# LEARNING AND ACTING IN UNKNOWN AND UNCERTAIN WORLDS

by

NOEL WELSH

A thesis submitted to

The University of Birmingham

for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

The University of Birmingham

June 2011

**Abstract**

This dissertation addresses the problem of learning to act in an unknown and uncertain world. This is a difficult problem. Even if a world model is available, an assumption not made here, it is known to be intractable to learn an optimal policy for controlling behaviour (Littman 1996).

Assuming no world model is known leads to two approaches: model-free learning, which attempts to learn to act without a model of the environment, and model learning, which attempts to learn a model of the environment from interactions with the world. Most earlier approaches make a priori assumptions about the complexity of the model or policy required, the upshot of which is that a fixed amount of memory is available to the agent. It is well known that in a noisy environment, the type assumed within, an environment specific amount of memory is required to act optimally. Fixing the capacity of memory before any interactions have occurred is thus a limiting assumption.

The theme of this dissertation is that representing multiple policies or environment models of varying size enables us to address this problem. Both model-free learning and model learning are investigated. For the former, I present a policy search method (usable with a wide range of algorithms) that maintains a population of policies of varying size. By sharing information between policies I show that it can learn near optimal policies for a variety of challenging problems, and that performance is significantly improved over using the same amount of computation without information sharing.

I investigate two approaches to model learning. The first is a variational Bayesian method for learning POMDPs. I show that it achieves superior results to the Bayes-adaptive algorithm (Ross, Chaib-draa and Pineau 2007) using their experimental setup. However, this experimental setup makes strong assumptions about prior information, and I show that weakening these assumptions leads to poor performance. I then address model learning for a simpler model, a topological map. I develop a novel non-parametric Bayesian map that sets no limit of the

model size, and show experimentally that maps can be learned from robot data with weak prior knowledge.

To Bree, Hilary, Rowan, and William.

# ACKNOWLEDGEMENTS

This work would not exist without the assistance of a great many people. I cannot hope to mention everyone here who has helped me along the way; my apologies to those I left out.

This thesis could not have been completed without the support of my wife, Bree. Thanks for your patience. My son Rowan and yet-to-be born son provided encouragement in another form, as well a much needed distraction.

Dave and Dave manned the fort while I was concentrating on other things. Thanks guys. I look forward to what we do next.

Finally, I wouldn't have achieved what I have without the guidance and support of my supervisor, Jeremy Wyatt.

# TABLE OF CONTENTS

# LIST OF ILLUSTRATIONS

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

## 1.1 Motivation and Fundamental Assumptions

In this dissertation I address the problem of learning to act in an unknown and uncertain world. This is an instance of the sequential decision making problem, a very general problem studied in machine learning and other fields. It is concerned with making a sequence of decisions to optimise some long-term measure of success. This problem statement covers a wide range of tasks. Examples of tasks that could be posed within the sequential decision making framework include choosing a sequence of treatments for an ill patient, choosing content to display to an Internet user, and choosing lines of dialogue for a robot in conversation with a human.

A defining characteristic of the sequential decision making problem is that the history of interactions affects the present situation. Thus a choice made in the past, for example a past conversation, affects the outcomes of actions now and in the future. The agent usually receives some information about the world, called an **observation**, before making a choice. If the agent has a model of world, solving it to find a course of actions is known as **planning**. If the agent lacks a model the problem is known as **reinforcement learning**.

I make a fundamental assumption that a world model is not available. Thus I am broadly concerned with reinforcement learning, and in particular with model-free policy learning and learning a model from interactions. This assumption holds in many realistic problems, where the environment is too complex to model, or it is too costly to produce a model. In particular

this assumption is key to developing autonomous intelligent agents.

The information the agent receives about the world, if any, may either contain all information relevant to making decisions, or it might be incomplete or noisy. For example, in the medical domain we might assume that the history of treatments and the results of medical tests are sufficient information to completely characterise the world. For the conversation domain we usually would assume that speech recognition is imperfect, so what the robot thinks it heard is not necessarily what its conversational partner uttered. The case where the information is complete is known as a **completely observable** world; otherwise it is **partially observable**.

I make another fundamental assumption: that the world is partially observable. This raises the issue of memory capacity. It is well known that in a partially observable world the performance of a decision maker is dependent on its memory capacity (Littman 1996, Singh, Jaakkola and Jordan 1994). For every problem there is some optimal, but unknown, amount of memory. Usually we cannot hope to achieve optimal performance but nonetheless performance is still related to memory capacity. Too much memory will both waste computational resources and lead to overfitting. Not enough memory will lead to poor decisions. A theme running throughout this dissertation is that maintaining a set of policies or models allows us to consider a range of memory capacities.

## 1.2 Structure of the Dissertation

Chapter 2 presents basic background information on the sequential decision making problem. The two primary formalisms, the Markov decision problem (MDP) and the partially observable Markov decision problem (POMDP), are presented along with foundational techniques for learning a policy given a model. In particular this chapter shows that learning an optimal policy from a partially observable Markov decision problem is intractable, and thus one must be prepared to approximate.

There are two ways to tackle sequential decision making given my assumption that a model is not available: either learn a model from data, or learn to act without a model. Background

material on these approaches is covered in Chapter 3 and Chapter 4 respectively.

Chapter 3 covers frequentist and Bayesian approaches to learning, and surveys their application to MDPs and POMDPs. Since there is relatively little work in this area it also covers techniques for a closely related model, the hidden Markov model.

Policy search is the main approach to reinforcement learning in POMDPs and surveyed in Chapter 4. In policy search the policy is represented as some computational machine, for example a finite state machine, and the agent attempts to optimise the parameters of that machine directly given interactions with the world. The two main approaches to this optimisation problem are gradient ascent and evolutionary computing. Relevant methods are surveyed for a variety of policy representations.

Chapter 5 is my first chapter of original work. This chapter addresses policy search, developing an algorithm that maintains a population of policies of varying size. By sharing information between policies I show that it can learn near optimal policies for a variety of challenging problems, and that performance is significantly improved over using the same amount of computation without information sharing.

Chapter 6 turns to model learning, exploring a variational Bayesian method for learning POMDPs from data. I show that this algorithm achieves superior results to the Bayes-adaptive algorithm for POMDPs (Ross et al. 2007). However, the experimental setup makes strong assumptions about prior information, and I show that weakening these assumptions leads to poor performance.

In Chapter 7 I address model learning for a simpler model, a topological map. A topological map in my formalism is a directed graph. I develop a novel Bayesian non-parametric model that sets no a priori limit on the number of states in the model. Experiments show that maps can be learned from robot data, and that my model outperforms an earlier model know as the probabilistic topological map (Ranganathan and Dellaert 2004, Ranganathan and Dellaert 2005, Ranganathan and Dellaert 2006).

## 1.3 Key Contributions

In Chapter 5 I show:

- A simple algorithm that uses a population of policies of varying size to find near optimal policies for a range of task.
- Sharing information between policies allows near optimal policies to be learned for a variety of challenging problems.
- Performance using information sharing is significantly improved over using the same amount of computation without information sharing.

In Chapter 6 I show:

- POMDP models learned using a variational Bayesian EM algorithm out perform those learned with the Bayes-adaptive algorithm under the same experimental setup.
- Without an informative prior useful models are not learned by the variational Bayesian EM algorithm.

In Chapter 7 I show:

- A novel Bayesian non-parametric model for topological maps represented as directed graph. My model does not require the number of nodes or connectivity between them be specified in advance.
- This model outperforms an earlier model, know as the probabilistic topological map, for learning maps from robot data.

## 1.4 Notation

Some mathematical notation used in this dissertation may be unfamiliar. It is frequently useful to treat a function on a finite domain as an array (specifically, a vector or matrix if the domain has

one or two dimensions, or a tensor for higher dimensions). For example the function $f : A \to \Re$ can be treated as a vector when $A$ is finite, and the function $g : A \times A \to \Re$ can be treated as a matrix. In such cases element access is indicated by a subscript. So $f_a$ is the same as $f(a)$, and $g_{a_1,a_2} = g(a_1, a_2)$.

When taking a slice of a matrix or tensor to obtain an object of lower dimension a dot (i.e. $\cdot$) will be used to indicate dimensions that are allowed to vary. For example $g_{\cdot,a_2}$ indicates the vector obtained from $g$ by varying the first parameter but fixing the second at $a_2$.

A variable collecting a series or sequence of data over a time span is indicated by $x_{1:T}$, where the subscripts indicate the starting time, in this case the first step, and the ending time, in this case the time step $T$ which means the last available time.

CHAPTER 2

# FOUNDATIONS: HOW TO ACT GIVEN A MODEL

## 2.1 Introduction

What sequence of decisions should I make now to get the best result in the future? This is a problem all of us face, and answering it is the aim of research into **sequential decision making**. In this chapter I describe the sequential decision making problem and introduce foundational methods for solving it given a model of the world is available.

Consider the classic mouse maze used in innumerable psychological experiments. An example is shown in Figure 2.1. The mouse attempts to navigate the maze to reach the tasty cheese. Imagine you are the mouse. What decisions would you make? Should you turn left or right at the start of the maze? Should you use the route that worked last time, or explore further to perhaps find a better route?

These are examples of the problems faced in sequential decision making. The essence of such problems is that decisions taken now affect future outcomes in both the long and short term. For example, if the mouse spends time exploring it might not find the cheese before the experimenter ends the experiment. On the other hand the mouse might find a faster route to the cheese, or another source of food it can exploit later. Finding the best sequence of decisions to optimise some measure of success is what is meant by **solving** a sequential decision making problem. Let us suppose for the mouse the measure of success is the total amount of cheese eaten during the period of the experiments, and so solving the problem means to eat the

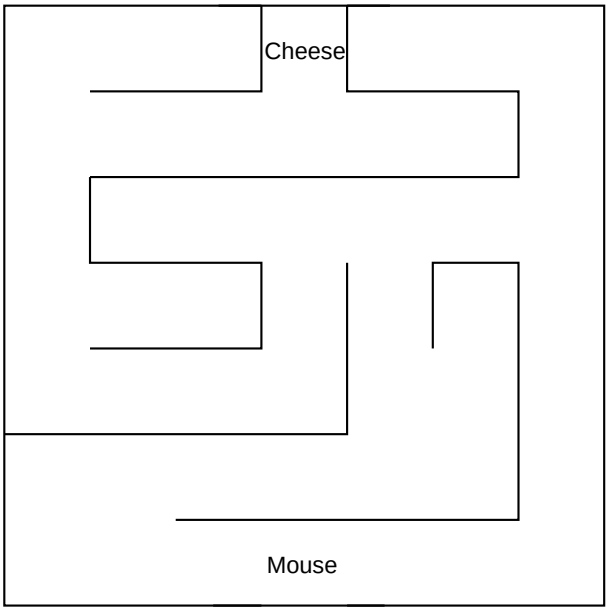


Figure 2.1: An example of a mouse maze as used in numerous psychology experiments. The mouse attempts to navigate the maze to find the tasty cheese.

maximum possible amount of cheese.

### 2.1.1 Overview

The basic terminology of sequential decision making is introduced in Section 2.2. There are two main formalisms used in the field: the Markov decision process (MDP) and the partially observable Markov decision process (POMDP). I am mainly addressing POMDPs in this dissertation, but MDPs provide a simpler introduction to many of the issues that also arise in POMDPs. Thus in Section 2.3 I describe the Markov decision process model, and in Section 2.4 I describe value iteration, a basic technique for solving MDPs. Having laid the groundwork with MDPs, POMDPs are discussed in Section 2.5 and solution techniques for them in Section 2.6. I conclude in Section 2.7.

## 2.2 Terminology

I introduce terminology in this section as a precursor to the formal model of sequential decision making.

### 2.2.1 The Agent

The **agent** is the entity that is acting in the world. In the mouse maze example the agent is the mouse. The agent must have some way to sense the environment so it can gather information from which to make decisions. The set of possible sensor readings or percepts is known as the **observations** and denoted $\mathcal{O}$. The agent must also have some way to affect the environment, so it can turn its decisions into actions. The set of **actions** is denoted $\mathcal{A}$.

### 2.2.2 The Environment

The **environment** is the arena in which the agent acts. At any time the environment is in a particular **state**. If we assume the topology of the maze and the location of the cheese are fixed, the state of the mouse maze consists solely of the location of the mouse. The set of possible states is denoted $\mathcal{S}$.

In some environments the agent cannot directly observe the true state. Such an environment is called **partially observable** and states that cannot be distinguished are called **perceptually aliased**. This is the situation for the mouse in the maze. To a human observer looking at the maze from above the complete state is known. Such an environment is called **completely observable**. Observability is thus really a function of the agent's sensors rather than some function of the world itself[1]. The observation function $\mathcal{Z}$ maps states to observations. In general it is stochastic, so the same state does not necessarily generate the same observation each time it is visited.

There is a set of rules governing how the state changes in response to the agent's actions. This is called the **transition** function $\mathcal{T}$, and maps from the current state and the agent's action to the next state. The transition function may be stochastic, meaning the next state can vary randomly given the same current state and action.

In many real world domains there are aspects of the environment that change very slowly. In the mouse maze, for example, the experimenter may change the layout of the maze from time to time. Such an environment is known as **non-stationary**. If the transition function chooses the

[1] Leaving aside quantum effects.

next state with fixed probabilities then the world is **stationary**. I will deal only with stationary worlds in this dissertation.

Assuming the environment only changes in response to the agent's actions makes it difficult to model other agents that have unpredictable actions and may act in adversarial ways. Modelling multi-agents interactions is beyond the scope of this disertation. The interested reader is referred to (Littman 1996) for more information on Markov games.

### 2.2.3 Reward

Implicit in our discussion of sequential decision making is some purpose or goal motivating the agent. The **reward** signal makes this purpose explicit. A reward is a real-valued number with larger values indicating more reward. The agent receives a reward after every action; the function that maps from states and actions to rewards is denoted $\rho$. Maximising some function of the rewards received over time is the agent's goal.

In an **infinite horizon** problem the agent can act for an indefinite period of time, and so must consider rewards extending infinitely far into the future. In **finite horizon** problems the agent can only take a fixed and finite number of actions.

The mouse's goal is straightforward: it is to maximise cheese consumption, which in turn is a function of the time taken to traverse the maze from the entrance to the cheese. Thus for the mouse an appropriate reward function is a small negative value at each time step, representing increasing hunger, and a large positive value on finding the cheese. If the experimenter gives the mouse a set period to find the cheese, the mouse faces a finite horizon problem. Otherwise, the mouse has an infinite horizon. All practical problems have a finite horizon (in this example, eventually either the experimenter will stop the experiment or the mouse will die of hunger) but if the horizon is far away modelling it as an infinite horizon is appropriate.

Often the feedback provided by the reward applies not to the immediately preceding action but to a preceding sequence of actions. For example, finding the cheese is the reward for all the actions that took the mouse to it. Determining how actions correspond to delayed rewards, the **credit assignment problem**, is a major issue when the agent does not have knowledge of $\rho$.

### 2.2.4 The Policy

The **policy** $\pi$ specifies the way in which agent acts. In general the policy may be arbitrarily complex, depending on the entire history of the interactions with the environment. However there are several kinds of restricted policies that are useful to study.

A **plan** is a fixed sequence of actions that the agent executes step by step without regard to feedback from with the environment. If the maze never changes the mouse will eventually learn a route that it could execute "with its eyes closed" and this will be a plan.

A **conditional plan** is a plan that branches at points where the agent uses observations to choose between different forks in the plan. A conditional plan could be used if the experimenters change part of the maze from time to time.

A **reactive policy** or **universal plan** is the ultimate expression of a conditional plan where the agent has decided on an action for every state. Such plans are usually not used by humans due to the colossal memory needed for all but the simplest cases. However they are quite simple for computers to implement and many algorithms for solving the sequential decision making problem construct a reactive policy. So long as the agent can determine the state a reactive policy can be optimal, as the agent can choose the best action for every eventuality it encounters.

Alternatively a policy can be represented using any formalism for a computational machine, such as a finite state machine or a recurrent neural network. **Finite state controllers** are a simple policy representation that, as their name suggests, are finite state machines augmented with input and outputs (also know as finite state transducers). Finite state controllers are quite different to reactive policies as their actions are determined by observations and their internal state. There are many other representations that have been used in sequential decision making problems. We will see later that these representations have some advantages over reactive policies in partially observable environments.

### 2.2.5 Problems

The information the agent has determines the problem it must solve. If the agent knows the environment, meaning it has a model of the transition, observation, and reward functions, it can use **model-based** algorithms to **plan**. If the agent lacks this information it is said to be **model-free** and must learn from interactions with the environment, a process known as **reinforcement learning**.

I consider reinforcement learning problems in this thesis. One method, called **policy search**, learns a policy directly from interactions with the environment. The other approach I explore, called **model learning**, tries to learn a model from interactions and then plans using this model.

## 2.3 Markov Decision Processes

So far I have been informal in describing the sequential decision making problem. In this section I formalise a simple model of the environment, the Markov decision process (MDP).

### 2.3.1 Definition

A Markov decision process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{I} >$ where:

- $\mathcal{S}$ is the set of states that the environment may be in.
- $\mathcal{A}$ is the set of actions that the agent can perform in the environment.
- $\mathcal{T}$ is the transition function, which determines the next state given the current state, and the chosen action with type $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. $\mathcal{T}(s, a, s')$ gives the probability of reaching state $s'$ given the agent takes action $a$ from state $s$.
- $\rho$ is the reward function that maps state and action pairs to reward: $\rho : \mathcal{S} \times \mathcal{A} \to \Re$
- $\mathcal{I}$ is the initial distribution over states $\mathcal{I} : \mathcal{S} \to [0, 1]$ defining the probability $\mathcal{I}(s) = P(s)$

### 2.3.2 Key MDP Variants

Observability is the main variation between MDP models with which I am concerned. In a fully or completely observable MDP the agent is aware of the true state of the world. In a partially observable MDP this is not the case. Intuitively this is the difference between the agent having perfect sensors, and hence always perceiving the world exactly as it is, and having imperfect sensors and hence some uncertainty about the state of the world. I turn to the partially observable case in Section 2.5. As we will see (and should expect!) the partially observable case is much harder than the completely observable one.

A MDP is **deterministic** if the next state is entirely determined by the current state and action. In other words there is no stochasticity in the transition function $\mathcal{T}$, and so it maps $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ to the set $\{0, 1\}$ rather than the interval $[0, 1]$. Deterministic MDPs are appropriate for some situations. For an example consider a robot moving around a world. Though locomotion is a difficult task, at the high level of, for example, planning routes, it may be appropriate to consider the job of actually moving the robot to be reliable enough as to be deterministic. As another example consider an agent designed to serve advertisements on a webpage. It must choose between several ads to optimise income. If the software is reliable enough that the agent can assume its chosen ads will be served, the environment can be assumed deterministic. There are specialised algorithms for deterministic MDPs such as (Madani 2002) and (McMahan and Gordon 2005). In this dissertation I only consider stochastic worlds.

If the set of states $\mathcal{S}$ has no structure I say the MDP is **enumerated**. If states are represented as tuples of values it is called a **factored** representation. If the states are represented by a graph of relations the model is a **relational** MDP. I am mainly concerned with enumerated representations in this dissertation, though I consider a factored representation in Chapter 7.

In some cases the state and action spaces may be **continuous**. These models are often used in, for example, robotic tasks where the environment model (location and so on) is naturally continuous. In general I consider only **discrete** MDPs, but there is an important application of continuous MDPs to solve POMDPs by representing the POMDP as a completely observable MDP in the so-called belief or information state space.

There are other variants on the MDP model that fall outside the scope of this dissertation but I mention here for completeness. If there is more than one agent the model is known as a **Markov game** when it is completely observable, and an **incomplete-information game** when it is partially observable. Finally note that all these models are inherently synchronous; time only advances when an agent makes an action. I do not consider any asynchronous models.

A generalised Markov decision process covering MDPs and Markov games is presented in (Littman 1996, p.50)

### 2.3.3 The Mouse Maze as a MDP

Let's consider the mouse maze as a MDP with an enumerated state representation. Recall from Section 2.2.2 that the state is the location of the mouse. To represent the mouse maze in this way one assigns a unique number to each square in the maze. The set of states $\mathcal{S}$ thus becomes the integers from 1 to the number of squares in the maze, and the state is the number of the square in which the mouse is currently located. An example of such a numbering is given in Figure 2.2.

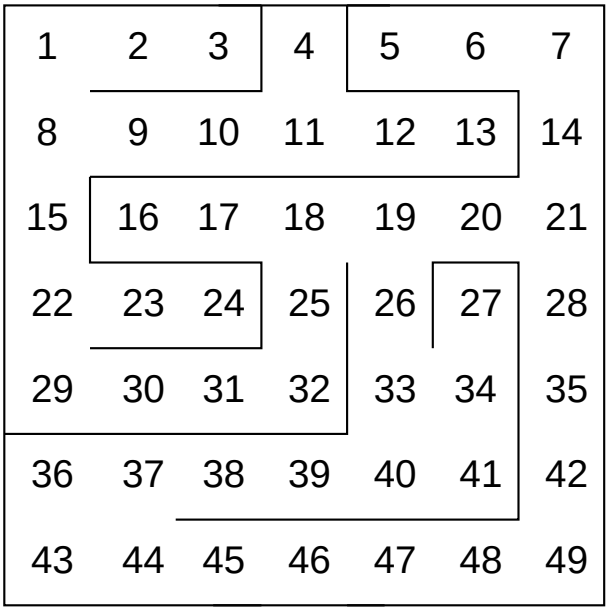


Figure 2.2: Numbering of states in the mouse maze, showing an enumerated representation

To complete the description of a MDP we must provide the set of actions, the transition

function, the reward function, and the initial state distribution. These aren't particularly important for the discussion that follows, but fully detailing the MDP will help ground the theoretical results presented next. Hence:

- $\mathcal{S}$ is the set of natural numbers from 1 to 49.
- $\mathcal{A}$ is the set $\{up, down, left, right\}$, the actions the mouse may take
- $\mathcal{T}$, the transition function, is described informally as follows: actions succeed with probability $0.9$ unless there is a wall in the way. If the action fails, or there is a wall in the way, the mouse stays where it is.
- The reward function $\rho$ gives the mouse a reward of 100 when it reaches the cheese (in state 4) and -1 at all other times.
- We assume the mouse always starts in state 46, and thus the initial state distribution $\mathcal{I}$ maps state 46 to 1 and all other states to 0.

## 2.4 Solving MDPs

Having defined Markov decision processes we now turn to solving them. In this section I introduce the basic concepts and algorithms for solving completely observable MDPs.

### 2.4.1 Acting Optimally

To begin the discussion of finding optimal solutions to MDPs we are going to look at the **value**, a measure of accumulated reward, that we might assign to a given policy. This will lead us to the infinite-horizon discounted sum of rewards, which is the objective function we seek to optimise. The infinite-horizon discounted sum of rewards is only one of many possible objective functions, but it is one that is easy to analyse and appropriate for many situations. From this definition we will define the optimal value function that enables us to solve an MDP; that is, to find a policy that maximises the infinite-horizon discounted sum of rewards.

A policy is the agent's way of acting. By the definition of MDPs the current state and action completely determine the probability of the next state and reward. Thus the current state encapsulates everything necessary to choose the best action. Since the MDP is completely observable, meaning the agent never has any doubt as to the current state, the agent doesn't need any memory to determine its actions. Thus we can assume a **stationary policy**, a policy where the choice of action is a deterministic function of the current state, and still find an optimal policy. This will become apparent as we explore the problem. For now it is sufficient to merely assume a stationary policy.

Formally a stationary policy $\pi$, introduced in Section 2.2.4, is a function from states to actions; $\pi : \mathcal{S} \to \mathcal{A}$. I write $\pi(s)$ to mean the action chosen by the policy given a state $s \in \mathcal{S}$.

Let's start by considering only a single time step. The environment is in state $s$ and we want to find the reward the policy receives for taking a single action without regard to what might happen afterwards. The reward is simply $\rho(s, a)$ for some action $a$. The action taken by a policy $\pi$ in state $s$ is $\pi(s)$, so we can write $\rho(s, a)$ as $\rho(s, \pi(s))$. It follows that the **one-step value** of a state $s$ under a policy $\pi$, the reward we will receive in $s$ following $\pi$ for a single step, is

$$V_1^\pi(s) = \rho(s, \pi(s)) \tag{2.1}$$

The optimal one-step value, written $V_1^*$, is the value of the action that maximises the one-step value:

$$V_1^*(s) = \max_{a \in \mathcal{A}} \rho(s, a) \tag{2.2}$$

I stated earlier that the agent is trying to maximise some measure of reward. If this measure is the one-step value, then an optimal policy $\pi_1^*$ can be constructed as

$$\pi_1^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \rho(s, a) \tag{2.3}$$

Note that the definition of the one-step optimal policy is identical to the definition of the one-step optimal value, except for the change in the term being maximised. Furthermore, note

there may be many possible one-step optimal policies for a given MDP.

Looking ahead only a single step is a bit foolish. By taking actions that maximise only the single-step value we could easily set ourselves on a path that is very bad in the long term. We should really consider at least some more steps into the future, and if we're going to consider some steps we might as well consider all steps. We can easily extend the one-step value to the value of all possible steps. Given a policy $\pi$ we can write the **value** of the policy as:

$$\begin{aligned} V^{\pi}(s) = \rho(s, \pi(s)) &+ \left( \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \rho(s', \pi(s')) \right) \\ &+ \left( \gamma^2 \sum_{s' \in \mathcal{S}} \sum_{s'' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \mathcal{T}(s', \pi(s'), s'') \rho(s'', \pi(s'')) \right) \\ &+ \ldots \end{aligned} \tag{2.4}$$

This equation deserves some explanation. Firstly, we include the transition function $\mathcal{T}$ to weight the future rewards by the probability of achieving them. The discount factor $\gamma$ serves two purposes: it keeps the sum of rewards bounded, and it determines how we weight future rewards relative to immediate rewards. If the discount factor is zero we completely ignore future rewards. If the discount factor is one, future rewards, no matter how far away they are, are considered equal to immediate rewards. Values between zero and one given proportionally more weight to future rewards, and values greater than one give more weight to rewards the further away they are – a nonsensical choice in any realistic setting. If the maximum attainable one-step reward is $M$ then the maximum infinite-horizon discounted sum of rewards is $\frac{1}{1-\gamma} M$. Finally note that the definition of value in Equation 2.4 has a recursive structure. This can be made more apparent:

$$V^{\pi}(s) = \rho(s, \pi(s)) + \left( \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \pi(s), s') \left( \rho(s', \pi(s')) + \gamma \sum_{s'' \in \mathcal{S}} \mathcal{T}(s', \pi(s'), s'') \left( \rho(s'', \pi(s'')) + \ldots \right) \right) \right) \tag{2.5}$$

Equation 2.5 can be easily rewritten into the standard definition of value, which is:

$$V^{\pi}(s) = \rho(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} (\mathcal{T}(s, \pi(s), s') V^{\pi}(s')) \tag{2.6}$$

This definition gives the value of a given policy. We can instead maximise value, yielding:

$$V^{*}(s) = \max_{a \in \mathcal{A}} \left( \rho(s, a) + \gamma \sum_{s' \in \mathcal{S}} (\mathcal{T}(s, a, s') V^{*}(s')) \right) \tag{2.7}$$

Given $V^*$ we can define the optimal policy as that which chooses the action giving the highest value:

$$\pi^{*}(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left( \rho(s, a) + \gamma \sum_{s' \in \mathcal{S}} (\mathcal{T}(s, a, s') V^{*}(s')) \right) \tag{2.8}$$

As with the one-step optimal policy, the definition of the optimal policy is nearly identical to that for the optimal value.

Equation 2.7 is known as a Bellman equation. It defines a set of simultaneous equations that we can solve to give the optimal value function for each state, and thus from which we can derive an optimal policy. The maximisation operator means the equations are not linear so Gaussian elimination cannot be used to solve them. I will now give a classic algorithm, Value Iteration, for solving MDPs. This algorithm works by iteratively refining an approximation of the optimal value function. Further information on this algorithm and the details of other algorithms for MDPs can be found in (Puterman 1994).

### 2.4.2 Value Iteration

The inner loop of value iteration, given in Function `oneValueIteration`, takes a $t-1$ step estimate of the value function and extends it a single time step into the future. This is essentially just an application of Equation 2.6. As a small optimisation we compute so-called $Q$-values, which are the value given a state *and action*. From these $Q$-values it is straightforward to compute both the $t$-step optimal policy and value.

**Function** oneValueIteration($< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{I} >$, $V_{t-1}$) Inner loop of value iteration for an MDP

**for** $s \in \mathcal{S}$ **do**
  **for** $a \in \mathcal{A}$ **do**
    $Q_t(s,a) \leftarrow \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V_{t-1}(s')$
  **end**
  $\pi_t(s) \leftarrow \text{argmax}_a \, Q_t(s,a)$
  $V_t(s) \leftarrow Q_t(s, \pi_t(s))$
**end**
**return** $(V_t, \pi_t)$

The full value iteration algorithm is given in Function `valueIteration`. It is just a small wrapper around the inner loop that keeps extending the horizon until the stopping condition is reached. In summary, value iteration works by computing $V_t$, a sequence of discounted finite-horizon optimal value functions with increasing horizon length, stopping when the change in the value function is less than some given value $\epsilon$.

**Function** valueIteration($< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{I} >$, $\epsilon$) Value iteration for an MDP

$t \leftarrow 0$
$V_0 \leftarrow 0$
**repeat**
  $t \leftarrow t + 1$
  $(V_t, \pi_t) \leftarrow$ `oneValueIteration` $(V_{t-1})$
**until** $\max_s |V_t(s) - V_{t-1}(s)| < \epsilon$
**return** $(V_t, \pi_t)$

## 2.5 Partially Observable Markov Decision Processes

I now turn to a more general model and the topic of this dissertation, the POMDP. As with the MDP this section gives the formal definition and Section 2.6 covers methods for solving POMDPs.

### 2.5.1 Definition

A partially observable Markov decision process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{O}, \mathcal{Z}, \mathcal{I} >$ where:

- $\mathcal{S}$ is the set of states that the environment may be in.
- $\mathcal{A}$ is the set of actions that the agent can perform in the environment.
- $\mathcal{T}$ is the transition function, which determines the next state given the current state, and the chosen action. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines the probability $\mathcal{T}(s, a, s') = P(s'|a, s)$
- $\rho$ is the reward function that maps state and action pairs to reward: $\rho : \mathcal{S} \times \mathcal{A} \rightarrow \Re$
- $\mathcal{O}$ is the set of observation symbols. These represent the range of observations that the agent may make.
- $\mathcal{Z}$ is the observation function that maps the current state of the environment to an observation. $\mathcal{Z} : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ defines the probability $\mathcal{Z}(s, o) = P(o|s)$
- $\mathcal{I}$ is the initial distribution over states $\mathcal{I} : \mathcal{S} \rightarrow [0, 1]$ defining the probability $\mathcal{I}(s) = P(s)$

If the observation function is a deterministic one-to-one map a POMDP is fully observable and hence a MDP.

### 2.5.2 The Mouse Maze as a POMDP

Let's now consider the mouse maze as a POMDP. The MDP representation, given in Section 2.3.3, still holds. To this we must add the observations $\mathcal{O}$ and the observation function $\mathcal{Z}$.

A simple representation for observations is to encode only information about the walls adjacent to the current location. We can represent the presence or absence of walls to the North, East, South and West as a binary digit giving $2^4 = 16$ possible observations. For example, square 1, which has a wall to the North and West, would be encoded as $1001 = 9$.

Building on this representation, one possible observation function is to simply map every state to its true observation with probability 1. This observation function has no noise but there are still states that have the same observation; in other words there are underlying states that are perceptually aliased together. For example, states 2, 6, 9, 10, 12, 17, 20, 23, 30, 31, 38, 45, 47, and 48 have the same observation and are therefore perceptually aliased. Many sensors, such as those used in robotics, are quite noisy. In such a case the observation function should include some noise. For example, it might randomly flip bits in the true observation to reflect the unreliability of its 'wall sensor'. This is not unreasonable. As we shall see in Chapter 7 even very precise laser range finders can miss walls or doors due to reflections, obstructions, and so on.

## 2.6 Solving POMDPs

We've previously looked at solving completely observable MDPs. We now look at fundamental algorithms for solving their partially observable counterparts, assuming again that a model is available. We'll see that there is an algorithm similar to value iteration in MDPs. The key difference, arising from partial observability, is that the agent no longer knows with certainty the state of the environment. Instead it can maintain a probability distribution over the possible underlying states, known as an **information** or **belief state**. Value can be defined in terms of this information state and from this we can derive algorithms for solving POMDPs. The high complexity of these exact algorithms (Littman 1996) makes them impractical for large problems, motivating the development of approximate algorithms.

### 2.6.1 The Belief State

Imagine the agent knows it is in a particular state. It performs an action and receives an observation. In a POMDP in general the observation will not identify the new state of the environment. However we can calculate a distribution over possible states. This distribution is known as the information or belief state.

Assume the POMDP is in a known state $s$ when the agent takes action $a$ and receives observation $o$. From Bayes rules and the POMDP dynamics the next state has probability:

$$P(s'|s,a,o) = \frac{P(o|s',a,s)P(s'|s,a)}{P(o|a,s)} \tag{2.9}$$

$$= \frac{P(o|s')P(s'|s,a)}{P(o|a,s)} \tag{2.10}$$

$$= \frac{\mathcal{Z}(s',o)\mathcal{T}(s,a,s')}{P(o|a,s)} \tag{2.11}$$

The term $P(o|a,s)$ is a normalising factor to make the probabilities sum to 1.

Now let us assume we have a vector of probabilities $b$, so that $b(s)$ gives the probability that the agent occupies state $s$. This vector is the belief state. Given action $a$ and observation $o$ we can update the belief state as follows:

$$
\begin{aligned}
b_{t+1}(s') &= P(s'|b_t, a, o) && (2.12)\\
&= \frac{P(o|s', a, b_t)P(s'|b_t, a)}{P(o|a, b_t)} && (2.13)\\
&= \frac{P(o|s')P(s'|b_t, a)}{P(o|a, b_t)} && (2.14)\\
&= \frac{P(o|s')\sum_{s\in\mathcal{S}} P(s'|s, a, b_t)P(s|a, b_t)}{P(o|a, b_t)} && (2.15)\\
&= \frac{P(o|b_t)\sum_{s\in\mathcal{S}} P(s'|s, a)P(s|b_t)}{P(o|a, b_t)} && (2.16)\\
&= \frac{\mathcal{Z}(s', o)\sum_{s\in\mathcal{S}} \mathcal{T}(s, a, s')b_t(s)}{P(o|a, b_t)} && (2.17)\\
&= SE(b_t, a, o) && (2.18)
\end{aligned}
$$

The function $SE$ is called the state estimator. The belief state is defined on the $|\mathcal{S}|$-dimensional probability simplex, which I write as $\mathcal{B}$. Thus the domain of $SE$ is $\mathcal{B} \times \mathcal{A} \times \mathcal{O}$ and its co-domain or range is $\mathcal{B}$.

A POMDP can be converted into an belief state MDP, where the state space consists of all possible belief states (i.e. $\mathcal{B}$). In other words the belief state is a sufficient statistic for summarising all past interactions with the POMDP. Most model-based algorithms for solving POMDPs make use of the belief state MDP. Unlike the MDPs in Section 2.3 the belief state MDP has a continuous state space, and therefore value iteration and other MDP algorithms are not directly applicable. We now explore the problem of acting optimally in a POMDP in more detail.

### 2.6.2 Acting Optimally

As for MDPs there are several definitions of optimality for a POMDP. The objective function used here is the infinite-horizon discounted sum of rewards, which is given for MDPs in Section 2.4.1. Similarly, we define the optimal value though this time over an belief state $b$ as:

$$V^*(b) = \max_{a \in \mathcal{A}} \left( \rho(b,a) + \gamma \int db' \, \mathcal{T}(b,a,b') V^*(b') \right) \tag{2.19}$$

Here I have extended the transition function $\mathcal{T}$ and reward function $\rho$ to operate on belief states. Equation 2.19 can be re-expressed in terms of state by noting that

$$\rho(b,a) = \sum_{s \in \mathcal{S}} b(s) \rho(s,a) \tag{2.20}$$

and

$$\begin{aligned}
\mathcal{T}(b,a,b') &= P(b'|b,a) && (2.21)\\
&= \sum_{o \in \mathcal{O}} P(b'|b,a,o) P(o|a,b) && (2.22)\\
&= \sum_{o \in \mathcal{O}} \mathbb{I}\left[b' - SE(b,a,o)\right] P(o|a,b) && (2.23)\\
&= \sum_{o \in \mathcal{O}} \mathbb{I}\left[b' - SE(b,a,o)\right] \sum_{s \in \mathcal{S}} \mathcal{Z}(s,o) b(s) && (2.24)
\end{aligned}$$

where $\mathbb{I}$ is the indicator function that is $1$ if its argument is $0$ and $0$ otherwise. Substituting Equation 2.20 and Equation 2.24 into Equation 2.19 gives

$$V^*(b) = \max_{a \in \mathcal{A}} \left( \sum_{s \in \mathcal{S}} b(s) \rho(s,a) + \gamma \sum_{o \in \mathcal{O}} \left[ \sum_{s \in \mathcal{S}} \mathcal{Z}(s,o) b(s) \right] V^*(SE(b,a,o)) \right) \tag{2.25}$$

As the belief state is a sufficient statistic we can define stationary policies in terms of it and still arrive at optimal policies. In Chapter 4 we will look at other policy representations. In the next section we turn to exact methods for finding an optimal policy for a POMDP.

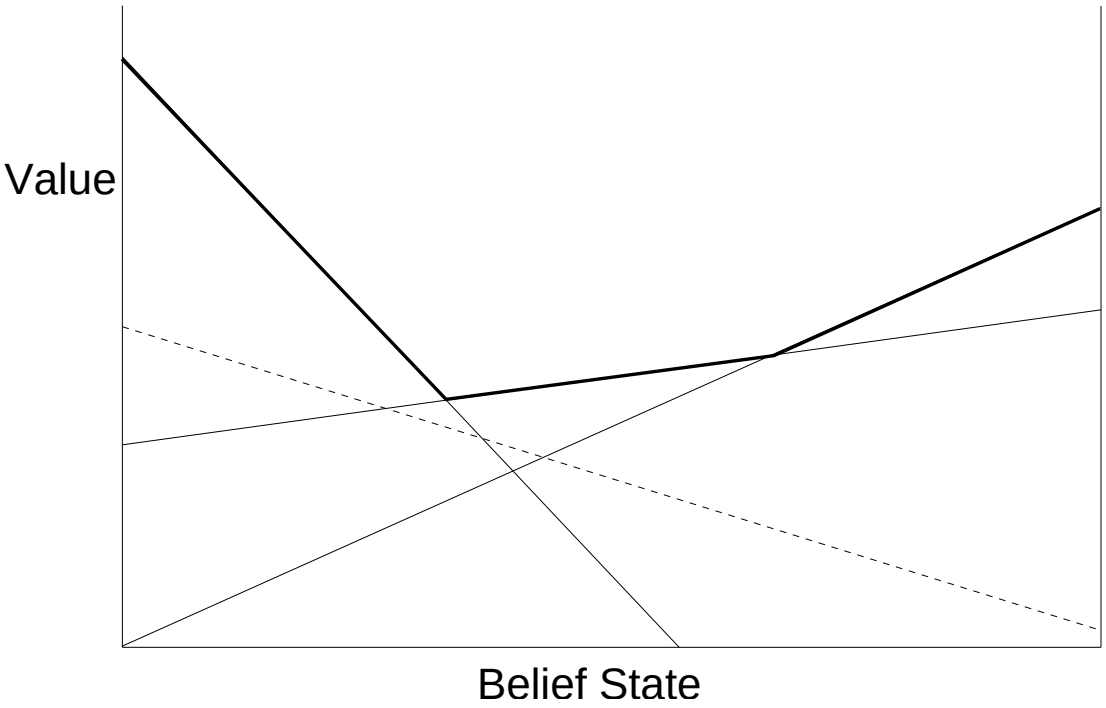


Figure 2.3: An example of a value function. The dark lines indicate the optimal value function. The dotted line indicates a hyperplane that is completely dominated by other hyperplanes.

### 2.6.3 Exact Methods

Exact methods for POMDPs are usually some variant of value iteration. This implies we want to compute the value function, as we did for MDPs. In MDPs we saw that the value function is a set of simultaneous equations that we could enumerate and solve by iteratively refining a $t$-step lookahead approximation. We cannot do this over the belief states. They are infinite in number and thus the simultaneous equations are infinite in number, and so we cannot enumerate them all.

It turns out there is no need to deal with the belief states directly. Rather the value function for finite-horizon belief state MDPs can be represented as a finite convex combination of hyperplanes in the belief state space (Smallwood and Sondik 1973). We can deal with these hyperplanes directly avoiding the infinite belief states. An example of these hyperplanes is shown in Figure 2.3. The value function, illustrated by the thick line, is constructed by choosing those pieces of the hyperplanes that have the highest value at each point. The value function thus constructed in known to be a piece-wise linear and convex function.

As can be seen in Figure 2.3 there are vectors that at every belief state have lower value than another vector. Such a vector is called **dominated** and should be removed from consideration, a process known as **pruning**. Pruning can be performed by solving a linear program; see (Cassandra 1998) for details.

From this we can construct a simple value iteration algorithm known as the Enumeration or Monahan's algorithm (Monahan 1982). Assume we have an optimal $t-1$ lookahead value function $V^*_{t-1}$. We can define an optimal $Q$-value as we did for MDPs (see Section 2.4.2) that gives the value of taking an action in a belief state:

$$Q_t(b,a) = \sum_{s\in\mathcal{S}} b(s)\rho(s,a) + \gamma \sum_{o\in\mathcal{O}} \left[\sum_{s\in\mathcal{S}} \mathcal{Z}(s,o)b(s)\right] V^*_{t-1}(SE(b,a,o)) \tag{2.26}$$

Now we know that $V_{t-1}$ can be represented as a set $\Gamma_{t-1}$ of $|\mathcal{S}|$-dimensional vectors $\alpha$. So

$$V^*_{t-1}(b) = \max_{\alpha\in\Gamma_{t-1}} b\cdot\alpha \tag{2.27}$$

$$= b\cdot\alpha^*_{t-1} \tag{2.28}$$

where I have defined $\alpha^*_{t-1}$ to be the optimal vector at the belief state $b$. We can substitute this into Equation 2.26 to give the $Q$-value in terms of the $\alpha$ vectors

$$Q_t(b,a) = \sum_{s\in\mathcal{S}} b(s)\rho(s,a) + \gamma \sum_{o\in\mathcal{O}} \left[\sum_{s\in\mathcal{S}} \mathcal{Z}(s,o)b(s)\right] b'\cdot\alpha^*_{t-1}(b') \tag{2.29}$$

Here, for brevity I have used $b' = SE(b,a,o)$. Now the equations become easier to manipulate if we switch to vector notation. I write $\rho_a$ for the vector of rewards for action $a$ and $\mathcal{Z}_o$ for the vector of observation probabilities for all states given an observation $o$. With this notation we can re-express Equation 2.29 as:

$$Q_t(b,a) = b \cdot \rho_a + \gamma \sum_{o \in \mathcal{O}} (\mathcal{Z}_o \cdot b) \left( b' \cdot \alpha^*_{t-1}(b') \right) \tag{2.30}$$

$$= b \cdot \rho_a + b \cdot \gamma \sum_{o \in \mathcal{O}} \mathcal{Z}_o \left( b' \cdot \alpha^*_{t-1}(b') \right) \tag{2.31}$$

$$= b \cdot \left( \rho_a + \gamma \sum_{o \in \mathcal{O}} \mathcal{Z}_o \left( b' \cdot \alpha^*_{t-1}(b') \right) \right) \tag{2.32}$$

If we define $\alpha_t^{a,o}(b) = \rho_a + \gamma \sum_{o \in \mathcal{O}} \mathcal{Z}_o \left( b' \cdot \alpha^*_{t-1}(b') \right)$ we can write:

$$Q_t(b,a) = b \cdot \alpha_t^{a,o}(b) \tag{2.33}$$

Now the optimal vector $\alpha_t^*$ is simply defined as:

$$\alpha_t^*(b) = \max_a b \cdot \left( \sum_{o \in \mathcal{O}} \alpha_t^{a,o}(b) \right) \tag{2.34}$$

This gives us a way to construct a $t$-step vector $\alpha_t^*$ given a $t-1$-step vector. We now need to decide which vectors to construct. Our solution is to construct all possible vectors and then prune away those that are dominated. The cross-sum operator $X \bigoplus Y$ returns the set of all possible pair-wise sums of vectors from sets $X$ and $Y$. Given a set of vectors $\Gamma_{t-1}$ we construct $\Gamma_t$ by first constructing all possible $\alpha_t^{a,o}$:

$$\Gamma_t^{a,o} = \left\{ \rho_a + \gamma \mathcal{Z}_o (b' \cdot \alpha^*_{t-1}) : \alpha^*_{t-1} \in \Gamma_{t-1} \right\} \tag{2.35}$$

Next we take the cross-sum over all possible observations:

$$\Gamma_t^a = \bigoplus_{o \in \mathcal{O}} \Gamma_t^{a,o} \tag{2.36}$$

Finally we prune the union of $\Gamma_t^a$ for all possible actions to give the set of optimal vectors:

$$\Gamma_t = prune\left(\bigcup_{a\in\mathcal{A}} \Gamma_t^a\right) \tag{2.37}$$

We have now constructed the optimal $t$-step value function. However, this algorithm is not practical. The size of the set $\Gamma$ grows with order $O(|\mathcal{A}|^{|\mathcal{O}|})$ in the worst case at each iteration. Pruning will decrease the size is most cases but we must still construct all these vectors before we prune them, and pruning itself is an expensive operation.

There are more efficient algorithms for solving POMDPs (see, for example, (Cassandra 1998)) but the basic problem of exponential growth in the set of hyperplanes remains for any exact algorithm. For this reason exact solution of POMDPs is generally infeasible.

### 2.6.4 Approximate Methods

Given the issues with exact solution methods for POMDPs there is a great deal of work in approximate solution methods. In my work I make use of two approximate POMDP solving algorithms: simple lookahead search and HSVI (Smith and Simmons 2004).

Lookahead search is a very simple algorithm. It builds a tree of reachable belief states starting from the current belief state. This tree is constructed by considering every possible action and observation up to a fixed depth. The value of the current belief state is then estimated by propagating values up from the leaf nodes in the tree. This algorithm is detailed in Function `lookaheadSearch`.

Lookahead search can be usefully combined with offline methods. This is the approach taken by heuristic search value iteration (HSVI) (Smith and Simmons 2004), an offline approximate value iteration algorithm. In HSVI offline methods are used to calculate upper and lower bounds on the value function, and lookahead search is used to refine the estimated value. A heuristic is used guide exploration of the tree of reachable belief states.

Specifically, a lower bound on the value function is calculated using a "blind policy", which is a policy that always chooses the same action. An upper bound is calculated by assuming the POMDP is fully observable (that is, converting it to a MDP). Lookahead search is then used to

**Function** lookaheadSearch($< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{O}, \mathcal{Z}, \mathcal{I} >$, $b$, $\gamma$, $depth$) Simple lookahead search algorithm for approximately solving POMDPs

**if** $depth = 0$ **then**
  **return** *0*
**end**
**else**
  **for** $a \in \mathcal{A}$ **do**
    $nextB \leftarrow \{SE(b, a, o) : o \in \mathcal{O}\}$
    $Q(b, a) \leftarrow \rho(b, a) + \gamma \sum_{b' \in nextB}$
    `lookaheadSearch` ($< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{O}, \mathcal{Z}, \mathcal{I} >$, $b'$, $\gamma$, $depth - 1$)
  **end**
  **return** $\max_{a \in \mathcal{A}} Q(b, a)$
**end**

refine the bounds. We can refine the upper bound by returning the current upper bound estimate at the terminal belief state (rather than returning 0 as in Function `lookaheadSearch`) and similarly for the lower bound.

Rather than expanding all nodes of the tree to a fixed depth, HSVI uses a heuristic to choose which child nodes to explore. There are two types of nodes in the tree, action nodes and observation nodes, and different heuristics are used for each. For action nodes, the action node with the highest upper bound on its Q-value is chosen. For observations, choice is the node with the highest weighted excess uncertainty:

$$o^* = \underset{o}{\operatorname{argmax}} \left( P(o|b, a) excess(SE(b, a, o), t + 1) \right) \tag{2.38}$$

The weight is the probability $P(o|b, a)$. The excess uncertainty $excess(b, t)$ of an information state $b$ at depth $t$ is:

$$excess(b, t) = (\bar{V}(b) - \underline{V}(b)) - \epsilon\gamma^{-t} \tag{2.39}$$

This is the difference between the upper bound on the uncertainty in the value function ($\epsilon\gamma^{-t}$ for some choice of $\epsilon$) and the actual uncertainty in the value function (where $\bar{V}(b)$ is the upper bound and $\underline{V}(b)$ is the lower bound).

A survey of online POMDP solution methods, focusing on lookahead search algorithms, is

given in (Ross, Pineau, Paquet et al. 2008).

## 2.7 Conclusions

In this chapter we have examined the sequential decision making problem, its formulation as a partially observable Markov decision process, and methods for finding a policy given a POMDP. We have seen that exact solutions are intractable, and so approximations must be made.

In an unknown world we do not have access to a model, so POMDP solution methods are not directly applicable. In the next chapter I discuss how one can learn a model from interactions with the environment. In Chapter 4 I survey work that learns a policy without the intermediate step of learning a model.

CHAPTER 3

# FOUNDATIONS: HOW TO LEARN A MODEL FROM DATA

## 3.1 Introduction

In this chapter I review methods for learning POMDPs from data. The two basic problems are the simpler parameter learning problem, and the more complex structure learning problem. Recall the definition of the POMDP from Section 2.5:

A partially observable Markov decision process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{O}, \mathcal{Z}, \mathcal{I} >$ where:

- $\mathcal{S}$ is the set of states that the environment may be in.
- $\mathcal{A}$ is the set of actions that the agent can perform in the environment.
- $\mathcal{T}$ is the transition function, which determines the next state given the current state, and the chosen action. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines the probability $\mathcal{T}(s, a, s') = P(s'|a, s)$
- $\rho$ is the reward function that maps state and action pairs to reward: $\rho : \mathcal{S} \times \mathcal{A} \rightarrow \Re$
- $\mathcal{O}$ is the set of observation symbols. These represent the range of observations that they agent may perceive about the environment.
- $\mathcal{Z}$ is the observation function that maps the current state of the environment to an observation. $\mathcal{Z} : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ defines the probability $\mathcal{Z}(s, o) = P(o|s)$

- $\mathcal{I}$ is the initial distribution over states $\mathcal{I} : \mathcal{S} \to [0, 1]$ defining the probability $\mathcal{I}(s) = P(s)$

The parameter learning problem assumes we know the number of states of the model and the connections between them. The task is to learn the transition and observation probabilities. More formally, we know $\mathcal{S}$ and $\mathcal{O}$ and we know for which values of $s, a$, and $s'$ $\mathcal{T}(s, a, s')$ is non-zero. The task is to learn $\mathcal{Z}$ and the non-zero values of $\mathcal{T}$.

The structure learning problem does not make the assumptions of the parameter learning problem. We do not know how many states are in the model and hence we cannot know the connections between them. So the structure learning problem adds learning $\mathcal{S}$ and $\mathcal{T}$ to solving the parameter learning problem. Usually we do not consider learning $\mathcal{O}$, as the set of observations is as much a property of the agent's sensors as of the environment.

POMDP learning is a small field, but there has been much work on a related model called the hidden Markov model (HMM). A HMM is simply a POMDP without actions and rewards. So $\mathcal{T}$ is $\mathcal{S} \times \mathcal{S} \to [0, 1]$ and $\rho$ is omitted. Where appropriate I'll refer to HMM methods with the understanding they can easily be adapted to POMDPs.

### 3.1.1 Overview

This chapter begins with an overview of maximum likelihood and Bayesian learning, the two main approaches to learning we'll consider. In Section 3.3 we turn to the two learning problems, parameter and structure learning, in Markov decision processes under the maximum likelihood and Bayesian frameworks. Although our ultimate goal is POMDP learning, tackling MDPs first gives a gentle introduction to the issues involved. Addressing MDPs introduces the Dirichlet distribution, which is a key tool for modelling MDPs and POMDPs in the Bayesian framework, and the Dirichlet and Beta processes, which I use in Chapter 7. We then turn to POMDPs in Section 3.4, discussing first general methods for learning with hidden state and then seeing how these methods can be applied to solve the two learning problems from the maximum likelihood and Bayesian perspectives. The chapter concludes with a brief overview of other model learning methods.

## 3.2 Maximum Likelihood and Bayesian Learning

Let's start by considering, in abstract, the model learning problem. We have some data $x$. Let a model $m$ with parameters $\theta$ be one of a family of generative models. A **generative model** defines a probability distribution $P(x|\theta)$. Usually a model will have some **hidden** or **latent variables** $y$ so that the probability of the data is defined by marginalising over them:

$$P(x|\theta) = \sum_y P(x|y,\theta)P(y|\theta) \tag{3.1}$$

Given data we can usually find many parameter settings or models that are consistent with it. How should we select between the available choices? One school of thought says we should choose the parameters or model most likely to generate the data we observed. Another school of thought says we should consider all possible choices, but weight them according to our prior beliefs and their consistency with the data. These two answers correspond respectively to the maximum likelihood and Bayesian approaches to learning.

### 3.2.1 Maximum Likelihood Learning

In the maximum likelihood, or frequentist, approach to parameter learning we seek the parameters $\theta_{ml}$ that make the observations most likely. That is, we try to find

$$\theta_{ml} = \operatorname*{argmax}_\theta P(x|\theta) \tag{3.2}$$

$$= \operatorname*{argmax}_\theta \sum_y P(x|y,\theta)P(y|\theta) \tag{3.3}$$

The distribution $P(x|\theta)$ is known as the **likelihood**. It will often be more convenient to deal with the **log likelihood** $\mathcal{L}(\theta) = \ln P(x|\theta)$. Maximising the log likelihood also maximises the likelihood, as the logarithm is a monotonic function.

**Maximum Likelihood Learning of a Binomial Distribution**

Consider fitting a binomial distribution to data. A binomial distribution is a distribution that chooses between 2 options, conventionally labelled 0 and 1. It is controlled by a single parameters $\theta$ that gives the probability of choosing a 1.[1] Let us say we have data $x = \{x_1, \ldots, x_N\}$, where $x_i \in \{0, 1\}$ indicates the option that was chosen for data point $x_i$. Our task now is to find the maximum likelihood setting of the parameters given this data.

The likelihood in this case is

$$P(x|\theta) = \prod_{i=1}^{N} P(x_i|\theta) \tag{3.4}$$

If we write $a$ for the number of 1s we observed and $b$ for the number of 0s, we can re-express the likelihood as

$$P(a, b|\theta) = \frac{(a+b)!}{a!b!}\theta^a(1-\theta)^b \tag{3.5}$$

Note the factor of $\frac{(a+b)!}{a!b!}$. This is to account for all the different combinations of sequences that could lead to the observed counts $a$ and $b$. The log likelihood follows as:

$$\mathcal{L}(\theta) = \ln(a+b)! - \ln(a!b!) + a\ln\theta + b\ln(1-\theta) \tag{3.6}$$

We can optimise the log likelihood by taking the derivative and setting to zero, giving

$$\frac{d\mathcal{L}(\theta)}{d\theta} = \frac{a}{\theta} + \frac{b}{1-\theta} \tag{3.7}$$

$$0 = \frac{a}{\theta} + \frac{b}{1-\theta} \tag{3.8}$$

$$\theta = \frac{a}{(a+b)} \tag{3.9}$$

Thus the maximum likelihood setting for the parameter $\theta$ of the binomial is $\frac{a}{(a+b)}$.

[1] As the probabilities of the two choices must sum to one the probability of choosing a 0 is completely determined by the value of $\theta$.

If the model contains unobserved variables we usually cannot analytically calculate a globally optimal solution. The standard alternative is to calculate a locally optimal solution using the expectation-maximisation (EM) algorithm. The EM algorithm is presented in Section 3.4.1.

A sufficiently complex model can fit any data arbitrarily closely. If we choose the model that maximises the data likelihood we are likely to fit the noise in the data rather than the true structure, a problem known as **overfitting**. There are several methods in the maximum likelihood framework one can use to combat overfitting, such as cross-validation or regularisation. The Bayesian framework provides an alternative that automatically controls for model complexity.

### 3.2.2 Bayesian Learning

Consider estimating the probability that a coin flip will be heads – that is, the binomial parameter estimation task we've just considered. If we see just one flip and it is tails, the maximum likelilhood estimate is that the probability of heads is zero! In many situations, and particularly those with scarce data, we would instead like to consider a range of possible values and assign a probability to these. Furthermore, if we were told beforehand that the coin is weighted towards heads we would like to incorporate this prior information. For these reasons, and more, Bayesian learning may be preferred.

In the Bayesian framework we treat all unknowns as random variables. For the parameter learning task we don't choose a single set of parameter values, but rather assign a probability $P(\theta|x)$ to all values under consideration. This distribution is known as the **posterior of the parameters**. Using Bayes rules we can relate the posterior to the likelihood $P(x|\theta)$ and a distribution $P(\theta)$ over parameters known as the **prior of the parameters**:

$$P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)} \tag{3.10}$$

$$= \sum_{y} \frac{P(x|y,\theta)P(y|\theta)P(\theta)}{P(x)} \tag{3.11}$$

If we were to pick a single solution we would choose

$$\theta_{MAP} = \underset{\theta}{\operatorname{argmax}} \frac{P(x|\theta)P(\theta)}{P(x)} \tag{3.12}$$

$$= \underset{\theta}{\operatorname{argmax}} P(x|\theta)P(\theta) \tag{3.13}$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{y} P(x|y,\theta)P(y|\theta)P(\theta) \tag{3.14}$$

This solution is the called the **maximum a posteriori** or MAP solution. If the prior is uniform – which is not always possible or desirable – then the MAP and ML solutions are equivalent.[1]

The Bayesian approach to structure learning is the same. We introduce a variable $m$ to represent the choice of model, and integrate over both hidden variables and parameters to achieve the **posterior of the models**:

$$P(m|x) = \frac{P(x|m)P(m)}{P(x)} \tag{3.15}$$

$$= \int d\theta \; \frac{P(x|\theta,m)P(\theta|m)P(m)}{P(x)} \tag{3.16}$$

If the prior over models is uniform then the posterior of the models is determined solely by the **marginal likelihood**:

$$P(x|m) = \int d\theta \; P(x|\theta,m)P(\theta|m) \tag{3.17}$$

The marginal likelihood is sometimes called the **evidence** for a model, as it represents the data dependent part of the posterior. Note that the marginal likelihood also appears as the normalisation constant in the denominator of Equation 3.10. Thus calculating the marginal likelihood is a key part of both the Bayesian parameter and model learning tasks.

The Bayesian approach automatically addresses overfitting. Every model has a fixed prob-

[1] Note that the MAP solution is not invariant to reparameterisation of the model. This makes it a rather arbitrary choice.

ability mass that it distributes amongst the data. A more complex model can represent a wide range of data, so it must necessarily assign a small mass to any particular sample. A simple model can represent few data, and so can give large mass to all. Both overly complex models and overly simple models are penalised by integrating over the parameters to compute the marginal likelihood. This information is reflected in the posterior over models, or, if a single model must be chosen, it can be used to make that choice.

### Choosing the Prior

The choice of prior is a major issue in Bayesian learning. The **subjective**, **objective**, and **empirical** approaches to priors represent the three main schools of thought.

**Subjective Priors** A subjective prior encodes all available prior knowledge as far as possible. It is often difficult to express beliefs in a suitable mathematical form, but one class of priors with a clear interpretation, as well as other favourable properties, is the class of **conjugate priors**. A conjugate prior is paired with a likelihood which together give a posterior of the same form, albeit with different parameters, as the prior. The combination of prior and likelihood is called a **conjugate pair**. For example, a normal distribution $\mathcal{N}(\mu, \sigma^2)$ with fixed variance $\sigma^2$ and mean drawn from a normally distributed prior $\mathcal{N}_p(\mu_p, \sigma_p^2)$ form a conjugate pair, and hence the posterior distribution over the mean is also normally distributed. This gives obvious advantages in maintaining tractable inference. The parameters of a conjugate prior also have a straightforward interpretation as imaginary or previous observations.

**Objective Priors** The objective approach tries to convey as little information as possible in the prior, so that inference is guided by data as far as possible. An objective prior is also called a **non-informative prior**. While appealing in theory non-informative priors are still an area of active research, and non-informative priors have not been developed for a wide range of distributions.

**Empirical Bayes** Oftentimes the prior is represented by a distribution itself parameterised by **hyperparameters**. In the example above of modelling a normal distribution with a fixed variance and normally distributed mean, the distribution on the mean must itself have a mean and variance. These are the hyperparameters. Sometimes the hyperparameters are fixed; otherwise they are distributed according to a **hyperprior**. The hyperprior itself may take parameters and so on, and **hierarchical priors** of this type can be constructed with an arbitrary number of levels.

Empirical Bayes refers to the practice of maximising the marginal likelihood of the data with respect to the hyperparameters at the top of the hierarchy. In this view Bayesian learning can be seen as a form of maximum likelihood learning, where it is the hyperparameters that are optimised rather than the parameters. Like maximum likelihood learning this approach ignores uncertainty, in this case uncertainty in the hyperparameter values. An alternative is to specify the hierarchical prior to a point where the top level priors are sufficiently vague that one is content to leave them unoptimised.

### Bayesian Learning of a Binomial Distribution

Consider again the same setup as in Section 3.2.1, but this time we want to learn using the Bayesian framework. Given a choice of parameter $\theta$, if we see a total of $N$ observations of which there are $a$ 1s and $b$ 0s, the likelihood is

$$P(a, b|\theta) = \frac{(a+b)!}{a!b!}\theta^a(1-\theta)^b \tag{3.18}$$

Now let's ask what the probability of $\theta$ is given $a$ and $b$. Bayes rule lets us compute this:

$$P(\theta|a,b) = \frac{P(a,N|\theta)P(\theta)}{P(a,N)} \tag{3.19}$$

$$= \frac{P(a,N|\theta)P(\theta)}{\int_0^1 d\theta\ P(a,N|\theta)P(\theta)} \tag{3.20}$$

$$= \frac{\frac{N!}{a!(b)!}\theta^a(1-\theta)^b P(\theta)}{\int_0^1 d\theta\ \frac{N!}{a!(b)!}\theta^a(1-\theta)^b P(\theta)} \tag{3.21}$$

$$= \frac{\theta^a(1-\theta)^b P(\theta)}{\int_0^1 d\theta\ \theta^a(1-\theta)^b P(\theta)} \tag{3.22}$$

If we assume that the prior $P(\theta)$ is uniform this reduces to

$$P(\theta|a,N) = \frac{\theta^a(1-\theta)^b}{\int_0^1 d\theta\ \theta^a(1-\theta)^b} \tag{3.23}$$

$$= \frac{1}{\mathrm{Beta}(a,b)}\theta^a(1-\theta)^b \tag{3.24}$$

The $\mathrm{Beta}$ function, the solution to the integral, has many forms, including $\mathrm{Beta}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$, where $\Gamma(x)$ is an extension of the factorial to the real numbers, such that for integer $x$, $\Gamma(x) = (x-1)!$.

The function in Equation 3.24 is known as the Beta distribution, and is the conjugate prior for the binomial. The parameters $a$ and $b$ must both be greater than 0. It is clear from the above derivation that they act as previous observations, as noted in Section 3.2.2. The posterior parameters, given observations $\alpha$ and $\beta$, are simply $a+\alpha$ and $b+\beta$.

### Practical Bayesian Learning

We have seen that the integral in calculating the marginal likelihood (Equation 3.17) is a key to the two tasks of parameter and model learning. For most models of interest it is not possible to analytically calculate this integral and so approximate methods must be used. One class of techniques, known as variational Bayesian expectation-maximisation, involve optimising a lower-bound on the posterior. This lower-bound is usually chosen to be more tractable than the

true posterior and thus more amenable to optimisation. Another class of techniques, known as Monte Carlo sampling, use a sample from the posterior as an approximate representation of the full posterior. One can, roughly, characterise the two classes of techniques as either exactly representing an approximate solution (variational Bayesian) or approximately representing an exact solution (Monte Carlo sampling). These two methods are further discussed in Section 3.4.2.

## 3.3 Learning Markov Decisions Processes

I now address learning the parameters and structure of MDPs. Learning MDPs is a simpler problem than learning POMDPs as there is never any doubt as to which state generated an observation. I use it as a lead in to the problem of learning POMDPs, which is addressed in Section 3.4.

### 3.3.1 Learning Parameters

Assuming we know $\mathcal{S}$, the set of states, reduces the learning problem to inferring the transition function $\mathcal{T}$, the initial state distribution $\mathcal{I}$, and the reward distribution $\rho$. This is the parameter learning problem for MDPs.

Learning the initial state distribution uses exactly the same methods as learning the transition function, so I do not consider it further. The reward function can be addressed in three ways:

- We can assume the reward function is known. In many cases, such as the mouse maze introduced in Chapter 2, this is a benign assumption as the reward function does not confer any information that helps with the rest of the model learning task.

- If we know that reward is drawn from a finite set (for example, in the mouse maze rewards always come from the set $\{-1, 100\}$) we can model the function as a set of multinomial distributions, one per state, and use the techniques discussed below. When we come to solve the MDP so learned we can use the mean of each reward distribution to construct the reward function.

- If we do not know that the rewards come from a finite set we can simply store all the rewards we experience and use their empirical mean when we need to construct the reward function.

We are left with learning the transition function. The transitions $\mathcal{T}_{ia\cdot}$ from a state $s_i$ given action $a$ are a multinomial distribution and thus the maximum likelihood solution is the maximum likelihood estimate for a multinomial. Let $n_{ij}^a$ be the number of times we have observed a transition from state $s_i$ to state $s_j$ following action $a$. The maximum likelihood estimate for $\mathcal{T}_{ia\cdot}$ is given by the element-wise estimates

$$\mathcal{T}_{iaj} = \frac{n_{ij}^a}{\sum_{j' \in \mathcal{S}} n_{ij'}^a} \tag{3.25}$$

The derivation follows that given in Section 3.2.1 for the binomial distribution, with Lagrange multipliers used to enforce normalisation.

If a transition has never been observed $n_{ij}^a$ will be zero, and hence the estimated probability will also be zero. This overconfidence can cause problems when learning a policy from the MDP. It is better to allow some small probability of connection before an observation is made. Such techniques are called **smoothing**. There are many smoothing methods. Add-one smoothing is a simple method that replaces the estimates of $\mathcal{T}_{iaj}$ with:

$$\mathcal{T}_{iaj} = \frac{n_{ij}^a + 1}{|\mathcal{S}| + \sum_{j' \in \mathcal{S}} n_{ij'}^a} \tag{3.26}$$

Bayesian methods can be seen as an alternative to ad-hoc smoothing. The conjugate prior for a multinomial distribution is the Dirichlet distribution, which is the multivariate generalisation of the Beta distribution given in Section 3.2.2. The Dirichlet is parameterised by pseudo-counts $\alpha_1, \ldots, \alpha_{|\mathcal{S}|}$, and is defined in terms of the multinomial Beta function. The multinomial Beta is:

$$\text{Beta}(\alpha_1, \ldots, \alpha_{|\mathcal{S}|}) = \frac{\prod_{i=1}^{|\mathcal{S}|} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{|\mathcal{S}|} \alpha_i)} \tag{3.27}$$

The probability density function for the Dirichlet distribution is:

$$P(\mathcal{T}_{ia1}, \ldots, \mathcal{T}_{ia|\mathcal{S}|} | \alpha_1, \ldots, \alpha_{|\mathcal{S}|}) = \frac{1}{\text{Beta}(\alpha_1, \ldots, \alpha_{|\mathcal{S}|})} \prod_{j=1}^{|\mathcal{S}|} \mathcal{T}_{iaj}^{\alpha_j - 1} \tag{3.28}$$

Like the Beta distribution the parameters should all be greater than zero. The expected value for any parameter $\mathbb{E}(\mathcal{T}_{iaj})$ is $\frac{\alpha_j}{\sum_{k=1}^{|\mathcal{S}|} \alpha_k}$

When we come to the structure learning problem we want to compare models with different numbers of states. The sum of the parameters represents the prior number of imaginary observations we have seen. This sum should be the same across all models. Setting all the counts to $\frac{1}{|\mathcal{S}|}$, giving one imaginary observation in total, is a common and reasonable choice that ensures all models have the same amount of prior information. The posterior distribution is:

$$Dirichlet(\alpha_1 + n^a_{i1}, \alpha_2 + n^a_{i2}, \ldots, \alpha_{|\mathcal{S}|} + n^a_{i|\mathcal{S}|}) \tag{3.29}$$

A MDP with Dirichlet priors over transitions is called a Bayes-adaptive MDP. A reinforcement learning algorithm for solving the Bayes-adaptive MDP is presented in (Duff 2003).

Maintaining sparsity is an issue with Dirichlet priors. Ideally our prior would prefer each state to be connected to a few other states (sparsity) but not put too much weight on a few observations (smoothness). With the standard Dirichlet we must trade-off sparsity for smoothness. An alternative hierarchical prior, introduced in (Friedman and Singer 1999), allows this control. It is used for learning MDPs in (Dearden, Friedman and Andre 1999).

### 3.3.2 Learning Structure

If $\mathcal{S}$ is not known we face the more difficult structure learning problem. In maximum likelihood learning (without smoothing) the likelihood of a transition is zero until it is observed, after which time it is updated in the usual way. This is straightforward to implement. In particular note there is no dependency between parameters, so when a new transition is observed there is no need to re-estimate the values for any other transition. If smoothing is used the normalising constant changes every time a new transition is observed, and hence all values must be re-estimated.

### The Dirichlet Process

We can address the structure learning problem in the Bayesian framework by extending the Dirichlet distribution to an infinite number of choices. The resulting distribution is called the Dirichlet process (DP) (Ferguson 1973, Antoniak 1974). Thus we can use the Dirichlet process to create a Bayesian MDP with an unbounded number of states. In this section I describe the Dirichlet process, and in the next section I describe a related distribution called the Beta process that I have used in my own work.

We are going to start by deriving the Dirichlet process as the infinite limit of the Dirichlet distribution, a derivation first presented in (Neal 1991). There are other ways of constructing the Dirichlet process: from the Gamma process (Ferguson 1973), as an urn scheme (Blackwell and MacQueen 1973), and the so-called stick-breaking process (Sethuraman 1994). We will discuss some of these later.

Let us start by considering a $K$-dimensional Dirichlet distribution with symmetric parameters $(\alpha/K, \ldots, \alpha/K)$. From this Dirichlet we can sample multinomial parameters $\mathcal{T}_{ia1}, \ldots, \mathcal{T}_{iaK} \sim Dirichlet(\alpha/K, \ldots, \alpha/K)$ in the usual way. The expected value of a parameter $\mathbb{E}(\mathcal{T}_{iaj})$ is $\frac{\alpha/K}{\alpha}$, so rather than sampling the multinomial parameters we can, if we so choose, integrate them out and work just with their expectations. This is what we are going to do with here. When we turn to the stick-breaking construction we will see how to recover these intermediate values.

After observing data $x$ we update the parameters in the standard way, with $n_{ij}^a$ being the number of observed transitions from state $i$ to $j$ following action $a$ and a total of $N$ observations. The expectation is now:

$$\mathbb{E}(\mathcal{T}_{iaj}|x) = \frac{n_{ij}^a + \alpha/K}{N + \alpha} \tag{3.30}$$

So far this is all standard for the Dirichlet distribution. Now consider what happens when the number of choices $K$ goes to infinity. The expectation converges to:

$$\mathbb{E}(\mathcal{T}_{iaj}|x) = \frac{n_{ij}^a}{N + \alpha} \tag{3.31}$$

Summing across all the transitions $\mathcal{T}_{iaj}$ the total probability mass is $\frac{N}{N+\alpha} < 1$. There is mass $\frac{\alpha}{N+\alpha}$ unassigned to any observed transition. No matter how many transitions we observe this will always be the case. We can interpret this as the probability of transitioning to a state we have not visited before. These unvisited states are all a priori the same, and there is an infinite number of them. This model is the Dirichlet process.

In the infinite Dirichlet view we have integrated out the intermediate component that is analogous to the multinomial in the finite case. We can retrieve this component via the stick-breaking construction (Sethuraman 1994). Before going into the details, let us consider what this construction will yield. A sample from the familiar Dirichlet distribution of dimension $|\mathcal{S}|$ is a set of $|\mathcal{S}|$ real numbers that sum to one. We can interpret these numbers as the parameters for a multinomial distribution. In effect we can sample a multinomial distribution from a Dirichlet distribution. We can write this as:

$$G \sim Dirichlet(\alpha_1, \ldots, \alpha_{|\mathcal{S}|}) \tag{3.32}$$

where $G = \{\mathcal{T}_1, \ldots, \mathcal{T}_{|\mathcal{S}|}\}$, $\mathcal{T}_i \in [0, 1]\ \forall i$, and $\sum_{i=1}^{|\mathcal{S}|} \mathcal{T}_i = 1$.

When we extend the Dirichlet distribution to the Dirichlet process the same basic characteristics hold, but now a sample from the Dirichlet process gives a countably infinite set of numbers, all in $[0, 1]$, which sum to $1$. We can interpret this set of numbers as the parameters for a multinomial distribution of infinite dimension. In effect, a sample from the Dirichlet process is itself a probability distribution. The stick-breaking construction allows us to construct these distributions.

So far we've been very informal when discussing probability distributions. Adding a small amount of formality is useful at this point. A probability distribution is defined on a sample space, which is the set of all possible outcomes, and it gives a probability or measure to every event. Some distributions, like the Gaussian, are defined on the reals. The multinomial is defined on some set of categories. If we have a multinomial with parameters $\{0.2, 0.5, 0.3\}$, to properly define the distribution we should specify the events that have probability of $0.2$, $0.5$, and $0.3$ of occurring though usually we finesse this issue by referring to the first, second,

and third choices. Samples from the Dirichlet process are themselves distributions and when working with the stick-breaking construction we must be explicit about the sample space on which these distributions are defined. We will see that the Dirichlet process is parameterised by a probability distribution, called the base distribution, that defines the sample space. If the base distribution is discrete, samples from the Dirichlet process are defined on exactly the same sample space as the base distribution. If the base distribution is continuous, samples from the Dirichlet distribution are defined on an infinite set of samples from the base distribution. Thus the distributions we sample from the Dirichlet process are always discrete. I sometimes refer to the elements of the sample space of a discrete distribution as the atoms, since they are the atomic units of samples.

We now have sufficient background to introduce the two parameters of the Dirichlet process:

- A **base distribution** $G_0$. This probability distribution specifies the sample space on which the Dirichlet process is defined, like the mean of the DP.
- A **strength** or **concentration parameter** $\alpha > 0$, which is like the inverse variance of the DP.

I write $G \sim DP(\alpha, G_0)$ to indicate $G$ is sampled from a Dirichlet process with parameters $\alpha$ and $G_0$.

Now let us return to the stick-breaking construction. There are two parts to it: we define how we construct the sample space from the base distribution, and we define how we give a probability or measure to each element of the sample space.

Constructing the sample space is simple: we use an infinite sequence of i.i.d. random variables $\phi_k$ sampled from $G_0$.

Constructing the measures is slightly more involved. We start with an infinite sequence of i.i.d. random variables $\pi'_k$, for $k = 1 \ldots \infty$ distributed according to $Beta(1, \alpha)$. This gives us a sequence of numbers between zero and one. To make sure they all sum to one, we derive weights $\pi_k$ as follows:

$$\pi_k = \pi'_k \prod_{l=1}^{k-1}(1 - \pi'_l) \tag{3.33}$$

Imagine we have a unit length stick. We break off a proportion of the stick equal to a $\pi'_1$, leaving $1 - \pi'_1$ of the stick remaining. The proportion we break off is $\pi_1 = \pi'_1$. From the remaining stick we break off a proportion equal to $\pi'_2$. This length of stick is $\pi_2 = \pi'_2(1 - \pi'_1)$. We continue this process untill, in the limit, there is no stick left.

Now we can define $G$ as a weighted set of samples from $G_0$:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\phi_k} \tag{3.34}$$

The function $\delta_{\phi_k}$ is the Dirac delta function centred at $\phi_k$. The stick lengths act as weights for the samples $\phi_k$ from $G_0$. Note that $G$ is a probability measure, not a number. The sum is over functions, not numbers, and so the result of the summation is itself a function.

### The Beta Process

All the distributions we've seen before the Dirichlet process are parametric distributions. That is, they are specified by a finite number of parameters. Parametric distributions are by far the most common in machine learning applications. Non-parametric distributions, on the other hand, adapt their effective number of parameters in response to data[1]. The Dirichlet process is an example of a Bayesian non-parametric distribution and the Beta process (BP) is another, which I use in my work in Chapter 7.

In the infinite Dirichlet representation of the DP a single feature (mixture component or category) is associated with each observation. In the Beta process equivalent, called the Indian Buffet Process[2] (IBP) (Griffiths and Ghahramani 2006), multiple features are associated with each observation. Imagine an binary matrix with rows being observed data and columns being

[1]More correctly one can view Bayesian non-parametric models as having an infinite number of parameters, only some of which are used to explain data.

[2]The Indian Buffet Process is infact the analogue of the Chinese Restaurant Process (CRP). The CRP is constructed from the infinite Dirichlet representation by considering the set partition induced by the infinite Dirichlet. Understanding this distinction is not important here.

features or categories. In the DP each observation is associated with a single feature or category. Therefore for each observation a single column contains a 1 and the (infinite) remaining columns contain 0. In the IBP each observation is associated with multiple categories, so each row has a number of 1s.

A sample from the IBP is a row in the binary matrix discussed above. Like the DP, to properly define the IBP we must also define the sample space – that is, what each column actually corresponds to. Thus the IBP has two parameters:

- A **base measure** $G_0$, which is like the mean of the IBP.
- A **concentration parameter** $\alpha > 0$, which is like the inverse variance of the IBP.

Note the similarity to the DP. $G_0$ defines the sample space and the mean sample, and $\alpha$ controls variation around the mean. I write $z \sim IBP(\alpha, G_0)$ to indicate a sample from an IBP with parameters $\alpha$ and $G_0$.

Samples from the IBP are generated in a similar manner to the DP. There are two components to the model: generating the binary values in the vector, and generating the atoms or sample space on which these vectors are defined.

The vectors are defined as follows: Let $k$ be the index of a column (a feature) and $i$ the index of a row (an observation), with $z_i$ the row and $z_{ik}$ be the element at row $i$ and column $k$. Assume we have $i-1$ observations, of which $n_k$ have feature $k$, and the largest value of $k$ is $K$. Then the conditional distribution for $z_i$ is:

$$z_{ik} \sim Bernoulli(\frac{n_k}{i}) \qquad k \leq K \tag{3.35}$$

$$n_{new} \sim Poisson(\frac{\alpha}{i}) \tag{3.36}$$

$$z_{ik} = 1 \qquad k \in \{K+1, \ldots, K+n_{new}\} \text{ and } n_{new} > 0 \tag{3.37}$$

The sample space is given by an infinite sequence of samples $\phi \sim G_0$. Associated with each column $k$ is a particular sample $\phi_i \sim G_0$. We typically assume there is some observation

function $F$ that is parameterised by $z$ and $\phi$. Let $x_i$ be the observation associated with $z_i$ and $\phi_:$ be all the samples $\phi_i$. Then $x_i \sim F(z_i, \phi_:)$ or equivalently $P(x_i|z_i, \phi_i) = F(z_i, \phi_i)$.

There is also a stick breaking representation for the IBP (Teh, Gorur and Ghahramani 2007) which generates an intermediate distribution analogous to the multinomial in the DP. It starts with an infinite sequence of i.i.d. random variables $\pi'_k \sim Beta(\alpha, 1)$. The stick weights $\pi_k$ are generated by the equation:

$$\pi_k = \pi'_k \pi_{k-1} = \prod_{i=1}^{k} \pi'_k \tag{3.38}$$

The IBP can be constructed from the Beta and Bernoulli processes, which form a conjugate pair like the Dirichlet and multinomial. This gives a slightly more general model than the IBP and provides some insight for constructing more complex models. Here I address the Beta process (Hjort 1990, Thibaux and Jordan 2006) directly. I restrict discussion to the discrete Beta process – it is far simpler than the continuous Beta process and sufficient for my purposes.

I write $C \sim BP(c, N)$ to indicate $C$ is drawn from a Beta process with **concentration parameter** $c$ and **base measure** $N$. I require that $N$ is discrete, so it can be written as $\sum_i n_i \delta_{\omega_i}$. The Beta process requires $n_i \in [0, 1]$. $C$ has atoms at the same location as $N$. Given an element $n_i \delta_{\omega_i}$, the corresponding element $c_i \delta_{\omega_i}$ has measure:

$$c_i \sim Beta(cn_i, c(1 - n_i)) \tag{3.39}$$

This generates the equivalent of the sticks in stick-breaking representation of the IBP. To generate the binary elements we see in the IBP it is necessary to use the Bernoulli process. I write $E \sim BeP(C)$ for $E$ distributed according to a Bernoulli process with **base measure** $C$. $E$ has atoms at the same location as $C$. Given an element $c_i \delta_{\omega_i}$, the corresponding element $e_i \delta_{\omega_i}$ has measure:

$$e_i \sim Bernoulli(c_i) \tag{3.40}$$

Informally, the Bernoulli process generates a sequence of weighted coin flips, the weights

of which are taken from the samples from the Beta process. The Beta-Bernoulli process form a conjugate pair. Given a set of observations $E_{1:n} = \{E_1, \ldots, E_n\}$

$$C|E_{1:n} \sim BP\left(c+n, \frac{c}{c+n}N + \frac{1}{c+n}\sum_{i=1}^{n} E_i\right) \tag{3.41}$$

## 3.4 Learning POMDPs

I now turn to learning POMDPs from data, considering again the parameter and structure learning problems. Both problems are made harder by no longer knowing the association between observations and states. Before discussing POMDP specific techniques I introduce general methods for learning models with hidden variables: the expectation-maximisation algorithm, variational Bayesian expectation-maximisation, and Markov chain Monte Carlo sampling.

### 3.4.1 The Expectation-Maximisation Algorithm

The expectation-maximisation (EM) algorithm (Dempster and Laird 1977) is the standard algorithm for maximum likelihood learning of models with hidden variables. My description is based on (Beal 2003), (Neal and Hinton 1998), and (Borman 2004) and presents the modern view that leads directly to variational approximations.

To quickly recap notation, in general the data $x$ is a set of $n$ i.i.d. values $\{x_1, \ldots, x_n\}$. The model we are trying to fit has hidden variables $y = \{y_1, \ldots, y_k\}$. The log likelihood can be written as a function of $\theta$ as:

$$\mathcal{L}(\theta) \equiv \ln P(x|\theta) = \sum_{i=1}^{n} \ln P(x_i|\theta) = \sum_{i=1}^{n} \ln \int dy\ P(x_i, y|\theta) \tag{3.42}$$

Marginalising out the hidden variables is necessary to make the likelihood of the parameters just a function of the observed data. For complex models the marginalisation of $y$ may be intractable, so we must seek approximate methods. We can approximate the likelihood by introducing *any* auxiliary distribution $q_y(y)$ over the hidden variables, giving a lower bound on $\mathcal{L}$. For each data point $x_i$ we can use a distinct distribution $q_{y_i}(y_i)$ over the hidden variables.

This gives a bound:

$$\mathcal{L}(\theta) = \sum_i \ln \int dy\ P(x_i, y|\theta) \tag{3.43}$$

$$= \sum_i \ln \int dy_i\ q_{y_i}(y_i) \frac{P(x_i, y_i|\theta)}{q_{y_i}(y_i)} \tag{3.44}$$

$$\geq \sum_i \int dy_i\ q_{y_i}(y_i) \ln \frac{P(x_i, y_i|\theta)}{q_{y_i}(y_i)} \tag{3.45}$$

$$= \sum_i \int dy_i\ q_{y_i}(y_i) \ln P(x_i, y_i|\theta) - q_{y_i}(y_i) \ln q_{y_i}(y_i) \tag{3.46}$$

$$\equiv \mathcal{F}(q_y(y), \theta) \tag{3.47}$$

The inequality follows from Jensen's inequality and the concavity of the logarithm.

**EM for exact optimisation**

The expectation-maximisation algorithm alternates between an E-step that calculates the posterior distribution over the hidden variables given the current parameter setting, and an M-step that maximises the parameter setting given the information calculated in the E-step. The E-step maximises $\mathcal{F}(q_y(y), \theta)$ with respect to $q_y(y)$ and the M-step maximises with respect to $\theta$. More formally:

$$\textbf{E step:}\ \ q_{y_i}^{t+1} = \underset{q_{y_i}}{\operatorname{argmax}}\, \mathcal{F}(q_y(y), \theta^t) \qquad \forall i \in \{1, \ldots, k\} \tag{3.48}$$

$$\textbf{M step:}\ \ \theta^{t+1} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{F}(q_y^{t+1}(y), \theta) \tag{3.49}$$

By direct substitution into Equation 3.45 it can be seen that the E-step is maximised when $q_{y_i}^{t+1}(x) = P(y_i|x_i, \theta)$ for all $i$. When this is the case $\mathcal{F}$ is the expectation of the log likelihood with respect to the hidden variables $x$, which indicates the origin of the name of this part of the EM algorithm.

The M-step optimisation can be achieved by taking the derivative of Equation 3.45 with respect to $\theta$ and setting it to zero. This is the same as optimising the first term of Equation 3.46, as the second term does not depend on $\theta$. Thus we can write the M-step as:

$$\textbf{M step: } \theta^{t+1} = \arg\max_{\theta} \sum_i \int dy_i \; P(y_i|x_i, \theta^t) \ln P(x_i, y_i|\theta) \tag{3.50}$$

Note that the first $\theta$ in Equation 3.50 is held fixed; the optimisation is over the second $\theta$.

After each E-step $\mathcal{F}(q_y^{t+1}(y), \theta^t) = \mathcal{L}(\theta^t)$. Each M-step finds a new $\theta^{t+1}$ which maximises the lower bound on the log likelihood and corresponds to a new and higher log likelihood. Since the E-step does not change the parameters the log likelihood is guaranteed to not decrease after each combined EM step.

### EM for approximate optimisation

In many practical models the posterior distribution over $P(y_i|x,\theta)$ is intractable, and thus the exact EM algorithm cannot be computed. However an approximate optimisation can still take place if $q_y(y)$ is constrained to a tractable form.

The lower bound $\mathcal{F}(q_y(y), \theta)$ can be written as:

$$\mathcal{F}(q_y(y), \theta) = \sum_i \int dy_i \; q_{y_i}(y_i) \ln \frac{P(x_i, y_i|\theta)}{q_{y_i}(y_i)} \tag{3.51}$$

$$= \sum_i \int dy_i \; q_{y_i}(y_i) \ln \frac{P(y_i|x_i, \theta)P(x_i|\theta)}{q_{y_i}(y_i)} \tag{3.52}$$

$$= \sum_i \int dy_i \; q_{y_i}(y_i) \ln P(x_i|\theta) + \sum_i \int dy_i \; q_{y_i}(y_i) \ln \frac{P(y_i|x_i, \theta)}{q_{y_i}(y_i)} \tag{3.53}$$

Note that $q_{y_i}$ in the first term of Equation 3.53 can be integrated out, leaving:

$$\mathcal{F}(q_y(y), \theta) = \sum_i \ln P(x_i|\theta) - \sum_i \int dy_i \; q_{y_i}(y_i) \ln \frac{q_{y_i}(y_i)}{P(y_i|x_i, \theta)} \tag{3.54}$$

From this we can see the E-step, maximising $q_y(y)$, is equivalent to minimising the quantity

$$\int dy_i \; q_{y_i}(y_i) \ln \frac{q_{y_i}(y_i)}{P(y_i|x_i, \theta)} \equiv \mathrm{KL}(q_{y_i}(y_i)||P(y_i|x_i, \theta)) \geq 0 \tag{3.55}$$

This is the KL-divergence between the approximation $q_y(y)$ and the exact posterior $P(y_i|x_i, \theta)$. The KL-divergence is a measure of the difference between two probability measures. Given two probability distributions $P$ and $Q$, the KL-divergence between $P$ and $Q$ is:

$$KL(P||Q) = \int dx \; P(x) \ln \frac{P(x)}{Q(x)} \tag{3.56}$$

Note that the KL-divergence is not symmetric, and thus KL-divergence is not a metric.

The M-step is the same as the exact optimisation Equation 3.50 but is based on the variational posterior over hidden variables.

$$\textbf{M step:} \;\; \theta^{t+1} = \sum_i \int dy_i \; q_{y_i}^{t+1}(y) \ln P(x_i, y_i|\theta) \tag{3.57}$$

So the approximate EM algorithm alternates between maximising the KL-divergence between the approximation $q_y(y)$ and the posterior over the hidden variables, and maximising the parameters given the approximation.

### 3.4.2 Methods of Approximate Bayesian Learning

As described in Section 3.2.2 we must often turn to approximate methods for Bayesian learning. In this section I describe two families of techniques, variational Bayesian EM and Monte Carlo sampling. One can, roughly, characterise the two classes of techniques as either exactly representing an approximate solution (variational Bayesian EM) or approximately representing

an exact solution (Monte Carlo sampling).

### Variational Bayesian EM

Variational Bayesian EM extends maximum likelihood EM to maintain a distribution over both the hidden variables $y$ and the parameters $\theta$. Recall in Bayesian learning we are attempting to compute the posterior over parameters $P(\theta|x,m)$, which is proportional to $\mathcal{L}(\theta)P(\theta|m)$, or the marginal likelihood $P(x|m)$. The focus here is on solving the latter problem as it requires a solution to the former and thus subsumes it.

If we treat the parameters as well as the hidden variables as uncertain quantities, as we do in Bayesian learning, the parameters and hidden variables become correlated in the posterior. The variational approach to Bayesian learning is to approximate the distribution over parameters and hidden variables with a simpler tractable distribution. Usually the simplification is to assume the parameters and hidden variables are independent.

We can get a lower bound on $P(y|m)$ by using Jensen's inequality in the same way as Equation 3.45:

$$\ln P(x|m) = \ln \int d\theta\, dy\; P(x,y,\theta|m) \tag{3.58}$$

$$= \ln \int d\theta\, dy\; q(y,\theta) \frac{P(x,y,\theta|m)}{q(y,\theta)} \tag{3.59}$$

$$\geq \int d\theta\, dy\; q(y,\theta) \ln \frac{P(x,y,\theta|m)}{q(y,\theta)} \tag{3.60}$$

$$\tag{3.61}$$

Here $q(y,\theta)$ is any distribution over the hidden variables and parameters with support where $P(y,\theta|x,m)$ does. Direct substitution shows the inequality is maximised when $q(y,\theta) = P(y,\theta|x,m)$, but this does not help matters as the normalising constant of $P(y,\theta|x,m)$ is the marginal likelihood. Progress can be made by constraining $q(y,\theta)$ to a factored form $q(y,\theta) \approx q_y(y)q_\theta(\theta)$.

$$\ln P(x|m) \geq \int d\theta\, dy\; q_y(y) q_\theta(\theta) \ln \frac{P(x,y,\theta|m)}{q_y(y) q_\theta(\theta)} \tag{3.62}$$

$$= \int d\theta\, q_\theta(\theta) \int dy\; q_y(y) \left( \ln \frac{P(x,y,\theta|m)}{q_y(y)} + \ln \frac{P(x,y,\theta|m)}{q_\theta(\theta)} \right) \tag{3.63}$$

$$= \mathcal{F}_m(q_y(y), q_\theta(\theta)) \tag{3.64}$$

$$= \mathcal{F}_m(q_{y_1}(y_1), \ldots, q_{y_n}(y_n), q_\theta(\theta)) \tag{3.65}$$

The last line follows from the i.i.d. data.

Variational Bayesian EM iteratively maximises $\mathcal{F}_m(q_y(y), q_\theta(\theta))$ with respect to $q_y(y)$ and $q_\theta(\theta)$. The update equations for $q_y(y)$ and $q_\theta(\theta)$ are given as Theorem 2.1 in (Beal 2003):

$$\textbf{VBE step: } q_{y_i}(y_i)^{t+1} = \frac{1}{\mathcal{Z}_{y_i}} exp\left( \int d\theta\; q_\theta^t(\theta) \ln P(x_i, y_i|\theta, m) \right) \qquad \forall i \tag{3.66}$$

$$\textbf{VBM step: } q_\theta^{t+1}(\theta) = \frac{1}{\mathcal{Z}_\theta} P(\theta|m) exp\left( \int dy\; q_y^{t+1}(y) \ln P(x, y|\theta, m) \right) \tag{3.67}$$

where $\mathcal{Z}_{y_i}$ and $\mathcal{Z}_\theta$ are normalising constants.

By rewriting Equation 3.62 it can be shown that maximising $\mathcal{F}_m(q_y(y), q_\theta(\theta))$ is equivalent to minimising the KL-divergence $\mathrm{KL}(q_y(y)q_\theta(\theta)||p(x, y|\theta, m))$ between $\mathcal{F}_m(q_y(y), q_\theta(\theta))$ and $P(x, y|\theta, m)$. Note the difference to Equation 3.55. The EM algorithm minimises the KL-divergence of the distribution over hidden variables, whereas the variational Bayesian EM minimises the KL-divergence of the distribution over hidden variables *and* the distribution over parameters.

### Monte Carlo Sampling

Monte Carlo techniques allow us to sample from the true posterior, and thus represent it as a set of samples. There are a large number of techniques that fall under the Monte Carlo designation. More information can be found in (MacKay 1998) and (Andrieu, Freitas, Doucet et al. 2003), for example. Here I focus on Markov chain Monte Carlo methods.

The goal of Markov Chain Monte Carlo (MCMC) methods is to sample from a distribution $Q(x)$. We assume we can evaluate $Q(x)$ up to a normalising factor $Z$ and denote this function $Q^*(x) = ZQ(x)$.

The key idea of MCMC methods that we generate samples from a Markov chain, and construct the chain in such a way that the samples converge to $Q(x)$ as time goes to infinity. We now formally define this concept.

A Markov Chain is a stochastic process with the property that:

$$P(x_{t+1}|x_t, \ldots, x_1) = P(x_{t+1}|x_t) \tag{3.68}$$

This means past is irrelevant given the present. We can represent a finite Markov chain by a transition matrix $T$, where $T(x'|x) = P(x'|x)$. In the continuous (infinite) case the Markov chain has a transition kernel $K$.

We want the probability of the Markov chain being in any state $x$ to converge to $Q(x)$ no matter what the starting state is. If this holds, we say that $Q(x)$ is the invariant, or stationary, distribution of the chain. We can guarantee there is a unique invariant distribution if the following two properties hold:

1. Irreducibility. For any state there is positive probability of (eventually) visiting all other states

2. Aperiodicity. The chain should not get caught in any cycles, so for all time after some time step there is a non-zero probability the chain can be in any state

A Markov chain satisfying these two conditions is known as ergodic. A sufficient, but not necessary, condition for an ergodic Markov chain to converge to $Q(x)$ is that detailed balance holds:

$$Q(x)T(x'|x) = Q(x')T(x|x') \tag{3.69}$$

Markov chains that satisfy detailed balance are also known as reversible.

**The Metropolis Algorithm** Metropolis sampling is the first example of a MCMC algorithm. The Markov chain is constructed from two parts:

1. A proposal distribution $R(x'|x)$, itself a Markov chain
2. A rejection rule for samples from $R$.

The distribution $R$ must be symmetric:

$$R(x'|x) = R(x|x') \tag{3.70}$$

The probability of accepting a state $x'$ given current state $x$ is:

$$A(x, x') = min\left(1, \frac{Q(x')}{Q(x)}\right) \tag{3.71}$$

If the proposed state $x'$ is not accepted the current state $x$ is retained as the sample for this time step.

The Metropolis algorithm is a form of stochastic gradient ascent. If $Q(x')$ is greater than or equal to $Q(x)$ the proposal is always accepted. Only if $Q(x')$ is less than $Q(x)$ might the proposal be rejected. This causes the samples to concentrate on regions of high probability.

**The Metropolis-Hastings Algorithm** We can generalise the Metropolis algorithm to cases where $R$ is not symmetric by using as the acceptance probability:

$$A(x, x') = min\left(1, \frac{Q(x')R(x|x')}{Q(x)R(x'|x)}\right) \tag{3.72}$$

This is known as the Metropolis-Hastings algorithm. Intuitively we want to move uphill in the target distribution $Q$ but downhill in the proposal distribution $R$. The downhill movement forces $R$ explore the entire state space.

The transition kernel of the Markov chain constructed by the Metropolis-Hastings algorithm is:

$$K_{MH}(x'|x) = R(x'|x)A(x,x') + \delta_x(x')r(x) \tag{3.73}$$

where $r(x)$ is the rejection probability:

$$r(x) = \int_x R(x'|x)(1 - A(x,x'))dx \tag{3.74}$$

The Metropolis algorithm explores $Q(x)$ by a random walk; an inefficient method of transportation. If the largest length scale of the space of probable states is $L$, and the Metropolis algorithm generates a random walk with step size $\epsilon$, then at least $(L/\epsilon)^2$ iterations must be run before the invariant distribution is reached. However, we can't set $\epsilon$ too high. Any higher than the smallest length scale and the rejection rate will rise. Hence reducing random walk behaviour is a key to efficient MCMC.

The time it takes an MCMC chain to converge to the invariant distribution is known as the mixing time. A chain that converges quickly is said to mix well. Note that MCMC has no dependency on the dimension of the space of $x$.

**Mixtures and Cycles of MCMC Kernels** Several MCMC samplers can be combined into one. If the transition kernels $K_1$ and $K_2$ both have invariant distribution $Q$ then the cycle hybrid kernel $K_1K_2$, and the mixture hybrid kernel $vK_1 + (1-v)K_2$, $v \in [0,1]$, also have invariant distribution $Q$.

**Gibbs Sampling** Gibbs sampling is a cycle hybrid method for $n$ dimensions ($n > 1$) when the full conditionals are known and are easy to sample from. Gibbs sampling samples a single variable at a time from the variable's conditional distribution given the current sampled values of all other variables. Let $x_j$ be the variable we are sampling. I write $x_{-j} = \{x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n\}$ to indicate all variables other than $x_j$. The conditional distributions of interest are thus:

$$Q(x_j|x_{-j}) \stackrel{def}{=} Q(x_j|x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) \tag{3.75}$$

We can now define the proposal distribution $R$ in terms of the conditional distributions:

$$R(x'|x) = \begin{cases} Q(x'_j|x_{-j}) & \text{if } x'_{-j} = x_{-j} \\ 0 & \text{otherwise} \end{cases} \tag{3.76}$$

What this means is we choose a single variable and sample a new value from the conditional distribution for that variable with all the other variables held constant. We can choose the variable at random, or simply cycle through all variables. The later choice known as the deterministic Gibbs sampler. The acceptance probability of the Gibbs sampler is always 1, and it is particularly simple to implement in Bayesian networks, such as POMDPs, that are expressed in terms of their conditional distributions.

### 3.4.3 Learning Parameters

Having explored general methods for learning in models with hidden variables we now return to learning POMDPs. The Baum-Welch algorithm (Rabiner 1989) is the classic algorithm for maximum-likelihood parameter learning of HMMs. The algorithm is in turn built on the forward-backward algorithm, which is a dynamic programming method for calculating the probability of occupying each state at each time step given the observation sequence. The Baum-Welch algorithm uses this information to calculate the expected frequency of each transition and observation pair which in turn determines the maximum likelihood parameter settings. This process iterates till convergence. Baum-Welch is efficient, taking $O(|\mathcal{S}|^2 T)$ time and $O(\mathcal{S})$ space, where $T$ is the length of the training sequence. It is a form of expectation maximisation (see Section 3.4.1), and as such only converges to a local maximum.

Baum-Welch has been applied to learning POMDPs in (Chrisman 1992) and (Nikovski and Nourbakhsh 2000). The models learned are very small, which makes it difficult to assess the generality of the algorithm in the POMDP situation. It is well known in the HMM case that the algorithm is very sensitive to the initial conditions (Ferrer, Alonso and Travieso 2000, Moghaddam and Piccardi 2009).

The primary issue with Bayesian methods is calculating the posterior. In particular, for

the HMM/POMDP the issue is the exponentially increasing number of pathways through the state space. Typically we model the transition probabilities with a Dirichlet/multinomial pair for each state. Recall the parameters of the Dirichlet represent counts of the number of times each transition is taken. To exactly calculate the posterior we should consider taking every possible pathway through the state space, the number of which increase exponentially with the sequence length. Then the posterior can be represented as a mixture with each component representing a single pathway via appropriate adjustment of the pseudo-counts. Clearly this is impractical so Bayesian methods have used various approximations. In this section I cover a Viterbi approximation, a variational approximation, and the Bayes-adaptive approximation.

The Viterbi approximation, used in (Stolcke and Omohundro 1993), uses the single highest probability path through the hidden states, calculated using the Viterbi algorithm. This choice allows the parameter posterior to remain Dirichlet. The MAP value of the parameter posterior is used to calculate the Viterbi sequence when the next observation sequence arrives.

The variational algorithm for HMMs (MacKay 1997, Beal 2003) is the Bayesian analog to Baum-Welch – so much so that the algorithm is almost identical. The approximation that makes the algorithm tractable is to assume that the posterior factorises into a component for the parameters and a component for the hidden state sequence. Under this condition the posteriors remain Dirichlet. I discuss this algorithm in more depth in Chapter 6, where I adapt it to POMDPs.

The Bayes-adaptive POMDP (Ross et al. 2007) maintains a posterior that is a mixture of Dirichlet distributions. It regains tractability by bounding the parameters to a subspace. It is shown that an $\epsilon$-optimal policy can still be learned in this bounded space. This algorithm is discussed further in Chapter 6.

### 3.4.4 Learning Structure

The most obvious way to find the structure is to simply learn the parameters for a number of models of different structure and then, optionally, pick the best one. This is inefficient; we might hope we could at least share information between models, or perhaps even grow a model

incrementally.

A common method is to start with a single state and split states when some test indicates this is beneficial. For example, CSSR (Shalizi and Klinkner 2004) splits states when a statistical test suggests a state violates the Markov property. The Utile-HMM (Wierstra and Wiering 2004) does a similar thing, though in this case it looks for returns that are not normally distributed.

Merging is the opposite of splitting: it starts with the maximum likelihood prefix tree and merges states that a test indicates are beneficial. Merging algorithms generally take more time than splitting ones as they start with a larger model and time complexity is typically polynomial in the number of states. The Bayesian model merging algorithm (Stolcke and Omohundro 1993) greedily merges when the posterior probability is improved.

Bayesian non-parametric distributions provide an alternative to splitting and merging. The infinite hidden Markov model (Beal, Ghahramani and Rasmussen 2002, Teh, Jordan, Beal et al. 2006, Gael, Saatci, Teh et al. 2008) is built upon the hierarchical Dirichlet process. We have seen how the DP can be used to build a mixture model with an infinite number of components. Each state of a HMM is a mixture model, the components being the other states of the HMM. So from the DP we can build an infinite HMM, with a top-level DP being necessary to ensure the state-level DP all share the same atoms. The model has also been applied to POMDPs, in the appropriately named infinite POMDP (Doshi-Velez 2009). There is some work exploring models suitable for sparse HMMs (Williamson, Wang, Heller et al. 2010). This work is presented in the context of topic models, and has yet to be applied to HMMs or POMDPs as far as I am aware.

## 3.5 Other Model Learning Methods

There are many other approaches to learning HMMs and POMDPs. Rather than describing the individual approaches I attempt to describe the general themes that have developed and categorise the works accordingly. I use the following categories:

- Objective function: What is the function that the algorithm tries to optimise?

- Fixed or variable structure: The simplest algorithms assume a fixed size model. In other words, they don't attempt to learn structure. Many algorithms do alter the structure of the model, for example by splitting or merging states according to some heuristic.

- Population versus single model: Most algorithms learn only a single model, but some learn a set of models allowing them to explore more alternatives.

### 3.5.1 Objective Functions

The data likelihood in a POMDP is calculated under a first-order Markov assumption. The Markov assumption ignores data dependencies beyond one-step. For example a Markov model would assign the same probability to the sequences of symbols $abaca$ and $acaba$. Anecdotally, some authors claim this is a source of poor performance and suggest longer-term dependencies must be accounted for (see, e.g., (Wierstra and Wiering 2004)). Hence some alternative objective functions have been considered.

**First Passage Time** One such objective function can be built using first passage times (FPTs), also known as first hitting times. Given a sequence of data, a start symbol $a$, and an end symbol $b$, the first passage time between $a$ and $b$ is the multiset containing the number of steps to the first occurrence of $b$ following each occurrence of $a$. For example, given the sequence:

$$\mathbf{a}c\mathbf{ab}c\mathbf{ba}cc\mathbf{b}$$

the first passage times from $a$ to $b$ are:

$$\{2, 0, 2\}$$

The first passage times can alternatively be represented as the set of pairs $\{(t_1, o_1), \ldots, (t_n, o_n)\}$ where $t_i$ is a passage time and $o_i$ is the number of occurrences of $t_i$. The above example in this representation is:

$$\{(0,1),(1,0),(2,2)\}$$

First passage times capture long-term relationships between symbols that are omitted in first-order features (Callut and Dupont 2007). First passage times are governed by phase-type distributions. (Callut and Dupont 2007) shows how a HMM variant, known as a partially observable Markov model (POMM), can be viewed as a phase-type distribution. Thus we have the data, the observed FPTs, the objective function, maximising the likelihood of the FPTs, and the model, the POMM from which we can calculate the likelihood of interest. The parameters of the POMM can be fit using an EM-type algorithm (Callut and Dupont 2007). Addition steps, involving adding new states and trimming infrequently used transitions, are used to learn the structure.

**Unlimited History Pairs** In (Abbeel and Ng 2005) the authors propose using an objective function that includes all possible pairs of observations. They are limited to fully observable models with deterministic actions. Given a sequence $(s_1, \ldots, s_t)$ they propose to optimise $\sum_{t=0}^{T-1} \sum_{k=1}^{T-k} \gamma^k log P(s_{t+k}|s_t)$. They present an EM-like algorithm for doing so.

**Viterbi Path Counting** The Viterbi Path Counting algorithm (Davis and Lovell 2004) uses the Viterbi algorithm to find the most likely allocation of observations to hidden states. In this way it uses an approximate first-order Markov objective function.

**KL-Divergence** The KL-divergence, or relative entropy, is a measure of difference between two probability distributions. The prefix tree constructed from an observation sequence is the maximum likelihood model for that sequence. The MDI algorithm (Thollard, Dupont and de La Higuera 2000) controls its search by computing the KL-divergence between a proposed model and the maximum likelihood prefix tree.

The generalised KL-divergence, or I-divergence, is the objective function for the non-negative matrix factorization at the heart of (Cybenko and Crespi 2011). The general principle is the same: minimising the distortion from the observed data sequence.

**$L_2$ one-step Markov error** The core of (Hsu, Kakade and Zhang 2009) is a singular value decomposition (SVD) of the matrix single step probabilities (that is, $P(x_2 = i, x_1 = j) \forall i, j \in \mathcal{O}$). The SVD is the optimal $L_2$-preserving matrix decomposition and hence this is the objective function of the algorithm.

### 3.5.2 Learning Structure

We have already seen state splitting, state merging, and non-parametric methods for learning structure. The method of (Cybenko and Crespi 2011) tries to "guess" the correct size of the HMM. This is achieved by calculating the eigenvalues of a matrix counting occurences of all prefix-suffix pairs. A large gap between eigenvalues indicates where the cut-off in states should occur.

### 3.5.3 Summary

Table 3.5.3 summarises the methods.

## 3.6 Conclusions

We have seen how MDPs and POMDPs can be learned from both the maximum likelihood and Bayesian perspectives. In an unknown and uncertain world, Bayesian methods are particularly attractive as they explicitly model the uncertainty in the true model. The variational algorithm for hidden Markov models has not previously been applied to POMDPs; in Chapter 6 I discuss its implementation and give experimental results. In the next chapter, however, I complete my survey of fundamental topics, addressing reinforcement learning and in particular policy search.

Table 3.1: A summary of methods for learning HMMs and related models.

| NAME | OBJECTIVE FN | STRUCTURE LEARN. | POPULATION |
|---|---|---|---|
| Baum-Welch [1] | First order Markov (FOM) | No | No |
| POMMPHit [2] | FPT | Splitting | No |
| (Abbeel and Ng 2005) | Unltd history pairs | No | No |
| Spectral HMM [3] | $L_2$ | No | No |
| (Cybenko and Crespi 2011) | I-divergence | Adaptive | No |
| CSSR [4] | FOM | Splitting | No |
| Utile-HMM [5] | FOM + return | Splitting | No |
| Viterbi Path Counting [6] | Modified FOM | No | Averaged |
| MDI [7] | KL-divergence | Merging | No |
| Variational Bayes [8] | FOM $\times$ prior | No | Yes (implicit) |
| Bayesian Model Merging [9] | FOM $\times$ prior | Merging | Yes (implicit) |
| Bayes-adaptive POMDP [10] | | No | Yes |
| iPOMDP [11] | FOM $\times$ prior | Infinite | Yes (sampling) |

[1] (Chrisman 1992)
[2] (Callut and Dupont 2007)
[3] (Hsu et al. 2009)
[4] (Shalizi and Klinkner 2004)
[5] (Wierstra and Wiering 2004)
[6] (Davis and Lovell 2004)
[7] (Thollard et al. 2000)
[8] (MacKay 1997, Beal 2003)
[9] (Stolcke and Omohundro 1993)
[10] (Ross et al. 2007)
[11] (Doshi-Velez 2009)

# CHAPTER 4

# FOUNDATIONS: HOW TO ACT WITHOUT A MODEL

## 4.1 Introduction

In this chapter I review reinforcement learning algorithms for unknown POMDPs. That is, methods for learning a policy without a model. We have seen in Section 2.4 that the optimal policy for an MDP requires no memory as the current state is a sufficient statistic. The information state is a sufficient statistic for a POMDP (see Section 2.6.1), but this cannot be computed without access to the environment model. Reinforcement learning algorithms operating in POMDPs attempt to approximate an information state by remembering their past interactions with the environment. The form this memory takes is one of the key axes for classifying algorithms. There are five main kinds of representations I consider:

- Memoryless policies avoid the issue of memory by having none.
- Finite history methods use a finite (but not necessarily fixed) window of past interactions to disambiguate the current state.
- Finite state policies use a finite state machine as their memory, and thus can represent events indefinitely far in the past.
- Recurrent neural networks are a Turing complete representation modelling the controller

as a neural network with feedback loops.

- Other representations, generally Turing complete, that have not been a major focus for the field and do not fit into the above categories.

The other main axis is the learning algorithm. Here there are two main kinds:

- Gradient ascent methods follow an estimated gradient in the parameter space to find parameter settings that give a locally maximum value.

- Evolutionary methods are stochastic search methods that use a population of policies to avoid local maxima.

Finally, generalisation over unseen histories is a concern all algorithms face. Due to observation noise it is often the case that the agent's current history of interactions will not exactly match any prior history. If the agent is able to use what it has learned from histories that are approximately the same (for some suitable definition of "approximately") it is said to generalise over histories. Algorithms differ in their ability to generalise. Some, like NSM (Section 4.3) do not generalise at all. Some, like recurrent neural networks, have generalisation built-in to their definition of memory. Finally some algorithms may or may not generalise according to the exact implementation used.

## 4.2 Memoryless Policies

We have seen in Section Section 2.4 that an optimal policy for an MDP requires no memory. Reinforcement learning algorithms for MDPs (see, for example, (Kaelbling, Littman and Moore 1996, Sutton and Barto 1998)) exploit this to learn an optimal memoryless (stationary) policy without a world model. These methods can be applied directly to POMDPs, treating the observations as if they were the underlying state. (Singh et al. 1994) shows some important results about such methods:

- Deterministic stationary policies learned in this way can have arbitrarily high loss with respect to the infinite-horizon discounted sum of rewards

- Stochastic stationary policies can perform arbitrarily better than deterministic stationary policies.
- Stochastic stationary policies can still have arbitrarily high loss.

Furthermore, it is NP-hard (Littman 1994) to find an optimal mapping between observations and actions. Nonetheless, several authors have applied these techniques to POMDPs. (Jaakkola, Singh and Jordan 1995) presents a gradient-ascent algorithm for learning Q-values from a Monte-Carlo estimate of value. This method can find a policy that is a local optimum given the starting conditions. (Loch and Singh 1998) shows that, empirically, eligibility trace techniques outperform Q-learning and can find good policies for POMDPs. These results are confirmed in (Crook and Hayes 2003).

We can see stochastic policies as a weak form of generalisation. A deterministic memoryless policy considers all states with the same observation the same. A stochastic policy relaxes this constraint, allowing the policy to represent how much perceptually aliased states do in fact warrant the same action.

## 4.3 Finite History Policies

Finite history methods use a finite, but not necessarily fixed, length window of past interactions to construct an information state.

Nearest Sequence Memory (NSM) (McCallum, Tesauro, Touretzky et al. 1995) is a nearest neighbour method for learning Q-values in POMDPs. It stores the agent's entire history of observations. To choose an action it finds the $k$-nearest neighbours for each possible future action, and chooses the action for which the average matching Q-value is highest. The metric for computing nearest neighbours is simply the length of the sub-sequences that match a subsequence of the history starting at the current time. Note that NSM offers no generalisation over histories. If two sequences differ due to a noisy observation they will be treated differently.

History may be compactly represented as a probabilistic suffix tree (Ron, Singer and Tishby 1996), which is the approach taken by the Utile Suffix Memory (USM) (McCallum 1995a).

The root of the tree represents the current time step. Child nodes represent alternatively observations and actions. The tree terminates at a fringe, which occurs when the agent cannot find a statistically significant difference in value given further history information. The tree grows as more utile distinctions, that is statistically significant differences in value, are observed. The leaf nodes of the tree represent the states of the world. Q-values, updated via value iteration, are used to decide on actions. Unlike NSM, the USM can generalise over histories.

There have been many extensions to USM. U-Tree (McCallum 1995b) extends USM to factored observations. Bayesian learning of history tree models (BLHT) (Suematsu and Hayashi 1999) finds models with the same structure as USM, but uses a Bayesian method to avoid overfitting. By maintaining only the MAP model and constraining the form of the prior they are able to use only constant computation at each step, which USM does not achieve. USM does not handle sensor noise (where a single state can appear different each time it is visited) as it will store such histories under different paths through the tree. NUSM (Brafman and Shani 2005) addresses this issue by storing histories in each possible path with a weight proportional to the probability that the observed history is actually the path corrupted by noise. (Dutech 2000) presents a similar algorithms to USM with an enhancement for the case when the POMDP has deterministic transitions. (Au and Maire 2004) splits the leaves based on the information gain in value replacing the statistical test used in the original USM. Experimental results show an improvement over the original USM.

Window-Q (Lin and Mitchell 1992) learns Q-values with an artificial neural network (ANN). The inputs to the ANN are the last $n$ observations and actions. Such a model naturally has some generalisation capabilities.

Finite history can only solve perceptual aliasing if the history extends far enough into the past. It is trivial to create a POMDP that can foil any finite history window. An example POMDP is shown in Figure 4.1. In this environment the agent must move from the Start location to the Load location and return. It requires one bit of state to determine in which direction the agent should move. If a finite history is long enough to remember the last state change a finite history method can solve this problem optimally. However it is trivial to extend the length of

the corridor to defeat any given finite history window.

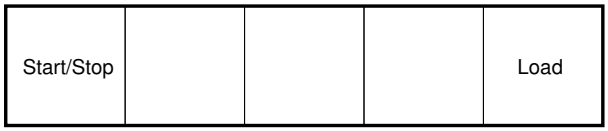


Figure 4.1: The Load/Unload Task. The agent must proceed to the Load state and return to the Start/Stop state to unload. A long enough corridor can defeat any finite memory policy.

## 4.4 Finite State Policies

A finite state policy, or finite state controller (FSC), represents the agent's memory with a state machine. They can be viewed as augmenting memoryless policies with an additional observation representing the internal memory state, or alternatively as a natural extension of finite history methods, replacing history sequences with a unique identifying state. By removing the dependence on history sequences finite state policies can remember events arbitrarily far in the past.

A finite state policy, like a finite state machine, is a set of states with connections, or transitions, between them. These states are termed **internal states** to avoid confusion with the states of the environment. At any time a finite state policy is in a particular internal state. This state determines a probability distribution over actions and transitions, and so determines both the action the policy performs at that time and the next internal state. Formally, a finite state policy consists of:

- $\mathcal{M}$ the set of internal states in the controller.
- $\mu_a : \mathcal{M} \times \mathcal{A} \to [0, 1]$ is the action function. Given an internal state and an action it gives the probability of the policy executing that action in that state.
- $\mu_c : \mathcal{M} \times \mathcal{O} \times \mathcal{M} \to [0, 1]$ is the transition function. Given two internal states and an observation it gives the probability of transitioning from the first to the second state given the observation.

- $m_0$ is the initial state in which the policy starts.

Gradient ascent methods, also known as policy-gradient methods, are popular for learning finite state controllers. The gradient of reward with respect to the transition and action probabilities can be estimated from interactions with the environment. The policy parameters can then be adjusted to move up the gradient. Policy-gradient methods have several attractions:

- As a local optimisation procedure the computation required is quite modest compared to POMDP solving methods.
- It is easy to incorporate prior knowledge by shaping the transition or action probabilities. The rewards can also be shaped using domain knowledge (Laud and Dejong 2002).
- Convergence to a local optimum can be guaranteed.

When optimising the action and transition functions using gradient ascent it is difficult to constrain the updates to lie within the probability simplex (i.e. to ensure that the values remain valid probabilities). It is more convenient to let the value range over the real numbers and project back into the probability simplex. The Boltzmann (softmax) function is a simple way to do this. Let $\theta_i, i = 1 \ldots n$ be a set of positive real numbers. Then the Boltzmann function maps $\theta_i$ to a probability $\mu_i$ as $\mu_i = exp(\theta_i/\tau)/\sum_j exp(\theta_j/\tau)$. $\tau$ is the so-called temperature parameter that varies the sensitivity of the Boltzmann function to differences between values of $\theta$.

GAPS (Meuleau, Peshkin, Kim et al. 1999) is a batch finite-horizon policy gradient algorithm for finite state controllers. As some of my experiments use GAPS (see Chapter 5) I detail the algorithm here.

### 4.4.1 The GAPS Algorithm

There are three main steps to the GAPS algorithm: defining the objective function to optimise, calculating the partial derivatives of the objective, and specialising the derivatives to the policy representation.

GAPS assumes a goal attainment task, with a goal state that, once obtained, all actions return to[1]. Furthermore the goal state is associated with a unique observation $o_G$, so the agent always knows when it has reached the goal.

Let $h_T = ((o_0, m_0, a_0, r_0), \ldots (o_T, m_T, a_T, r_T))$ be a history of interactions with the environment, consisting of tuples of observation, internal state, action, and reward. The value of such a history sequence, using discounted sum of rewards is:

$$V(h_T) = \sum_{t=0}^{T} \gamma^t r_t \tag{4.1}$$

Let $H_T$ be the set of all history sequences that are $T$ steps long. The objective function with respect to a policy $\pi$ is the value of all history sequences:

$$F_\pi = \sum_{T=0}^{\infty} \sum_{h_T \in H_T} V(h_T) P(h_T|\pi) \tag{4.2}$$

The policy is parametrised by the values of $\mu_a$ and $\mu_c$. We'll label these, generically, as $\mu$ with components $\mu_k$. Taking the partial derivative of the objective function with respect to $\mu_k$ gives:

$$\frac{\partial}{\partial \mu_k} F_{pi} = \sum_{T=0}^{\infty} \sum_{h_T \in H_T} \frac{\partial}{\partial \mu_k} \left( V(h_T) P(h_T|\pi) \right) \tag{4.3}$$

$$= \sum_{T=0}^{\infty} \sum_{h_T \in H_T} \frac{\partial}{\partial \mu_k} V(h_T) P(h_T|\pi) + \frac{\partial}{\partial \mu_k} V(h_T) P(h_T|\pi) \tag{4.4}$$

$$= \sum_{T=0}^{\infty} \sum_{h_T \in H_T} \frac{\partial}{\partial \mu_k} V(h_T) P(h_T|\pi) \tag{4.5}$$

Now we can write $P(h_T|\pi)$ as

$$P(h_T|\pi) = \prod_{t=0}^{T} P(o_j|h_T^{0:j-1}) \mu_c(m_{t-1}, o_t, m_t) \mu_a(m_t, a_t) P(r_t|h_T^{0:j-1}, o_j, a_j) \tag{4.6}$$

where $h_T^{0:j-1}$ means the portion of $h_T$ from time 0 to $j-1$ (equivalently, the belief state at

[1] That is, using the terminology of Markov processes, an absorbing state.

time $j-1$). By definition $h_T^{0:-1} = \emptyset$ and $\mu_c(m_{-1}, o_0, m_0) = m_0$. We can now take the partial derivative of $P(h_T|\pi)$ by using a little trick related to the product rule. Let $f$, $u$, and $v$ be functions of $x$ and $f = uv$. Then:

$$\ln f = \ln u + \ln v \tag{4.7}$$

$$\frac{1}{f}\frac{df}{dx} = \frac{1}{u}\frac{du}{dx} + \frac{1}{v}\frac{dv}{dx} \tag{4.8}$$

$$\frac{df}{dx} = f\left(\frac{1}{u}\frac{du}{dx} + \frac{1}{v}\frac{dv}{dx}\right) \tag{4.9}$$

With this we can turn the product in Equation 4.6 into a sum and differentiate:

$$\begin{aligned}\frac{\partial}{\partial \mu_k} P(h_T|\pi) = P(h_T|\pi)\Bigg(&\sum_{t=0}^{t} \frac{\partial}{\partial \mu_k} \ln P(o_j|h_T^{0:j-1}) + \frac{\partial}{\partial \mu_k} \ln \mu_c(m_{t-1}, o_t, m_t) \\ &+ \frac{\partial}{\partial \mu_k} \ln \mu_a(m_t, a_t) + \frac{\partial}{\partial \mu_k} \ln P(r_t|h_T^{0:j-1}, o_j, a_j)\Bigg)\end{aligned} \tag{4.10}$$

Removing terms that are not a function of $\mu_k$ gives

$$\frac{\partial}{\partial \mu_k} P(h_T|\pi) = P(h_T|\pi)\left(\sum_{t=0}^{t} \frac{\partial}{\partial \mu_k} \ln \mu_c(m_{t-1}, o_t, m_t) + \frac{\partial}{\partial \mu_k} \ln \mu_a(m_t, a_t)\right) \tag{4.11}$$

Substituting into Equation 4.5 yields

$$\frac{\partial}{\partial \mu_k} F_{pi} = \sum_{T=0}^{\infty} \sum_{h_T \in H_T} V(h_T) P(h_T|\pi)\left(\sum_{t=0}^{t} \frac{\partial}{\partial \mu_k} \ln \mu_c(m_{t-1}, o_t, m_t) + \frac{\partial}{\partial \mu_k} \ln \mu_a(m_t, a_t)\right) \tag{4.12}$$

Recall that $\mu_c$ and $\mu_a$ are computed using the Boltzmann function. The partial derivative of the natural logarithm of the Boltzmann is:

$$\frac{\partial}{\partial \mu_k} \ln \frac{e^{\frac{\mu_k}{\tau}}}{\sum_i e^{\frac{\mu_i}{\tau}}} = \frac{1}{\tau} \left( 1 - \frac{e^{\frac{\mu_k}{\tau}}}{\sum_i e^{\frac{\mu_i}{\tau}}} \right) \tag{4.13}$$

Therefore stochastic gradient ascent on the objective function can be done using the agent's interactions with the environment. From a single interaction with the environment we can calculate an estimate of the partial derivatives of $\mu_c$ and $\mu_a$. Averaging estimates over multiple interactions naturally factors in $P(h_T|\pi)$. During each trial the weights are kept constant and gradients at each time step $t$ accumulated. At the end of an interaction the immediate gradients are summed to given the gradient for the whole trial. The current values of the weights are then moved a fraction $\alpha$ in the direction of the gradient. The immediate gradients are:

$$\Delta \mu_c^{m,o,m'}(t) = \gamma^t r_t \frac{N^{m,o,m'}(t)}{\tau} \left( 1 - \frac{e^{\frac{\mu_c^{m,o,m'}}{\tau}}}{\sum_i e^{\frac{\mu_i}{\tau}}} \right) \tag{4.14}$$

$$\Delta \mu_a^{m,a}(t) = \gamma^t r_t \frac{N^{m,a}(t)}{\tau} \left( 1 - \frac{e^{\frac{\mu_a^{m,a}}{\tau}}}{\sum_i e^{\frac{\mu_i}{\tau}}} \right) \tag{4.15}$$

where $N^{m,o,m'}(t)$ gives the number of times that state $m$ has transitioned to state $m'$ following observation $o$ up to time $t$, and $N^{m,a}(t)$ the number of times state $m$ has executed action $a$ up to time $t$.

The update equations at the end of an interaction are:

$$\mu_c^{m,o,m'} = \mu_c^{m,o,m'} + \alpha \Delta \mu_c^{m,o,m'}(T) \tag{4.16}$$

$$\mu_a^{m,a'} = \mu_c^{m,a} + \alpha \Delta \mu_a^{m,a}(T) \tag{4.17}$$

Gradient ascent is extended to infinite horizon tasks in (Aberdeen and Baxter 2002). The Exp-GPOMDP algorithm makes two innovations: using discounted traces to solve the credit assignment problem and so extend gradient ascent to infinite horizon problems, and using a

belief state over the (completely observable) internal state of the controller to reduce variance in the gradient estimates.

The gradient computed in Equation 4.14 does not always point in the direction of steepest ascent. A policy defines a distribution over paths through the model. By changing the parameters of the model a manifold over distributions arises. The correct gradient to use is the gradient on this manifold, the so-called natural gradient computed using the Fisher information matrix at the point of interest (Amari 1998). This idea has been applied to policy search in (Kakade 2002, Bagnell and Schneider 2003, Peters, Vijayakumar and Schaal 2003) and to POMDPs specifically in (Aberdeen, Buffet and Thomas 2007, Boularias and Chaib-draa 2009).

## 4.5 Recurrent Neural Networks

A recurrent neural network (RNN) is a neural network with feedback loops, allowing the network to maintain an internal state. Recurrent neural networks have been applied to reinforcement learning tasks with a variety of network structures and training algorithms. Here I discuss some of the major strands of work.

### 4.5.1 Gradient Descent Methods

Back-propagation through time (BPTT) (Rumelhart, Hinton and Williams 1986) is one method for training recurrent neural networks. It unfolds the network over $T$ steps and uses the standard back-propagation algorithm from artificial neural networks to adjust the network weights. This means the network may only be explicitly trained over a $T$-step history window. Real-time recurrent learning (Williams and Zipser 1989) addresses this issue by training the network back to time 0.

Long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) introduces complex neurons that learn to turn on or off their memory inputs and outputs. Learning is performed by gradient descent. LSTM has been applied to a grid world navigation task and to pole balancing (Bakker 2002). The LSTM network learns Q-values, and a separate non-recurrent neural net-

work is trained to estimate the variance of the Q-values. This allows greater control over the exploration/exploitation trade-off. The natural gradient (see Section 4.4) can also be used for training the parameters of LSTM networks (Wierstra, Forster, Peters et al. 2009).

### 4.5.2 Evolutionary Methods

(Glickman and Sycara 2001) is an early work applying evolutionary methods to learning RNNs for reinforcement learning tasks. Their networks have a fixed topology, and their evolutionary algorithm used only mutation.

Cooperative Synapose Neuroevolution (CoSyNe) (Gomez and Schmidhuber 2008, Koutník, Gomez and Schmidhuber 2010) is a method for evolving networks with fixed topology, and hence fixed memory. It maintains a number of sub-populations from which components are draw to create a complete RNN. In the original paper individuals represent network parameters directly. (Koutník et al. 2010) represents the network by a discrete cosine transform. By filtering high frequency components the complexity of the network can be controlled. CoSyNe has been successfully applied to several pole-balancing tasks.

NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen 2002) evolves both network weights and topology. Mutations in NEAT can add nodes or connections. NEAT tracks the lineage of each node by assigning every new node a globally unique identifier. When comparing two individuals, corresponding nodes can found by matching these identifiers. This allows crossover to occur in a meaningful way without performing expensive topological analysis on the networks. These identifiers also allow rapid computation of a similarity measure between individuals. This similarity function is used to divide the population into sub-populations. Individuals only have to compete within their sub-populations, allowing time for innovations to be evaluated. NEAT has been applied to pole-balancing tasks.

Recurrent neural networks are best suited to domains with continuous observation and action spaces. This characterises many low-level control tasks, but higher-level tasks often have discontinuities in their action space – that is, good policies must choose very different actions in nearby parts of the internal state space. NEAT has been modified for such domains in (Kohl and

Miikkulainen 2008). The key change is to use radial basis function nodes. Each node's output is determined by a Gaussian and thus it is active only in a localised area of the state space.

## 4.6 Other Representations

The are many other methods that have been applied to reinforcement learning problems. Here I briefly survey some of the more prominent ones.

Evolutionary methods have been used with representations other than recurrent neural networks. A review, now dated, is given in (Moriarty, Schultz and Grefenstette 1999). Learning classifer systems (LCSs) are Turing-complete representations combining genetic algorithms and reinforcement learning to learn a population of rules (called classifiers) that meet some goal. LCSs are inherently sequential decision making systems, as the rules are learned over a sequence of interactions with the environment. Recent work within the LCS community is directly tackling reinforcement learning problems, where LCSs can be seen as simultaneously learning both a model and a policy (Sigaud, Butz, Kozlova et al. 2009, Hamzeh and Rahmani 2008). See (Sigaud and Wilson 2007) for a recent survey.

Several reinforcement learning algorithms based on Levin search claim to be asymptopically optimal (Schmidhuber 2004, Hutter 2005). They have not been widely applied so it is difficult to appraise their performance in practice.

Finally, predictive state representations (PSRs) (Singh, James and Rudary 2004) are a recently introduced model that are more expressive and potentially more compact than POMDPs. Recent research has addressed model inference, planning, and learning in PSRs (e.g. (Aberdeen et al. 2007), (Boots, Siddiqi and Gordon 2009)).

## 4.7 Conclusions

We have seen a variety of methods for tackling reinforcement learning in POMDPs. Most of these methods attempt to learn a policy directly, and so are known as **policy search**. The exception are the finite-history methods, which are an intermediate step between a model learning

and policy search method. Policy search is attractive when the policy representation is more compact than the model, as policy search typically scales in the policy size. This is, of course, a problem dependent characteristic.

There is a trade-off between policy expressivity and tractability. For example, finite state controllers are less powerful than recurrent neural networks but are easier to analyse, both in terms of learning algorithms and in user interpretability of the learned policy. For this reason I have used finite state controllers in my work. In the next chapter I show how finite state controllers may be combined in a population to search over policy size and learn better policies than can be learned without.

# CHAPTER 5

# MODEL-FREE POPULATION-BASED REINFORCEMENT LEARNING

## 5.1 Introduction

In the previous chapters we have reviewed the sequential decision making problem, model-based policy learning, model learning, and model-free policy learning. In this chapter I present my first piece of original work. Throughout this dissertation I assume the agent does not have access to a model of the environment. Thus it must either learn a world model or learn to act without a model. In this chapter I take the latter route, developing an algorithm that learns a population of policies directly from interactions with the environment.

Policy search, reviewed in Chapter 4, is the general term for methods that learn directly from interactions. Finite state contollers (FSCs; see Section 4.4) are a popular representation for policies used by many policy search algorithms, and the representation I use herein. All previous work using finite state controllers, and the vast majority of work on policy search in general, learns controllers of a fixed size. The number of internal states in the FSC (i.e. it's size) determines how much memory it has, and as we've seen, choosing the right amount of memory is crucial for solving a POMDP. Fixing the size of the memory before interacting with the environment is thus problematic when the environment is not known.

My primary contribution is showing that a population of policies of varying size can be used to find near optimal policies for a range of tasks. Thus it is not necessary to fix the size

of a finite state controller in advance. By maintaining a population of policies we can consider policies with a range of sizes and so find the best size of policy for the particular environment under consideration.

Another issue with the gradient ascent algorithms typically used in policy search is the high variance in their gradient estimates. This is due to the inescapable fact that gradients are estimated from (usually) a few interactions with the environment, and are very noisy as a result. I also show that my population-based algorithm can improve over the performance of pure gradient ascent policy search using the same amount of computation.

### 5.1.1 Overview of the Chapter

As described above this chapter explores how a population of policies may be exploited to improve model-free policy learning. The core of this work is the definition and evaluation of an algorithm that shares information between a population of policies. To focus on the effect of the population I have deliberately chosen a naive way of sharing information, with the expectation that more sophisticated algorithms will only improve the results. I show that this algorithm is:

- Capable of finding near-optimal policies on challenging problems.
- More efficient than using equivalent resources to search without sharing information.
- Compatible with, and can improve upon the performance of existing local search techniques such as gradient ascent.

Furthermore, I do not require the size of the policy be known in advance.

I begin with an informal description of the algorithm (Section 5.2). In Section 5.3 I formalise the description. Benchmark problems are presented in Section 5.4 and experimental results in Section 5.5. Conclusions are in Section 5.6.

## 5.2 An Informal Description of The Policy Learning Algorithm

I begin with an informal description of my policy-learning algorithm. As mentioned in the Introduction the algorithm uses a population of policies. The two problems we face with policy learning, high variance and fixed size policies, will be overcome by using information from one policy to inform others and maintaining different sized policies within the population, respectively. I want to isolate the effect of the population as far as possible so I deliberately aim for a simple algorithm.

The setup is this: there is a population of policies, each of which has experienced one or more episodes in the world, and thus has an estimate of its expected reward (I assume an episodic task). To share information between policies we want to focus the search on the policies we believe are performing better. At the same time we want to maintain exploration of the search space to meet our second goal of finding the best policy size. The algorithm meets these goals by using several operators that transform one population into another. To share information about a good policy we simply clone it, replacing some other member of the population with the copy. This has the effect of focusing more resources on the point in the search space given by that policy. To ensure we continue to explore we use several different ways to perturb a policy: gradient ascent learning, a random change to the transition or action probabilities, or a change in the number of states in the policy. Finally to balance exploration and exploitation we run these operations within an algorithm known as simulated annealing. Simulated annealing starts off allowing any operation to succeed, but over time operations failing to improve estimated return are rejected with an increasing frequency. In this way simulated annealing begins with open exploration and slowly transitions towards pure optimisation. Figure 5.1 gives a schematic overview of one iteration of our policy-learning algorithm.

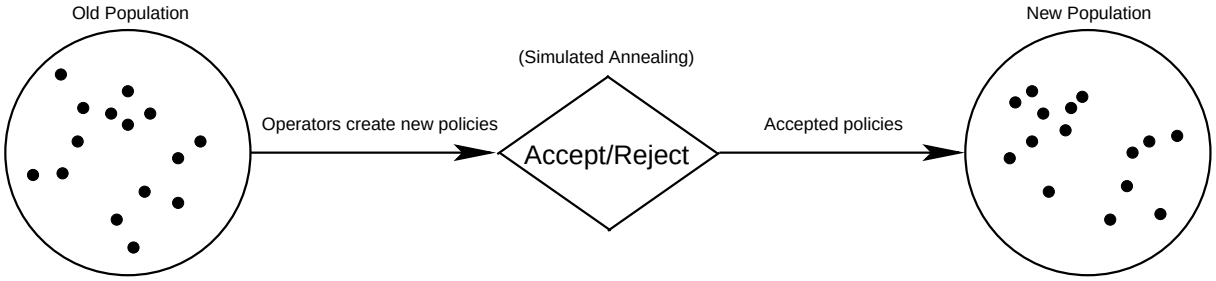


Figure 5.1: A high-level view of one iteration of my policy learning algorithm.

## 5.3 Policy Search Algorithm

As discussed in Section 5.2 the basic components of the algorithm are the population, the search operators that create new policies, and the simulated annealing algorithm that ties everything together. I address each in turn.

### 5.3.1 The Population

The population $P$ is a set of finite state controller policies $\pi$. Finite state controllers are covered in Section 4.4. To summarise, a finite state controller consists of:

- $\mathcal{M}$ the set of internal states in the controller.
- $\mu_a : \mathcal{M} \times \mathcal{A} \rightarrow [0, 1]$ is the action function. Given an internal state and an action it gives the probability of the policy executing that action in that state.
- $\mu_c : \mathcal{M} \times \mathcal{O} \times \mathcal{M} \rightarrow [0, 1]$ is the transition function. Given two internal states and an observation it gives the probability of transitioning from the first to the second state given the observation.
- $m_0$ is the initial state in which the policy starts.

As discussed in Section 4.4, when optimising the action and transition functions using gradient descent it is difficult to constrain the updates to lie within the probability simplex. For this reason the parameters are allowed to be reals and projected back into the simplex using the Boltzmann (softmax) function.

I write $\pi_i$ for the $i^{th}$ policy. I assume an episodic task, meaning that the task terminates after a goal state is reached (or after some fixed number of steps to prevent policies running forever.) Each policy experiences at least one episode and so can approximate its expected return:

$$\mathbb{E}[V^{\pi}] \approx \left( \sum_{i \in episodes} \sum_{\tau=0}^{t_i-1} \gamma^{\tau} r^i_{\tau+1} \right) \tag{5.1}$$

where $t_i$ is the number of time steps in the $i^{th}$ episode, $r^i_t$ is the reward received at time step $t$ of episode $i$, and $\gamma$ is the discount factor.

#### Initialisation

For the experiments discussed here, the population was initialised by generating random policies with between one and five states. Each element of the action and transition weights (that is, the values before they are converted to probabilities) was set to one.

### 5.3.2 Search Operators

A search operator is a function that produces a policy. I use two types of search operators: local operators having only a single policy as input, and global operators having the entire population as input. All operators generate a single policy. In my experiments I use several local operators but only a single global operator.

#### Local Search Operators

The first local search operator is called **perturbation**. This operator is used to search the area around a particular policy. A perturbation changes a single randomly chosen value in either the transition or action matrix. Recall the Boltzmann function is used to map real numbers to probabilities. As the softmax involves an exponential it is very sensitive to the relative difference between the numbers. Therefore the perturbation cannot be too great or the resulting probabilities will be extremely skewed. I use the following rule: for a value $v$ the new value is given by $0.5v + uniformRandom(0.0, v)$. The expected value is $v$ and the maximum range is

| NAME | TYPE | DESCRIPTION |
|---|---|---|
| Perturbation | Local | Random change to obs. or trans. function |
| Jumping | Local | Increase or decrease size |
| GAPS | Local | Gradient ascent |
| Cloning | Global | Change to other policy in current population |

Table 5.1: The search operators used in the population reinforcement learning algorithm.

$[0.5v, 1.5v]$, which tends to keep the values reasonable.

The second search operator, called **jumping**, moves to a randomly generated policy with one more or one less state than the chosen policy. The new policy is initialised with transition and action weights uniformly drawn from the range $[0.1, 2]$. To prevent policies becoming very large I set a maximum number of states that no policy can exceed.

The final local operator is the gradient ascent learning algorithm **GAPS** (Peshkin, Meuleau and Kaelbling 1999). GAPS is described in more detail in Section 4.4.1.

#### Global Search Operators

The single global search operator is called **cloning**. It simply replaces one member of the population with another. The policy is sampled from the current population with probability proportional to estimated return. This is a very simple method – remember that my goal is to isolate the effect of the population rather than focus on cleverness in the algorithm.

The search operators are summarised in Table 5.1.

### 5.3.3 Population Simulated Annealing

Having defined the search operators we can now combine them in a search algorithm. The algorithm used is simulated annealing (Kirkpatrick, C. D. Gelatt and Vecchi 1983), which is modified to incorporate a population.

Simulated annealing is a stochastic hill climbing algorithm. The exploration/exploitation trade-off is controlled by a temperature parameter. At high temperatures up hill moves are only weakly favoured over down hill moves, while at low temperatures the algorithm approaches pure hill climbing. The temperature parameter is initially set high, to encourage exploration, and

then gradually reduced towards zero using an algorithm known as the cooling schedule. Thus simulated annealing initially allows exploration but gradually comes to prefer exploitation.

Each temperature setting is called an iteration. During an iteration a new population is constructed from the current population. This is a two stage process. First, for every policy in the current population a proposed replacement is generated by applying a search operator. Then either the proposed policy or the original policy is placed in the new population. The choice made is a function of the current temperature and the estimated return of the two policies. If no proposals are accepted during the iteration the search halts, otherwise the temperature is decreased and the process repeats.

The rule for generating a proposed policy is simple: an operator is chosen with a fixed probability dependent on the experimental setup, and then applied to the current policy or the entire population as appropriate. A proposal is always accepted if its estimated return is greater than or equal to the estimated return of the policy from which it was generated. If this is not the case it is accepted with a probability inversely proportional to the difference in estimated return and proportional to the current temperature.

Given enough proposals and a slow enough cooling schedule, simulated annealing is guaranteed to converge to the global optimum. This theoretical guarantee only holds for cooling schedules that are too slow to use in practice. Geometric cooling, where the next temperature is some fixed proportion of the current temperature, is commonly used though no guarantee of convergence holds in this case.

The complete algorithm is given in Algorithm 5.1.

## 5.4 An Empirical Study

To evaluate the algorithm I ran it on four problems. In order of complexity they are Load/Unload, the M-Maze, the Small Maze, and the Tunnels Maze. All the problems are goal attainment tasks. That is, the agent begins in one of a number of starting positions and has to find its way to a fixed goal state. Pertinent characteristics of the problems are summarised in Table 5.2.

**Algorithm 5.1:** Population simulated annealing

**Data**: $temp$: the starting temperature; $max$: the maximum number of states, $proposals$: the number of proposed policies per temperature step, $size$: the number of individuals in the population.

**Result**: A population of optimised solutions

$p \leftarrow createInitialPopulation(size)$
**while** $temp > finalTemp$ **do**
  $p' \leftarrow \emptyset$
  **for** $s \leftarrow p$ **do**
    **for** $i = 1$**to**$proposals$ **do**
      $proposed \leftarrow proposeNewSolution(s, p)$
      $d \leftarrow quality(proposed) - quality(s)$
      $a \leftarrow min(1, \frac{exp(d)}{temp})$
      $u \leftarrow uniformRandom(0, 1.0)$
      **if** $u < a$ **then**
        $s \leftarrow proposed$
      **end**
    **end**
    $p' \leftarrow s \cup p'$
  **end**
  **if** $p = p'$ **then**
    **return** $(p)$
  **end**
  **else**
    $p \leftarrow p'$
    $temp \leftarrow decreaseTemperature(temp)$
  **end**
**end**
**return** $p$

The Load/Unload problem is shown in Figure 5.2. The agent starts at the left end of a corridor five units long and must move to the right end (load) and then return to the start (unload). The agent may only move left or right. Actions always succeed unless the action would take the agent into a wall in which case the agent stays where it is. The agent receives a reward of 1 when it loads (if it was not previously loaded), a reward of 1 when it unloads (if it was previously loaded), and a reward of -1 at all other times. Hence the maximum total (undiscounted) reward is -4.[1]

[1]This formulation of the Load/Unload problem follows (Aberdeen 2003). This differs slightly from the formulation given in (Peshkin et al. 1999) as the agent receives a reward every time it loads and unloads rather than only every time it unloads

| PROBLEM | STATES | OBSERVATIONS | ACTIONS |
|---|---|---|---|
| Load/Unload | 10 | 3 | 2 |
| M-Maze | 11 | 6 | 4 |
| Small Maze | 44 | 38 | 4 |
| Tunnels | 165 | 44 | 4 |

Table 5.2: Summary of key characteristics of the problems on which the algorithm was evaluated.

The optimal policy for Load/Unload requires two states (alternatively, one bit of memory) to record if the agent is loaded.

The M-Maze (McCallum 1995b) is shown in Figure 5.3. The agent starts in either of the positions labelled `Start` and must navigate to the `Goal` position. The corner positions are the only means the agent has for distinguishing its starting place. The optimal policy for the M-Maze requires 4 states. The agent receives a reward of 1 when it reaches the goal and a reward of -1 at all other times, so the maximum total reward is -4.0

The Small Maze is a 10x10 grid world shown in Figure 5.4. I use a deterministic variant and a stochastic variant of this maze. In the deterministic variant the agent starts in the lower right corner and transitions are deterministic. In the stochastic variant the agent can start in the lower left or lower right corner, and actions have only a 0.875 probability of succeeding and a 0.125 probability of keeping the agent in the same square. In both variants the goal is the upper left corner.

The Tunnels Maze, shown in Figure 5.5, has the same rules as the stochastic Small Maze with more states and observations.

The optimal policy for the deterministic Small Maze has a return of -12.0. To compute an upper bound on the return for the stochastic Small Maze we implemented an oracle optimal policy – that is, a policy with full knowledge of the true underlying state – and calculated an average return of -10.0 over 1000 runs. To establish a lower bound we ran 1000 randomly initialised polices with between one and five internal states, which had an average return of -681.3. For the Tunnels Maze the estimated upper bound is -33.0 and the lower bound is -917.8.

All problems have underlying deterministic transitions, except the stochastic Small Maze and the Tunnels Maze, and a significant degree of aliasing amongst observations. They all use

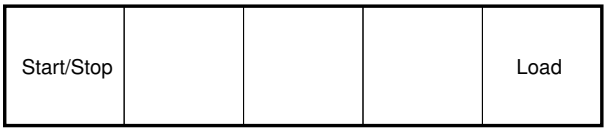


Figure 5.2: The Load/Unload Task

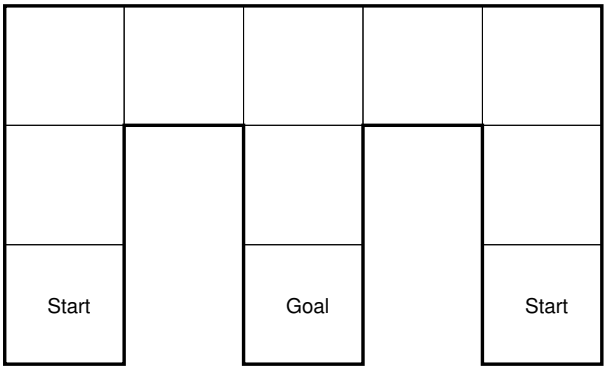


Figure 5.3: The M-Maze

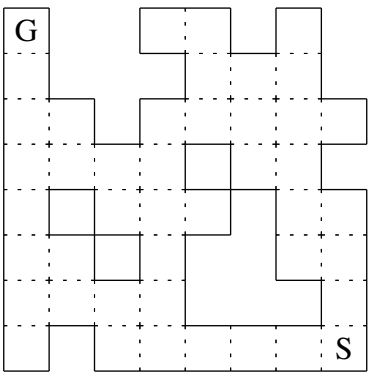


Figure 5.4: The Small Maze

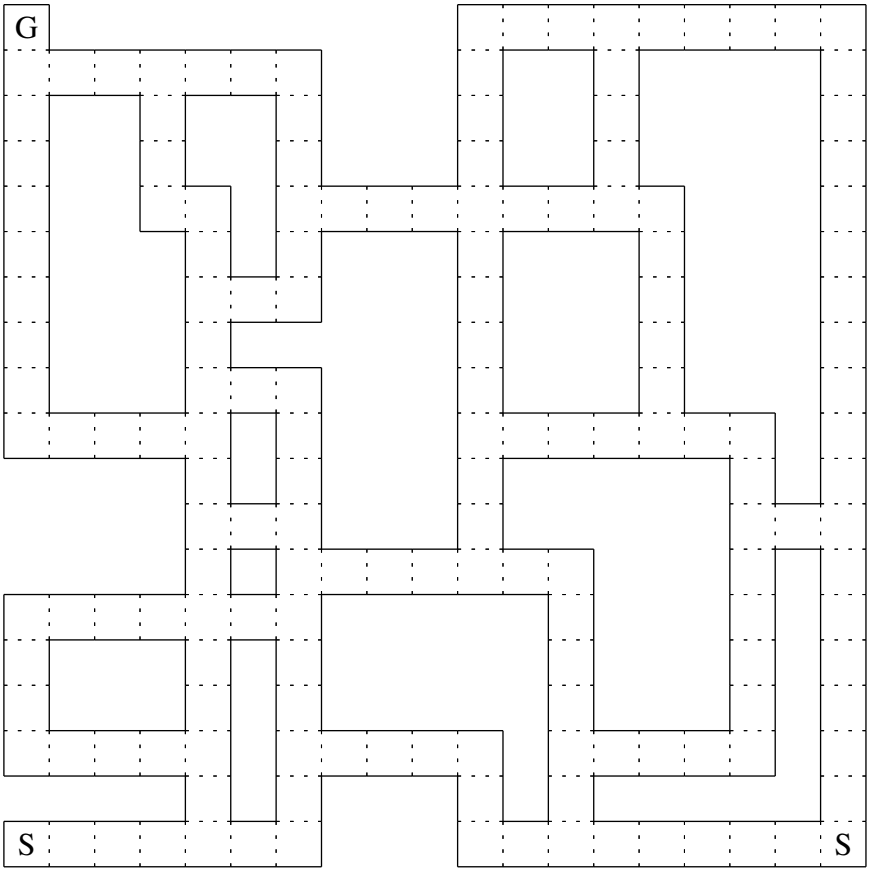


Figure 5.5: The Tunnels Maze

an enumerated representation; observations are encoded as numbers with no internal structure apparent to the agent.

For the Load/Unload and M-Maze problems the observations are constructed by encoding the presence or absence of walls to the north, south, east, and west in a binary string. Since all possible observations are not present in either maze the distinct binary strings are mapped to consecutive integers. So, for example, there are only three possible observations in the Load/Unload problem (the two end points and the middle corridor sections), and they are mapped to the integers 1, 2, and 3 respectively. There are six distinct observations in the M-Maze using this encoding, and thus they are mapped to the integers from 1 to 6.

A similar encoding is used for the other mazes, but here the binary digits indicate whether each of the eight squares surrounding the current position is occupied by a barrier (the mazes are constructed out of $1 \times 1$ square barriers). The same mapping to consecutive integers is used to generate the observations the agent observes.

## 5.5 Experimental Results

To validate the claims made for the algorithm I ran three groups of experiments. For all the experiments the initial temperature is 100.0, there are 1000 proposals per iteration, and return is estimated by averaging over 10 episodes.

For all runs the initial policy has one state, and parameters uniformly set to one. The number of states may vary between 1 and 5. Annealing is stopped when the temperature reaches 0.001 or if no new policy is accepted during an iteration. The cooling schedule is to set the next temperature to 0.9 of the current temperature (geometric cooling). Any episode that executes for more than 1000 steps without completing the task is halted.

Unless otherwise specified the probability of cloning is 0.10, the probability of jumping a dimension is 0.05, and the policy is perturbed otherwise.

To show the scalability of the algorithm I ran population simulated annealing on all the problems given in Section 5.4. The previous largest problem tackled using finite state controller

and policy search that I am aware of is the 52 state pentagon problem addressed by (Aberdeen 2003). This is roughly the size of the Small Maze, and about a third the size of the Tunnels Maze. For Load/Unload, the M-Maze, and the Small Maze I also varied the population size to see how it affected performance. For all these runs I used perturbation, jumping, and cloning. The results are shown in Table 5.3.

| POP. | RUNS | BEST | WORST | $\mu$ | $\sigma$ | Optimal |
|---|---|---|---|---|---|---|
| LOAD / UNLOAD | | | | | | |
| 10 | 30 | **-4.3** | -12.9 | **-5.36** | 1.57 | *-4.0* |
| M-MAZE | | | | | | |
| 10 | 39 | **-4.0** | -14.4 | -7.62 | 2.20 | *-4.0* |
| 20 | 39 | -4.0 | -12.2 | -7.76 | 2.04 | *-4.0* |
| 40 | 40 | -4.0 | -10.8 | -6.67 | 1.79 | *-4.0* |
| 80 | 7 | -4.0 | -8.4 | **-5.91** | 1.69 | *-4.0* |
| SMALL MAZE | | | | | | |
| 10 | 30 | -13.0 | -27.2 | -17.52 | 3.39 | *-12.0* |
| 20 | 27 | **-12.0** | -20.3 | -15.80 | 3.22 | *-12.0* |
| 40 | 3 | -12.5 | -12.6 | **-12.57** | 0.06 | *-12.0* |
| STOCHASTIC SMALL MAZE | | | | | | |
| 10 | 28 | **-12.3** | -26.3 | **-17.17** | 3.34 | *-10.0* |
| TUNNELS | | | | | | |
| 10 | 30 | **-38.4** | -101.3 | **-57.41** | 14.58 | *-33.0* |

Table 5.3: Results for scalability experiments, which demonstrate the ability of the algorithm to tackle large problems. These experiments use perturbation, jumping, and cloning as search operators. Reported above are the population size, the number of runs, the best and worst policy in the final population, the mean and standard deviation of the estimated return in the final population, and the return of the optimal policy.

From these results we see that 1) the algorithm is capable of learning good policies for large problems, and 2) increased population size is associated with improvement in the mean policy and decreased variance across the population. It is unsurprising that increasing the population size improves the algorithm. The probability of finding a near optimal solution is directly proportional to the amount of effort spent searching, which in turn is directly proportional to the population size. However, it is worth noting that the results suggest that a given amount of computing power is better invested in increasing the population size than increasing the number of runs with a given population size. Note that I was unable to complete Stochastic Small Maze or Tunnels runs with larger populations due to the long runtime on these problems, in the order

of days per run.

To show that sharing information between members of the population is an important contribution to the algorithms success I set the probability of cloning to zero for Load/Unload and the M-Maze, and compare these results to those obtained with cloning in the scalability experiments above. The results are collected in Table 5.4. The difference between both pairs of results is significant at the 0.05 level and clearly show information sharing (cloning) outperforms runs without it.

| VARIANT | POP. | RUNS | MAX | MIN | $\mu$ | $\sigma$ |
|---|---|---|---|---|---|---|
| LOAD / UNLOAD | | | | | | |
| Information sharing | 10 | 30 | **-4.3** | -12.9 | **-5.36** | 1.57 |
| No information sharing | 10 | 35 | -4.9 | -14.0 | -10.09 | 3.33 |
| M-MAZE | | | | | | |
| Information sharing | 10 | 39 | **-4.0** | -14.4 | **-7.62** | 2.20 |
| No information sharing | 10 | 30 | -7.7 | -13.7 | -9.86 | 1.48 |

Table 5.4: Results showing the effectiveness of information sharing. Experiments compare perturbation and jumping with and without cloning.

To show that population simulated annealing may be used with existing local search algorithms from the literature, I implemented the GAPS gradient ascent algorithm (Meuleau et al. 1999) and then ran the following experiments:

- I ran simulated annealing with GAPS on the M-Maze, with the probability of cloning set to zero. The initial policy always has a single state, so I maintain a non-zero probability of jumping to allow the search to vary the number of states. The effect of jumping is to generate a random policy, and thus jumping will initiate GAPS from a variety of starting points.

- I re-ran the above experiments with the probability of cloning set to 0.1, jumping set to 0.05, and probability of GAPS set to 0.85. This experiment will demonstrate if cloning is beneficial to GAPS.

- I re-ran the above with a hybrid local search: the probability of GAPS was set to 0.425 and the probability of perturbation was set to 0.425. This experiment determines if GAPS (with cloning) benefits from perturbation.

These results are shown in Table 5.5. The differences between GAPS with and without information sharing, and between GAPS with information sharing and the hybrid method are both significant at the 0.05 level. These results demonstrate that information sharing improves GAPS, as expected, and interestingly perturbation also improves GAPS. It has been shown that the initial gradient estimates for fully connected finite state controllers are often uninformative, and so gradient ascent may fail in this case (Aberdeen 2003). I hypothesise that the hybrid method outperforms pure GAPS by using perturbation to move controllers to regions where the gradient is informative.

| VARIANT | POP. | RUNS | MAX | MIN | $\mu$ | $\sigma$ |
|---|---|---|---|---|---|---|
| M-MAZE | | | | | | |
| GAPS, no information sharing | 10 | 23 | -4.0 | -9.2 | -6.25 | 1.49 |
| GAPS | 10 | 26 | -4.0 | -6.9 | -5.14 | 0.81 |
| GAPS and perturbation | 10 | 25 | -4.0 | -6.8 | **-4.69** | 0.86 |

Table 5.5: Results comparing GAPS without cloning, to GAPS and GAPS/perturbation hybrid with cloning. This demonstrates the information sharing algorithm improves existing policy search algorithms.

I started this chapter by claiming that the population based simulated annealing algorithm was:

- Capable of finding near-optimal policies on challenging problems.
- More efficient than using equivalent resources to search without sharing information.
- Compatible with, and can improve the performance of, existing local search techniques such as gradient ascent.

The experiments have validated these claims. Results on the Tunnels Maze (Table 5.3) show the algorithm is effective for POMDPs three times the size of previously reported results. The results with and without information sharing (Table 5.4) clearly show that sharing information within the population is a more efficient way to search, achieving a higher mean and lower variance. The results with GAPS (Table 5.5) demonstrate that other policy search methods are compatible with and improved by information sharing.

The hybrid GAPS/perturbation runs suggest that gradient information can be usefully combined with the random walks of perturbation to achieve performance better than either. The hybrid (or Hamiltonian) Monte Carlo method (see, e.g., (Neal 1993)) is a general framework for such approaches.

## 5.6 Conclusions

The experimental results have shown that the simple population-based reinforcement learning algorithm is an easy way to improve the performance of existing policy search algorithms. Although the algorithm is successful on quite large problems, issues remain. The most important issue is the running time required to produce results. Runs, particularly with a large population or problem, took days to complete. There are three factors to consider here: the generic nature of the algorithm, the problem representation, and the very nature of model-free search.

A generic algorithm, such as mine, will always be beaten by an algorithm specialised to the policy representation and problem domain. Scaling reinforcement learning to real-world tasks will require specialised algorithms and representations.

There is no structure in the enumerated representation of observations, but there is structure in the underlying model that the algorithm could utilise if it was available. For example if the observations were factored into separate readings for each four directions in the grid, policies might learn to generalise observations that share some properties in common (e.g. a wall in the same place). Observation representation affects scaling in other ways as well. Policy search is attractive as it scales in the size of the policies rather than the size of the POMDP. In my algorithm I allow the size to vary so in theory the FSCs might grow very large. However, since each additional state leads to a quadratic increase in the number of parameters in the FSC exploring the space of larger models is correspondingly more difficult, acting as a pressure to reduce the size of the FSCs. This does mean performance could be poor if the problem has a large number of observations that are relevant to solving the problem. In these situations a factored representation will help to avoid explosion in the size of the search space.

The members of the population can be seen as samples from a distribution over the space of FSCs, and the algorithm as a mechanism for fitting that distribution to the areas with highest return. We represent the distribution via point samples, and thus lose information everytime a policy is removed from the population. We might try to place an explicit distribution over the search space from which we sample. Preliminary experiments with a simple mixture model that chooses first a number of states, and then transmission and emission parameters showed that the number of states alone provides no useful information. An alternative is to use a hierarchical Dirichlet process prior in the manner of the infinite HMM (Beal et al. 2002, Teh et al. 2006), allowing states to be shared amongst the population.

# CHAPTER 6

# VARIATIONAL LEARNING OF POMDPS

## 6.1 Introduction

In Chapter 5 we explored model-free reinforcement learning and saw that, while capable of solving large problems with an unknown memory requirement, the algorithm required an excessive number of trials to do so. The core problem is that the model-free algorithm is inefficient with data. Each policy represents knowledge of some interactions with the environment, but that knowledge is not represented in a way that can easily be exploited. Furthermore, when a policy is removed from the population that knowledge is lost. If instead we were to learn a model of the environment that model could store the knowledge of all interactions, and so we might reasonably hope to achieve faster reinforcement learning.

Deciding to learn a model raises its own issues, namely the choice of model learning algorithm and the method used to find a policy given the model. Bayesian methods (reviewed in Section 3.2.2) are appropriate for learning the model given the goal of explicitly modelling uncertainty. POMDP model learning algorithms were surveyed in Section 3.4, where we saw that sampling algorithms have been applied to learning POMDPs, but the variational algorithm for hidden Markov models has not. Variational algorithms are generally faster than sampling approaches, so it is worth determining if a variational approach can learn a good model of a POMDP. Variational algorithms require the size of the model is fixed, but there are reasons to believe this won't be a problem in practice. Experiments using the variational HMM algorithm

have shown that excess states are assigned posterior probability close to zero (McGrory and Titterington 2006). Thus I hope that by choosing a large model the variational algorithm with converge to a model with the "right" number of states.

I use standard algorithms to construct a policy from the learned models. While we might expect better performance from using an algorithm tailored to the Bayesian model representation I learn, I found the models learned were poor and do not justify additional effort in this direction.

### 6.1.1 Overview

Section 6.2 presents the Bayesian POMDP, and Section 6.3 shows how it may be learned using a variational algorithm. In Section 6.4 I describe experiments where the utility for reinforcement learning of models learned using the variational algorithm is compared to that learned using the Bayes-adaptive algorithm (Ross et al. 2007). The models learned using the variational algorithm outperform those learned using the Bayes-adaptive algorithm in all cases. Following the experimental setup in (Ross et al. 2007), the two algorithms are compared under a strong prior. I then investigate the effect of the prior on the quality of the learned models, and find that as the prior becomes less informative the utility of the models rapidly decreases. I conclude in Section 6.5 with discussion of the limitations of the variational approach and the difficulties of learning POMDPs in general.

## 6.2 Bayesian POMDPs

The POMDP was defined in Section 2.5. To recap, a partially observable Markov decision process is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \rho, \mathcal{O}, \mathcal{Z}, \mathcal{I} >$ where:

- $\mathcal{S} = \{s_1, \ldots, s_K\}$ is the set of states that the environment may be in.
- $\mathcal{A}$ is the set of actions that the agent can perform in the environment.

- $\mathcal{T}$ is the transition function, which determines the next state given the current state, and the chosen action. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines the probability $\mathcal{T}(s, a, s') = P(s'|a, s)$
- $\rho$ is the reward function that maps state and action pairs to reward: $\rho : \mathcal{S} \times \mathcal{A} \rightarrow \Re$
- $\mathcal{O} = \{o_1, \ldots, o_N\}$ is the set of observation symbols. These represent the range of observations that they agent may perceive about the environment.
- $\mathcal{Z}$ is the observation function that maps the current state of the environment to an observation. $\mathcal{Z} : \mathcal{S} \times \mathcal{O} \rightarrow [0, 1]$ defines the probability $\mathcal{Z}(s, o) = P(o|s)$
- $\mathcal{I}$ is the initial distribution over states $\mathcal{I} : \mathcal{S} \rightarrow [0, 1]$ defining the probability $\mathcal{I}(s) = P(s)$

The Bayesian POMDP allows uncertainty in the transitions, observation, and initial state distributions by placing a Dirichlet prior over them:

$$\mathcal{I} \sim Dirichlet(\phi_1^{\mathcal{I}}, \ldots, \phi_K^{\mathcal{I}}) \tag{6.1}$$

$$\mathcal{T}_{s_i,a_j,\cdot} \sim Dirichlet(\phi_1^{\mathcal{T}}, \ldots, \phi_K^{\mathcal{T}}) \tag{6.2}$$

$$\mathcal{Z}_{s_i,\cdot} \sim Dirichlet(\phi_1^{\mathcal{O}}, \ldots, \phi_N^{\mathcal{O}}) \tag{6.3}$$

where $\phi_1^{\mathcal{I}}$ represents the parameter for state $s_1$ for the initial state distribution $\mathcal{I}$ and so on.

The Dirichlet distribution is defined in Section 3.3. To recap, it is a distribution over the parameters of the multinomial distribution. The Dirichlet has as many parameters as outcomes in the multinomial distribution it models. Let $(p_1, \ldots, p_K)$ be the parameters of a $K$-dimensional multinomial. We write $(p_1, \ldots, p_K) \sim Dirichlet(\phi_1, \ldots, \phi_K)$ if $(p_1, \ldots, p_K)$ is distributed according to a Dirichlet distribution with the given parameters. The expected value of the parameter $p_i$ is $\frac{\phi_i}{\sum_{j=1}^{K} \phi_j}$.

Dirichlet distributions are conjugate to the multinomial, so if the prior distribution is a Dirichlet and the data is distributed according to a multinomial, the posterior is also Dirichlet (see Section 3.2.2 for more on conjugate distributions). Let $n_i$ be the number of data that

take on value $i \in 1, \ldots, K$. Then the posterior for a Dirichlet prior $Dirichlet(\phi_1, \ldots, \phi_K)$ is $Dirichlet(\phi_1 + n_1, \ldots, \phi_K + n_K)$. It is for this reason that the parameters of the Dirichlet are often referred to as experience counts.

The key problem with learning Bayesian POMDPs is deciding how to update the experience counts in the Dirichlet when, due to partial observability, we do not know what state we are in. The correct solution is to consider all possible updates, giving a posterior that is a mixture of Dirichlets. This quickly becomes impractical, as the number of components in the mixture grows exponentially with the sequence length. Some approximation is needed and this is the problem addressed here, and the problem that has been addressed by the Bayes-adaptive framework (Ross et al. 2007).

### 6.2.1 The Bayes-Adaptive POMDP

The Bayes-Adaptive POMDP (BAPOMDP) framework of (Ross et al. 2007) is one solution to the problem of updating a Bayesian POMDP in response to data. Two approximations are necessary to regain tractability:

Firstly, the experience counts of the Dirichlets are restricted to a finite subspace. The size of the subspace is shown to bound the error in the value function, allowing arbitrarily close approximation to the true value function with increasing size.

Let $\phi^{\mathcal{T}}$ and $\phi^{\mathcal{O}}$ be the parameters defining the Dirichlet distributions over transitions and observations respectively, with the sum of the elements being $N^{\mathcal{T}}$ and $N^{\mathcal{O}}$. (Note that there is a vector of transition parameters for each state and action, and of observation parameters for each state. For clarity of notation I have not indicated this.) To bound the error in the value function to $\epsilon$ we can limit $N^{\mathcal{T}}$ and $N^{\mathcal{O}}$ to:

$$N^{\mathcal{T}} \leq \max\left(\frac{|\mathcal{S}|(1+\epsilon')}{\epsilon'}, \frac{1}{\epsilon''}\right) \tag{6.4}$$

$$N^{\mathcal{O}} \leq \max\left(\frac{|\mathcal{Z}|(1+\epsilon')}{\epsilon'}, \frac{1}{\epsilon''},\right) \tag{6.5}$$

where

$$\epsilon' = \frac{\epsilon(1-\gamma)^2}{8\gamma||\rho||_\infty} \tag{6.6}$$

$$\epsilon'' = \frac{\epsilon(1-\gamma)^2 \ln(\gamma^{-e})}{32\gamma||\rho||_\infty} \tag{6.7}$$

These restricted spaces can still be very large, so a second approximation is introduced. In the experiments reported here only the 16 most probable updates are kept after each experience.

Note the BAPOMDP method is online, meaning the model is updated with each experience. The variational algorithm I propose is a batch method.

## 6.3 Variational Learning

My solution to the posterior update problem is a variational approximation to the true posterior. This section describes the variational Bayesian EM algorithm for learning POMDPs. It will turn out that this algorithm is a simple modification of the classic EM algorithm for maximum likelihood learning of HMM parameters, the Baum-Welch algorithm. The Baum-Welch algorithm itself makes use of the forward-backward algorithm to compute the E-step. I present first the forward-backward algorithm in Section 6.3.1, then the Baum-Welch algorithm (Section 6.3.2), and finally the variational Bayesian EM algorithm (Section 6.3.3). The variational Bayesian EM algorithm for POMDPs is a simple modification to the same algorithm for HMMs. The variational HMM algorithm was first developed in (MacKay 1997) and refined in (Beal 2003).

A model learning algorithm must have data as input. In my model the agent first receives an observation from its current state and then takes an action transitioning to a new state. The combination of the origin state and action determine the award.

, interactions are ordered observation, then action, then reward, except for the final interaction when the agent receives both an observation and reward but takes no action[1]. Let $o_t \in \mathcal{O}$

[1]This is to allow the agent to receive an observation and reward upon reaching a goal state. If the agent were to take an action, that action would not lead to any other state, as the goal state is absorbing. This is, in fact, how

be the observation at time $t$, and likewise $a_t \in \mathcal{A}$ and $r_t \in \Re$ be the action and reward. Then a sequence of actions terminating at time $T$, otherwise known as an **episode**, is a list of tuples:

$$\text{episode} \equiv [(o_1, a_1, r_1), (o_2, a_2, r_2), \ldots, (o_T, r_T)] \tag{6.8}$$

A tuple in an episode is an experience.

### 6.3.1 The Forward-Backward Algorithm

Given a sequence of observations the forward-backward algorithm calculates $P(s_t|o_{1:T}, a_{1:T-1})$ for all $s$ at all times $t$. The state occupancy probability can be factored using the Markov property and Bayes rule:

$$\begin{aligned}
P(s_t|o_{1:T}, a_{1:T-1}) &= P(s_t|o_{1:t}, a_{1:t-1}, o_{t+1:T}, a_{t:T-1}) && (6.9)\\
&= \frac{P(o_{t+1:T}|s_t, o_{1:t}, a_{1:t-1}, a_{t:T-1})P(s_t, o_{1:t}|a_{1:t-1}, a_{t:T-1})}{P(o_{t+1:T}|a_{1:t-1}, a_{t:T-1})} && (6.10)\\
&= \frac{P(o_{t+1:T}|s_t, a_{t:T-1})P(s_t, o_{1:t}|a_{1:t-1})}{P(o_{t+1:T}|a_{1:T-1})} && (6.11)\\
&\propto P(o_{t+1:T}|s_t, a_{t:T-1})P(s_t, o_{1:t}|a_{1:t-1}) && (6.12)
\end{aligned}$$

The two terms $P(s_t, o_{1:t}|a_{1:t-1})$ and $P(o_{t+1:T}|s_t, a_{t:T-1})$ can be calculated using the forward and backward recursions respectively. These terms are traditionally called the $\alpha$ and $\beta$ values.

The $\alpha$ values $\alpha(s_t) \equiv P(s_t, o_{1:t}|a_{1:t-1})$ are recursively defined by:

it implemented – all actions in the goal state lead to the goal state. When learning a model this final action is dropped.

$$\alpha(s_t) = P(s_t, o_{1:t}|a_{1:t-1}) \tag{6.13}$$

$$= P(o_t|s_t) \sum_{s_{t-1}} P(s_t|s_{t-1}, a_{t-1}) P(s_{t-1}, o_{1:t-1}|a_{1:t-2}) \tag{6.14}$$

$$= P(o_t|s_t) \sum_{s_{t-1}} P(s_t|s_{t-1}, a_{t-1}) P(s_{t-1}, o_{1:t-1}|a_{1:t-2}) \tag{6.15}$$

$$= P(o_t|s_t) \sum_{s_{t-1}} P(s_t|s_{t-1}, a_{t-1}) \alpha(s_{t-1}) \tag{6.16}$$

The base case for the recursion is:

$$\alpha(s_1) = P(s_1, o_1) = P(o_1|s_1) P(s_1) = \mathcal{Z}_{\cdot,o_1} \mathcal{I}(s) \tag{6.17}$$

Note that the base case is not dependent on any action.

The $\beta$ values $\beta(s_t) \equiv P(o_{t+1:T}|s_t, a_{t:T-1})$ are recursively defined by:

$$\beta(s_t) = P(o_{t+1:T}|s_t, a_{t:T-1}) \tag{6.18}$$

$$= \sum_{s_{t+1}} P(o_{t+1}|s_t) P(o_{t+2:T}, s_{t+1}|s_t, a_{t:T-1}) \tag{6.19}$$

$$= \sum_{s_{t+1}} P(o_{t+1}|s_t) P(o_{t+2:T}|s_{t+1}, a_{t+1:T-1}) P(s_{t+1}|s_t, a_t) \tag{6.20}$$

$$= \sum_{s_{t+1}} P(o_{t+1}|s_t) P(s_{t+1}|s_t, a_t) \beta(s_{t+1}) \tag{6.21}$$

The base case for the recursion is:

$$\beta(s_t) = 1 \tag{6.22}$$

Numeric underflow will occur if the $\alpha$ and $\beta$ variables are calculated as presented above. To avoid this one can work with $\ln \alpha(s_t)$ and $\ln \beta(s_t)$ or normalise at each step.

Finally, to calculate $\gamma(s_t) \equiv P(s_t|o_{1:T}, a_{1:T-1})$ one simply takes the normalised product of $\alpha$ and $\beta$:

$$\gamma(s_t) = P(s_t|o_{1:T}, a_{1:T-1}) = \frac{\alpha(s_t)\beta(s_t)}{\sum_{s_i \in \mathcal{S}} \alpha(s_i)\beta(s_i)} \tag{6.23}$$

### 6.3.2 The Baum-Welch Algorithm

Recall from Section 3.4.1 that for exact maximum likelihood EM the E-step and M-step have the following forms:

$$\textbf{E step: } q_{y_i}^{t+1} = P(y_i|x_i, \theta) \tag{6.24}$$

$$\textbf{M step: } \theta^{t+1} = \arg\max_{\theta} \sum_i \int dy_i \; P(y_i|x_i, \theta^t) \ln P(x_i, y_i|\theta) \tag{6.25}$$

where $x$ are the observations, $\theta$ the parameters, and $y$ the hidden variables.

Addressing the M-step first gives the forms of the expectations we need from the E-step.

Rephrasing the M-step in terms of the POMDP, the parameters being optimised are the initial, transition, and observation functions $\mathcal{I}$, $\mathcal{T}$, and $\mathcal{Z}$ respectively. The parameters are optimised with respect to the log complete-data likelihood $\ln P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T})$ and the posterior $P(s_{1:T}|o_{1:T}, \mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T})$.

The complete-data likelihood for a POMDP is given by:

$$P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T}) = P(s_1)P(o_1|s_1) \prod_{t=2}^{T} P(s_t|s_{t-1}, a_{t-1})P(o_t|s_t) \tag{6.26}$$

The log of the complete-data likelihood can be written as:

$$\ln P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T}) = \ln P(s_1) + \ln P(o_1|s_1) + \sum_{t=2}^{T} (\ln P(s_t|s_{t-1}, a_{t-1}) + \ln P(o_t|s_t)) \tag{6.27}$$

We can achieve a compact representation in terms of the parameters if we use vector notation. I represent the state occupancy $s_t$ using a binary column vector $\mathbf{s}_t$ such that when $s_t = i$ the $i^{th}$ entry of $\mathbf{s}_t$ is 1 and all other entries are zero, and similarly for $\mathbf{o}_t$. I denote the matrix formed from the transition function by holding the action $a_t$ constant as $\mathcal{T}_{\cdot,a_t,\cdot}$. With this representation the log complete-data likelihood becomes:

$$\ln P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T}) = \mathbf{s}_1^\top \ln \mathcal{I} + \sum_{t=2}^{T} \mathbf{s}_{t-1}^\top \ln \mathcal{T}_{a_{t-1}} \mathbf{s}_t + \sum_{t=1}^{T} \mathbf{s}_t^\top \ln \mathcal{Z} \mathbf{o}_t \tag{6.28}$$

Note that Equation 6.28 has factored into separate terms for $\mathcal{I}$, $\mathcal{T}$, and $\mathcal{Z}$ and thus we can optimise them separately, giving:

$$\mathcal{I}_j = \mathbb{E}(\mathbf{s}_{1,j}) \tag{6.29}$$

$$\mathcal{T}_{ijk} = \frac{\sum_{t=2}^{T} \mathbb{E}(\mathbf{s}_{t-1,i}\mathbf{s}_{t,k}|a_{t-1} = a_k)}{\sum_{t=2}^{T} \mathbb{E}(\mathbf{s}_{t-1,i}|a_{t-1} = a_k)} \tag{6.30}$$

$$\mathcal{Z}_{im} = \frac{\sum_{t=1}^{T} \mathbb{E}(\mathbf{s}_{t-1,i}, \mathbf{o}_{t,m})}{\sum_{t=1}^{T} \mathbb{E}(\mathbf{s}_{t-1,i})} \tag{6.31}$$

The expectations above are taken with respect to $P(s_{1:T}|o_{1:T}, \mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T})$, the posterior over the hidden states. These values are available from the forward-backward iterations as follows:

$$\mathbb{E}(\mathbf{s}_{t,j}) = \gamma(\mathbf{s}_{t,j}) \tag{6.32}$$

$$\mathbb{E}(s_{t-1,i}, s_{t,k}|a_{t-1} = a_k) = \frac{\alpha(\mathbf{s}_{t-1,i})P(\mathbf{s}_t|\mathbf{s}_{t-1}, a_{t-1})\beta(\mathbf{s}_{t,,j})}{\sum_{s'_t}\sum_{s'_{t+1}} \alpha(\mathbf{s}'_{t-1,i})P(\mathbf{s}'_t|\mathbf{s}'_{t-1}, a_{t-1})\beta(\mathbf{s}_{t,,j})} \tag{6.33}$$

$$\mathbb{E}(\mathbf{s}_{t-1,i}, \mathbf{o}_{t,m}) = \gamma(\mathbf{s}_{t,i})P(o_t = o_m|s_t = s_i) \tag{6.34}$$

Thus we have all the needed information from the forward-backward algorithm and the EM algorithm can proceed.

### 6.3.3 Variational Bayesian EM for POMDPs

Having seen maximum likelihood parameter learning for POMDPs we now derive its variational Bayesian equivalent.

As described in Section 3.2.2 the marginal likelihood is the key quantity to compute for model selection. Written in terms of the POMDP the marginal likelihood is:

$$P(o_{1:T}|m, a_{1:T-1}) = \int d\mathcal{I}\ P(\mathcal{I}) \int d\mathcal{T}\ P(\mathcal{T}) \int d\mathcal{Z}\ P(\mathcal{Z}) \sum_{\mathbf{s}_{1:T}} P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}) \quad (6.35)$$

This integral is intractable to compute. Given any particular parameter setting we can compute the likelihood as we've seen above. However now we are integrating over all parameters this is not useful. Similarly, given a hidden state sequence the parameters can be analytically integrated out, but the number of possible sequences grows exponentially with the length of the episode.

We can derive a variational Bayesian EM algorithm starting with the log marginal likelihood, and approximating it with some distribution $q$ over the hidden states and parameters.

$$\ln P(o_{1:T}) = \ln \int d\mathcal{I}\ P(\mathcal{I}) \int d\mathcal{T}\ P(\mathcal{T}) \int d\mathcal{Z}\ P(\mathcal{Z}) \sum_{s_{1:T}} P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}) \quad (6.36)$$

$$\geq \int d\mathcal{I}\ P(\mathcal{I}) \int d\mathcal{T}\ P(\mathcal{T}) \int d\mathcal{Z}\ P(\mathcal{Z}) \sum_{s_{1:T}} q(\mathcal{I}, \mathcal{T}, \mathcal{Z}, s_{1:T}) \ln \frac{P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z})}{q(\mathcal{I}, \mathcal{T}, \mathcal{Z}, s_{1:T})} \quad (6.37)$$

where the approximation follows from Jensen's inequality. Note we have dropped the conditioning on the model $m$ for concision; it is assumed throughout.

Assuming that $q(\mathcal{I}, \mathcal{Z}, \mathcal{T}, s_{1:T})$ can be factored into the form:

$$q(\mathcal{I}, \mathcal{Z}, \mathcal{T}, s_{1:T}) \approx q(\mathcal{I}, \mathcal{Z}, \mathcal{T}) q(s_{1:T}) \quad (6.38)$$

allows us to factor the inequality:

$$\ln P(o_{1:T}) \geq$$

$$\int d\mathcal{I} \int d\mathcal{T} \int d\mathcal{Z}\; q(\mathcal{I},\mathcal{Z},\mathcal{T}) \left[ \ln \frac{P(\mathcal{I},\mathcal{Z},\mathcal{T})}{q(\mathcal{I},\mathcal{Z},\mathcal{T})} + \sum_{s_{1:T}} q(s_{1:T}) \ln \frac{P(s_{1:T}, o_{1:T}|\mathcal{I},\mathcal{T},\mathcal{Z})}{q(s_{1:T})} \right] \tag{6.39}$$

$$= \mathcal{F}(q(\mathcal{I},\mathcal{Z},\mathcal{T}), q(s_{1:T})) \tag{6.40}$$

In (Beal 2003) it is shown that $q(\mathcal{I},\mathcal{Z},\mathcal{T})$ can be factored without further approximation into $q(\mathcal{I})q(\mathcal{Z})q(\mathcal{T})$.

**The VBM Step**

The VBM step is calculated by taking functional derivatives of $\mathcal{F}$ with respect to $q(\mathcal{I})$, $q(\mathcal{Z})$, and $q(\mathcal{T})$ and equating to zero. Solving gives updates to the elements of the Dirichlet distributions defining $\mathcal{I}$, $\mathcal{Z}$, and $\mathcal{T}$ as follows:

$$\phi^{\mathcal{I}}_{s_i} = \phi^{\mathcal{I}}_{i} + \mathbb{E}_{q(s_{1:T})}(\delta(s_1, i)) \tag{6.41}$$

$$\phi^{\mathcal{T}}_{s_i,a_j,s_k} = \phi^{\mathcal{T}}_{s_i,a_j,s_k} + \sum_{t=2}^{T} \mathbb{E}_{q(s_{1:T})}(\delta(s_{t-1}, s_i)\delta(s_t, s_k)\delta(a_{t-1}, a_j)) \tag{6.42}$$

$$\phi^{\mathcal{Z}}_{s_i,o_j} = \phi^{\mathcal{Z}}_{s_i,o_j} + \sum_{t=1}^{T} \mathbb{E}_{q(s_{1:T})}(\delta(s_t, s_i)\delta(o_t, o_i)) \tag{6.43}$$

**The VBE Step**

The VBE step is derived is the same manner as the VBM step, except in this case we hold $q(\mathcal{I})$, $q(\mathcal{Z})$, and $q(\mathcal{T})$ constant and take the functional derivative with respect to $q(s_{1:T})$. Equating to zero and solving gives:

$$\ln q(s_{1:T}) = \int d\mathcal{I} \int d\mathcal{T} \int d\mathcal{Z}\ q(\mathcal{I})q(\mathcal{Z})q(\mathcal{T}) \ln P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T}) - \ln \mathcal{Z}(o_{1:T}) \tag{6.44}$$

$$= \mathbb{E}_{q(\mathcal{I})q(\mathcal{Z})q(\mathcal{T})}(\ln P(s_{1:T}, o_{1:T}|\mathcal{I}, \mathcal{T}, \mathcal{Z}, a_{1:T})) - \ln \mathbb{Z}(o_{1:T}) \tag{6.45}$$

where $\mathbb{Z}(o_{1:T})$ is a normalising constant.

Using the factorisation of Equation 6.28 we can rewrite Equation 6.45 as:

$$\ln q(\mathbf{s}_{1:T}) = \mathbb{E}_{q(\mathcal{I})q(\mathcal{Z})q(\mathcal{T})} \left( \mathbf{s}_1^\top \ln \mathcal{I} + \sum_{t=2}^{T} \mathbf{s}_{t-1}^\top \ln \mathcal{T}_{a_{t-1}} \mathbf{s}_t + \sum_{t=1}^{T} \mathbf{s}_t^\top \ln \mathcal{Z} \mathbf{o}_t \right) - \ln \mathcal{Z}(o_{1:T}) \tag{6.46}$$

$$= \mathbf{s}_1^\top \ln \mathbb{E}_{q(\mathcal{I})}(\mathcal{I}) + \sum_{t=2}^{T} \mathbf{s}_{t-1}^\top \ln \mathbb{E}_{q(\mathcal{T})}(\mathcal{T}_{a_{t-1}}) \mathbf{s}_t + \sum_{t=1}^{T} \mathbf{s}_t^\top \ln \mathbb{E}_{q(\mathcal{Z})}(\mathcal{Z}) \mathbf{o}_t - \ln \mathcal{Z}(o_{1:T}) \tag{6.47}$$

Equation 6.47 now appears identical to Equation 6.28 except that we are taking expectations over the logarithm of the parameters. Therefore we can use the forward-backward algorithm to compute $q(\mathbf{s}_{1:T})$ so long as we can compute the expectation of the log parameters under the Dirichlet prior. This can be done by making use of the integral

$$\int d\mathbf{p} Dirichlet(\mathbf{p}; u) \ln p_i = \psi(u_i) - \psi \left( \sum_{i=1}^{k} u_i \right) \tag{6.48}$$

where $\psi$ is the digamma function. From this it follows that we run the forward-backward algorithm with parameters

$$\hat{\mathcal{I}}_i = \exp\left[\psi(\phi^{\mathcal{I}}_i) - \psi\left(\sum_{j=1}^{K} \phi^{\mathcal{I}}_j\right)\right] \tag{6.49}$$

$$\hat{\mathcal{T}}_{ijk} = \exp\left[\psi(\phi^{\mathcal{T}}_{s_i,a_j,s_k}) - \psi\left(\sum_{l=1}^{K} \phi^{\mathcal{T}}_{s_i,a_j,s_l}\right)\right] \tag{6.50}$$

$$\hat{\mathcal{B}}_{im} = \exp\left[\psi(\phi^{\mathcal{Z}}_{s_i,o_m}) - \psi\left(\sum_{n=1}^{N} \phi^{\mathcal{Z}}_{s_i,o_n}\right)\right] \tag{6.51}$$

Note the exponential in the above equations. Equation 6.47 gives us $\ln q(\mathbf{s}_{1:T})$ but we want $q(\mathbf{s}_{1:T})$. We can take the exponential independently of each parameter as they are uncoupled in the complete data likelihood. Further note that by taking geometric averages we will end up with sub-normalised probabilities (i.e. summing to $\leq 1$). This has no effect on the forward-backward algorithm other than to change the normalisation factor used in the recursions.

#### Incorporating Rewards

So far I have left rewards out of the model learning process. Rewards can be treated as another observation if we have some model for the reward generating process that can compute a likelihood. This is reasonable for many goal attainment tasks when the agent receives some large reward for reaching a goal, and no or negative reward at all other states. Alternatively we can distribute the reward $r_t$ over the states in proportion to $\mathbf{s}_t$, the state occupation probability which we have approximately calculated as part of the variational algorithm.

## 6.4 Experiments

To assess the variational approximation in practice I performed three experiments. The first experiment compares the performance of the variational algorithm to the Bayes-adaptive POMDP algorithm (Ross et al. 2007) on the Tiger problem (Kaelbling, Littman and Cassandra 1998). The second experiment investigates the effect of prior knowledge on the performance of the variational algorithm alone, using the 4x3 grid problem (Russell and Norvig 2003). For both

experiments I use a simple online $d$-step lookahead search to compute a policy from the MAP model. The third experiment continues investigating the effect of the prior, for the first time including a uniform prior, this time using on the 11-state MMaze (Figure 5.3) with deterministic transitions and observations. For this case lookahead search was too time-consuming, so I switched to HSVI, an approximate dynamic programming algorithm (Smith and Simmons 2004).

### 6.4.1 Comparison to BAPOMDP

The Tiger problem is a very simple POMDP. The agent is in a room with one door to the left and one to the right. Behind one door is a hungry tiger, and behind the other is a pile of loot. The agent may listen (for reward -1), hoping to hear the sounds of hungry tiger and thus learn which door to avoid, or open a door. An episode ends when the agent opens a door and learns their fate, receiving reward of -100 upon mauling, and reward 10 otherwise.

For the Tiger problem I follow the experimental setup described in (Ross et al. 2007) and assume that only the observation probabilities are not completely specified. Hence there are four parameters to be learned, $O_{Ll}$, $O_{Lr}$, $O_{Rl}$, $O_{Rr}$, where, for example, $O_{Lr}$ means $Pr(z = hear\_right|s = tiger\_Left, a = listen)$. We define the observation count vector as $\psi = (\psi_{Ll}, \psi_{Lr}, \psi_{Rl}, \psi_{Rr})$, and initialise it to the value $\psi_0 = (5, 3, 3, 5)$. The expected sensor accuracy under this prior is 62.5%, while the true sensor accuracy is 85%.

Each run of the Tiger problem consisted of 100 episodes, and results were averaged over 1000 runs. The discount rate was $0.75$, and 3-step lookahead search was used. The variational algorithm was allowed a maximum of 100 iterations of expectation-maximisation per call, and the Most Probable variant of the Bayes-adaptive algorithm was used. The average total discounted return is shown in Figure 6.1. These results clearly show the variational algorithm outperforms the Bayes-adaptive algorithm.

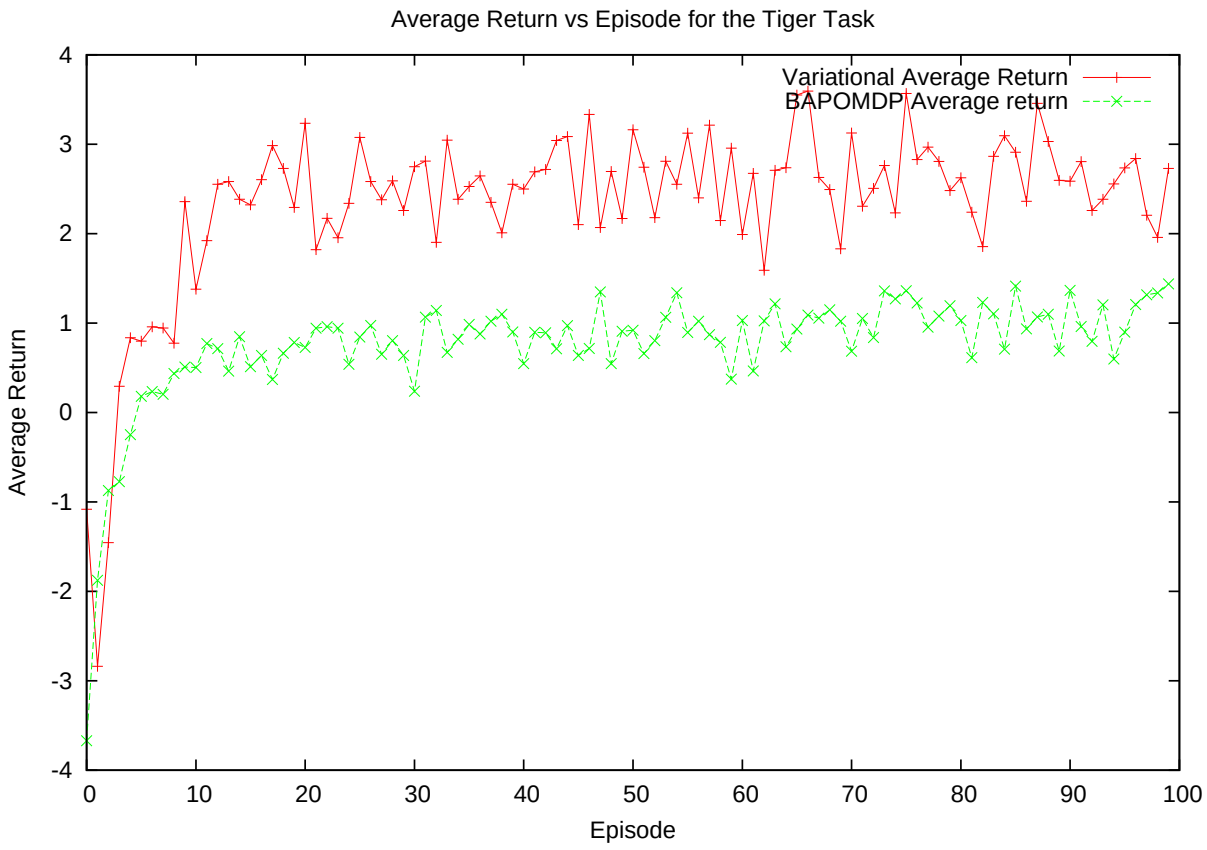


Figure 6.1: Average return over 1000 runs for the variational and Bayes-adaptive algorithms on the Tiger problem

### 6.4.2 Initial Experiments with Priors

The experiment with the Tiger problem above uses a prior that is strongly biased to the correct model. I now investigate how well the variational algorithm performs when the prior is not so informative. These experiments are performed on the 4x3 grid task, shown in Figure 6.2, where the observations, rewards, and initial state distributions are known, but the transitions are not. The priors are constructed by mixing together the true transition matrix and a uniform matrix. There are two parameters to vary: the mixing proportion between the true and uniform matrices, and the number of samples given to the prior. Let $T$ be the true transition matrix, and $U$ be a matrix of the same size with all elements equal to $\frac{1}{|\mathcal{S}|}$ (for the 4x3 grid $|\mathcal{S}|$, the number of states, is 11). Define two parameters: $p$, the mixing proportion, and $n$, the number of 'virtual' samples to assign to the prior. Given $p$ and $n$, the prior is given by:

$$prior = \frac{n}{2}(pT + (1-p)U) \tag{6.52}$$

In this experiment I fixed $n$ to 10, and chose values of $p \in \{0.2, 0.4, 0.6, 0.8\}$. In this way I keep the strength of the prior constant, but vary how informative of the true model it is.

I used a discount rate of 0.75, and a lookahead depth of 7. Each run consisted of 50 episodes, and results were averaged over 10 runs. Two results are reported: the average total discounted return, and the L1 distance between the expectation of the transition matrix and the true transition matrix. Results are given in Figures 6.3 and 6.4, and the average L1 distance at the start and end of the runs is given in Table 6.1.

The results for average return are noisy and difficult to interpret. The results for L1 distance have less noise and are perhaps more informative than the average return. These results show steady learning throughout the course of the runs, though the weaker priors never catch up with the stronger priors.

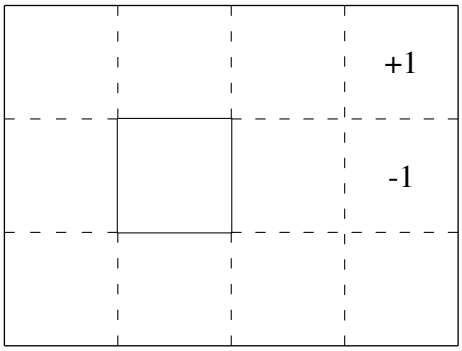


Figure 6.2: The 4x3 Maze. The are 11 possible locations, with the agent starting in any of the nine empty squares. The two terminal states are labelled '+1' and '-1', which is the reward received upon reaching them. All other states have a reward of -0.04. Observations indicate if there is a wall to agent's left and right. Transitions have a 0.8 probability of succeeding and a 0.2 probability of taking the agent in a perpendicular direction.

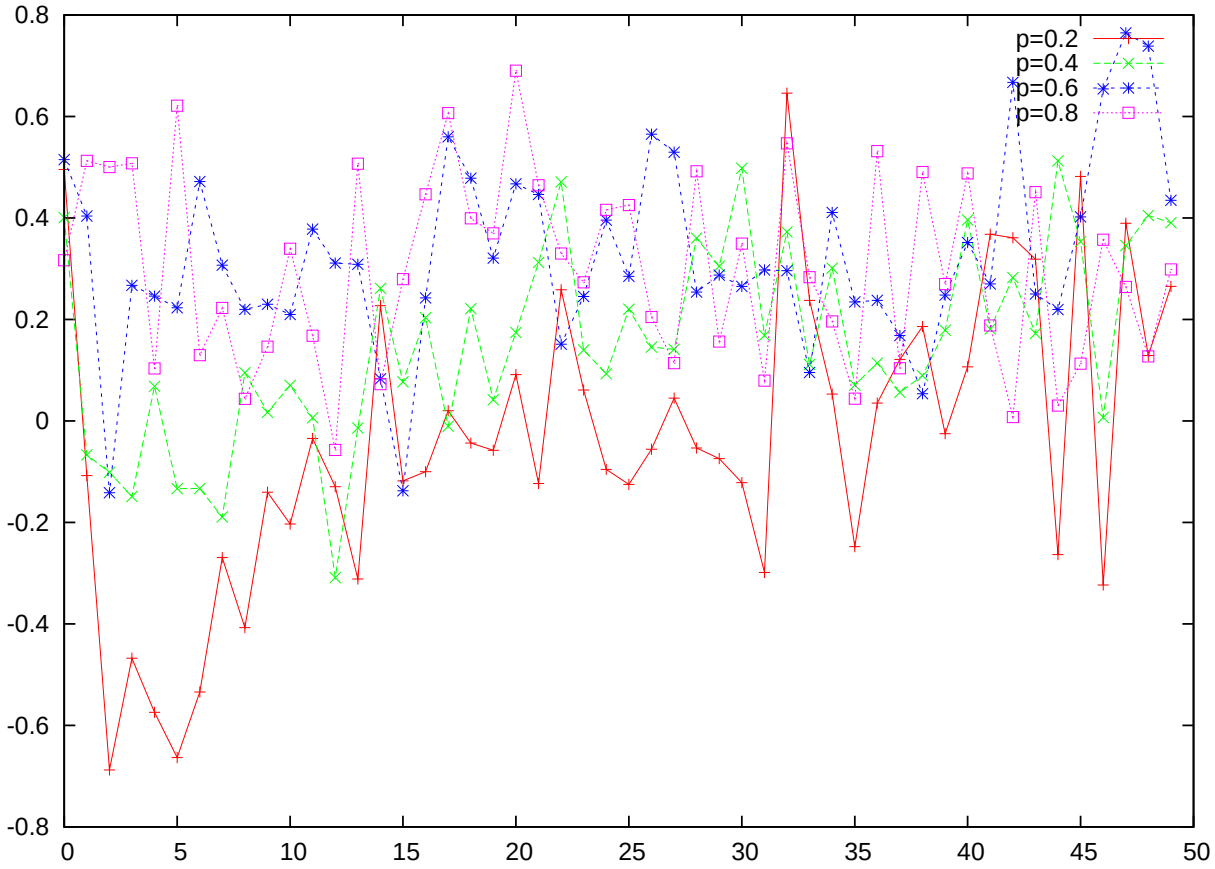


Figure 6.3: Average return over 10 runs with varying prior on the 4x3 grid problem

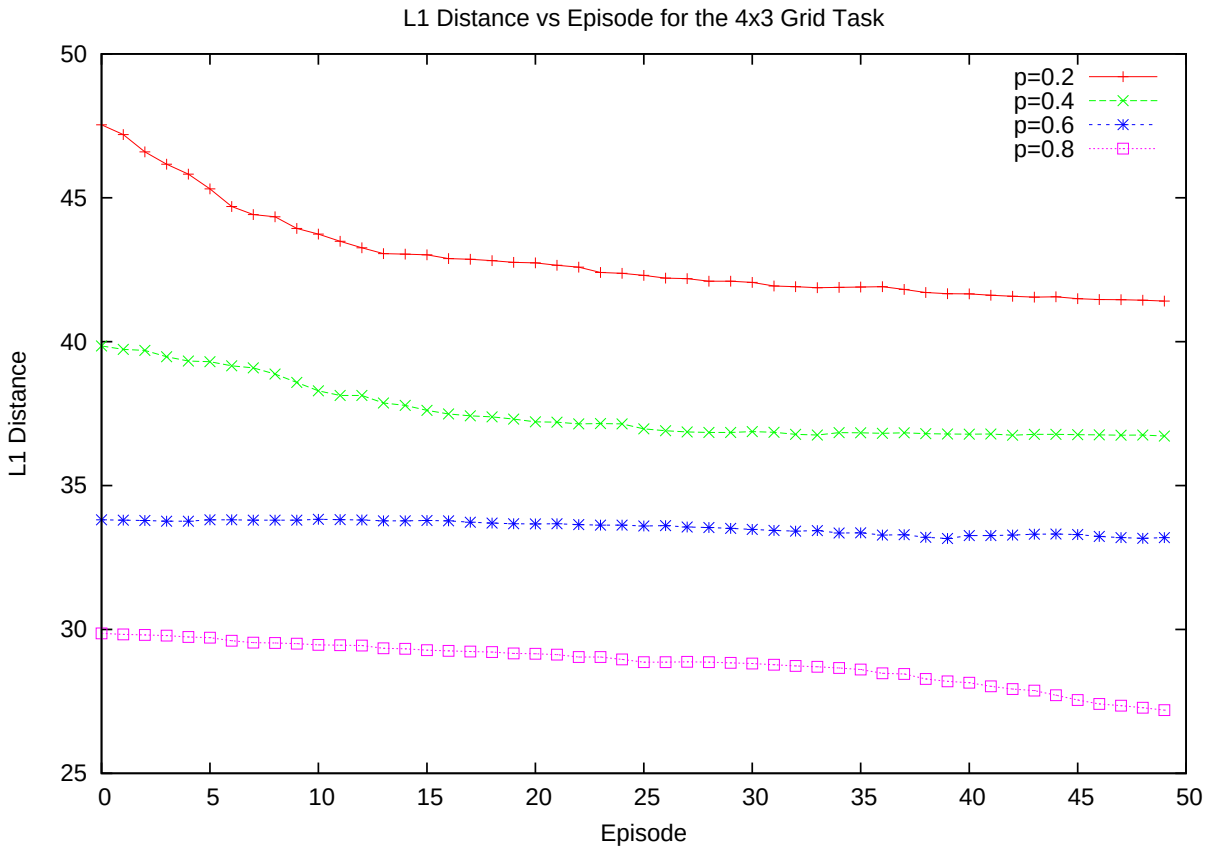


Figure 6.4: Average L1 distance between expectation and true transition matrix over 10 runs with varying prior on the 4x3 grid problem

| $p$ | START | END |
|---|---|---|
| 0.2 | 47.53 | 41.41 |
| 0.4 | 39.85 | 36.72 |
| 0.6 | 33.81 | 33.19 |
| 0.8 | 29.86 | 27.19 |

Table 6.1: Beginning and Ending L1 Distance for Various Priors on the 4x3 Grid Task

### 6.4.3 Further Experiments with Priors

Encouraged by these results I ran another experiment to test the effect of the prior, using the 11-state M-Maze (see Section 5.4) with deterministic transitions and observations. In this experiment I tested unbiased priors for the transition and observation functions as well as biased priors as tested before. To vary the prior more smoothly between the uniform prior and the true distribution the following formula was used:

$$prior(\mathbf{v}, S, B) = \frac{(\mathbf{v} * B) + S}{|\mathbf{v}| + B} \tag{6.53}$$

where $\mathbf{v}$ is the true distribution. The samples parameter $S$ determines how many effective samples we construct the prior from. The bias parameter $B$ determines how the samples are distributed amongst the elements in the vector. For example

$$prior(< 0, 1, 0, 0 >, 1, 0.5) =< 0.22, 0.33, 0.22, 0.22 > \tag{6.54}$$

$$prior(< 0, 1, 0, 0 >, 5, 20) =< 0.21, 1.041, 0.21, 0.21 > \tag{6.55}$$

Experiments were run with three prior distributions: a uniform distribution, a prior weakly biased to the true model (1 sample, bias of 0.5), and a prior strongly biased towards the true model (5 samples, bias of 20). Two different policies were used to collect the data for training the model: the optimal policy, and a mixture that chooses an action with probability 0.5 from the optimal policy, and otherwise at random. The former is known as the exploiting policy, and the later as the exploring policy.

Each data collecting policy was run through the M-Maze 100 times. The variational algorithm was run on the collected data for a maximum of 100 iterations, but in all cases it terminated before this limit was reached. Lookahead search took too long to plan in the learned models, so HSVI was used instead. It was run for a maximum of 10 minutes to generate a policy. Each experiment was repeated forty times.

The results are shown in Table 6.2. Exactly two cases occur – the learned policy is optimal

and achieves a return of -3.95, or the learned policy is hopeless with a return of -91.98. The number of policies that are optimal is shown in Table 6.3.

| | EXPLOITING POLICY | EXPLORING POLICY |
|---|---|---|
| UNIFORM PRIOR | -91.98 (0.00) | -91.98 (0.00) |
| WEAK BIAS | -91.98 (0.00) | -17.16 (31.83) |
| STRONG BIAS | -91.98 (0.00) | -25.96 (38.60) |

Table 6.2: Mean (and standard deviation) over 40 runs of return for policies learned from data collected from the deterministic M-Maze

| | EXPLOITING POLICY | EXPLORING POLICY |
|---|---|---|
| UNIFORM PRIOR | 0 | 0 |
| WEAK BIAS | 0 | 34 |
| STRONG BIAS | 0 | 30 |

Table 6.3: Number of runs out of 40 that achieve the optimal (-3.95) return on the M-Maze

To gain insight into this unusual performance I analysed the learned models and policies. Figure 6.5 shows one model learned from data collected from the exploring policy with the strongly biased prior. This model is very good, and a policy learned from it achieves the optimal return. The combination of the exploring policy that explores the entire model and a strong bias towards the true model work to learn a model with transition probabilities that in most cases are almost equal to the true values. Note however that even in this case some transitions are quite badly misestimated. These transitions are ones for which there are fewer samples.

Figure 6.6 shows a model learned from the exploiting policy with a strong bias. The policy for this model fails to achieve the optimal return. Inspecting the policy shows it chooses to move down in the initial belief state, but this action will not move the agent any closer to the goal. There is a simple explanation for this behaviour: the model implies that moving down leads to all states with equal probability, so the planner believes it is extremely likely (probability of $\frac{9}{11}$) that this move will get the agent to a better location than simply moving up. This occurs because exploiting data collection policy does not explore all transitions and so, even with the biased prior, the model learned from this data does not have good estimates for transition probabilities.

The models learned from the uniform prior are extremely messy, and I haven't found a layout to make it interpretable. The general structure of one model is as follows:

- Three states are correctly identified as bottom states, but two are identified as the goal state.
- The remaining states play are variety of roles; none are strongly biased to any particular state in the true M-Maze.
- The transitions are complex and messy. Geometric constraints are violated. For example, moving up from state 4 has a probability of 0.44 of arriving in state 8. Moving up from state 8 has a probability of 0.95 of arriving in state 4. Clearly this violates a geometric constraint – two states cannot both be above one another.

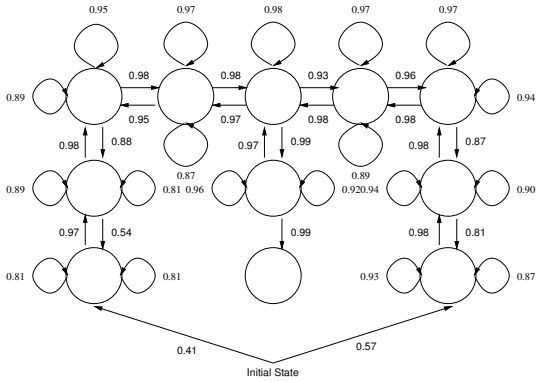


Figure 6.5: Simplified POMDP inferred from data collected by the mixture policy and with a strongly biased prior. Where transitions are not shown they are uniform to all states

## 6.5 Conclusions

The experiments have shown the variational approximation is capable of learning a accurate posterior distribution over models given a weakly informative prior. My comparison with the Bayes-Adaptive framework shows the variational algorithm out-performs it. This is perhaps due to the bounds placed on the Dirichlet parameters in the Bayes-Adaptive formalism, which prevents the accuracy of the model improving with data beyond a certain point.

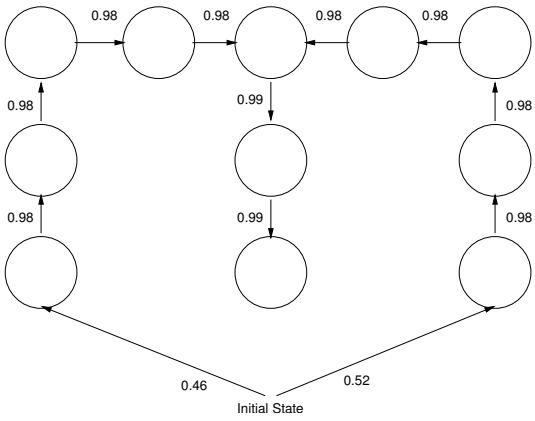


Figure 6.6: Simplified POMDP inferred from data collected by the optimal policy and with a strongly biased prior. Where transitions are not shown they are uniform to all states

Further work is required to more fully explore the performance of the variational POMDP algorithm in larger models and with less informative priors. My results show poor performance with an uninformative prior, but there are several factors that remain to be untangled. Firstly there is the issue of the prior. There are two variables here: the strength of the prior – the number of pseudo or virtual samples – and the bias towards the true model. My experiments cover a very small part of this parameter space, and in particular do not investigate the effect of changing the strength of a uniform prior.

Another major issue is the interaction between the policy and the model. My results show that without sufficient exploration a good policy cannot be learned. However I do not use the model uncertainty to guide exploration, instead sampling the MAP model and planning in it. Better would be to use a Bayesian planning approach (e.g. (Poupart and Vlassis 2008)) to account for the uncertainty in the model and so trade-off exploration and exploitation.

Finally, it is well known that hidden Markov models are difficult to learn so we should expect the same to apply to POMDPs. In (Beal 2003) the Baum-Welch algorithm is used to initialise the variational algorithm. The same technique may improve the performance of POMDP learning.

# CHAPTER 7

# TOPOLOGICAL MAPS

## 7.1 Introduction

In earlier chapters I have laid out two approaches to acting under uncertainty. In the first I directly explore the space of policies. The other method is to learn a model. The idea behind model learning is to allow the agent to identify accurately where it is when it has enough history to do so. The model can thus be thought of as a way to render the hidden state of the world know again. Given a known world state and a model of how the state changes under actions, the agent can then plan how to achieve goals. However this leads to the problem of inferring a model of an unknown world from data. In Chapter 6 we saw how hard this is in the general POMDP case. One route therefore is to adjust the model formalism to make it easier to learn. In this chapter this is exactly what we do. In particular I show how to learn topological maps for worlds with an unknown number of states. The model is learn is similar to a POMDP with a single action (or, a hidden Markov model) and uniform transition probabilities to all neighbours. A particular feature of my maps is that they are sparsely connected. That is, every node is connected to only a few other nodes. This provides some advantages, as we shall see.

A topological map is a map that does not precisely represent metric information of the world. For example, the familiar London Underground, or Tube, map, a fragment of which is shown in Figure 7.1, is a topological map.

The Tube map retains some metric information, however this metric information is dis-

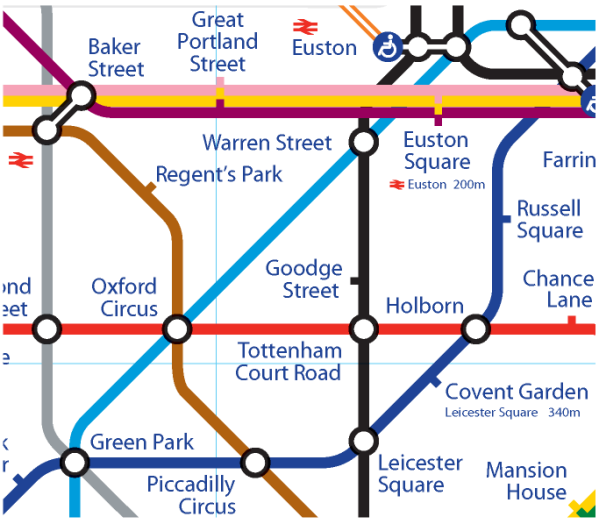


Figure 7.1: A fragment of the London Underground map, an example of a topological map.

torted to make the task of routing planning easier to perform with the human visual system. In particular:

- The Tube map compresses the outer suburbs of London, where there are few lines, and expands the city centre ("Zone 1") where the lines converge.
- The stylised layout and the absence of information such as the street network make lines easier to follow.

The overarching idea behind these properties is the appropriate abstraction of information. The properties of the human visual cortex aren't relevant to robotic applications, but robots none-the-less can benefit from topological maps. Consider a topological map, like the Tube map represented as a graph, but without necessarily embedding it in the two dimensional plane[1]. Planning a route in such a map is a simple application of a graph search algorithm such as Dijkstra's algorithm. Planning a route in a metrically accurate map is a much more involved problem, and one that is typically solved by sampling from the metric map to form a graphical representation, known as a probabilistic roadmap, that is precisely the topological map representation I am proposing (LaValle 2006). Furthermore, scaling metric maps is difficult as

[1]If a topological map was built only from, say, images, it would not contain any metric information and therefore could not be embedded into the two dimensional plane. This is the approach taken by some work such as FAB-MAP (Cummins and Newman 2009, Cummins and Newman 2008). In all my experiments I use odometry data, so my maps can be embedded in the plane.

small errors in measurements accumulate over distance, making it difficult to reconcile measurements taken at different times in the same place. An alternative approach is to divide the map into distinct regions of interest, and represent the area around each region with a local coordinate system, thereby preventing the problem of accumulating error. This approach is known as a hybrid metric-topological map; the topological map specifies the areas of interest and the connections between them, while the metric maps give details at each region. I do not address hybrid maps in this chapter, but my method for learning topological maps can be used as the basis for constructing a hybrid map.

### 7.1.1 Overview of the Chapter

The core of this chapter is the definition of topological maps as directed graphs and the development of an inference technique to learn maps from data. Others have developed similar methods (see Section 7.4) but there are several features that together make this work unique:

- The model of a topological map, a directed graph, explicitly models both places and the connections between them. Much prior work only models places, with connections being implicit in the model.
- There is no a priori limit on the number of places in the map, or the structure of the connections between them.
- I give a complete Bayesian generative model of maps. Learning maps from data involves inferring the posterior distribution over all possible maps.

These features give several advantages:

- Maps are general. That is, they are not restricted to a particular number of states or connectivity structure.
- By explicitly modelling the connections we can use them during inference. I will empirically demonstrate that this is an important condition for dealing with real world data, due

to a phenomenon known as perceptual aliasing. Perceptual aliasing occurs when sensor noise or the structure of the environment makes different places appear the same to the robot. When this occurs the robot must use the history of observations to distinguish places. If the connectivity structure is not constrained during inference this is impossible as any state can reach all others and so there are no paths through the graph that can distinguish aliased states.

- The Bayesian formulation explicitly represents uncertainty in the true map which can be utilised by higher level processes that use the map. Explorer robots will want to know where there is high uncertainty in the map, indicating where further exploration should take place. Reinforcement learning agents can use this information to make optimal decisions trading off exploration and exploitation of existing knowledge.

I call the model the infinite topological map (ITM). In Section 7.2 I present a description of the infinite topological map and develop an inference algorithm in Section 7.3. Related work is discussed in Section 7.4. Of particular importance is the probabilistic topological map (PTM) framework (Ranganathan and Dellaert 2004, Ranganathan and Dellaert 2005, Ranganathan and Dellaert 2006) which is the closest work to mine and to which I compare. This comparison (Section 7.5) is done on simulated data and data taken from real robot runs. I present a variety of results showing that the ITM outperforms the PTM.

## 7.2 The Infinite Topological Map Model

The ITM is a generative Bayesian model. Bayesian learning is reviewed in Section 3.2.2 and briefly recapped here. Being generative means that one part of the ITM is a machine, called the likelihood, for generating observations from maps. Being Bayesian means there is another machine, called the prior, that generates maps. These machines define distributions over the objects they produce, which gives us a declarative way to view the ITM.

Our goal is to adjust the map generating machine, the prior, to account for the observations we've seen so that we are more likely to construct maps consistent with the data. The adjusted

prior is called the posterior. I write $P(Obs|Map)$ for the likelihood and $P(Map)$ for the prior. The posterior is denoted $P(Map|Obs)$. We can calculate the posterior via Bayes rule:

$$P(Map|Obs) \propto P(Obs|Map)P(Map) \tag{7.1}$$

This section defines the likelihood – the observation generating machine – and the prior – the map generating machine. It interleaves the algorithmic and declarative presentation, with the aim of giving the reader both an intuitive and formal understanding of their workings. In Section 7.3 I show how to calculate the posterior – that is, how to adjust the map generating machine given data. There is no closed form expression for the posterior. Instead I build an approximate representation by drawing samples from it. The particular method I use is known as Gibbs sampling.

### 7.2.1 Generating Observations

The likelihood is a machine that generates observations given a map. Thus we must first define a map before we can define how to generate observations. In my representation a map is a directed graph. The set of nodes in the map, denoted $N = \{n_1, n_2, \ldots\}$, are the significant places in the world. The edges are the connections between the places. The set of edges $E$ are represented as binary variables so that $E_{ij} = 1$ if node $i$ is connected to $j$, and $E_{ij} = 0$ otherwise. I write $E_i$ to indicate the binary vector of edges from node $i$.

An example topological map, called Two Squares, is shown in Figure 7.2. The circles indicate the places in the world. The symbols inside the circles are the observations associated with the places. The robot is assumed to have four obstacle detectors, such as thresholded infra red or laser range finders, that indicate if there is an object within a set distance from the robot. The symbols indicate how each place appears to these sensors if they are free of noise. Arrows indicate connections between places.

In the Two Squares example the set of nodes is

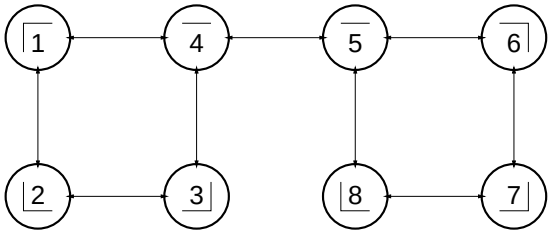


Figure 7.2: The Two Squares map. Circles represent the places in the world. Inside each circle is a symbol, which represents how the place appears to object detectors, and a number, which is the name for the place. Arrows indicate connections between places.

$$N = \{1, 2, 3, 4, 5, 6, 7, 8\} \tag{7.2}$$

and the edges are represented by the binary matrix

$$E = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{7.3}$$

The likelihood machine works as follows: it has a notion of the current place in the map and uses this to generate an observation. It then chooses a connection to take, updating the current place. An important practical consideration is that we don't generate any observations while travelling between places. This means that when we process real data we must filter out any uninteresting observations – that is, observations that don't come from a place as we have defined them above. There are existing ways to do this feature selection such as (Ranganathan and Dellaert 2009) and (Bosse and Zlot 2009).

We handle observations by first building a probabilistic model of the robot's sensors and assigning to each place a parametrisation of this sensor model. The likelihood then samples from

the current place's parametrised sensor model to create an observation. In the Two Squares map, for example, the robot has four obstacle sensors, each emitting a single binary value indicating if the sensor detected an obstacle. Assume each obstacle detector has a fixed probability of generating a correct reading of, say, $0.9$. Then each place in the map is associated with four parameters giving the probability of detecting an obstacle for each of the sensors. For place 1 the parameters would be $(0.9, 0.1, 0.1, 0.9)$. From this we can generate noisy observations. Formally:

- $X$ is the space from which observations $x$ are drawn.
- $F : X, \Theta \mapsto [0, 1]$. $F$ is the observation likelihood function, or sensor model, with parameters $\theta$ drawn from $\Theta$. For an observation $x$ and some parameters $\theta$, $P(x|\theta) = F(x, \theta)$.
- Every node $n_i$ is associated with a single sample $\theta_i \in \Theta$.

Connections are chosen with equal probability from all available choices:

$$P(n_t = j|n_{t-1} = i) = \frac{E_{ij}}{\sum_k E_{ik}} \tag{7.4}$$

When we consider observations we obtain the full likelihood:

$$P(n_t = j|x_t, n_{t-1} = i) = \frac{1}{\mathcal{Z}} F(x_t, \theta_j) \frac{E_{ij}}{\sum_k E_{ik}} \tag{7.5}$$

where $\frac{1}{\mathcal{Z}}$ is a normalising factor.

An example sequence of observations in shown in Figure 7.3. The top line shows the places visited in order. The second line is the observations as they would appear without noise. The third line, which is the only data the robot can access, is the sequence of observations corrupted by noise.

A map has an unbounded number of places and connections, though only a finite number will be involved in generating any finite data set. When I diagram a map I show only the places and connections that generated the data.

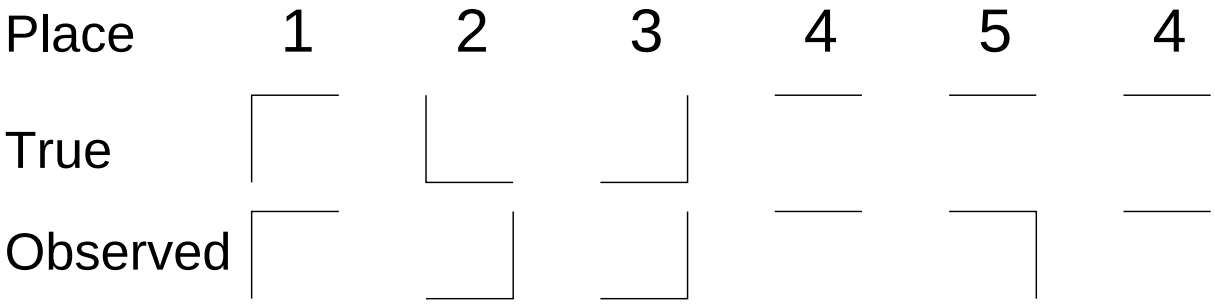


Figure 7.3: An example sequence of noisy observations from the Two Squares map. The top line shows the places that are visited. The second line is the noise-free observations, and the third line is the observations corrupted by noise.

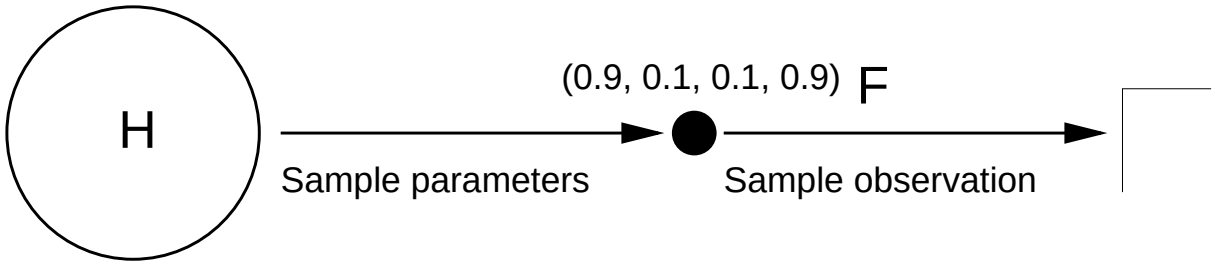


Figure 7.4: The process by which an observation is generated. First parameters are sampled from the space $H$. Then observations are generated from the likelihood function $F$.

Maps exist purely in the space of observations, and are not necessarily embedded in $\Re^2$ or $\Re^3$. If the observations include odometry we will be able to embed the map into (typically) the Euclidean plane. If the observations consist solely of, say, images than this is not possible without external data. My method is entirely general and can be used for any data, real or artificial, for which an sensor model can be built.

### 7.2.2 Generating Maps

I now describe the prior, the machine for generating maps. To generate a map requires generating the sensor model parameters and the connections.

To generate sensor model parameters I require a distribution $H$ over the space of parameters from which the prior samples. For the object detectors in the Two Squares map a natural distribution is a tuple of four Beta distributions. The complete sensor model (prior and likelihood) is illustrated in Figure 7.4. The parameters $\theta$ are drawn from $H$, and observations $x$ are drawn from $F(\theta)$.

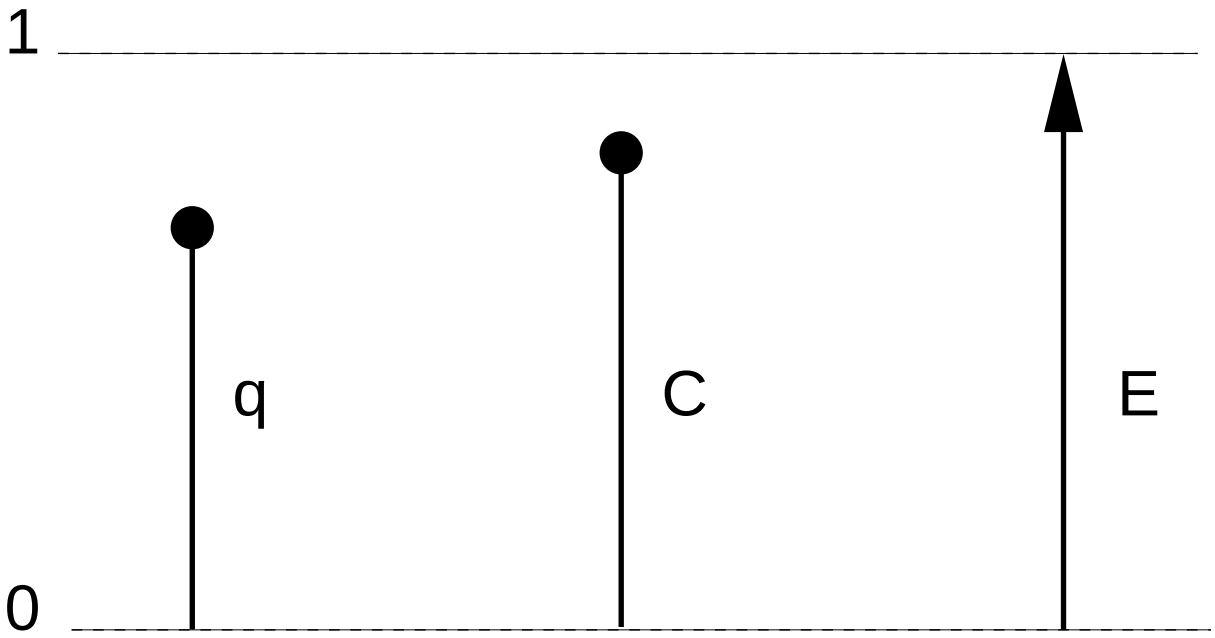


Figure 7.5: The process by which an edge is generated. The weight $q_{ij}$ is given by the geometric sequence. From this weight the connection probability $C_{ij}$ is drawn from $\mathrm{Beta}(q_{ij}, 1 - q_{ij})$. In the illustrated case it is higher than $q_{ij}$ but this need not occur in general. Finally the edge $E$ is sampled from $\mathrm{Bernoulli}(C_{ij})$, in this case giving a 1, indicating an edge exists.

I assume connections are independent, meaning the presence or absence of one connection has no effect on any others. Therefore the prior can generate $E_i$ by a sequence of weighted coin flips. For tractable inference I desire only a finite number of connections, so the infinite sum of the weights should be bounded. A geometric sequence satisfies this property. The elements $q_i$ of a geometric sequence are given by $q_i = g^i$ where $g \in (0, 1)$ is the ratio defining the sequence. The sum of the sequence is given by $\frac{1}{1-g}$. I denote this sequence of weights by $C_i$. Note that this biases edges to connecting to the nodes that occur earlier in the sequence. I return to this issue in Section 7.2.4.

To allow the weights $C_i$ to be adjusted in response to data I place a distribution over them. A Beta distribution is the natural choice here. I make the weights $q_i$ the expected value of the Beta distribution by parametrising it as $\mathrm{Beta}(q_i, 1 - q_i)$. The complete process for creating edges is shown in Figure 7.5.

### 7.2.3 Model Summary

In summary the ITM consists of two parts: the likelihood or observation model that describes how observations are generated given a map, and the prior or network model which describes how maps are generated. The observation model is defined by:

- $X$

  $X$ is the space from which observations $x$ are drawn.

- $\theta \sim \Theta$

  The parameters $\theta$ for the likelihood function are drawn from a space $\Theta$.

- $F : X, \Theta \mapsto [0, 1]$

  $F$ is the observation likelihood function with parameters $\theta$ drawn from $\Theta$. For an observation $x$ and some parameters $\theta$, $P(x|\theta) = F(x, \theta)$. Alternatively we may write $x_i \sim F(\theta)$.

- The prior over $\theta$ is labelled $H$.

The network model is as follows:

- $g \in [0, 1]$

  The ratio defining the geometric series is a number between 0 and 1.

- $q_j = g^j$

  The geometric series is labelled $q$.

- $N : \{n_1, n_2, \ldots\}$

  $N$ is the set of nodes or places. This set is infinite, allowing us to consider an a priori unbounded number of places. Every node is associated with a sample $\theta_i \sim \Theta$.

- $C_{ij} \sim \mathrm{Beta}(q_j, 1 - q_j)$

  Each node $n_i$ has an infinite connectivity vector $C_i$. The element $C_{ij}$ gives the probability of an edge from node $n_i$ to $n_j$.

  The elements of the connectivity vector are drawn from a Beta distribution parametrised by the geometric series.

- $E_{ij} \sim \text{Bernoulli}(C_{ij})$

  For each node $n_i$ the edge function $E_i$ maps every node to a binary digit, indicating the absence or presence of a connection. Alternatively we may view $E_i$ as an infinite binary vector with elements $E_{ij}$, and the complete edge matrix $E$ as an infinite binary matrix.

  $E_i$ is a sequence of draws from a Bernoulli distribution parametrised by $C_i$.

An edge function $E_i$ gives the available connections from a node. To choose between them I assume all connections are equally likely. Thus the probability of taking a transition from node $i$ to $j$, before accounting for observations, is:

$$P(n_t = j | n_{t-1} = i) = \frac{E_{ij}}{\sum_k E_{ik}} \tag{7.6}$$

When we add in observations this distribution becomes:

$$P(n_t = j | x_t, n_{t-1} = i) = \frac{1}{\mathcal{Z}} F(x_t, \theta_j) \frac{E_{ij}}{\sum_k E_{ik}} \tag{7.7}$$

where $\frac{1}{\mathcal{Z}}$ is a normalising factor.

### 7.2.4 Relationship to the Indian Buffet and Beta Processes

The ITM model is a variant of the Indian Buffet Process (Griffiths and Ghahramani 2006), and a particular case of the Beta-Bernoulli process (Hjort 1990, Thibaux and Jordan 2006). The weights $C_i$ are samples from a discrete Beta process with discrete base measure $q$, and the edges $E_i$ are samples from a Bernoulli process with base measure $C_i$. The geometric sequence constructs the discrete base measure $q$ from the sample space $\Theta$.

One effect of the geometric sequence is to give higher weight to the first few places. I have not found this asymmetry to be a problem in practice and it substantially simplifies inference, as described in Section 7.3. However we can obtain a uniform prior distribution over connections by replacing the geometric sequence with a Beta process, giving a hierarchical Beta process model (Thibaux and Jordan 2006) at the cost of more complex inference. The hierarchical Beta

process model can be simplified by integrating out the intermediate Beta process, giving the Indian Buffet Process. In Table 7.1 the three models are compared. I do not further develop these alternate models but discuss possible applications in Section 7.6.

| ITM | HBP | IBP |
|---|---|---|
| $\theta_j \sim H$ | $\theta_j \sim H$ | $\theta_j \sim H$ |
| $x\|\theta_j \sim F(\theta)$ | $x\|\theta_j \sim F(\theta)$ | $x\|\theta_j \sim F(\theta)$ |
| $N = g$ | $N \sim BP(c, H)$ | |
| $C_i\|q, N \sim BP(c_i, N)$ | $C_i\|q, N \sim BP(c_i, N)$ | |
| $E_i\|C_i \sim BeP(C_i)$ | $E_i\|C_i \sim BeP(C_i)$ | $E_i \sim IBP(\alpha)$ |

Table 7.1: The ITM model compared to two alternative models built on the hierarchical Beta process and the Indian Buffer Process.

## 7.3 Inference of Topological Maps

In the previous section I described the prior and likelihood model. In this section I describe how we can adjust the prior to account for data; in other words compute the posterior or perform inference. The network model of the ITM (the process of generating connections) is the same across all uses, but the observation model is dependent on the robot's sensors. In Section 7.2 I treated the observation model in abstract, specifying the existence of $X$, $H$, $\Theta$, and $F$ but not giving their form. Before we can calculate the posterior we must have a concrete observation model, so in this section I start by presenting the models I used for obstacle detectors, odometry data, and laser range finders. In Section 7.3.2 I turn to inference, showing how we can approximate the posterior by a set of samples that are generated using a method known as Gibbs sampling.

### 7.3.1 Observation Models

#### The Obstacle Sensor

A very simple observation model is the obstacle sensor presented in Section 7.2.1 in the context of the Two Squares map. To recap, this sensor outputs a single binary value indicating if there is

an obstacle within a set distance from the sensor. We assume the robot has four of these sensors, and a digital compass allowing it to know its true orientation. This gives the following model:

- $X$ is the space of binary four-tuples $\{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}$.
- $H$ is the cartesian product of four $\mathrm{Beta}$ densities. For convenience I set the initial parameters of the distributions to $(1,1)$ in my experiments. So $H = \mathrm{Beta}(1,1) \times \mathrm{Beta}(1,1) \times \mathrm{Beta}(1,1) \times \mathrm{Beta}(1,1)$.
- $\Theta$ is the subset of $\Re^4$ where all dimensions are restricted to $[0,1]$ (the four dimensional unit cube). Alternatively we can write $\Theta = [0,1] \times [0,1] \times [0,1] \times [0,1]$.
- The $\theta$ are four-dimensional points sampled from $\Theta$ which we write $(\theta^1, \theta^2, \theta^3, \theta^4)$ for some values $\theta^1, \theta^2, \theta^3, \theta^4 \in [0,1]$.
- If we write observations $x$ as $(x^1, x^2, x^3, x^4)$ for $x^1, x^2, x^3, x^4 \in \{0,1\}$ then $F(x,\theta) = \prod_{i=1,2,3,4} x^i\theta^i + (1-x^i)(1-\theta^i)$

The posterior for this sensor model follows directly from its definition, as the likelihood is conjugate to the prior. Say we have some sample observations $\{x_1, x_2, \ldots, x_n\}$. I write $n_0^i$ to indicate the number of observations where $x^i$ was 0, and similarly for $n_1^i$ to indicate the number of observations where $x^i$ was 1. Then the posterior is simply $H' = \mathrm{Beta}(1+n_1^1, 1+n_0^1) \times \mathrm{Beta}(1+n_1^2, 1+n_0^2) \times \mathrm{Beta}(1+n_1^3, 1+n_0^3) \times \mathrm{Beta}(1+n_1^4, 1+n_0^4)$

### Odometry

Odometry readings from a mobile robot typically consist of coordinates in a two-dimensional plane and a bearing. I drop the bearing and use just the $x$ and $y$ coordinates.

A very simple model for odometry that I have found works well in practice is to model the location of each place as a two-dimensional Normal distribution with a fixed spherical covariance. This allows another two-dimensional Normal distribution to be used as a conjugate prior for the mean. Again I used a spherical covariance for the covariance matrix of the prior. I have not observed this restriction to have any affect in practice. The formal definition is:

- $X$ is the two-dimensional plane $\Re^2$.
- $H$ is a two-dimensional Normal distribution $\mathcal{N}(\mu, \sigma_H)$ where $\mu$ is the two-dimensional prior mean and $\sigma_H$ the two-dimensional prior covariance matrix. I restrict $\sigma_H$ to be spherical; that is the off-diagonal elements of the covariance matrix are 0.
- $\Theta$ is the two-dimensional plane $\Re^2$, and so $\theta$ are two-dimensional points.
- $\sigma$ is the fixed covariance matrix for the likelihood, not to be confused with $\sigma_H$, the covariance for the prior on $\Theta$. $\sigma$ is also spherical, with non-zero elements $\sigma_{x,x}$ and $\sigma_{y,y}$
- $F$ is the Normal distribution likelihood given the sampled mean $\theta$ and the fixed variances $\sigma_{x,x}$ and $\sigma_{y,y}$:

$$F(x, y, \theta, \sigma) = \frac{1}{2\pi\sigma_{x,x}\sigma_{y,y}} \exp\left(-\frac{1}{2}\left(\frac{(x-\theta_x)^2}{\sigma_{x,x}^2} + \frac{(y-\theta_y)^2}{\sigma_{y,y}^2}\right)\right)$$

The posterior update is:

$$\mu' = \left(\sigma_H^{-1} + n\sigma^{-1}\right)^{-1}\left(\sigma_H^{-1}\mu + n\sigma^{-1}\bar{x}\right)$$
$$\sigma_H = \left(\sigma_H^{-1} + n\sigma^{-1}\right)^{-1}$$

where $\bar{x}$ is the sample mean and $n$ is the number of observations.

### The Laser Range Finder

Laser scans consists of a sequence of range estimates taken at regular points along an arc. It is uncommon to find a laser range scanner that scans in a full circle. This, along with the fact that rarely will the same point be scanned twice, means that to determine if two laser scans come from the location they must be 'aligned'.

Iterative Dual Correspondence (IDC) (Lu and Milios 1997) is a popular algorithm for aligning laser scans. It aims to find a translation and rotation that minimises the squared error be-

tween two scans. It does this by matching points in the scans (using two methods, hence the name) and then finding an optimal transformation given these correspondences. The algorithm iterates till a local maximum is reached.

I convert the match into a probability as follows:

- I normalise the squared error between scans by dividing by the number of corresponding points. This is so scans with few matching points do not appear to have lower error than scans with many matches
- I model the normalised error as a Gaussian distribution with zero mean and standard deviation calculated from all inter-scan distances. The probability of a scan belonging to a place is the probability of the average distance to all other scans already assigned to the place under this Gaussian model. If there are no scans yet assigned to a place I use the standard deviation over all scans as the distance.

This is clearly an ad-hoc model, but I have found it works given suitable pre-processing of the data. IDC works best when two scans are already very close to one another. This is the situation in typical SLAM uses where it is run on sequences of scans. However it does not describe my setup where there may be considerable distance between scans. In this situation IDC often gets trapped in local minima and some pre-processing is necessary. I clustered the scans for each place using the squared error as a distance metric. Each cluster contained scans within an empirically determined distance, which was 40 in my case. I then selected the largest cluster to use to generate the observations for my data. A better scan matching algorithm would avoid this step. I discuss this matter further in Section 7.6.

### 7.3.2 Inferring Network Structure

There is no closed-form expression for the posterior, so to represent it I resort to sampling. Sampling methods are reviewed in Section 3.4.2. The fundamental idea is to represent the posterior by a set of samples taken from it. The challenge them becomes to draw these samples.

Luckily this is particularly easy with the ITM (and most generative models) as the model is specified in terms of its conditional distributions so we use Gibbs sampling.

To create a sample means to find a value for every variable in the model. The ITM contains many variables but as we shall see we can integrate out $\theta$, and $C$ and $E$, a process known as collapsing or marginalisation.

**Integrating out $\theta$**

I assume that $F$ and $H$ in the observation model are a conjugate pair. When this is the case we can integrate out $\theta$. This is explained in more detail in Section 7.3.1 above.

**Integrating out $C$ and $E$**

From the properties of the Beta process we know the distribution of $C_{ij}$ is $\mathrm{Beta}(q_j, 1 - q_j)$ and that for $E_{ij}$ is $\mathrm{Bernoulli}(C_{ij})$. The expectation for $C_{ij}$ is thus:

$$\mathbb{E}(C_{ij}) = \frac{q_{ij}}{q_{ij} + (1 - q_{ij})} \tag{7.8}$$

$$= q_{ij} \tag{7.9}$$

$$= g^j \tag{7.10}$$

The expectation for $E_{ij}$ is simply $C_{ij}$.

When generating data we don't really care about samples from $E$ as they are only an intermediate step towards choosing which edge we follow. If we can compute the normalising factor $\sum_k E_{ik}$ we can go directly to the probability of choosing an edge, $\frac{E_{ij}}{\sum_k E_{ik}}$, without actually sampling edges.

As we know the expectation of $E_{ij}$ is $C_{ij}$, and we know the expectation of $C_{ij}$ is $g^j$, the normalising factor is, in expectation, the sum of the geometric series. This is the well known equation:

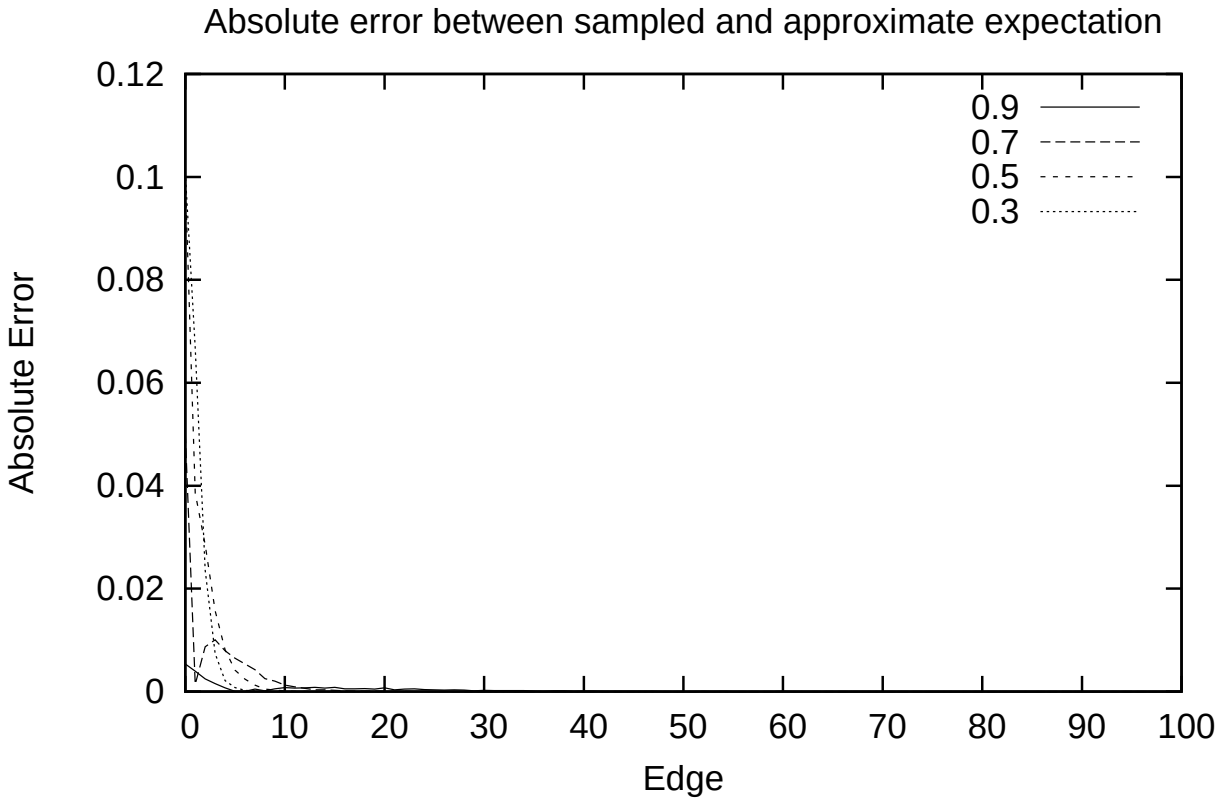


Figure 7.6: Absolute difference between the calculated and approximate expectation for various values of $g$.

$$\sum_{i=1}^{\infty} g^i = \frac{1}{1-g} \tag{7.11}$$

From this we can approximate the expectation

$$\mathbb{E}\left(\frac{E_{ij}}{\sum_k E_{ik}}\right) \approx \frac{g^j}{1-g} \tag{7.12}$$

To evaluate the quality of this approximation with $g \in \{0.3, 0.5, 0.7, 0.9\}$ I sampled 100,000 sequences of $C$, and from them $E$, with length 100. I then computed $\mathbb{E}\left(\frac{E_{ij}}{\sum_k E_{ik}}\right)$ and compared this to the approximation in Equation 7.12. In Figure 7.6 the absolute difference between the calculated and approximate expectations in shown. The error drops off quickly, indicating that Equation 7.12 is a very good approximation to the true expectation.

The above suggests we can remove $C$ and $E$, but we haven't yet considered the effect of observations. It may turn out that we need to keep these variables when we add observations in.

I now analyse the posterior distribution to see what information we need to retain.

The posterior update for $C$ is standard for the beta distribution. If we choose transition $E_{ij}$ then the posterior distribution becomes $\mathrm{Beta}(q_{ij}+1, 1-q_{ij})$. We do not adjust the distribution for any other $C_{ik}, k \neq j$ as we do not have any additional information about these transitions.

After $t_{ij}$ samples of $C_{ij}$ the posterior will be:

$$C_{ij} \sim \mathrm{Beta}(q_{ij}+t_{ij}, 1-q_{ij}) \tag{7.13}$$

and so the posterior expectation of $C_{ij}$ becomes:

$$\mathbb{E}(C_{ij}) = \frac{q_{ij}+t_{ij}}{q_{ij}+t_{ij}+1-q_{ij}} \tag{7.14}$$

$$= \frac{q_{ij}+t_{ij}}{t_{ij}+1} \tag{7.15}$$

$$= \frac{q_{ij}}{t_{ij}+1} + \frac{t_{ij}}{t_{ij}+1} \tag{7.16}$$

$$= \frac{g^j}{t_{ij}+1} + \frac{t_{ij}}{t_{ij}+1} \tag{7.17}$$

We can view this as a mixture model which gives increasingly less weight to the prior value $g^j$ and increasingly more to a Dirac delta function centred at the same point. The normalisation factor must be increased by an amount:

$$\frac{q_{ij} + t_{ij}}{t_{ij} + 1} - q_{ij} \tag{7.18}$$

$$= \frac{q_{ij} + t_{ij}}{t_{ij} + 1} - \frac{q_{ij}(t_{ij} + 1)}{t_{ij} + 1} \tag{7.19}$$

$$= \frac{t_{ij} - t_{ij} q_{ij}}{t_{ij} + 1} \tag{7.20}$$

$$= \frac{t_{ij}(1 - q_{ij})}{t_{ij} + 1} \tag{7.21}$$

$$= \frac{t_{ij}(1 - g^j)}{t_{ij} + 1} \tag{7.22}$$

The normalising factor thus becomes:

$$\frac{1}{1-g} + \sum_j \frac{t_{ij} - t_{ij} g^j}{t_{ij} + 1} \tag{7.23}$$

$$= \frac{1}{1-g} + \sum_j \frac{t_{ij}}{t_{ij} + 1} - \frac{1}{1-g} \sum_j \frac{t_{ij}}{t_{ij} + 1} \tag{7.24}$$

$$= \frac{1}{1-g} + \frac{g}{g-1} \sum_j \frac{t_{ij}}{t_{ij} + 1} \tag{7.25}$$

Given this we only need to record the number of times each edge has been taken, and thus we can remove $C$ and $E$ completely.

### Collapsed Gibbs Sampling

We've seen that we can remove $\theta$, $C$, and $E$ from the model. To complete the inference algorithm I introduce the variables $t_{ij}$ to record the number of times the edge from place $i$ to place $j$ has been taken. The collapsed model is:

$$n_t = i | x_t, n_{t-1} = j, t_i, g \sim F(x_t, H) \left( \frac{g^j + t_{ij}}{\frac{1}{1-g} + \frac{g}{g-1} \sum_k \frac{t_{ik}}{t_{ik}+1}} \right) \tag{7.26}$$

Our challenge now is to sample from the posterior distribution of this model. In other words, to find assignments of observations to places that will in frequency follow the posterior

distribution. It turns out this is easy given the form of Equation 7.26; in fact it is sufficient to sample directly from the conditional distribution of $n$. This method is known as Gibbs sampling (Gelfand and Smith 1990, Geman and Geman 1984) and given sufficient iterations the distribution of samples is sure to converge to the posterior. The algorithm works as follows:

- Start by assigning all observations to random places. Update observation model posterior and counts $t$ appropriately.
- Choose an observation (at random or in order doesn't matter). Remove its current assignment, updating the observation model and counts.
- Sample a new assignment using Equation 7.26.
- Continue resampling assignments for some specified number of iterations.

## 7.4 Related Work

The closest work to mine is the Probabilistic Topological Map (PTM) framework (Ranganathan and Dellaert 2004, Ranganathan and Dellaert 2005, Ranganathan and Dellaert 2006). In PTM a topological map is a partition of the observations. For example, given the sequence of observations

$$(A, B, C, D, E)$$

a map might be the partition

$$\{\{A, C, D\}\{B, E\}\}$$

This implies a map like that shown in Figure 7.7. Note that transitions are not modelled in the PTM (alternatively, the PTM considers every place to be connected to all others). Given a partition we can deduce edges: if two observations occur in sequence but are assigned to different subsets then there must be an edge between them. However edges play no role in

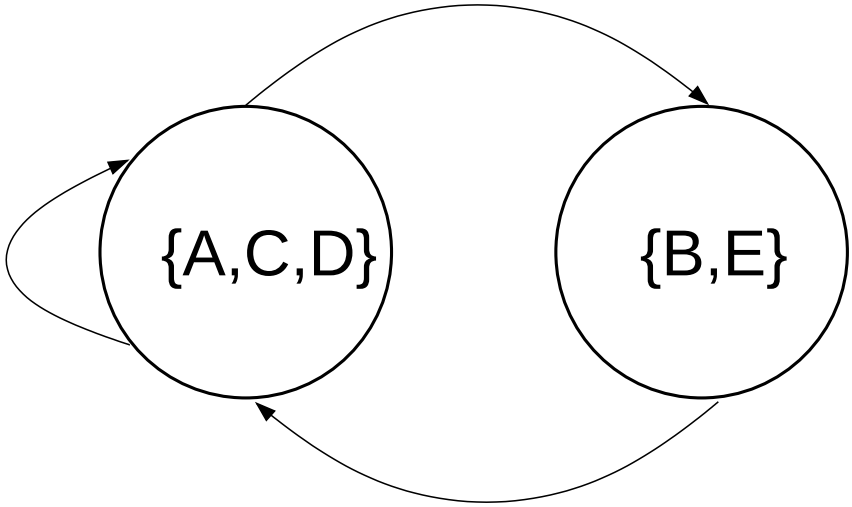


Figure 7.7: An example topological map in the PTM framework induced by the partition $\{\{A, C, D\}\{B, E\}\}$. The connections between places are not used in inference but inferred later given the partition and order of the observations.

the inference process; they are purely an after-the-fact construct. This is the major difference between the ITM and the PTM frameworks. Otherwise from a high-level perspective they are quite similar. In particular the PTM is a Bayesian approach and some instances use a Bayesian non-parametric distribution (specifically the Dirichlet process).

The Hybrid Spatial Semantic Hierarchy (HSSH) (Beeson, Modayil and Kuipers 2010) is a system for building a hierarchy of metric and topological maps from starting from raw robot sensor data. The hierarchy consists of four levels: local metric and topological maps, and a global metric and topological map, of which the global topological map is the closest equivalent to the ITM. The HSSH relies on processing at lower levels to remove perceptual ambiguity so that the process for building the global topological map can work with deterministic actions and observations. The algorithm expands a tree of all possible maps till it finds one that is consistent with the data. It is likely this method will fail to scale well, when there are too many maps to make expanding the tree feasible or there is insufficient data to make removing perceptual ambiguity possible. The ITM works with stochastic observations and uses a sampling method so it doesn't face these scaling problems, but it assumes observations have already been segmented into places.

The topological SLAM technique of (Werner, Maire, Sitte et al. 2009) aims to build maps

that are consistent with all n-grams extracted from the environment. Their motivation is similar to ours: n-grams capture a notion of neighbourhood just like edges. However they require a complete set of n-grams from the world before inference can commence. This is a very strict requirement that is difficult to meet in large worlds. Although their model of a map contains edges, these are not used in inference. Furthermore they do not define a true generative model of maps. They have no prior over nodes, edges, or observations.

An E-M algorithm for inferring topological structure is presented in (De, Lee and Cowan 2008). The algorithm unfortunately is restricted to graphs with one or two cycles and requires the number of nodes is known in advance.

Another approach to topological map inference builds a similarity matrix between all observations, and seeks structure in the matrix to define the map. Markov random field inference is applied to the similarity matrix in (Anati and Daniilidis 2009). Spectral clustering is applied to the similarity matrix in (Valgren, Duckett and Lilienthal 2007) and (Blanco, González and Fernández-Madrigal 2006). These methods generally find a single map, not a distribution over maps as I do, and require a similarity measure which my method does not.

FAB-MAP (Cummins and Newman 2009, Cummins and Newman 2008) focuses almost exclusively on the observation model. The core of FAB-MAP is a generative model for images in a bag-of-words representation. This means that images are represented as unordered sets of binary variables or words indicating the presence of features drawn from some vocabulary. In FAB-MAP the vocabulary is learned by clustering SURF (Bay, Ess, Tuytelaars et al. 2008) or SIFT (Lowe 2004) descriptors taken from a training set of images. Dependencies between words are represented by a tree-structured Bayes net. Locations are represented as independent binary random variables over the vocabulary. The primary operation in FAB-MAP is computing the probability of an observation given a location. FAB-MAP's model has proven very robust in practice, and through a variety of optimisations has been scaled to image data from a tour approximately 1000km long.

A feature-based representation for laser scans is used to build topological maps in (Bosse and Zlot 2009). Like FAB-MAP the main focus on the work is the observation model. Fea-

tures based around histograms of statistical properties of scan points are used to characterise a place. Feature-based representations are more robust to noise and allow sub-linear matching techniques to be used.

## 7.5 Experimental Results

To evaluate the ITM model and inference algorithm I tested it on two data sets, one artificial and one collected from a real robot. Both data sets have a high degree of perceptual aliasing, and on both I compare performance against the PTM. This comparison shows the improvement that reasoning about the connections brings to the quality of the maps inferred.

### 7.5.1 Two Squares

I have used the Two Squares map as a running example throughout this Chapter. To summarise, Two Squares is a simplified map of the School of Computer Science at the University of Birmingham. Each observation consists of a tuple of readings: one from a object detector, and one from odometry. The object detector emits four binary values, indicating if there is a clear path (0) or an object (1) to the North, East, South, and West of the robot. The odometry reading consists of two floating point numbers giving the $x$ and $y$ coordinates of the robot. I drew 39 samples from this model. Figure 7.8 shows the map and data, with crosses indicating the sampled odometry locations, and circles with width 3m indicating the places.

This model has significant aliasing amongst states. Several states appear the same to the object detector. There is less noise in the odometry data but it is still significant.

Observations are generated as follows:

Each of the four elements of the object detector has a probability of 0.1 of generating an incorrect reading. Note that this is a significant amount of noise; the probability of any one reading being completely correct is only 0.6561.

The odometry is generated using a simple motion model: I assume the robot moves exactly between the places, but its odometry readings are corrupted by noise. Each turn is perturbed

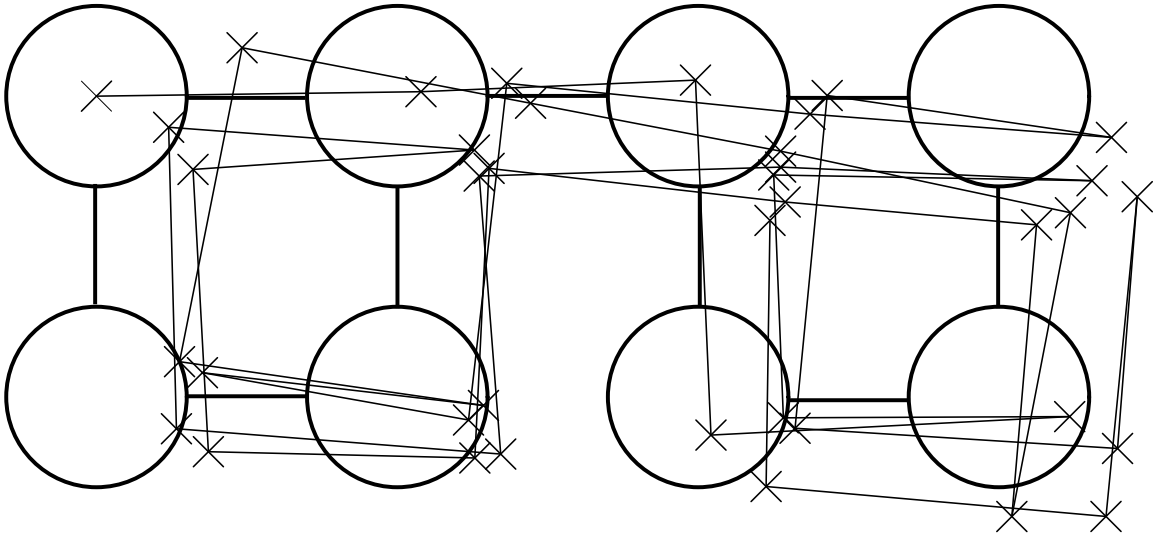

Figure 7.8: The Two-squares data set showing sampled odometry locations and the true locations of the places. The circles enclosing the places are 3 metres wide.

with zero mean and 0.01 standard deviation Gaussian noise per radian turn. Forward motion is perturbed with zero mean and 0.1 standard deviation Gaussian noise per metre in the direction of travel, and zero mean and 0.05 standard deviation Gaussian noise perpendicular to the direction of travel. That is, I use the algorithm shown in Algorithm 7.1.

**Algorithm 7.1:** The algorithm used to synthesis odometry data

$$turn(bearing,\ a) = bearing + a + a * \mathcal{N}(0,\ 0.01)$$

$$\begin{aligned}
&forward((x,\ y),\ bearing,\ s) = \\
&\textbf{let} \\
&\quad fNoise = s * \mathcal{N}(0,\ 0.1) \\
&\quad nNoise = s * \mathcal{N}(0,\ 0.05) \\
&\quad fMult = fNoise + s \\
&\quad xOffset = (nNoise * sin(bearing)) + fMult * cos(bearing) \\
&\quad yOffset = (nNoise * cos(bearing)) + fMult * sin(bearing) \\
&\textbf{in} \\
&(x + xOffset\ ,\ y + yOffset)
\end{aligned}$$

I evaluated the ITM and PTM (using a Dirichlet process partition model) on the Two Squares data. I used a multinomial/Dirichlet pair for the object detector, and a Gaussian/Gaussian pair (unknown mean, known covariance) to model the odometry. The prior for the object detector model was a uniform $(1, 1)$ for each direction. The odometry prior has a mean at $(15, 5)$ (ap-

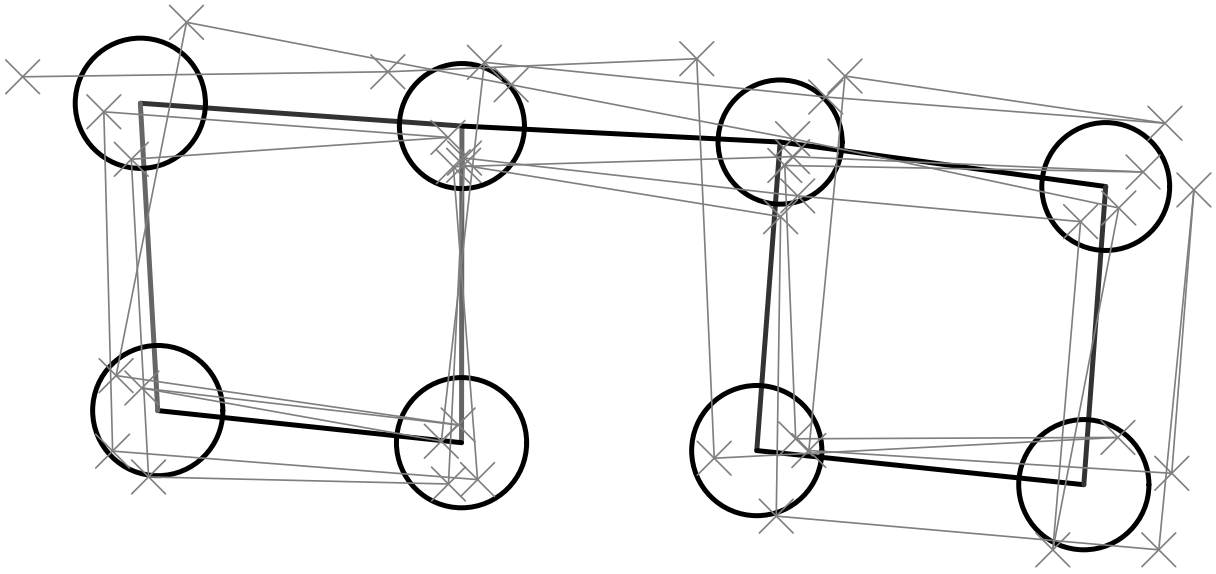

Figure 7.9: The best map found by the ITM algorithm for the Two Squares data. Crosses indicate where odometry samples were taken. The circles, indicating places, are one standard deviation wide. Lines indicate connections, with intensity proportional to the probability of the connection. Connections in the ITM are directed; the illustrated line is the average of the connections in either direction.

proximately the mean of the data) and a covariance of $(10, 10)$. The known covariance was set to $(3, 3)$. For the PTM I set $\alpha$ to 5. For the ITM I set $g$ (the ratio defining the geometric series) to 0.5. I ran Gibbs sampling for 5000 iterations for each model.

Of the 5000 ITM samples 4668 were distinct. For the PTM samples there were 4965 distinct samples. The sample variance in log-likelihood for the ITM was 76.97, and 15.40 for the PTM.

The best model found by the ITM is shown in Figure 7.9. The best map found by the PTM is shown in Figure 7.10. Compared to the PTM map, the ITM map is clearly closer to the ground truth. In particular the PTM is confused by states that are close to one another and have similar observations, whereas the ITM is able to use transition information to distinguish between them.

Figure 7.11 shows the distribution over the number of places for the ITM and PTM. The ground truth has eight places and it is clear the ITM samples are closer to this than the PTM.

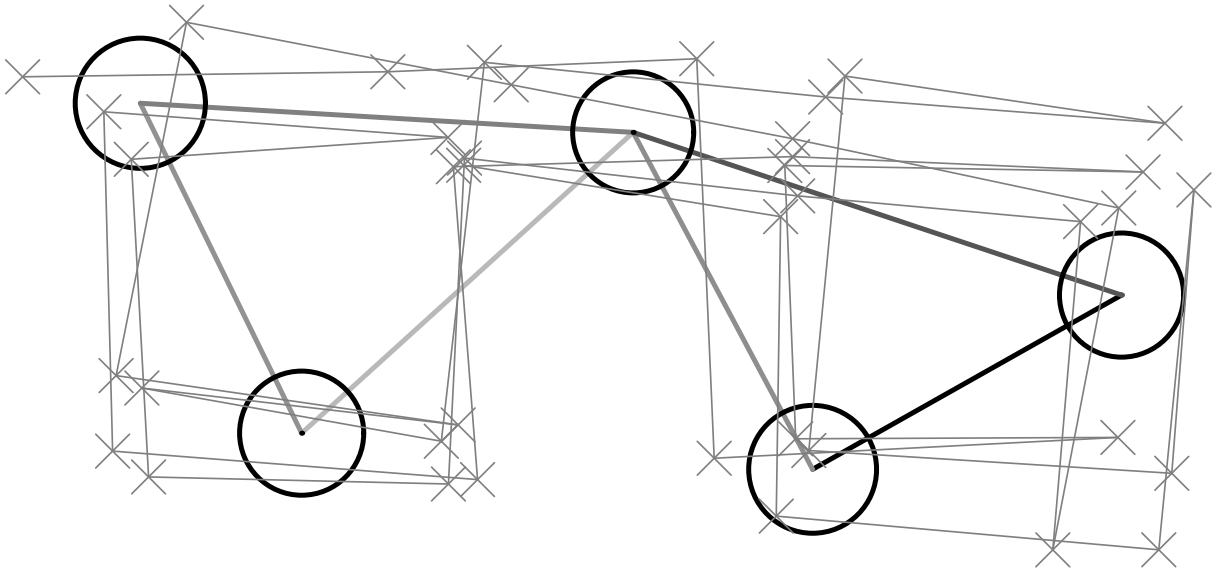

Figure 7.10: The best map found by the PTM algorithm for the Two Squares data. Crosses indicate where odometry samples were taken. The circles, indicating places, are one standard deviation wide. Lines indicate connections, with intensity proportional to the strength of the connection. Connections are synthesised from the data and map as explained in Section 7.4.

### 7.5.2 Austin

The Austin data set is taken from a tour of the ACES building on the University of Texas at Austin campus. This data was obtained from the Robotics Data Set Repository (Radish) (Howard and Roy 2003). A map showing corrected laser scans from the Austin data set is shown in Figure 7.12. The map covers an area 40m$\times$40m. The complete data set consists of laser scans and odometry measurements. I retained the laser scans, but since the odometry covered only a single traversal of the map I generated odometry using the same algorithm as for the Two Squares data. I selected nine points as the places for the topological map, and selected laser scans from a small area about each place.

For the observation model I used the iterative dual correspondence (IDC) algorithm (Lu and Milios 1997) to match laser scans, and assumed a Gaussian noise model. I use the same Gaussian/Gaussian model for odometry as for Two Squares with the prior mean set to the mean of the odometry. I ran the Gibbs sampler for 5000 iterations.

Of the 5000 samples I recorded the ITM sampled 4830 distinct maps, and the PTM sampled 4874 distinct maps. The variance in log-likelihood was 224.13 and 172.16 for the ITM and

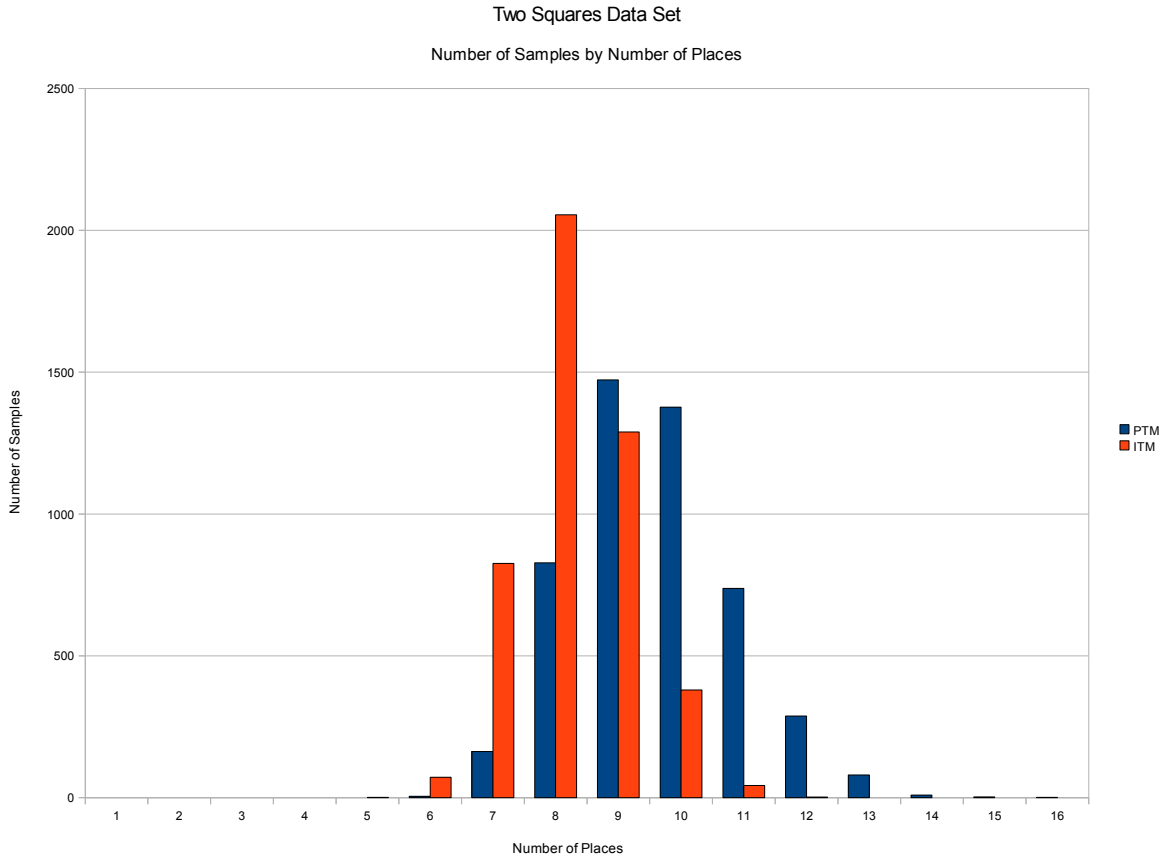


Figure 7.11: The distribution over the number of places in the PTM and ITM samples for the Two Squares data set. The ground truth data has eight places.

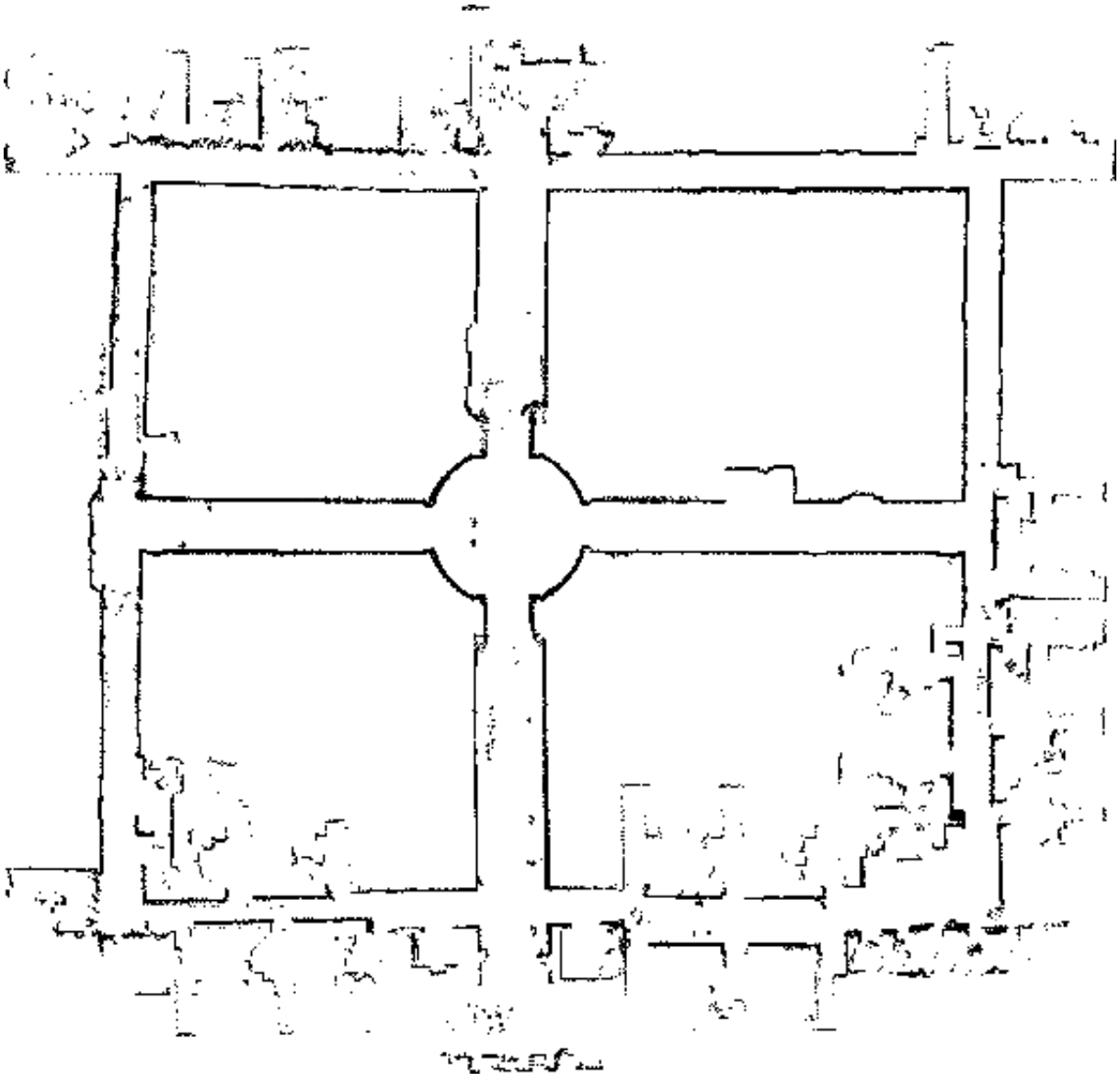

Figure 7.12: The Austin data set, showing the corrected laser scans in the original data set

PTM respectively.

The best model found by the ITM is shown in Figure 7.13 and for the PTM in Figure 7.14. It is clear that a good map is found and again that the ITM outperforms the PTM.

Figure 7.15 shows how many distinct samples had a given number of places for the ITM and PTM. The ground truth data has nine places. Although the two distributions are close the PTM algorithm clearly finds more models with the correct number of states. However, consider the models that are found. Figure 7.16 and Figure 7.17 show the second to fifth best maps found by the ITM and PTM algorithms respectively. There is not a great deal of variance between the samples for either algorithm suggesting we can have some confidence that inferences we draw from this set will carry over to the entire set of samples. It appears that the PTM samples have eight places that correspond to true places in the world, with the remaining places fitting noise. The ITM samples have nine places fitting the true places, with extra places fitting noise. Thus the ITM finding more maps with nine places it not necessarily indicative that it is finding better maps.

## 7.6 Discussion and Conclusions

I have presented a novel Bayesian nonparametric model of directed graphs and used this model to infer topological maps from real and simulated robot data. My experimental results show that this model outperforms the PTM framework validating my claim that explicit modelling of connections is important for resolving perceptual aliasing.

The log-likelihood results suggests that a better inference algorithm will go some way to improving the ITM. One idea is to adjust the forward-backward algorithm, as successfully used in hidden Markov models (HMMs) (Rabiner 1989), to the ITM. The forward-backward algorithm has been adapted to a Dirichlet process model called the infinite HMM (Gael et al. 2008). An ITM version could be built on the stick breaking construction for the IBP (Teh et al. 2007), motivating the alternate model we developed in Section 7.2.4. Alternatively a particle filter would allow on-line inference, which is useful for active robot mapping.

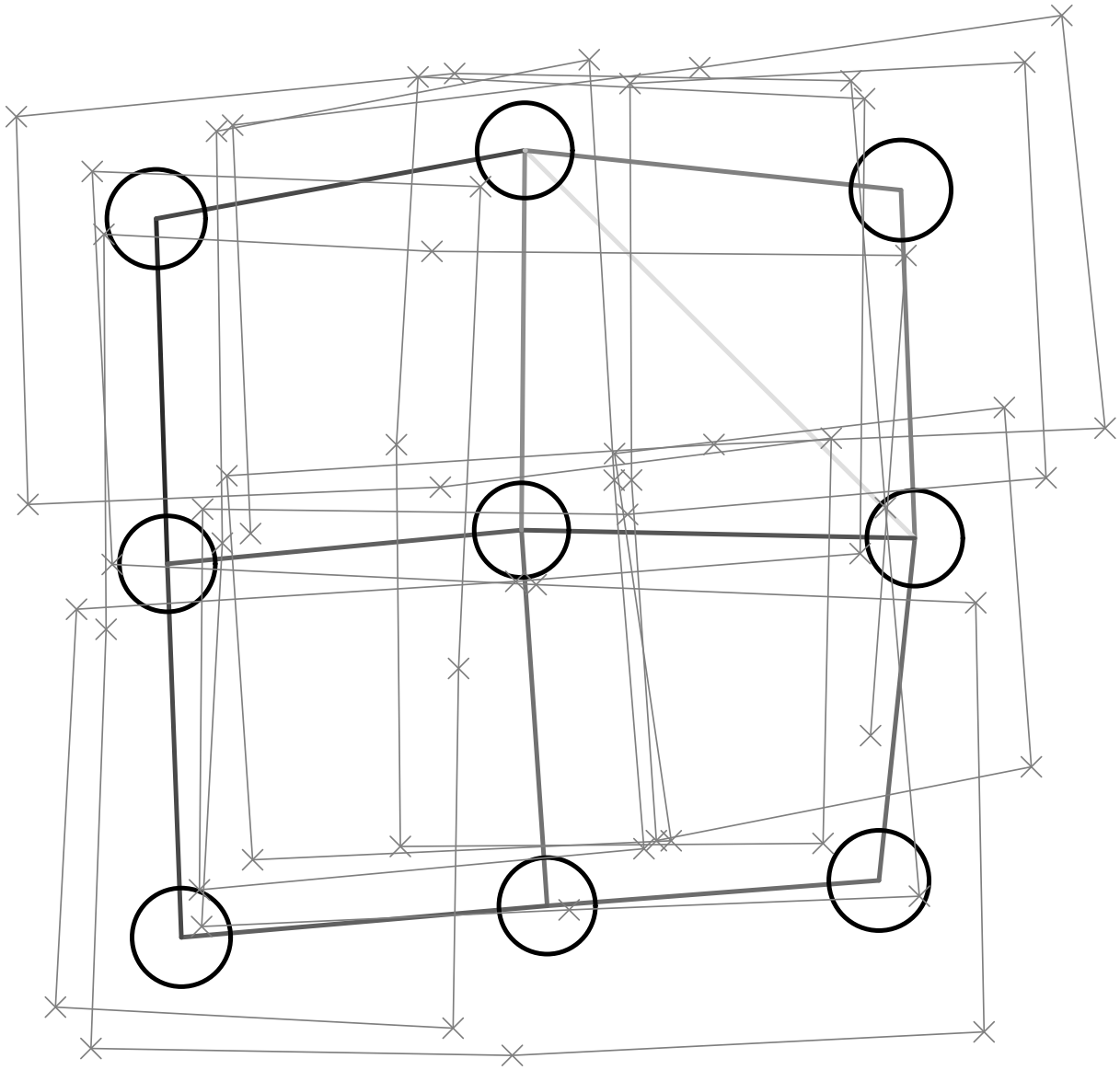

Figure 7.13: The best map found by the ITM algorithm for the Austin data. Crosses indicate where odometry samples were taken. Laser scans are not shown as they create too much clutter. The circles, indicating places, are one standard deviation wide. Lines indicate connections, with intensity proportional to the probability of the connection. Connections in the ITM are directed; the illustrated line is the average of the connections in either direction.

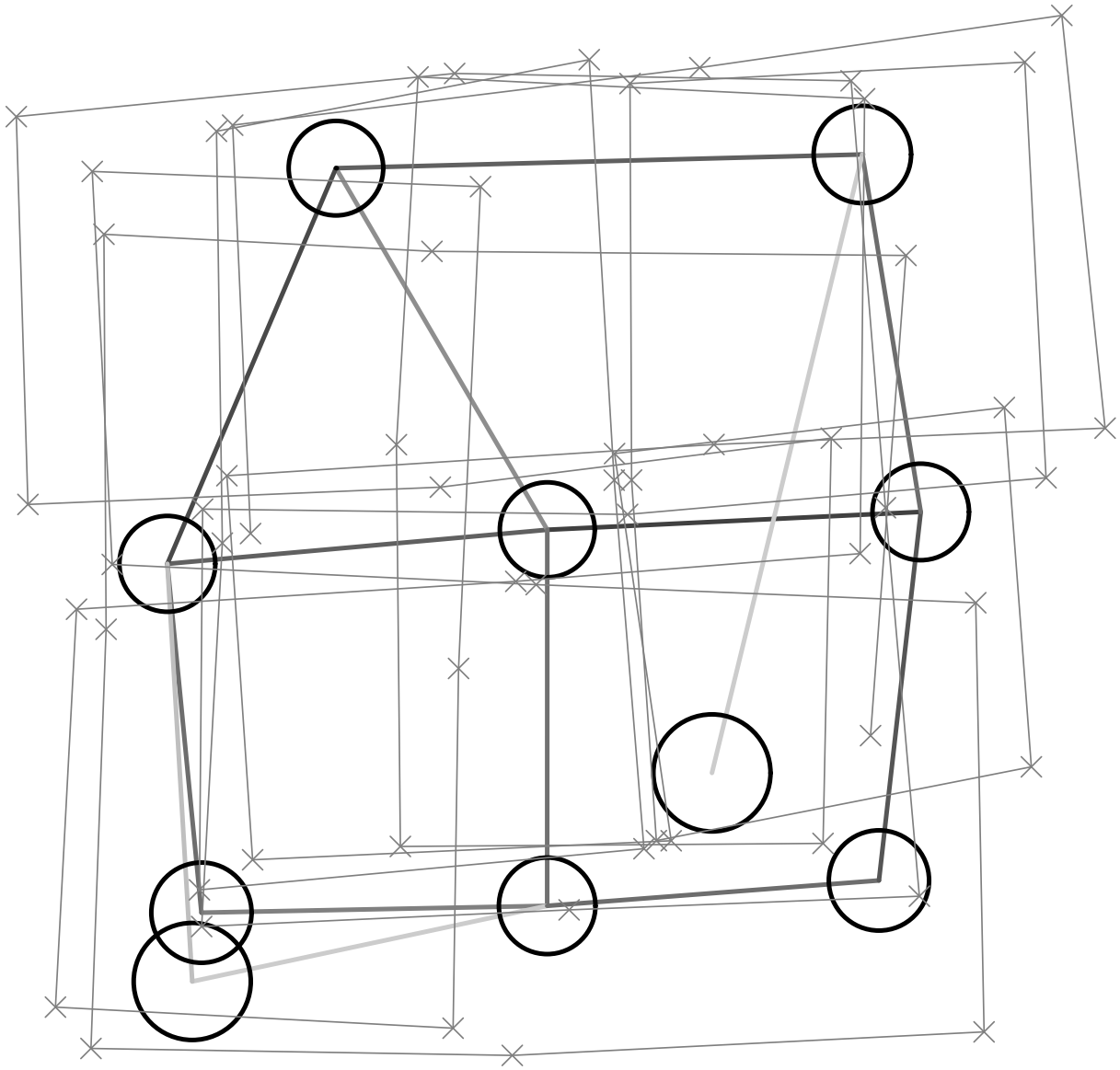

Figure 7.14: The best map found by the PTM algorithm for the Austin data. Crosses indicate where odometry samples were taken. Laser scans are not shown as they create too much clutter. The circles, indicating places, are one standard deviation wide. Lines indicate connections, with intensity proportional to the strength of the connection. Connections are synthesised from the data and map as explained in Section 7.4.

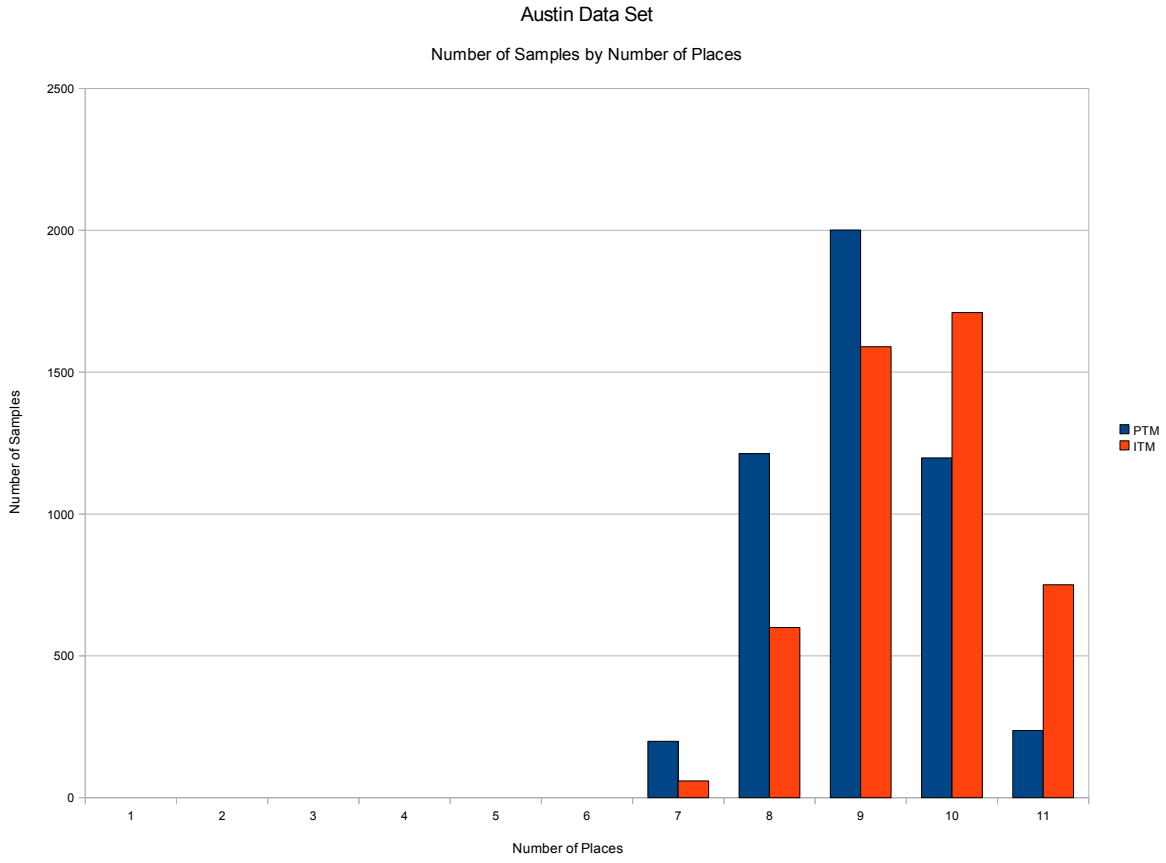


Figure 7.15: The distribution over the number of places in the PTM and ITM samples for the Austin data set. The ground truth data has nine places.

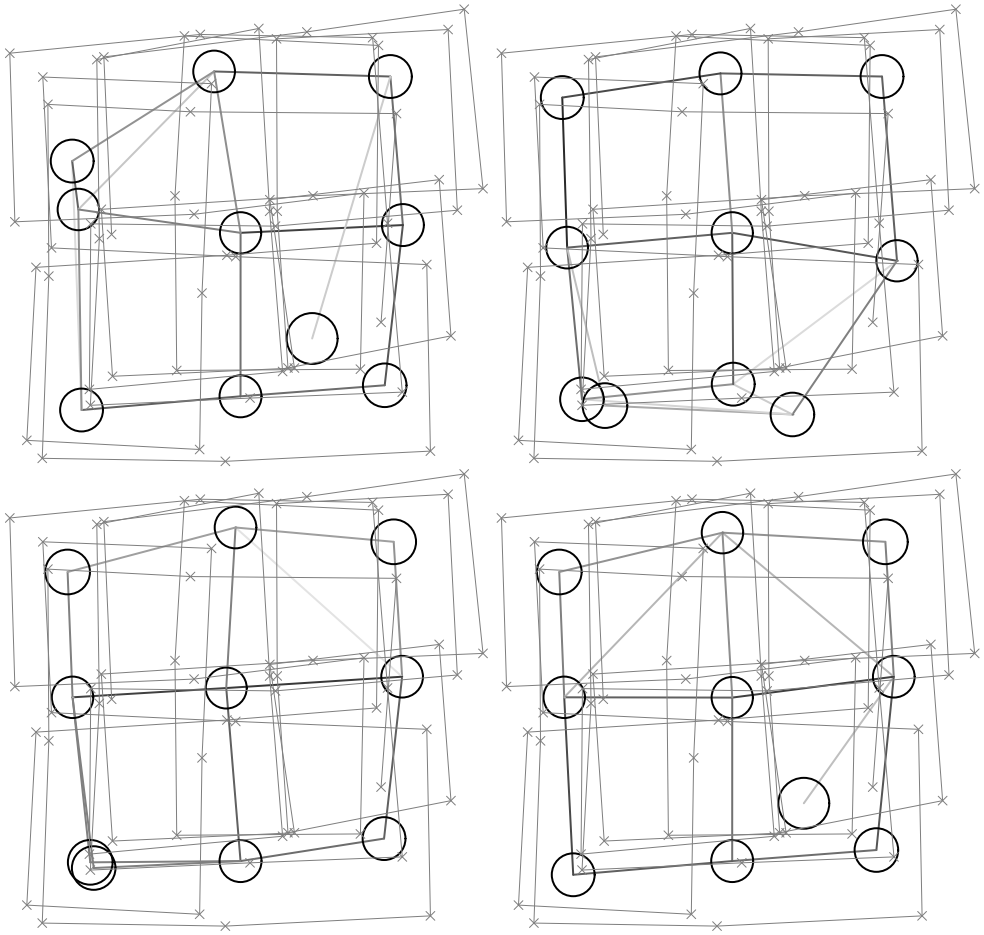

Figure 7.16: The second to fifth best maps found by the ITM algorithm for the Austin data.

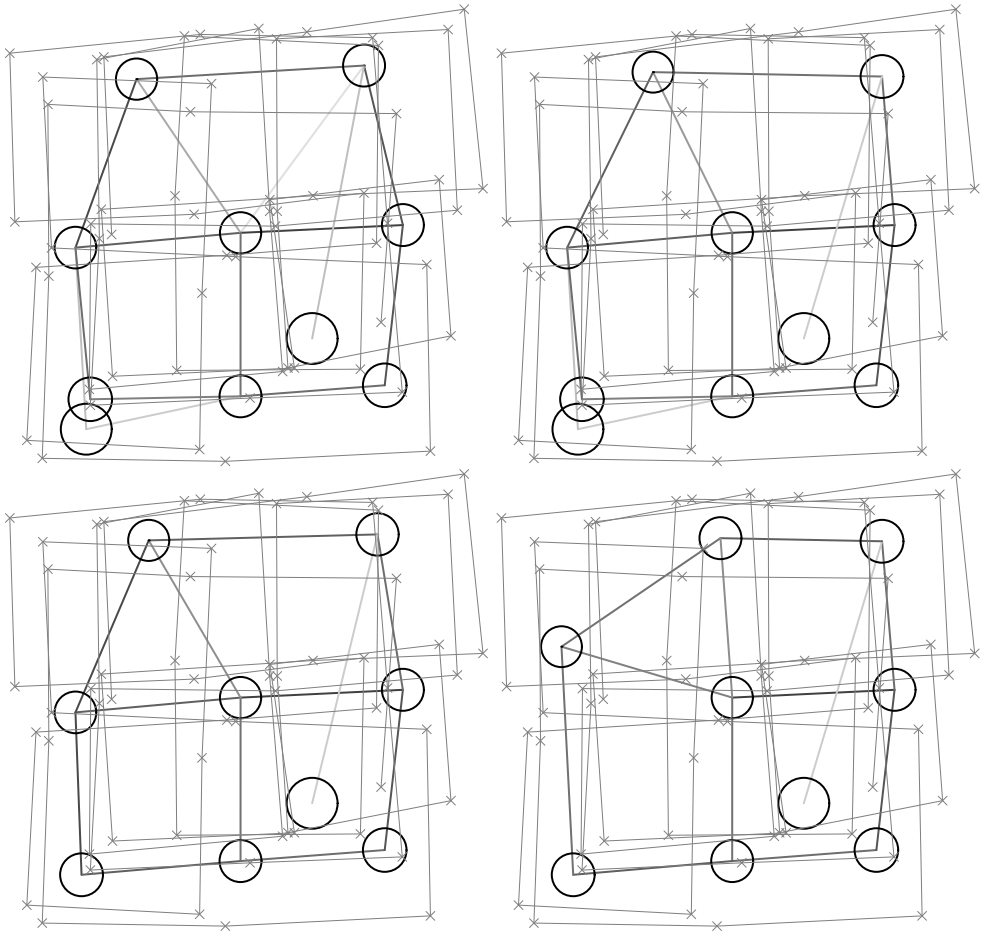

Figure 7.17: The second to fifth best maps found by the PTM algorithm for the Austin data.

The ITM is only as good as the information extracted from the observations. The IDC scan matching algorithm we used for the Austin experiment is designed to align successive scans, where the expected translation and rotation is quite small. Our data set requires large transformations to match scans. In this case IDC and other iterative methods easily get caught in local minima. A better approach is correlative scan matching (Olson 2009) which searches the full (discretised) space of translations and rotations and is shown to be insensitive to the magnitude of the transformation. I am working on a Bayesian version of this method.

I am interested in extending the ITM in two directions: including transition probabilities to build a sparse HMM model, and developing a hierarchical model to build maps out of components such as corridors and rooms. The technique used in the Focused Topic Model (Williamson, Wang, Heller et al. 2009) could be used to do the former. The later is more of a step; I envisage building it using the hierarchical Beta process but there are many details to work out.

Finally I note that the ITM requires the data is segmented into places. Automatic segmentation would allow the ITM to run directly from the raw robot sensor data, and so should be investigated.

# CHAPTER 8

# CONCLUSIONS AND FUTURE WORK

The central thesis of this dissertation is that maintaining a set of policies or models allows us to effectively tackle the sequential decision making problem in unknown and uncertain worlds. This has been shown to be the case, with some caveats.

I have shown that reinforcement learning with a population of policies can achieve near-optimal return on a variety of problems, and outperforms using the same computational resources to learn without the population. However the running time of this policy search algorithm is extremely high. This can be partly alleviated by making the inner loop of the algorithm – the search operators – more efficient. Ongoing work on gradient methods for policy search has led to new algorithms that could replace GAPs (Peshkin et al. 1999) in my implementation, for example. However this does not overcome the fundamental problem that the policy search algorithm is wasteful of data.

I investigated a variational Bayesian algorithm for learning POMDPs, based on a similar algorithm for hidden Markov models (MacKay 1997, Beal 2003). The algorithm learns a model from which a good policy can be found, but only if given a prior biased towards the true model. This is a unrealistic assumption in practice, but even in deterministic worlds dropping this assumption leads to very poor performance. There are several possible explanations for this poor performance and further investigation is required to untangle the interactions between the policy, the model learning algorithm, and the prior.

I have created a novel model for topological maps, the ITM, and shown that maps close

to the ground truth can be learned from robot data. Furthermore that this model outperforms earlier work in the probabilistic topological map framework (Ranganathan and Dellaert 2004, Ranganathan and Dellaert 2005, Ranganathan and Dellaert 2006). This is the most promising work reported here. The results are good with only weak prior assumptions and the running time is moderate.

Addressing my work on policy search, as stated above it makes inefficient use of data. A better policy search method may explicitly place a distribution over policies. For example, one could build an infinite finite state controller model from the hierarchical Dirichlet process, and learn using a artificial likelihood of the form $e^{-V^\pi}$, where $V^\pi$ is the estimated expected return of a controller $\pi$. This immediately raises the issue of how to represent the posterior distribution, of which the most straightforward way is to sample. Thus it appears we've returned to the same problem. However, this is not entirely the case. My policy search algorithm does implicitly define a distribution over policies though this fact is not leveraged by the algorithm. By making the distribution explicit we can ensure that the way we sample from it is consistent – that is, given enough samples we'll get the correct result. Without doing this step it is not obvious that the algorithm will be correct even given infinite data.

Considering the distribution over policies raises another point, which is that is not the policies themselves that are of interest but rather the distributions they define over sequences of interactions. It is possible for what appear to be very different policies to actually define the same such distribution. Ideally we would deal directly with the distribution over sequences, but then we have the issue of how to represent this. It is not apparent to me that there is any compact representation that does not have the same problem with non-identification of parameters.

More powerful observation models will allow the ITM to be extended to larger data sets. Recent results for FAB-MAP (Cummins and Newman 2009) show that observations alone are sufficient for mapping with thousands of images, but a motion model becomes beneficial when dealing with hundreds of thousands of images. Conversely my own prelimiary experiments have shown that the ITM cannot learn a good map with only one of the laser or odometry models I used in my experiments. The two should be complementary – FAB-MAPs motion model

would be improved by the ITM, while more powerful observation models would improve the ITM. Scaling up also requires an appropriate inference algorithm. Gibbs sampling is known to perform poorly for sequential data. Dynamic programming inspired techniques could be used, as in (Gael et al. 2008), and or the techniques in (Bratieres, Van Gael, Vlachos et al. 2010) adapted to scale inference to large data sets.

There are many ways in which the ITM could be extended. Adding actions is straightforward. Adding transition probabilities, perhaps as in (Williamson et al. 2009), would give a sparse infinite HMM or POMDP model. It would be interesting to compare this model to the equivalents based on the Dirichlet process. Sparsity is induced in (Gael et al. 2008), the most successful sampling algorithm for the Dirichlet process models, by truncating the model using slice sampling. In a Beta process based model this sparsity arises naturally, though likely slice sampling or some other technique would be necessary at a higher level of the model to control the complexity of the sampling algorithm.

So far I have said nothing about the model solving. Recent work in reinforcement learning in MDPs has produced frequentist algorithms giving tight regret bounds (e.g. (Szita and Szepesvári 2010, Jaksch, Ortner and Auer 2010)). Comparable results for Bayesian reinforcement learning in completely observable worlds do not exist, to the best of my knowledge, but there is some promising recent work (Tesauro, Rajan and Segal 2010, Dimitrakakis 2008). It is perhaps the case that with a powerful enough observation model we can reduce perceptual aliasing to a level where the world is almost completely observable, at least in robotics applications where sensor processing is rapidly improving. This seems to me the best route to solving Bayesian POMDPs, particularly if we wish to bound performance. There will still be cases where observations are perceptually aliased. Two promising approaches to Bayesian planning in full POMDPs are (Poupart and Vlassis 2008) and (Toussaint, Harmeling and Storkey 2006).

# LIST OF REFERENCES


Abbeel, P. and Ng, A.Y. (2005) "Learning first-order Markov models for control." In Saul, L.K., Weiss, Y. and Bottou, L. (eds.) **Advances in Neural Information Processing Systems 17**. Cambridge, MA: MIT Press. pp. 1–8

Aberdeen, D. and Baxter, J. (2002) Scalable Internal-State Policy-Gradient Methods for POMDPs. **Proceedings of the Nineteenth International Conference on Machine Learning**, p. 3. URL http://portal.acm.org/citation.cfm?id=655827.

Aberdeen, D., Buffet, O. and Thomas, O. (2007) "Policy-gradients for PSRs and POMDPs." In **Proceedings of the 11th International Conference on Artificial Intelligence and Statistics (AIStats)**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.3908\&rep=rep1\&type=pdf.

Aberdeen, D.A. (2003) **Policy-Gradient Algorithms for Partially Observable Markov Decision Processes**. Ph.D. thesis, The Australian National University.

Amari, S.i. (1998) Natural gradient works efficiently in learning. **Neural Computation**, 10 (2): 251–276. URL http://www.mitpressjournals.org/doi/abs/10.1162/089976698300017746.

Anati, R. and Daniilidis, K. (2009) "Constructing topological maps using Markov random fields and loop-closure detection." In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I. and Culotta, A. (eds.) **Advances in Neural Information Processing Systems 22**. pp. 37–45

Andrieu, C., Freitas, N.D., Doucet, A. et al. (2003) An introduction to MCMC for machine learning. **Machine Learning**, 50 (1–2): 5–43. URL http://www.cs.berkeley.edu/~jordan/papers/mlintro.ps.gz.

Antoniak, C.E. (1974) Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems. **The Annals of Statistics**, 2 (6): 1152–1174. URL http://projecteuclid.org/euclid.aos/1176342871.

Au, M. and Maire, F. (2004) "Automatic state construction using decision tree for reinforcement learning agents." In **International Conference on Computational Intelligence for Modelling Control and Automation**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.7597\&rep=rep1\&type=pdf.

Bagnell, J.A. and Schneider, J. (2003) Covariant policy search. **International Joint Conference On Artificial Intelligence**, pp. 1019–1024. URL http://portal.acm.org/citation.cfm?id=1630805.

Bakker, B. (2002) "Reinforcement learning with long short-term memory." In Dietterich, T., Becker, S. and Ghahramani, Z. (eds.) **Advances in Neural Information Processing Systems 14**. Cambridge, MA: MIT Press. vol. 2, pp. 1475–1482. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.3533\&rep=rep1\&type=pdf.

Bay, H., Ess, A., Tuytelaars, T. et al. (2008) SURF: Speeded up robust features. **Computer Vision and Image Understanding (CVIU)**, 110 (3): 346–359

Beal, M. (2003) **Variational algorithms for approximate Bayesian inference**. Ph.D. thesis, University College London. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.9951\&rep=rep1\&type=pdf.

Beal, M., Ghahramani, Z. and Rasmussen, C.E. (2002) "The infinite hidden Markov model." In **Advances in Neural Information Processing Systems 14**. Cambridge, MA: MIT Press. pp. 577–585

Beeson, P., Modayil, J. and Kuipers, B. (2010) Factoring the mapping problem: Mobile robot map-building in the Hybrid Spatial Semantic Hierarchy. **International Journal of Robotics Research**.

Blackwell, D. and MacQueen, J.B. (1973) Ferguson Distributions Via Polya Urn Schemes. **The Annals of Statistics**, 1 (2): 353–355. URL http://projecteuclid.org/euclid.aos/1176342372.

Blanco, J.L., González, J. and Fernández-Madrigal, J.A. (2006) "Consistent observation grouping for generating metric-topological maps that improves robot localization." In **IEEE International Conference on Robotics and Automation (ICRA'06)**.

Boots, B., Siddiqi, S. and Gordon, G. (2009) Closing the learning-planning loop with predictive state representations. URL http://arxiv.org/pdf/0912.2385.

Borman, S. (2004) The Expectation Maximization Algorithm - A short tutorial.

Bosse, M. and Zlot, R. (2009) Keypoint design and evaluation for place recognition in 2d lidar maps. **Robotics and Autononmous Systems**, 57 (12): 1211–1224

Boularias, A. and Chaib-draa, B. (2009) "Predictive representations for policy gradient in POMDPs." In **Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09)**. New York, New York, USA: ACM. URL http://portal.acm.org/citation.cfm?id=1553374.1553383.

Brafman, R. and Shani, G. (2005) "Resolving perceptual aliasing in the presence of noisy sensors." In Saul, L.K., Weiss, Y. and Bottou, L. (eds.) **Advances in Neural Information Processing Systems 17**. Cambridge, MA: MIT Press. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.4994\&rep=rep1\&type=pdf.

Bratieres, S., Van Gael, J., Vlachos, A. et al. (2010) "Scaling the iHMM: Parallelization versus Hadoop." In **Workshop on Scalable Machine Learning and Applications**. URL http://eprints.pascal-network.org/archive/00006994/.

Callut, J. and Dupont, P. (2007) Learning Partially Observable Markov Models from First Passage Times. **Lecture Notes In Artificial Intelligence; Vol. 4701**. URL http://portal.acm.org/citation.cfm?id=1421665.1421679.

Cassandra, A. (1998) **Exact and Approximate Algorithms for Partially Observable Markov Decision Processes**. Ph.D. thesis, Brown University. URL http://portal.acm.org/citation.cfm?id=926710.

Chrisman, L. (1992) "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach." In **Proceedings of the Tenth National Conference on Artificial Intelligence**. pp. 183–188

Crook, P. and Hayes, G. (2003) Learning in a state of confusion: Perceptual aliasing in grid world navigation. **Towards Intelligent Mobile Robots**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.5004\&rep=rep1\&type=pdf.

Cummins, M. and Newman, P. (2008) FAB-MAP: Probabilistic Localization and Mapping in the Space of Appearance. **The International Journal of Robotics Research**, 27 (6): 647–665

Cummins, M. and Newman, P. (2009) "Highly scalable appearance-only SLAM - FAB-MAP 2.0." In **Proceedings of Robotics: Science and Systems**. Seattle, USA.

Cybenko, G. and Crespi, V. (2011) Learning hidden Markov models using non-negative matrix factorization. **IEEE Transactions on Information Theory**, 1 (In submission). URL http://arxiv.org/pdf/0809.4086.

Davis, R.I.A. and Lovell, B.C. (2004) Comparing and Evaluating HMM Ensemble Training Algorithms Using Train and Test and Condition Number Criteria. **Pattern Analysis and Applications**, 6 (4): 327—-336. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.2069.

De, A., Lee, J. and Cowan, N.J. (2008) "Towards SLAM on graphs." In **Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)**. Guanajuato, Mexico.

Dearden, R., Friedman, N. and Andre, D. (1999) "Model-based bayesian exploration." In **Proceedings of the Proceedings of the Fifteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)**. San Francisco, CA: Morgan Kaufmann.

Dempster, A. and Laird, N. (1977) Maximum likelihood from incomplete data via the EM algorithm. **Journal of the Royal Statistical Society. Series B (Methodological)**, 39 (1): 1–38. URL http://www.ams.org/leavingmsn?url=http://links.jstor.org/sici?sici=0035-9246(1977)39:1⟨1:MLFIDV⟩2.0.CO;2-Z\&origin=MSN.

Dimitrakakis, C. (2008) "Tree exploration for Bayesian RL exploration." In **Computational Intelligence for Modelling, Control and Automation 2008 (CIMCA'08)**. URL http://www.computer.org/portal/web/csdl/doi/10.1109/CIMCA.2008.32.

Doshi-Velez, F. (2009) "The Infinite Partially Observable Markov Decision Process." In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C.K.I. and Culotta, A. (eds.) **Advances in Neural Information Processing Systems 22**. vol. 22. URL http://people.csail.mit.edu/finale/papers/finale_ipomdp_nips09.pdf.

Duff, M. (2003) "Design for an optimal probe." In **Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)**.

Dutech, A. (2000) "Solving POMDPs using selected past events." In **Proceedings of the 14th European Conference on Artificial Intelligence (ECAI 2000)**. URL http://scholar.google.co.uk/scholar?cluster=12623713220824949425\&hl=en\&as_sdt=2000\#0.

Ferguson, T.S. (1973) A Bayesian Analysis of Some Nonparametric Problems. **The Annals of Statistics**, 1 (2): 209 – 230. URL http://www.jstor.org/stable/2958008.

Ferrer, M., Alonso, I. and Travieso, C. (2000) Influence of initialisation and stop criteria on HMM based recognisers. **Electronics Letters**, 36 (13): 1165. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=850484.

Friedman, N. and Singer, Y. (1999) "Efficient Bayesian parameter estimation in large discrete domains." In **Advances in Neural Information Processing Systems 11**.

Gael, J.V., Saatci, Y., Teh, Y.W. et al. (2008) "Beam sampling for the infinite hidden Markov model." In **25th International Conference on Machine Learning (ICML 2008)**.

Gelfand, A.E. and Smith, A.F.M. (1990) Sampling-based approaches to calculating marginal densities. **Journal of the American Statistical Association**, 85 (410): 398–409

Geman, S. and Geman, D. (1984) Stochastic relaxation, Gibbs distribution and Bayesian restoration of images. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 6: 721–741

Glickman, M.R. and Sycara, K. (2001) "Evolutionary search, stochastic policies with memory, and reinforcement learning with hidden state." In **Proceedings of the Eighteenth International Conference on Machine Learning**. pp. 194–201

Gomez, F. and Schmidhuber, J. (2008) Accelerated neural evolution through cooperatively coevolved synapses. **Journal of Machine Learning Research**, 9 (May): 937–965. URL http://portal.acm.org/citation.cfm?id=1390712.

Griffiths, T.L. and Ghahramani, Z. (2006) "Infinite latent feature models and the indian buffet process." In **Advances in Neural Information Processing Systems 18**.

Hamzeh, A. and Rahmani, A. (2008) A new architecture for learning classifier systems to solve POMDP problems. **Fundamenta Informaticae**, 84 (3,4): 329–351

Hjort, N.L. (1990) Nonparametric Bayes Estimators Based on Beta Processes in Models for Life History Data. **The Annals of Statistics**, 18 (3): 1259 – 1294. URL http://www.jstor.org/stable/2242052.

Hochreiter, S. and Schmidhuber, J. (1997) Long short-term memory. **Neural Computation**, 9 (8): 1735–1780. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735.

Howard, A. and Roy, N. (2003) The robotics data set repository (Radish). URL http://radish.sourceforge.net/.

Hsu, D., Kakade, S. and Zhang, T. (2009) "A Spectral Algorithm for Learning Hidden Markov Models." In **Twenty-Second Annual Conference on Learning Theory (COLT 2009)**. URL http://arxiv.org/pdf/0811.4413.

Hutter, M. (2005) **Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability**. Springer. URL http://www.amazon.co.uk/Universal-Artificial-Intelligence-Algorithmic-Probability/dp/3540221395.

Jaakkola, T., Singh, S. and Jordan, M. (1995) Reinforcement learning algorithm for partially observable Markov decision problems. **Advances in Neural Information Processing Systems**, pp. 345–352. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.2793\&rep=rep1\&type=pdf.

Jaksch, T., Ortner, R. and Auer, P. (2010) Near-optimal Regret Bounds for Reinforcement Learning. **Journal of Machine Learning Research**. URL http://www.jmlr.org/papers/volume11/jaksch10a/jaksch10a.pdf.

Kaelbling, L.P., Littman, M.L. and Cassandra, A.R. (1998) Planning and acting in partially observable stochastic domains. **Artificial Intelligence**, 101: 99–134

Kaelbling, L.P., Littman, M.L. and Moore, A. (1996) Reinforcement learning: A Survey. **Journal of Artificial Intelligence Research**, 4 (237-285): 102–138. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.44.3732\&rep=rep1\&type=pdf.

Kakade, S. (2002) "A Natural Policy Gradient." In **Advances in Neural Information Processing Systems 14**. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.8133.

Kirkpatrick, S., C. D. Gelatt, J. and Vecchi, M.P. (1983) Optimization by simulated annealing. **Science**, 220 (4598): 671–680

Kohl, N. and Miikkulainen, R. (2008) "Evolving neural networks for fractured domains." In **Genetic And Evolutionary Computation Conference**. URL http://portal.acm.org/citation.cfm?id=1389095.1389366.

Koutník, J., Gomez, F. and Schmidhuber, J. (2010) "Evolving Neural Networks in Compressed Weight Space." In **Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-10)**.

Laud, A. and Dejong, G. (2002) "Reinforcement learning and shaping: Encouraging intended behaviors." In **Proceedings of the Nineteenth International Conference on**

**Machine Learning (ICML '02)**. San Francisco, CA: Morgan Kaufmann Publishers Inc. pp. 355–362. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.3195.

LaValle, S.M. (2006) **Planning Algorithms**. Cambridge University Press.

Lin, L.j. and Mitchell, T.M. (1992) Memory Approaches to Reinforcement Learning in Non-Markovian Domains. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.319.

Littman, M.L. (1994) "Memoryless policies: theoretical limitations and practical results." In **Proceedings of the Third International Conference on Simulation of Adaptive Behavior : from Animals to Animats 3**. URL http://portal.acm.org/citation.cfm?id=189893.

Littman, M.L. (1996) **Algorithms for Sequential Decision Making**. Ph.D. thesis, Brown University.

Loch, J. and Singh, S. (1998) "Using Eligibility Traces to Find the Best Memoryless Policy in Partially Observable Markov Decision Processes." In **Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)**. In Proceedings of the Fifteenth International Conference on Machine Learning. pp. 323–331. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.3975.

Lowe, D.G. (2004) Distinctive image features from scale-invariant keypoints. **International Journal of Computer Vision**, 60 (2): 91–110

Lu, F. and Milios, E. (1997) Robot pose estimation in unknown environments by matching 2d range scans. **Journal of Intelligent and Robotic Systems**, 18 (3): 249–275

MacKay, D.J.C. (1997) Ensemble learning for hidden Markov models. URL http://www.inference.phy.cam.ac.uk/mackay/abstracts/ensemblePaper.html.

MacKay, D.J.C. (1998) "Introduction to Monte Carlo methods." In Jordan, M.I. (ed.) **Learning in Graphical Models**. NATO Science Series, Kluwer. pp. 175–204. URL http://www.inference.phy.cam.ac.uk/mackay/erice.ps.gz.

Madani, O. (2002) "Polynomial value iteration algorithms for deterministic MDPs." In **18th Conference on Uncertainty in Artificial Intelligence**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.7.4535\&rep=rep1\&type=pdf.

McCallum, A. (1995a) "Instance-Based Utile Distinctions for Reinforcement Learning with Hidden State." In **Proceedings of the Twelfth International Conference on Machine Learning (ML '95)**. pp. 387–395. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.132.

McCallum, A. (1995b) **Reinforcement learning with selective perception and hidden state**. Ph.D. thesis, University of Rochester. URL http://portal.acm.org/citation.cfm?id=922924.

McCallum, A., Tesauro, G., Touretzky, D. et al. (1995) Instance-based state identification for Reinforcement Learning. **Advances in Neural Information Processing Systems 7**, pp. 377–384. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.2521\&rep=rep1\&type=pdf.

McGrory, C.A. and Titterington, M. (2006) Variational Bayesian analysis for hidden Markov models. **Australian and New Zealand Journal of Statistics**.

McMahan, H. and Gordon, G. (2005) Generalizing Dijkstra's Algorithm and Gaussian Elimination for Solving MDPs. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.2070\&rep=rep1\&type=pdf.

Meuleau, N., Peshkin, L., Kim, K.e. et al. (1999) "Learning finite-state controllers for partially observable environments." In **Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence**. pp. 427–436. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.102.8052.

Moghaddam, Z. and Piccardi, M. (2009) "Deterministic Initialization of Hidden Markov Models for Human Action Recognition." In **Digital Image Computing: Techniques and Applications 2009**. IEEE. pp. 188–195. URL http://www.computer.org/portal/web/csdl/doi/10.1109/DICTA.2009.37.

Monahan, G.E. (1982) A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. **Management Science**, 28 (1): 1 – 16. URL http://www.jstor.org/stable/2631070.

Moriarty, D.E., Schultz, A.C. and Grefenstette, J.J. (1999) Evolutionary algorithms for reinforcement earning. **Journal of Artificial Intelligence Research**, 11: 241–276

Neal, R. (1991) Bayesian Mixture Modelling by Monte Carlo Simulation.

Neal, R. and Hinton, G.E. (1998) **A View Of The EM Algorithm That Justifies Incremental, Sparse, And Other Variants**. Kluwer Academic Publishers. pp. 355–368. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.2557.

Neal, R.M. (1993) Probabilistic inference using Markov chain Monte Carlo methods. Tech. rep., Department of Computer Science, University of Toronto.

Nikovski, D. and Nourbakhsh, I. (2000) "Learning probabilistic models for decision-theoretic navigation of mobile robots." In **Proceedings of the International Conference on Machine Learning**. pp. 266–274

Olson, E. (2009) "Real-time correlative scan matching." In **Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)**. Kobe, Japan. pp. 4387–4393

Peshkin, L., Meuleau, N. and Kaelbling, L.P. (1999) "Learning policies with external memory." In **Proceedings of the Sixteenth International Conference on Machine Learning**. pp. 307–314

Peters, J., Vijayakumar, S. and Schaal, S. (2003) "Reinforcement learning for humanoid robotics." In **EEE-RAS International Conference on Humanoid Robots (Humanoids2003)**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.7959\&rep=rep1\&type=pdf.

Poupart, P. and Vlassis, N. (2008) "Model-based bayesian reinforcement learning in partially observable domains." In **Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM)**.

Puterman, M.L. (1994) **Markov Decision Processes: Discrete Stochastic Dynamic Programming**. New York, NY: John Wiley and Sons.

Rabiner, L.R. (1989) A tutorial on hidden Markov models and selected applications in speech recognition. **Proceedings of the IEEE**, 77 (2): 257–286

Ranganathan, A. and Dellaert, F. (2004) "Inference in the space of topological maps: An MCMC-based approach." In **IROS**. IEEE.

Ranganathan, A. and Dellaert, F. (2005) "Data driven mcmc for appearance-based topological mapping." In **Robotics: Science and Systems**.

Ranganathan, A. and Dellaert, F. (2006) "A Rao-Blackwellized particle filter for topological mapping." In **ICRA**. IEEE.

Ranganathan, A. and Dellaert, F. (2009) "Bayesian surprise and landmark detection." In **2009 IEEE International Conference on Robotics and Automation, ICRA 2009**. pp. 2017–2023. URL http://ananth.in/pub/Ranganathan09icra.pdf.

Ron, D., Singer, Y. and Tishby, N. (1996) The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. **Machine Learning**, 25 (2): 117–149. URL http://www.springerlink.com/content/u43u01836770w366.

Ross, S., Chaib-draa, B. and Pineau, J. (2007) "Bayes-adaptive pomdps." In **Proceedings of the 21st conference on Neural Information Processing Systems (NIPS'07)**.

Ross, S., Pineau, J., Paquet, S. et al. (2008) Online Planning Algorithms for POMDPs. **The Journal of Artificial Intelligence Research**, 32 (2): 663–704. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2748358\&tool=pmcentrez\&rendertype=abstract.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) **Learning internal representations by error propagation**. chap. 8, pp. 318–362. URL http://portal.acm.org/citation.cfm?id=104293.

Russell, S. and Norvig, P. (2003) **Artificial Intelligence: A Modern Approach**, second ed. New Jersey, USA: Pearson Education. chap. 17, p. 614

Schmidhuber, J. (2004) Optimal Ordered Problem Solver. **Machine Learning**, 54 (3): 211–254. URL http://www.springerlink.com/content/l6v96242k51117w3/.

Sethuraman, J. (1994) A constructive definition of Dirichlet priors. **Statistica Sinica**. URL http://www.computing.edu.au/\~{}phung/nonparam/localpapers/sethuraman.pdf.

Shalizi, C. and Klinkner, K. (2004) "Blind construction of optimal nonlinear recursive predictors for discrete sequences." In **Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference**. pp. 504–511

Sigaud, O., Butz, M.V., Kozlova, O. et al. (2009) Anticipatory learning classifier systems and factored reinforcement learning. **Lecture Notes in Computer Science**, pp. 321–333

Sigaud, O. and Wilson, S.W. (2007) Learning classifier systems: a survey. **Soft Comput.**, 11 (11): 1065–1078

Singh, S., James, M.R. and Rudary, M.R. (2004) "Predictive state representations: A new theory for modeling dynamical systems." In **Uncertainty in Artificial Intelligence: Proceedings of the Twentieth Conference (UAI)**. pp. 512–519. URL http://www.eecs.umich.edu/~baveja/Papers/uai2004psr.pdf.

Singh, S.P., Jaakkola, T. and Jordan, M.I. (1994) "Learning Without State-Estimation in Partially Observable Markovian Decision Processes." In **Intl Conf on Machine Learning**. Morgan Kaufmann. pp. 284–292. URL citeseer.ist.psu.edu/singh94learning.html.

Smallwood, R.D. and Sondik, E.J. (1973) The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon. **Operations Research**, 21 (5): 1071–1088. URL http://www.jstor.org/stable/168926.

Smith, T. and Simmons, R.G. (2004) "Heuristic search value iteration for POMDPs." In **Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)**.

Stanley, K. and Miikkulainen, R. (2002) "Efficient reinforcement learning through evolving neural network topologies." In **Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)**. vol. 1, p. 3. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.7467\&rep=rep1\&type=pdf.

Stolcke, A. and Omohundro, S. (1993) "Hidden Markov model induction by Bayesian model merging." In **Advances in Neural Information Processing Systems 5**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.470\&rep=rep1\&type=pdf.

Suematsu, N. and Hayashi, A. (1999) A reinforcement learning algorithm in partially observable environments using short-term memory. **Advances in neural information processing ...**. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.5242\&rep=rep1\&type=pdf.

Sutton, R.S. and Barto, A.G. (1998) **Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)**. The MIT Press. URL http://www.amazon.com/

Reinforcement-Learning-Introduction-Adaptive-Computation/dp/0262193981.

Szita, I. and Szepesvári, C. (2010) "Model-based reinforcement learning with nearly tight exploration complexity bounds." In **International Conference on Machine Learning (ICML 2010)**. pp. 1031–1038. URL http://dblp.uni-trier.de/db/conf/icml/icml2010.html\#SzitaS10.

Teh, Y., Gorur, D. and Ghahramani, Z. (2007) "Stick-breaking construction for the indian buffet process." In Meila, M. and Shen, X. (eds.) **Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics**.

Teh, Y.W., Jordan, M.I., Beal, M.J. et al. (2006) Hierarchical Dirichlet processes. **Journal of the American Statistical Association**, 101 (476): 1566–1581

Tesauro, G., Rajan, V. and Segal, R. (2010) "Bayesian Inference in Monte-Carlo Tree Search." In **The 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)**. URL http://event.cwi.nl/uai2010/papers/UAI2010_0219.pdf.

Thibaux, R. and Jordan, M.I. (2006) Hierarchical beta processes and the Indian buffet process. Tech. Rep. 719, Department of Statistics, University of California, Berkeley.

Thollard, F., Dupont, P. and de La Higuera, C. (2000) "Probabilistic DFA inference using Kullback-Leibler divergence and minimality." In **Proceedings of the Seventeenth International Conference on Machine Learning**. pp. 975–982. URL http://scholar.google.co.uk/scholar?q=minimal+divergence+inference\&hl=en\&as_sdt=0\&as_vis=1\&oi=scholart\#0.

Toussaint, M., Harmeling, S. and Storkey, A. (2006) Probabilistic inference for solving (PO) MDPs. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.527\&rep=rep1\&type=pdf.

Valgren, C., Duckett, T. and Lilienthal, A.J. (2007) "Incremental spectral clustering and its application to topological mapping." In **ICRA**. Roma, Italy: IEEE. pp. 4238–4288

Werner, F., Maire, F.D., Sitte, J. et al. (2009) "Topological SLAM using neighbourhood information of places." In **IEEE/RSJ International Conference on Intelligent Robots and Systems**.

Wierstra, D., Forster, A., Peters, J. et al. (2009) Recurrent policy gradients. **Logic Journal of the IGPL**. URL http://jigpal.oxfordjournals.org/cgi/content/abstract/jzp049v1.

Wierstra, D. and Wiering, M. (2004) "Utile distinction hidden Markov models." In Brodley, C.E. (ed.) **ICML**. ACM. ACM International Conference Proceeding Series, vol. 69.

Williams, R. and Zipser, D. (1989) Experimental analysis of the real-time recurrent learning algorithm. **Connection Science**, 1 (1): 87–111. URL http://www.informaworld.com/index/776610600.pdf.

Williamson, S., Wang, C., Heller, K. et al. (2009) "Focused topic models." In **Applications for Topic Models: Text and Beyond. (NIPS Workshop)**. Vancouver, Canada.

Williamson, S., Wang, C., Heller, K.A. et al. (2010) "The IBP Compound Dirichlet Process and its Application to Focused Topic Modeling." In **Proceedings of the 27th International Conference on Machine Learning,**. URL http://scholar.google.co.uk/scholar?q=TheIBP-compoundDirichletprocessanditsapplicationtofocusedtopicmodel&oe=utf-8\&rls=org.mozilla:en-GB:official\&client=firefox-a\&um=1\&ie=UTF-8\&sa=N\&hl=en\&tab=ws\#0.