deterministic model $p_0(v|\theta)$ at each iteration, unless we are at convergence. Note $n(h)$ can be chosen arbitrarily at each iteration of the EM-algorithm, which can help escape local minima.

MDP EM-learning

To translate the 'anti-freeze' idea into the MDP framework consider an objective

$$F(\theta) = \sum_{s \in \mathcal{S}} R(s)p(s|\theta)$$

for a positive function $R(\cdot) : \mathcal{S} \to \mathbb{R}^+$ with our task being to maximise F with respect to θ. An EM style bounding approach can be derived by defining the auxiliary distribution

$$\tilde{p}(s|\theta) = \frac{R(s)p(s|\theta)}{F(\theta)}$$

so that by considering $KL(q(s)|\tilde{p}(s|\theta))$ for some variational distribution $q(s)$ we obtain the bound

$$\log F(\theta) \geq \mathbb{E}_{q(s)}\Big[\log q(s)\Big] + \mathbb{E}_{q(s)}\Big[\log R(s)\Big] + \mathbb{E}_{q(s)}\Big[\log p(s|\theta)\Big].$$

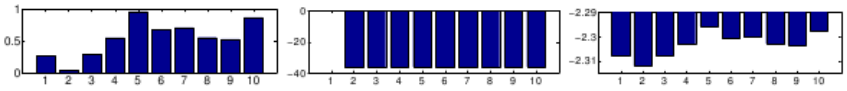


Figure D.4: Maximising a utility $L(\theta) = \sum_{s=1}^{10} \log R(s) p(s|\theta)$, where $p(s|\theta)$ is deterministic, placing all its mass in the state $s = \theta$. Here $R(s)$ is given, being positive and drawn at random. The task is to find the optimal θ, which is equivalent in this case to finding the state s that maximises the given $R(s)$. (a) True utility $L(\theta)$ as a function over the 10 values of θ. The optimal state is $\theta = 5$. (b) The energy for $\epsilon = 0$ and $\theta_{\text{old}} = 1$. The energy is $-\log \infty$ (cut off here at -36) for all but $\theta = \theta_{\text{old}} = 1$, where the energy is zero, displaying the characteristic EM-freezing. (c) Energy of modified distribution using $\epsilon = 0.99$ and $\theta_{\text{old}} = 1$. The new energy has the optimum at the correct place, $\theta = 5$.

The M-step states that the optimal q-distribution is given by $q(s) = \tilde{p}(s|\theta_{\text{old}})$. At the E-step of the algorithm the new parameters θ_{new} are given by maximising the 'energy' term

$$\theta_{\text{new}} = \underset{\theta}{\text{argmax}}\, \mathbb{E}_{p(s|\theta_{\text{old}})} \Big[\log p(s|\theta) \Big].$$

For a deterministic distribution $p(s|\theta) = \delta(s, f(\theta))$ the E-step fails since the energy is negative infinity unless $\theta_{\text{new}} = \theta_{\text{old}}$, in which case the energy is zero. We can attempt to heal this by using the alternative objective

$$F_\epsilon(\theta) = \sum_s R(s) p_\epsilon(s|\theta),$$

with

$$p_\epsilon(s|\theta) = (1-\epsilon) p(s|\theta) + \epsilon n(s), \qquad 0 \leq \epsilon \leq 1,$$

and an arbitrary distribution $n(s)$. Our task is to maximise F with respect to θ. Since

$$F_\epsilon(\theta) = (1-\epsilon) F_0(\theta) + \sum_s n(s) R(s) \tag{D.2}$$

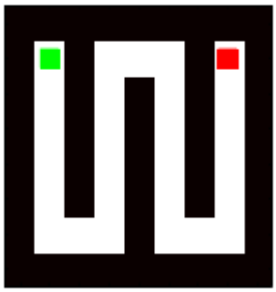
it is clear that $F_\epsilon(\theta)$ has the same optimum as $F_0(\theta)$. Furthermore, since

$$F_\epsilon(\theta_{\text{new}}) - F_\epsilon(\theta_{\text{old}}) = (1-\epsilon)[F_0(\theta_{\text{new}}) - F_0(\theta_{\text{old}})]$$

provided that for $\epsilon > 0$ we can find a θ_{new} such that $F(\theta_{\text{new}}) > F(\theta_{\text{old}})$, then necessarily $F_0(\theta_{\text{new}}) > F_0(\theta_{\text{old}})$. Using this result, we may derive an EM-style algorithm that guarantees to increase $F(\theta)$, unless we are already at the optimum, for $\epsilon > 0$ and can therefore guarantee to increase $F_0(\theta)$. To do so we use

$$\tilde{p}_\epsilon(s|\theta) \equiv \frac{R(s) p(s|\theta)}{F_\epsilon(\theta)},$$

in place of equation (15), and then derive an EM algorithm as before. Currently this proof only holds for a planning horizon of 1 and in longer planning horizons the added noise term no longer separates in the additive manner it does in (D.2). However, this result still provides intuitive justification for the



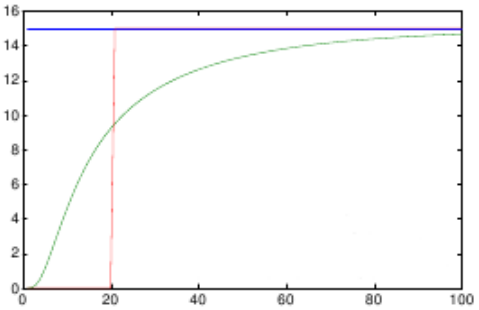


Figure D.5: (Left) Maze considered in the deterministic transition and deterministic policy experiments. The colour scheme is the same as previously, that is the walls are black, the initial state is green (top left corner) and the goal state is red (top right corner), with the remaining states in white. The discount factor was set to $\gamma = 1$ and the horizon set to $H = 40$. There are a total of 27 states. (Right) Anti-freeze experiment for a deterministic policy and environment. We compare the deterministic policy EM-algorithm with anti-freeze, $\epsilon = 0.35$ to the standard EM-algorithm and policy iteration. The two EM algorithms were initialized with the same deterministic policy. The plot shows the results for the deterministic policy EM-algorithm (blue), the standard EM-algorithm (green) and policy iteration (red).

extension of the 'anti-freeze' idea to longer planning horizons.

Applying anti-freeze on a toy problem

To demonstrate that the effect of adding on noise to the deterministic distribution is non-trivial and can heal the EM-algorithm, we carried out a simple experiment, see fig(D.3). We define a distribution $p(s|\theta) \propto \exp(\gamma \mathbb{I}[s = \theta])$ over the states $\mathbb{N}_5$. For each experiment the reward for each state $R(s)$, $s \in \mathbb{N}_5$ is drawn from a uniform distribution between 0 and 1. The task is to find θ that maximises $\sum_s R(s)p(s|\theta)$ using an EM style algorithm. To attempt to resolve freezing we added uniform noise by an amount ϵ to the distribution $p(s|\theta)$. In fig(D.3) we compute the number of times that starting from a random starting state we will, under EM, converge to the correct optimum state. As we can see, for no noise added, $\epsilon = 0$, and in a deterministic limit (γ large) we are in the optimum state only 20% of the time, since no updating occurs, and we start in the correct state with probability 0.2. As we increase ϵ, EM unfreezes and we begin to find the correct optimum. One may have the impression that for a truly deterministic $p(s|\theta)$ we would need to set $\epsilon = 1$ to unfreeze EM and thereby destroy the problem in the process. To show that this is not the case, consider the example in fig(D.4) which considers a truly deterministic $p(s|\theta)$ yet, by applying antifreeze with $\epsilon < 1$, we find the optimum at the correct place.

Applying anti-freeze on a MDP

To illustrate the validity of the 'anti-freeze' method in a MDP setting we consider the simple maze problem in fig(D.5). The transitions are deterministic and the policy is initialised in the worst possible way, taking an action that moves in the opposite direction of the optimal policy, e.g. when the agent is in the initial state it will move upwards instead of downwards. Since the environment is deterministic, the normal deterministic EM-algorithm would perform trivial updates on this problem and freeze. When implementing the anti-freeze idea one has to select the amount of noise and the form of the noise dis-

tribution. In the experiments we use an anti-freeze distribution $\Gamma_s(s')$ to be uniform for all states that satisfy the condition $Q_\pi(a, s) = 0, \forall a \in \mathcal{A}$, i.e. states that have zero probability of receiving a reward under the current policy. The transitions of the remaining states where left unchanged. When adding noise to the transitions was set to 0.35. The results of the experiment are shown in fig(D.5) where we can see that our algorithm converged in a single M-step. We also ran the standard EM-algorithm and policy iteration on this maze problem with the transition probabilities. Policy iteration converges to the optimal policy after roughly 20 policy updates and the EM-algorithm converges more slowly, since it does not explicitly seek a deterministic policy. It should be noted that policy iteration also converges quickly if we add a small amount of noise to the transitions.

Summary

In an attempt to increase the rate of convergence of the EM-algorithm we have considered an EM-algorithm that restricts its search space to the space of deterministic policies. This algorithm uses the insight that it is sufficient to consider the space of deterministic policies to obtain global optimality of the MDP objective. A previous attempt was made at such an extension in [170], but their algorithm is not a true EM-algorithm but actually a reformulation of policy iteration. Perhaps unsurprisingly this restriction to the space of deterministic policies is able to increase the rate of convergence, and sometimes dramatically so, but at the cost of robustness of the algorithm. An important limitation of all EM approaches is that in a deterministic environment (in a standard EM problem this corresponds to the observation distribution being deterministic, and in the MDP case the analog is that the environment transitions are deterministic) EM freezes, and no updating occurs. This can also happen in low noise environments. We introduced a 'anti-freeze' method, that is similar to other 'exploration' techniques, that potentially heals this problem by considering a modified environment being a convex combination of the true environment and a noise distribution. To date we have only applied this 'anti-freeze' idea to toy problems and it would be interesting to study its viability in larger scale problems, as well as other settings such as independent component analysis. Additionally, it is of interest that this 'anti-freeze' approach also increased the convergence of other optimisation techniques, such as policy iteration, and it is of interest to better understand this phenomena.