Compact Parametric Models for Efficient Sequential Decision Making in High-dimensional, Uncertain Domains

by

Emma Patricia Brunskill

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author . . ..
Department of Electrical Engineering and Computer Science
May 21, 2009

Certified by . ...
Nicholas Roy
Assistant Professor
Thesis Supervisor

Accepted by . . ..
Terry P.Orlando
Chairman, Department Committee on Graduate Theses

Compact Parametric Models for Efficient Sequential Decision Making in High-dimensional, Uncertain Domains

by

Emma Patricia Brunskill

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2009, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Within artificial intelligence and robotics there is considerable interest in how a single agent can autonomously make sequential decisions in large, high-dimensional, uncertain domains. This thesis presents decision-making algorithms for maximizing the expcted sum of future rewards in two types of large, high-dimensional, uncertain situations: when the agent knows its current state but does not have a model of the world dynamics within a Markov decision process (MDP) framework, and in partially observable Markov decision processes (POMDPs), when the agent knows the dynamics and reward models, but only receives information about its state through its potentially noisy sensors.

One of the key challenges in the sequential decision making field is the tradeoff between optimality and tractability. To handle high-dimensional (many variables), large (many potential values per variable) domains, an algorithm must have a computational complexity that scales gracefully with the number of dimensions. However, many prior approaches achieve such scalability through the use of heuristic methods with limited or no guarantees on how close to optimal, and under what circumstances, are the decisions made by the algorithm. Algorithms that do provide rigorous optimality bounds often do so at the expense of tractability.

This thesis proposes that the use of parametric models of the world dynamics, rewards and observations can enable efficient, provably close to optimal, decision making in large, high-dimensional uncertain environments. In support of this, we present a reinforcement learning (RL) algorithm where the use of a parametric model allows the algorithm to make close to optimal decisions on all but a number of samples that scales polynomially with the dimension, a significant improvement over most prior RL provably approximately optimal algorithms. We also show that parametric models can be used to reduce the computational complexity from an exponential to polynomial dependence on the state dimension in forward search partially observable MDP planning. Under mild conditions our new forward-search POMDP planner maintains prior optimality guarantees on the resulting decisions. We present experimental results on a robot navigation over varying terrain RL task and a large global driving POMDP planning simulation.

Thesis Supervisor: Nicholas Roy
Title: Associate Professor

Acknowledgments

As I pass out of the hallowed hallows of the infamous Stata center, I remain deeply grateful to those that have supported my graduate experience. I wish to thank my advisor Nicholas Roy, for all of his insight, support and particularly for always urging me to think about the potential impact of my work and ideas, and encouraging me to be broad in that impact. Leslie Kaelbling and Tomas Lozano-Perez, two of my committee members, possess the remarkable ability to switch their attention instantly to a new topic and shed their immense insight on whatever technical challenge I posed to them. My fourth committee member, Michael Littman, helped push me to start thinking about the theoretical directions of my work, and I greatly enjoyed a very productive collaboration with him, and his graduate students Lihong Li and Bethany Leffler.

I have had the fortune to interact extensively with two research groups during my time at MIT: the Robust Robotics Group and the Learning and Intelligent Systems group. Both groups have been fantastic environments. In addition, Luke Zettlemoyer, Meg Aycinena Lippow and Sarah Finney were the best officemates I could have ever dreamed of, always willing to read paper drafts, listen attentively to repeated practice presentations, and share in the often essential afternoon treat session. I wish to particularly thank Meg and Sarah, my two research buddies and good friends, for proving that an article written on providing advice on how to be a good graduate student was surprizingly right on target, at least on making my experience in graduate school significantly better.

Far from the work in this thesis I have been involved in a number of efforts related to international development during my stay at MIT. It has been my absolute pleasure to collaborate extensively with Bill Thies, Somani Patnaik, and Manish Bhardwaj on some of this work. The MIT Public Service Center has generously provided financial assistance and advice on multiple occasions for which I am deeply grateful. I also appreciate my advisor Nick's interest and support of my work in this area.

I will always appreciate my undergraduate operating systems professor Brian Bershad for encouraging me to turn my gaze from physics and seriously consider computer science. I credit he, Gretta Smith, the graduate student that supervised my first computer science research project, and Hank Levy for showing me how exciting computer science could be, and convincing me that I could have a place in this field.

My friends have been an amazing presence in my life throughout this process. Molly Eitzen and Nikki Yarid have been there since my 18th birthday party in the first house we all lived in together. The glory of Oxford was only so because of Chris Bradley, Ingrid Barnsley, Naana Jumah, James Analytis, Lipika Goyal, Kirsti Samuels, Ying Wu, and Chelsea Elander Flanagan Bodnar and it has been wonderful to have several of them overlap in Boston during my time here. My fantastic former housemate Kate Martin has the unique distinction of making even situps seem fun. Chris Rycroft was a fantastic running partner and always willing to trade math help for scones. I've also had the fortunate to cross paths on both coasts and both sides of the Atlantic pond with Marc Hesse since the wine and cheese night when we both first started MIT. There are also many other wonderful individuals I've been delighted to meet and spend time with during my time as a graduate student.

Most of the emotional burden of me completing a PhD has fallen on only a few unlucky souls and I count myself incredibly lucky to have had the willing ear and support of two of my closest friends, Sarah Stewart Johnson and Rebecca Hendrickson. Both have always inspired me with the joy, grace and meaning with which they conduct their lives.

My beloved sister Amelia has been supportive throughout this process and her dry librarian wit has brought levity to many a situation. My parents have always been almost unreasonably supportive of me throughout my life. I strongly credit their consistent stance that it is better to try and fail than fail to try, with their simultaneous belief that my sister and I are capable of the world, to all that I have accomplished.

Most of all, to my fiance Raffi Krikorian. For always giving me stars, in so many ways.

To my parents and sister,
for always providing me with a home, no matter our location,
and for their unwavering love and support.

Contents

List of Figures

List of Tables

The central problem of our age is how to act decisively in the absence of certainty.

Bertrand Russell

1 Introduction

Consider driving on the freeway, trying to reach some far off destination. The state of the environment is large and high-dimensional: the driver must consider the current location of her own car, the location of all the other cars, the weather, if the other car drivers are talking on their cell phones, the time of day, the speed limit, and a number of other variables (or dimensions) each of which can take on a large number of values. At each step the world changes in a stochastic manner: even if the driver knows her own car's behavior, it is likely she can only approximately predict how all the other drivers are going to behave, and it is equally tricky to predict future weather changes, or traffic accidents that may have a significant impact on how the world changes between time steps. In addition, the driver cannot simply teleport to her final destination: she must make a series of decisions in order to try to reach her destination as quickly as possible without colliding with any other vehicles.

This thesis will focus on algorithms to autonomously make a sequence of decisions in such large, high-dimensional, uncertain domains. In particular the work will consider two decision making scenarios. In the first case, imagine that the driver is learning how to drive, or has recently moved to a new city and so is learning the standard traffic behavior of a new area. Imagine also that this learner is equipped with a GPS unit, has the radio on to hear of upcoming traffic delays, construction and weather changes, and has excellent mirrors and cameras that provide her with high quality estimates of all the surrounding cars. This driver is learning the best way to get home given she doesn't know exactly how her car or the dynamics of the world works, but she does have a very good estimate of the current world state at each time step. We will frame this problem as one of learning in fully observable large, high-dimensional Markov decision processes.

Now imagine our driver has passed her driver's test, or has lived in her new city long enough to become well acquainted with the local driving culture and customs. Unfortunately one day she loses her GPS unit and her radio breaks. The driver has a good model of how the dynamics of the world works but now the world is only partially observable: the driver can only gain state information through her local car sensors, such as by checking her right side mirror to determine if a car is on her right. When planning her route home, the driver must now consider taking actions specifically to try to gather information about the world state, such as checking whether it is safe to change lanes. This second problem of interest is one of planning in a large, high-dimensional partially observable Markov decision process.

In this thesis we will provide algorithms for handling both of these scenarios. Though

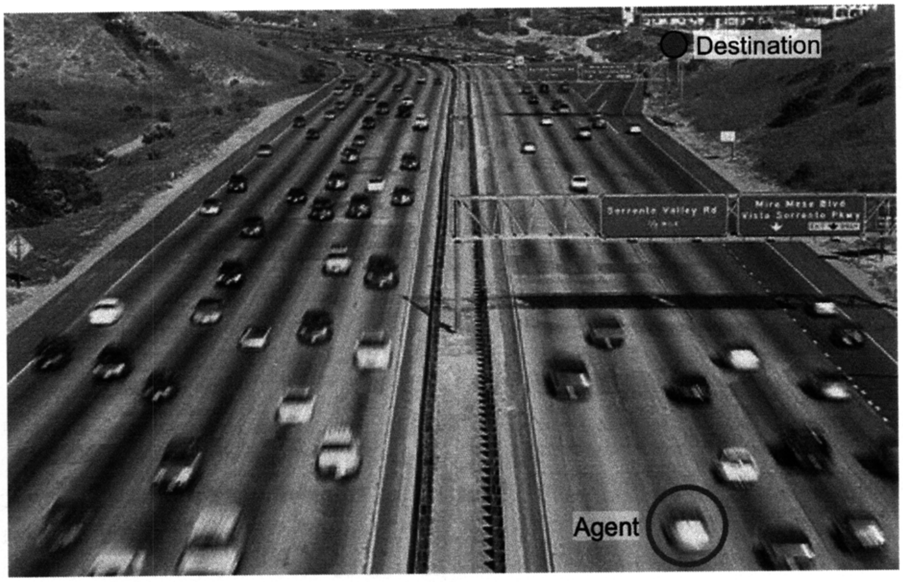


Figure 1-1: Driving to a destination: a large, high-dimensional, uncertain sequential decision making problem.

we have introduced these problems using the particular example of autonomous navigation, sequential decision making in large, high-dimensional, uncertain domains arises in many applications. For example, using robots for autonomous scientific exploration in extreme environments can require examining whether a sample is likely to be interesting, as well as collecting samples believed to be useful, and decisions made must also account for energy and communication constraints (Bresina et al., 2002; Smith, 2007). Similar issues arise in many other robotics problems, including robust motion planning in a crowded museum environment (Thrun et al., 2000) or learning new complicated motor tasks such as hitting a baseball (Peters & Schaal, 2008). Figuring out the right sequence of investment decisions (buying, selling, investing in CDs, etc.) in order to try to retire young and wealthy is another example of sequential decision making in an uncertain, very large domain. Sequential decision making under uncertainty frequently arises in medical domains, and algorithms have or are being developed for ischemic heart disease treatment (Hauskrecht & Fraser, 2000), adaptive epilepsy treatment (Guez et al., 2008) and learning drug dosing for anemia treatment (Martin-Guerrero et al., 2009). Therefore approaches for making efficiently making good decisions in high-dimensional, uncertain domains are of significant interest.

1.1 Framework

The start of modern decision theory came in the 17th century when Blaise Pascal gave a pragmatic argument for believing in God.[1] The key idea behind Pascal's wager was to

[1] *"God is, or He is not... But to which side shall we incline? Reason can decide nothing here... A game is being played... where heads or tails will turn up. What will you wager? Let us weigh the gain and the loss*

evaluate different decisions by comparing their expected benefit. Each decision can lead to one of a number of potential outcomes, and the benefit of a decision is calculated by summing over the probability of each outcome multiplied by the benefit of that outcome. The decision to be taken is the one with the highest expected benefit.

Later Richard Bellman developed the popular Markov decision process (MDP) framework that extends this idea of expected benefit to sequential decision making (Bellman, 1957). In sequential decision making a human or automated system (that we refer to as an *agent* in this document) gets to make a series of decisions. After each decision, the environment may change: a robot may hit a ball, or the driver may take a left turn. The agent must then make a new decision based on the current environment. Sequential decision making immediately offers two new complications beyond the framework illustrated by Pascal's wager. The first is how to represent the environmental changes over time in response to decisions. Bellman's MDP formulation makes the simplifying assumption that the environment changes in a Markovian way: the environmental description after an agent takes an action is a function only of the world environment immediately prior to acting (and no prior history), and the action selected. Second, sequential decision making requires a re-examination of the methodology for choosing between actions. Instead of making decisions based on their immediate expected benefit, Bellman proposed that decisions should be taken in order to maximize their long term expected benefit, which includes both the immediate benefit from the current decision, and the expected sum of future benefits. Maximizing the sum of rewards allows agents to make decisions that might be less beneficial temporarily, such as waiting to make a left hand turn, in order to achieve a larger benefit later on, such as reaching the driver's destination. To compute this quantity, Bellman introduced the idea of a value function that represents for each world state the expected sum of future benefits, assuming that, at each time step the best possible decision is made for the resulting state.

One of the limitations of Bellman's approach is that it both assumes that the world models, and the world state are perfectly known. Unfortunately, this is not always the case: it may be hard or impossible to specify in advance a perfect model of a robot arm's dynamics when swinging a baseball bat, an investor may not be able to predict a coming subprime crisis, and a driver may only be able to check one of her mirrors at a time to view surrounding traffic. Problems that exhibit model uncertainty form part of the field of Reinforcement Learning (Sutton & Barto, 1998). Sequential decision making within reinforcement learning is often referred to simply as "learning" since the algorithm has to learn about how the world operates.

In some situations the world state is only accessible through an observation function. For example, a driver with a broken radio and no GPS unit must rely on local observations, such as checking her right mirror, in order to gain partial information about the world state. In order to represent such domains in which the agent receives information about the world state solely through observations, the Markov Decision Process formalism has been extended to the Partially Observable MDP (POMDP) framework (Sondik, 1971). Within the POMDP framework, the history of observations received and decisions made is often

in wagering that God is. Let us estimate these two chances. If you gain, you gain all; if you lose, you lose nothing. Wager, then, without hesitation that He is." Blaise Pascal, Pensées

	Models Known, State Known	Models Unknown and/or State Partially Observed
Single Decision	Pascal's Wager	
Multiple Decisions	Markov Decision Processes	Reinforcement Learning, POMDPs

Table 1.1: Overview of some types of decision making under uncertainty problems.

summarized into a probability distribution which expresses the probability of the agent currently being in any of the world states. Since the true world state is unknown, instead this distribution over states is used to select a particular decision, or action, to take at each time step. As in MDPs, we will again be interested in computing a strategy of mapping distributions to decisions, known as a policy, in order to maximize the expected discounted sum of future benefits. Within the POMDP setting we will focus on computing policies when the world models are known: this process is typically known as POMDP planning. Table 1.1 illustrates some of the relationship between different types of decision making problems.

One of the key challenges of sequential decision making in environments with model or sensor uncertainty is balancing between decisions that will yield more information about the world against decisions that are expected to lead to high expected benefit given the current estimates of the world state or models. An example of an information-gathering decision is when a driver checks her blind spot or when a robot arm takes practice air swings to better model the dynamics. In contrast, actions or decisions such as when the driver changes lanes to move towards her destination, or when the robot arm takes a swing at the baseball, are expected to yield a direct benefit in terms of achieved reward. This tradeoff greatly increases the challenge of producing good solution techniques for making decisions.

1.2 Optimality and Tractability

When developing an algorithm for sequential decision making, a key consideration is the evaluation objective: what are the desired characteristics of the final algorithm? As a coarse generalization, the field of sequential decision making under uncertainty within the artificial intelligence community has been predominantly concerned with optimality, tractability and expressive power.

Expressive power refers to the type of dynamics and reward models it is possible to encode using a particular representation. For example, representing the world state by discretizing each variable into a large number of bins, and representing the full state by the cross product of each variable's current bin, is a very expressive model because with a fine enough discretization it can represent almost any possible dynamics or reward model. In contrast, a model which represents the world dynamics as a linear Gaussian, has lower expressive power because it can only represent well a subset of world models, namely those in which the dynamics are linear Gaussian.

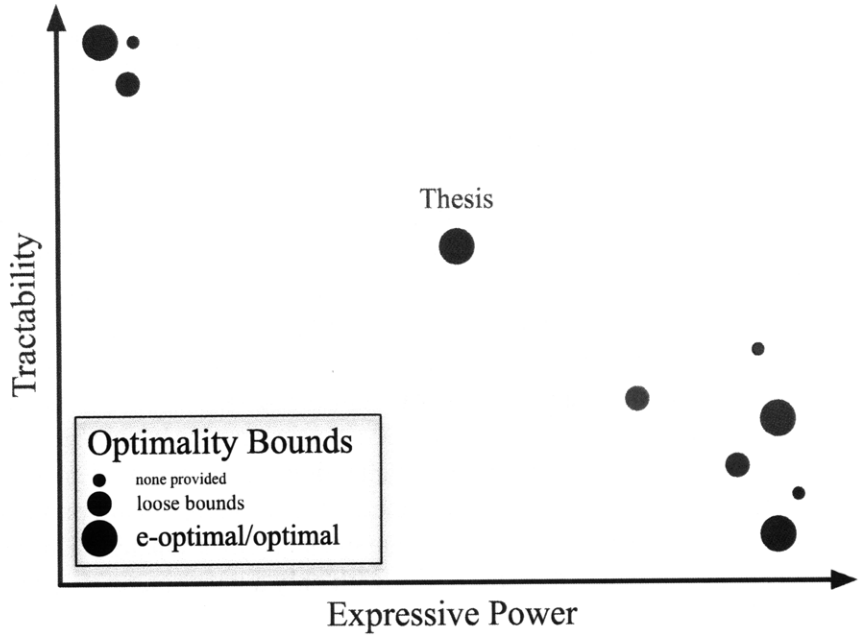


Figure 1-2: The three criteria of interest in this document for evaluating sequential decision making under uncertainty algorithms: tractability, optimality and expressive power. Most prior approaches tend to either have large expressive power, but low tractability in large, high-dimensional environments, or good tractability but low expressive power. The thesis provides algorithms with provable guarantees on their optimality which lie in the middle ground of this spectrum between tractability and expressive power.

Typically, one objective comes at the sacrifice of the other, as illustrated graphically in Figure 1-2. The computational cost required to exactly solve a generic discrete-state POMDP problem is computationally intractable for the driving example and other high-dimensional, large domains since the number of discrete states typically increases exponentially with the number of dimensions. However, a discrete-state representation may have more expressive power than is necessary to represent the regularities in many large, high-dimensional environments, including the driving example. The linear quadratic Gaussian controller (Burl, 1998) achieves high tractability by assuming an identical dynamics model across the state space, but this may be too limited to express some domains of interest, such as navigation over varying terrain.

It remains an open challenge to create tractable algorithms that make provably good decisions in a large number of high-dimensional, large domains. Tractability is important for both learning and planning: if the world models are unknown, the amount of data required to achieve good performance should scale gracefully as the state dimensionality increases. The computational complexity of selecting the right action given known world models (planning) should also scale well in high-dimensional state spaces. We also seek optimality guarantees on the performance of these algorithms. Finally, the representations used should be expressive enough to adequately capture a variety of environments. Prior approaches are not tractable in high-dimensional, large state spaces, do not provide optimality analysis, or are too limited in their expressive power to generalize to some of our problems of interest.

This thesis presents progress towards these goals by using compact parametric functions to represent the world models. Compact parametric functions allow values to be specified over large or infinite domains with a small number of parameters [2] by making assumptions about the particular shape of the function. Representing world models using compact parametric functions offers three advantages over prior model representations. First, restricting the expressive power of the representation by limiting the number of function parameters leads to faster generalization during learning: intuitively if there are fewer parameters to learn, then the amount of data required to learn them is reduced. This makes learning in high-dimensional environments more tractable. Second, in certain domains restricting the number of model parameters can reduce the computational complexity of planning since there are less parameters to manipulate and update. Third, hierarchical combinations of compact parametric models, such as switching state models (Ghahramani & Hinton, 2000), provide a significant increase in expressive power with only a linear increase in the number of model parameters.

1.3 Thesis Statement

The central claim of this work is that compact parametric models of the world enable more efficient decision making under uncertainty in high-dimensional, large, stochastic domains. This claim will be supported through the presentation of three algorithms, proofs, and experimental results.

1.4 Contributions

The first contribution of this thesis is the Switching Mode POMDP (SM-POMDP) planner which makes decisions in large (infinite) partially-observable environments (Chapter 3). SM-POMDP generalizes a prior parametric planner by Porta et al. (2006) to a broader class of problems by using a parametric switching-state, multi-modal dynamics model. This allows SM-POMDP to obtain better solutions that Porta et al.'s parametric planner in a unmanned aerial vehicle collisions avoidance simulation, and a robot navigation simulation where the robot has faulty actuators. We also empirically demonstrate the benefit of continuous parametric models over discrete-state representations. However, in general the number of parameters required to represent the SM-POMDP models increases exponentially with the state dimension, making this approach most suitable for planning in a broad range of low-dimensional continuous-domains.

In large fully-observable high-dimensional environments, a similar parametric model can enable much more efficient learning in continuous-state environments where the

[2] Our focus will be on representations where the number of parameters required scales polynomially with the state space dimension.

world models are initially unknown. This is demonstrated in the Continuous-state Offset-dynamics Reinforcement Learning (CORL) algorithm (Chapter 4). We represent the dynamics using a switching, unimodal parametric model that requires a number of parameters that scales polynomially with the state dimension. CORL builds good estimates of the world models, and consequently make better decisions, with less data: CORL makes ϵ-close to optimal decisions on all but a number of samples that is a polynomial function of the number of state dimensions. CORL learns much more quickly in high-dimensional domains, in contrast to several related approaches that require a number of samples that scales exponentially with state dimension (Brafman & Tennenholtz, 2002; Kearns & Singh, 2002; Leffler et al., 2007b). CORL also has the expressive power to handle a much broader class of domains than prior approaches (e.g. Strehl and Littman (2008)) with similar sample complexity guarantees. Our chosen typed parametric model allows CORL to simultaneously achieve good expressive power and high learning tractability. We show the typed noisy-offset dynamics representation is sufficient to achieve good performance on a robot navigation across varying terrain experiment, and on a simulated driving task using a dynamics model built from real car tracking data.

In the final contribution of the thesis, we consider how to plan in large and high-dimensional, partially observed environments assuming the world models are known, and present the Low Order Parametric Forward Search POMDP (LOP-POMDP) algorithm (Chapter 5). Forward search planning is a powerful method that avoids explicitly representing the value function over the full state space (see e.g. Ross et al. 2008a). However, prior methods have typically employed representations that cause the computational complexity of the forward search operators to be an exponential function of the number of state dimensions. In LOP-POMDP we use low order parametric models to significantly reduce the computational complexity of the forward search operators to a polynomial function of the state dimension. We present several example parametric representations that fulfill this criteria, and also present a theoretical analysis of the optimality of the resulting policy, as well as the total computational complexity of LOP-POMDP. We demonstrate the strength of LOP-POMDP on a simulated driving problem. In contrast to past approaches to traffic driving (McCallum, 1995; Abbeel & Ng, 2004), this problem is formulated as a global partially-observable planning problem, where the agent must perform local collision avoidance and navigation in order to reach a goal destination. A naïve discretization of this problem would yield over 10^{21} discrete states, which is many orders of magnitude larger than problems that can be solved by offline discrete-state POMDP planners.

Together these three contributions demonstrate that compact parametric representations can enable more efficient decision making in high dimensional domains and also support performance guarantees on the decisions made. These claims are empirically demonstrated on a large driving simulation and robot navigation task.

1.5 Organization

Chapter 2 provides an overview of the related background and prior research. Chapter 3 presents the SM-POMDP algorithm, and Chapter 4 presents the CORL algorithm. The LOP-POMDP algorithm is discussed in Chapter 5. To conclude, Chapter 6 outlines future work, and summarizes the thesis contributions.

Uncertainty and expectation are the joys of life.
William Congreve

2 Background

Sequential decision making under uncertainty has a rich history in economics, operations research, robotics and computer science. The work of this thesis takes as its starting point the artificial intelligence view of sequential decision making under uncertainty, where the main objective is to create algorithms that allow an agent to autonmously make decisions in uncertain environments.

When describing a decision making process, a key component is modeling how the environment (or *state*) changes over time and in response to actions. Throughout this document we wil make the common assumption that the way the world state changes at each time step is Markov: namely that the probability of transitioning to the next world state is strictly a function of the current state of the environment and the action chosen, and any prior history to the current state is irrelevant:

$$p(s_{t+1}|s_0, a_0, s_1, a_1, \ldots, s_t, a_t) = p(s_{t+1}|s_t, a_t).$$

Figure 2 depicts a graphical model for a Markov transition model.

Decision-making problems which are Markov are know as Markov decision processes (MDPs) and will be discussed shortly. This thesis will focus on two particular problems with the MDP framework. The first problem is that of planning when the models are unknown. This problem arises in a number of applications, including the example presented in the introduction of the driver just learning how to drive, or the driver who has recently moved to a new city and is learning the local traffic customs and patterns. This problem is known as reinforcement learning (RL) since the autonomous agent must, implicitly or explicitly, learn how the world works while simultaneously trying to determine the best way to act to gather reward. In this case the state is assumed to be fully observable. The other contributions of this thesis will consider planning (or control) algorithms for when the models are known but the state is only partially observable, such as when an experienced driver has lost her GPS unit and has a broken radio, and so must therefore rely on local sensing, such as checking her rear-view mirror, in order to estimate the world state.

We will first introduce Markov decision processes formally, and outline a few techniques for solving them: such solution algorithms often constitute a subpart of RL or POMDP planning algorithms. We will then discuss reinforcement learning before moving on to the POMDP framework and POMDP planning techniques. We do not attempt to cover the vast array of work on sequential decision making under uncertainty, and we will focus our attention on the most relevant related research, while trying to provide enough

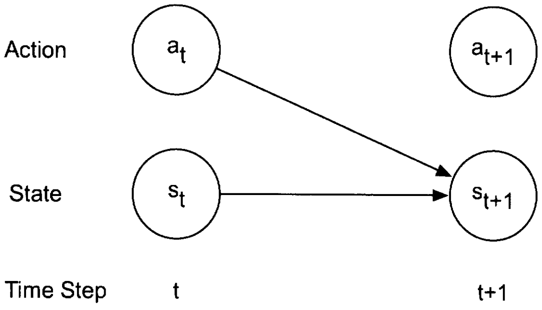


Figure 2-1: Markov transition model

context to center our own approaches.

2.1 Markov Decision Processes

Markov decision processes were first introduced by Richard Bellman (1957) and have since been used in a variety of applications, including finance (Schäl, 2002), manufacturing processes, and communications networks (Altman, 2002). In this framework, we assume the existence of an agent that is trying to make decisions, which will be referred to as actions. Formally, a discrete-time[1] MDP is a tuple $\langle S, A, T, R, \gamma \rangle$ where

- S: a set of states s which describe the current state of the world
- A: a set of actions a
- $p(s'|s,a) : S \times A \times S \rightarrow \mathbb{R}$ is a transition (dynamics) model which describes the probability of transitioning to a new state s' given that the world started in state s and the agent took action a
- $R : S \times A \times S \rightarrow \mathbb{R}$ is a reward function which describes the reward $r(s, a, s')$ the agent receives for taking action a from state s and transitioning to state s'. In many scenarios the reward will only be a function of s, a or simply the current state s.
- $\gamma \in [0, 1]$: the discount factor which determines how to weight immediate reward versus reward that will be received at a later time step.

The objective of MDP planning is to produce a policy $\pi : S \rightarrow A$ which is a mapping of states to actions in order to maximize a function of the reward. One very common objective (Sutton & Barto, 1998) is to compute a policy that maximizes the expected discounted sum of future rewards over a time horizon M (which could be infinite):

$$\arg\max_{\pi} E\left[\sum_{t=0}^{M-1} \gamma^t R(s^{t+1}, \pi(s^t), s^t)|s^0\right] \tag{2.1}$$

where the expectation is taken over all potential subsequent states $\{s^1, \ldots, s^{M-1}\}$ assuming the agent started in an initial state s^0. The value of a particular policy π for a particular state

[1] All of the new work presented in this thesis will focus on the discrete time setting and all algorithms should be assumed to be discrete-time algorithms unless specifically stated otherwise.

s, $V^{\pi}(s)$, is the expected sum of discounted future rewards that will be received by starting in state s and following policy π for M steps. In some situations we are interested in the infinite horizon performance ($M = \infty$): the expected sum of discounted future rewards is still finite for infinite horizons as long as the discount factor γ is less than 1.

The value of a particular policy can be calculated using the Bellman equation which separates the value of a state into an expectation of the immediate received reward at the current timestep, and the sum of future rewards:

$$V^{\pi}(s) = \int_{s'} p(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}(s')]ds'. \tag{2.2}$$

If the reward model is only a function of the current state and action selected ($r(s,a,s') = r(s,a)$) then the Bellman equation becomes

$$V^{\pi}(s) = R(s,\pi s) + \gamma \int_{s'} p(s'|s,\pi(s))V^{\pi}(s')ds'. \tag{2.3}$$

If the number of states is discrete and finite, then writing down the Bellman equation for each state gives a set of $|S|$ (the cardinality of S) linear equations which can be analytically solved to compute the values $V^{\pi}(s)$ for all $s \in S$. This procedure is known as policy evaluation (Howard, 1960).

Typically we will be interested in finding the optimal policy, rather than only evaluating the value of a particular policy. One common procedure to find the optimal policy is policy iteration. This procedure consists of alternating between evaluating the current policy π using policy evaluation, and improving upon the current policy to generate a new policy. Policy improvement consists of finding the best action for each state given that the agent takes any action on the first step, and then follows the current policy π for all future steps:

$$\pi'(s) = \arg\max_{a} \int_{s'} p(s'|a,s)(R(s,a,s') + \gamma V^{\pi}(s')]ds'. \tag{2.4}$$

Policy evaluation and policy improvement continue until the policy has converged. The resulting policy has been proved to be optimal (Howard, 1960).

Another popular procedure for computing the optimal policy in an MDP is value iteration. Value iteration focuses directly on trying to estimate the optimal value of each state in an iterative fashion, where at each step the previous value function V is used to create a new value function V' using the Bellman operator:

$$\forall s V'(s) = \max_{a} \int_{s'} p(s'|a,s)(R(s,a,s') + \gamma V(s'))ds'. \tag{2.5}$$

Due to the max over actions, this is not a set of linear equations, and therefore value iteration consists of repeatedly computing this equation for all states until the value of each state has stopped changing. The Bellman operator is a contraction and therefore this procedure is guaranteed to converge to a fixed point (Banach's Fixed-Point Theorem, presented as Theorem 6.2.3 in Puterman (1994)). Briefly, the Bellman operator can be seen to be a contraction by considering any two value functions V_1 and V_2 for the same MDP and denoting

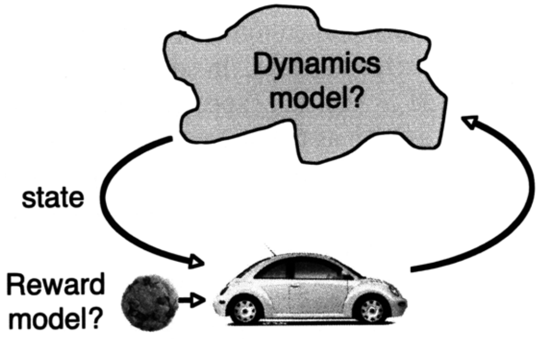


Figure 2-2: Reinforcement Learning. An agent must learn to act to optimize its expected sum of discounted future rewards without prior knowledge of the world models.

the new value function after the Bellman operator is applied as V_1' and V_2' respectively. Let $||f||_\infty$ denote the max-norm of a function f. Then

$$\begin{aligned} ||V_1' - V_2'||_\infty &= ||\int_{s'} p(s'|a,s)(R(s,a,s') + \gamma V_1(s'))ds' - \int_{s'} p(s'|a,s)(R(s,a,s') + \gamma V_2(s'))ds'||_\infty \\ &= ||\gamma \int_{s'} p(s'|s,a)(V_1(s') - V_2(s'))ds'||_\infty \\ &\leq \gamma ||V_1 - V_2|| \end{aligned}$$

which since $\gamma \in [0,1]$ immediately proves that the Bellman operator is a contraction.

The value function at the fixed point of value iteration is guaranteed to be the optimal value function V^* (Puterman, 1994). The optimal policy can be extracted from the optimal value function by performing one more backup:

$$\pi^*(s) = \operatorname*{argmax}_a \int_{s'} p(s'|a,s)(R(s,a,s') + \gamma V^*(s')]ds'. \tag{2.6}$$

Both policy iteration and value iteration involve operations with a computational complexity that scales polynomially as a function of the size of the state space ($O(|S|^2|A|)$). While these approaches work well in discrete-state MDPs with a small number of states, they do not scale well to large or infinite state spaces, such as the driving domain discussed in the introduction. There do exist special cases with analytic solutions, such as the famous linear-quadratic-gaussian (LQG) controller for linear Gaussian dynamics and quadratic rewards (Burl, 1998), However, the general case of continuous-state MDPs is an active area of research in its own right, and is known to be provably hard (Chow & Tsitsiklis, 1989). Some of the recent advances in this area include work by Kocsis and Szepesvári(2006), Kveton and Hauskretcht (2006), and Marecki and Tambe (2008).

2.2 Reinforcement Learning

In reinforcement learning an agent must learn how to act in a MDP without prior knowledge of the world reward and dynamics models (see Figure 2-2). One of the central challenges in reinforcement learning is how to balance between *exploration* actions that improve the agent's (implicit or explicit) estimates of the world models, with *exploitation* actions that are expected to yield high reward given the agent's current estimates of the world models. An agent that exploits too early may make decisions based on estimated world models that are quite different than the true underlying models, and therefore the agent may make suboptimal decisions. On the other hand, an agent that explores forever will never get to use its knowledge to actually select actions that are likely to yield a large sum of rewards, which is the ultimate goal of the agent.

Before we discuss several different approaches to handling exploration and exploitation in reinforcement learning, it is worth noting that there has recently been some interesting developments in the related field of apprenticeship learning (see eg. Abbeel and Ng (2004; 2005)) which take the slightly different approach of assuming that an expert has provided some training examples which an agent can learn from. This approach has had some impressive experimental success on helicopter flying and quadruped robot locomotion, and is a promising approach for when some expert training samples are available. In our work we will assume the samples are only generated by the learning agent and so our work falls within the standard reinforcement learning framework.

There are three broad theoretical approaches to handling the exploration/exploitation tradeoff which we now enumerate:

1. *Infinite or No Guarantee RL*: Place no guarantees on the optimality of the actions selected, or only guarantee that the actions selected as the number of samples grows to infinity will be optimal.

2. *Optimal RL*: Optimally balance exploration and exploitation from the first action chosen and guarantee that in expectation the rewards achieved will be the highest possible in the particular environment given the models are initially uncertain.

3. *Probably Approximately Correct RL*: Allow a finite number of exploratory actions during which the action selected may have arbitrarily bad reward, but require that the actions selected at all other time steps have a value which is ϵ-close to the value of the optimal action with high probability.

Naturally the second objective seems the most appealing, but there are some significant challenges that mean that it is generally intractable for most domains of interest. We now discuss each approach in more depth.

2.2.1 Infinite or No Guarantee Reinforcement Learning

Many reinforcement learning techniques offer no optimality guarantees on their performance, or only guarantee that under certain conditions, the actions chosen will be optimal as the number of time steps grows to infinity. For example, a simple popular approach

Algorithm 1 Q-learning

```
Initialize all Q values
Set state $s$ to be drawn from starting state distribution
loop
  Generate $r$, a random number between 0 and 1
  if $r \leq \epsilon$ then
    $a$ = RandomAction
  else
    $a$ = $\mathrm{argmax}_a \, Q(s, a)$
  end if
  Take action $a$, receive reward $r$, transition to new state $s'$
  Update the state action value $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + \mathrm{argmax}_{a'} \, \gamma Q(s', a'))$
  Update $\alpha$ and $\epsilon$
  $s = s'$
end loop
```

is the Q-learning algorithm (Watkins, 1989). In finite state and action spaces, Q-learning typically uses a tabular representation over the estimated values of each state-action pair. After each transition (s, a, s') Q-learning updates the corresponding entry for state s from which the action a was taken as follows

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R + \gamma \max_{a'} Q(s', a)) \tag{2.7}$$

where R is the reward received after taking action a from state s and transitioning to state s', and α is a learning parameter which controls how much to alter the current state-action estimates. These estimated Q values are used to select actions at each time step. There are a number of action-selection methods possible, but a popular approach is known as ϵ-greedy: at each time step the agent selects the action which maximizes $Q(s, a)$ with probability $1 - \epsilon$ and otherwise selects an action at random. Algorithm 1 displays this version of Q-learning. If ϵ is decayed over time such that there is always some probability of exploration, and the learning parameter α is reduced by no larger than a factor of $1/t^2$ as the number of time steps t increases, then Q-learning is guaranteed to converge to the optimal policy as the number of samples grows to infinity.

Though appealing in its simplicity and computational speed per update, Q-learning typically exhibits extremely slow convergence to a good policy, in large part because at each step only a single state-action pair is updated. For example, if an agent finally found a location with an enormous cookie, only the prior state would have its value updated to reflect the benefit of that action: all other states leading up to the cookie would still have their original values. It can therefore take a long time for changes in the state-action values to propagate back to precursor states.

In addition, when the number of states is extremely large or infinite, such as in the driving example, the standard tabular representation of the state-action values will not be possible. The state space can be discretized to an arbitrary resolution and standard Q-learning can be applied to this discrete representation. Such an approach has a high degree of expressive power, since any potential dynamics or reward model can be represented with an arbitrary accuracy if a sufficiently fine grid resolution is employed. The resolution used does not affect the per-step computational cost in Q-learning, since Q-learning only updates a single state-action tuple at each time step. In this sense Q-learning is quite computationally tractable for large environments. However its slow convergence becomes even more exaggerated as the state space gets larger, making its learning time intractable in large environments.

A common alternate approach to discretization in large environments is to use some type of function approximation in order to estimate the value function over the large or continuous state space. Sutton and Barton (1998) provide an overview of some of these approaches. The majority of these approaches provide no formal guarantees on the optimality of the actions selected. Indeed, even in the limit of an infinite number of samples, some of these methods will not even converge to a single policy, as demonstrated by Baird (1995) and Boyan and Moore (1995). Figure 2-3 presents an early example from Boyan and Moore that demonstrates the divergence of the value function on a (now) standard reinforcement learning problem, where linear weighted regression is used as a function approximator. However, Gordon (1995) proved that if the function approximator is an averager, which means that the distance between two approximated points is always smaller than the original distance between the two points, then reinforcement learning using that function approximation is guaranteed to converge. In parallel Tsitsiklis and Van Roy (1996) proved similar results about function approximators satisfying what they called interpolative representations. Note that these results guarantee that using these function approximators during reinforcement learning will allow the computed policy to eventually converge, but do not guarantee the resulting policy is optimal.

Most more recent RL approaches are guaranteed to converge, but without necessarily providing guarantees on the resulting optimality of the solution. This is often an issue with function approximation techniques since the particular function used to approximate the value function constrains the space of representable value functions. If the true optimal value function does not lie in the class of functions used to approximate the value function, it is hard to estimate the resulting error. However, a number of these approaches have good empirical performance and are applicable to a large number of domains (they have high expressive power). For example, Least Squares Policy Iteration (Lagoudakis & Parr, 2003) has good empirical performance but no theoretical guarantees on the policy optimality. There has also been some interesting work in using Gaussian Processes in continuous-state and action reinforcement learning (Engel et al., 2003; Engel et al., 2005; Rasmussen & Kuss, 2004; Deisenroth et al., 2008). One of the interesting advantages of Gaussian Processes is that they automatically encode the degree of uncertainty on the resulting value function estimates, which can be potentially used when selecting actions during exploration. Gaussian Processes are a nonparametric approach and as such have a high degree of expressive power, since they can use represent any function output by the sampled data. However, in general they require that all the data is stored, which means their computational tractability in large domains with long horizons is limited. They also do not provide theoretical guarantees on the resulting actions selected.

Jong and Stone (2007) recently presented Fitted R-max, an approach that can learn in continuous environments which assumes that the dynamics model between nearby states

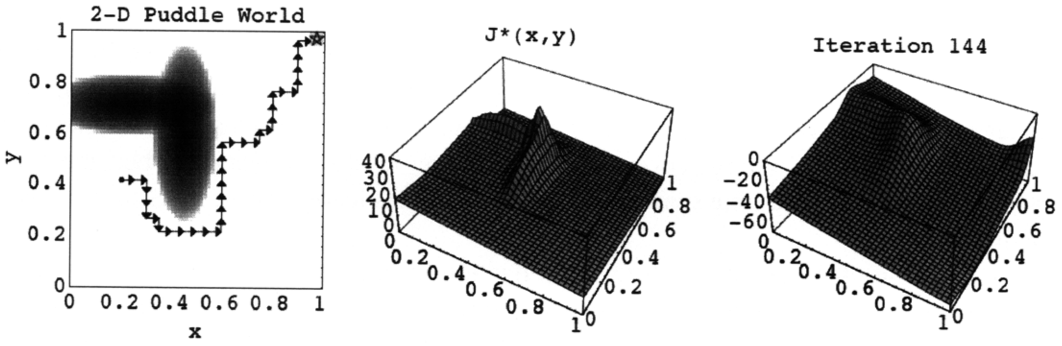


Figure 2-3: *(Reproduced with permission from Boyan and Moore (1995), figure 4).* An example where a standard function approximator, local weighted regression, combined with dynamic programming on a continuous-state problem, causes the value function to diverge. The state space is shown on the left: it is a navigational problem where an agent must learn to avoid the oval puddles and travel to a goal in the right upper corner. The middle figure shows the optimal value function and the right figure shows the approximate value function after 144 iterations. One of the interesting points about this example is that although linear weighted regression (LWR) can be used to provide a very good approximation to the optimal value function, if LWR is used to approximate the value function during value iteration, it causes the value function to diverge.

is similar, This allows Fitted R-max to learn more quickly in continuous environments by explicitly assuming a certain amount of generalization between states. Their experimental results were encouraging in low-dimensional environments. However, the amount of samples needed for learning would typically scale exponentially with the state-space dimension since Fitted R-max is an instance-based approach. Therefore, while their approach can represent a large class of problems, meaning that it has a high degree of expressive power, it is not computationally tractable in large domains, and it does not provide formal optimality guarantees.

Recent work by Peters and Schaal (2008) uses a policy gradient approach in which the policy is slowly modified until a local optimum is reached. Their approach works well for allowing a seven degree of freedom robot to learn how to hit a baseball, an impressive demonstration of tractability. However the approach is not guaranteed to find a global optimum.

2.2.2 Optimal Reinforcement Learning

The second potential objective in balancing exploration and exploitation is to attempt to optimally balance between the two from the first action. Most such approaches frame reinforcement learning within a Bayesian perspective, and consider a distribution over possible model parameters. As the agent acts, receives rewards, and transitions to new states, the distribution over model parameters is updated to reflect the experience gathered in the world. When selecting actions, the agent considers the distribution over model parameters, and selects actions that are expected to yield high reward given this distribution: such actions could either be information gathering/exploratory or exploitative in nature. In particular, Duff (2002) and a number of other researchers have framed the problem as a partially observable Markov decision process (POMDP) planning problem. We will discuss POMDP planning in detail later in this chapter, but in this context the model dynamics and reward parameters are considered to be hidden states, and the agent maintains a belief over the distribution of potential values of these hidden parameters. In Bayesian RL the bulk of the computational effort is to compute a plan for the POMDP, typically in advance of acting in the real world. This plan essentially specifies the best action to take for any distribution over model parameters, or belief, the agent might encounter. When the agent is acting in the real world, the agent must simply update these distributions (known as belief updating) at each time step to reflect the transition and reward it just experienced, and then use these distributions to index the correct action using the previously computed plan.

One of the challenges in this style of approach is that even if the states in the MDP are discrete, the transition parameters will typically be continuous-valued. Poupart et al. (2006) show how the belief over the hidden model parameters can be represented as a product of Dirichlets: the Dirichlet distribution is the conjugate prior for the multinomial distribution. The authors then prove that the POMDP planning operators can be performed in closed form, and that the value function is represented as a set of multivariate polynomials. The number of parameters needed to represent the value function increases during planning, and so the authors project the updated value function to a fixed set of bases iteratively during the computation. Rather than re-projecting the value at each step, the authors transform the problem to a new altered POMDP over a different set of bases. This greatly reduces the computational load, which allows BEETLE to scale to a handwashing task motivated by a real application, highlighting its computational tractability and expressive power. However, it is unclear how much error is caused by the bases approximation or how to select bases to reduce the approximation error, and the final algorithm does not have bounds on the resulting actions selected.

Castro and Precup (2007) also assume a fully observed discrete state space with hidden model parameters but represented the model parameters as counts over the different transitions and reward received, thereby keeping the problem fully observable. Doshi et al. (2008) consider a Bayesian approach for learning when the discrete state space is only partially observable. Both of these approaches have good expressive power but will only be tractable in small environments. Also, both are approximate approaches, without formal optimality guarantees on the resulting solution.

Both Wang et al. (2005) and Ross et al. (2008a) present Bayesian forward-search planners for reinforcement learning (RL) problems. Despite the useful contributions of Wang et al. and Ross et al., in both cases the work is unlikely to be computationally tractable for large high-dimensional domains. For example, Ross et al. used a particle filter which has good performance in large, low-dimensional environments, but typically requires a number of particles that scales exponentially with the domain dimension. Like the prior Bayesian approaches outlined, the work of Wang et al. and Ross et al. make few assumptions on the structure of the underlying model structure (so they have good expressive power) but they do not provide guarantees on the optimality of the resulting solutions, and it is unclear how well these approaches will scale to much larger environments.

All the approaches outlined face inherent complexity problems and produce algorithms that only approximately solve their target optimality criteria. Therefore, while the Bayesian approach offers the most compelling objective criteria, to our knowledge there do not yet exist algorithms that succeed in achieving this optimal criteria.

2.2.3 Probably Approximately Correct RL

The third middle ground objective is to allow a certain number of mistakes, but to require that the policy followed at all other times is near-optimal, with high probability. This objective is known as probably approximately correct (PAC). Another very recent and slightly more general objective is the Knows What it Knows (KWIK) learning framework which can handle adversarial environments (Li et al., 2008). Due to the more extensive literature on PAC RL we will focus our discussion on the PAC RL formalism.

The tractability of PAC RL algorithms is typically designated as learning complexity, and consists of sample complexity and computational complexity (Kakade, 2003). Computational complexity refers to the number of operations required at each step in order for the agent to compute a new action. Sample complexity refers to the number of time steps at which the action chosen is not close to optimal.

PAC RL was introduced by Kearns and Singh (2002) and Brafman and Tennenholtz (2002) who created the E^3 and R-max algorithms, respectively. These algorithms were guaranteed to achieve near optimal performance on all but a small number of samples, with high probability. R-max and E^3 were developed for discrete state spaces, and their sample and computational complexity is a polynomial function of the number of discrete states. These algorithms can also be applied to continuous state spaces by discretizing the continuous values. However, the R-max and E^3 algorithms assume that the dynamics model for each state–action tuple is learned independently. Since each state–action can have entirely different dynamics, this approach has a high expressive power, but as there is no sharing of dynamics information among states, it has a very low level of generalization. This causes the algorithms to have an extremely large sample complexity in large environments, causing them to have low tractability. In contrast, the work of Strehl and Littman (2008) and the classic linear quadratic Gaussian regulator model (see Burl, 1998) assume that the dynamics model is the same for all states, greatly restricting the expressive power of these models in return for more tractable (fast) learning in large environments.

Kearns and Koller (1999) extended E^3 to factored domains and proved that when the dynamics model can be represented a dynamic Bayesian network (DBN), the number of samples needed to achieve good performance is only a polynomial function of the number of DBN parameters. Their approach slightly limits the expressive power of the E^3 algorithm to a smaller subclass of dynamics models, but makes a significant improvement in tractability, while still providing formal theoretical guarantees on the quality of the resulting solution. However, the authors assume access to an approximately optimal planner for solving the resulting MDP but do not provide a particular planner that fulfills their assumptions. Though the algorithm has attractive properties in theory, no experimental results were provided.

Another approach that explores the middle ground between expressive power and (sample complexity) tractability is Leffler et al. (2007b)'s work on domains in which the dis-

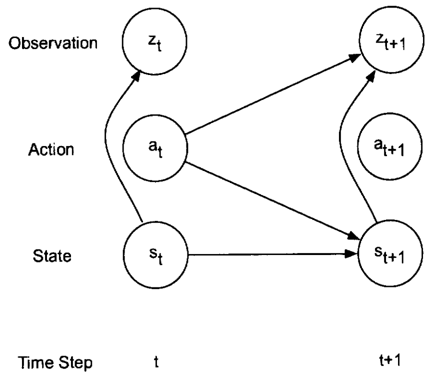


Figure 2-4: POMDP transition and observation graphical model

crete state space is divided into a set of types. States within the same type are assumed to have the same dynamics. The authors proved that a typed representation can require significantly less experience to achieve good performance compared to a standard R-max algorithm that learns each state–action dynamics model separately. Our work in Chapter 4 will demonstrate how using parametric representations can significantly improve the sample complexity of typed representations.

One of the appealing aspects of the PAC objective over a Bayesian RL approach is that it allows precise statements about the efficiency and performance of algorithms: if the MDP or POMDP used in the Bayesian RL approach could be solved exactly with an informative prior, then this approach would likely outperform PAC-MDP approaches. However, when a Bayesian RL problem is only approximately solved or when the prior information is incorrect, it is unknown how far the resulting solution is from the optimal behavior. It is for this reason that in this thesis we will focus on this third RL objective.

We will now turn our attention to the second type of problem considered in this thesis, planning when the state is only partially observable, such as when a driver is planning a route and can only make local sensing observations.

2.3 POMDPs

Partially Observable MDPs (Sondik, 1971) are an extension of the MDP framework. A POMDP consists of a tuple $\langle S, A, T, R, Z, p(z|s', a), \gamma \rangle$ where S,A,T,R, and γ are the same as in the MDP model, and

- Z is a set of observations $z \in Z$
- $p(z|s,a) : S \times A \times Z \rightarrow \mathbb{R}$ is an observation function which encodes the probability of receiving observation z after taking action a and transitioning to state s'

Figure 2.3 show the relationship between the dynamics and observation model in POMDPs.

Unlike an MDP, an agent acting in a POMDP no longer has access to the direct state and instead must make decisions based on the prior history of observations it has received, $z_{1:t}$, and actions it has taken, $a_{1:t}$, up to the present time point t. Instead of maintaining an ever-expanding list of past observations and actions, a sufficient statistic $b_t(s)$ of the past

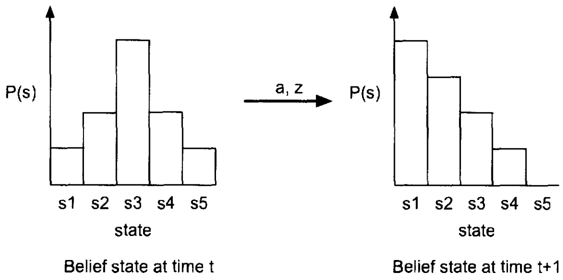


Figure 2-5: An example belief over state space of 5 states, and how it might change after taking action a and receiving observation z.

history of observations and actions is used which summarizes the probability of the world being in each state given the past history:

$$b_t(s) = p(s_t|a_0, z_1, a_1, z_2, \dots a_{t-1}, z_t).$$

This sufficient statistic b_s is known as the belief.

After each time step, a new belief b_{t+1} is constructed based on the action a_t just taken, the most recent observation z_{t+1}, and the prior belief b_t. Figure 2-5 shows an example belief state, and how it might change after the agent takes an action and receives an observation. The new belief can be calculated using a Bayesian filter (Kaelbling et al., 1998). The new probability of the world being in a particular state s' depends on how the new state s' could have been reached given the transition model for the chosen action a and the prior distribution over states b_t, weighted by the probability of s' generating the received observation z:

$$b_{t+1}(s') = \frac{p(z|s') \int_s p(s'|s,a) b_t(s) ds}{\int_{s'} p(z|s') \int_s p(s'|s,a) b_t(s) ds ds'}.$$

The denominator ensures that the new belief is a proper probability distribution:

$$\int_{s'} b_{t+1}(s') ds' = 1.$$

The goal of POMDP planning is to construct a policy π that maps prior histories of observations and actions (or beliefs) to actions ($\pi : b \rightarrow a$) in order to maximize the discounted expected sum of future rewards. In MDPs we compute the value of a state under a particular policy: here we are interested in the value of a belief under a particular policy. The value of a belief is expected sum of future rewards from starting in that belief and acting according to a given policy π from then onwards.

A policy is often found by computing the value function over the space of beliefs:

$$V(b) = \max_a \left[R(b,a) + \gamma \int_{b' \in B} p(b'|a,b) V(b') \right]. \tag{2.8}$$

Since beliefs are probability distributions, there is an infinite number of possible beliefs, which means the value function cannot be computed by enumeration.

Within the context of evaluating POMDP planners we will use tractability to refer to the computational tractability of a planning algorithm.

2.3.1 Planning in Discrete-State POMDPs

The POMDP formulation described thus far is agnostic about whether the underlying world states, actions, and observations are discrete or continuous. Starting with the important work of Sondik (1971) there has been significant research on discrete state, action and observation POMDPs. Sondik proved that the optimal finite horizon value function in this case consists of a finite set of hyperplanes over the belief space.

Before discussing the multi-step planning in discrete state, action and observation POMDPs, we will first start with a simple example of how to plan in a POMDP if the agent only gets to make a single decision. We will then show how the 1-step solution or plan can be used to construct a plan for acting when the agent gets to act for 2 time steps: where the agent first takes an action, receives an observation, and then gets to take one more action. This example will help illustrate how longer term plans can be constructed from shorter-horizon plans. A number of other authors have provided excellent introductions to discrete-state POMDP planning, including Kaelbling, Littman and Cassandra (1998), Roy (2003), and Smith (2007), and the explanation presented here draws upon these past explanations.

For ease of exposition, let us assume that the reward received at each step is a function only of the current state s and action chosen a. If there is only a single action to be made, then there exist $|A|$ potential policies, one for each possible action. The value for each policy is a $|S|$-length vector that specifies the immediate reward that would be received if that action was taken from each possible $s \in S$. For example, the value for the policy of taking action a_i is:

$$\alpha_{a_i} = \{R(s_1, a_i), R(s_2, a_i), \ldots, R(s_{|S|}, a_i)\}.$$

As a concrete example, let us consider a two state POMDP with 2 actions and 2 observations where the reward model is specified as follows:

$R(s,a)$	a_1	a_2
s_1	2	0
s_2	0	1

Here the two initial polcies, corresponding to each action, would have a value of

$$\begin{aligned} V_1 &= \{2, 0\} \\ V_2 &= \{0, 1\} \end{aligned}$$

as displayed in Figure 2-6.

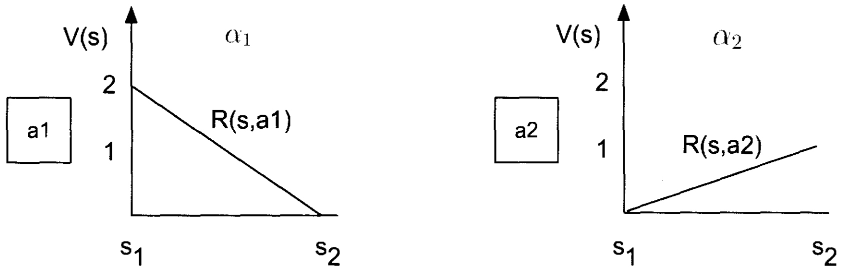


Figure 2-6: The 1-step α-vectors and policy trees for a 2-state, 2-action POMDP. Note that the policy (specified by the square) is simply a single action, and the corresponding α-vector for each policy is the vector of immediate rewards received if that action is taken, where the vector spans the state space.

If the agent knew its state, then it could simply choose the policy with the highest value for its current state. However the agent does not know its state, and instead only has access to its belief b_0. The agent must therefore choose an policy based on the expected value under each policy:

$$\operatorname*{argmax}_p \sum_s V_p(s)b(s)$$

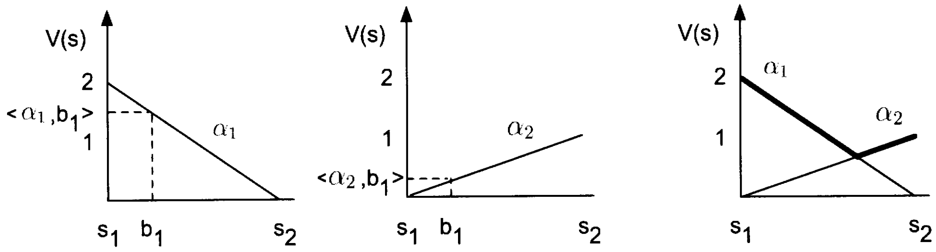
where p indexes the policy.

Following Sondik (1971) the value of each policy will be defined as an $|S|$-length α-vector: each entry $\alpha(s_i)$ of the α-vector specifies the value of following that policy if the agent is currently in state s_i.

The value function over the whole belief space for a one-step horizon consists of the supremum of the value of all policies (the supremum of all α-vectors for those 1-step policies). Figure 2-7 displays the expected value for an example belief b_1 under the two 1-step policies, as well as the the value function across the entire belief space. Note that the value function is a piecewise linear function, consisting of the supremum of a set of vectors.

We can now use the set of policies from the 1-step value function in order to construct a plan for acting when the time horizon is two steps. Note that we will only use 1-step policies whose α-vectors make up part of the supremum which represents the 1-step value function: if the α-vector for a 1-step policy does not appear in the supremum set, it indicates that for all beliefs there is at least one other policy which yields a higher expected value. Let Γ_1 be the set of all α-vectors (each associated with a particular 1-step policy) that make up part of the optimal 1-step value function.

To compute the two-step horizon value function, we first create a set of all potential two step conditional policies. A two-step conditional policy consists of an action at its



(a) $V_p(b)$ for both policies

(b) 1-step V

Figure 2-7: (a) Computes the expected value $\langle \alpha_i, b_1 \rangle$ under both 1-step horizon α-vectors for a particular belief b_1. Here α_1 yields a higher expected value. (b) The 1-step value function consists of the supremum over all 1-step α-vectors. Here the supremum is specified by thick black lines.

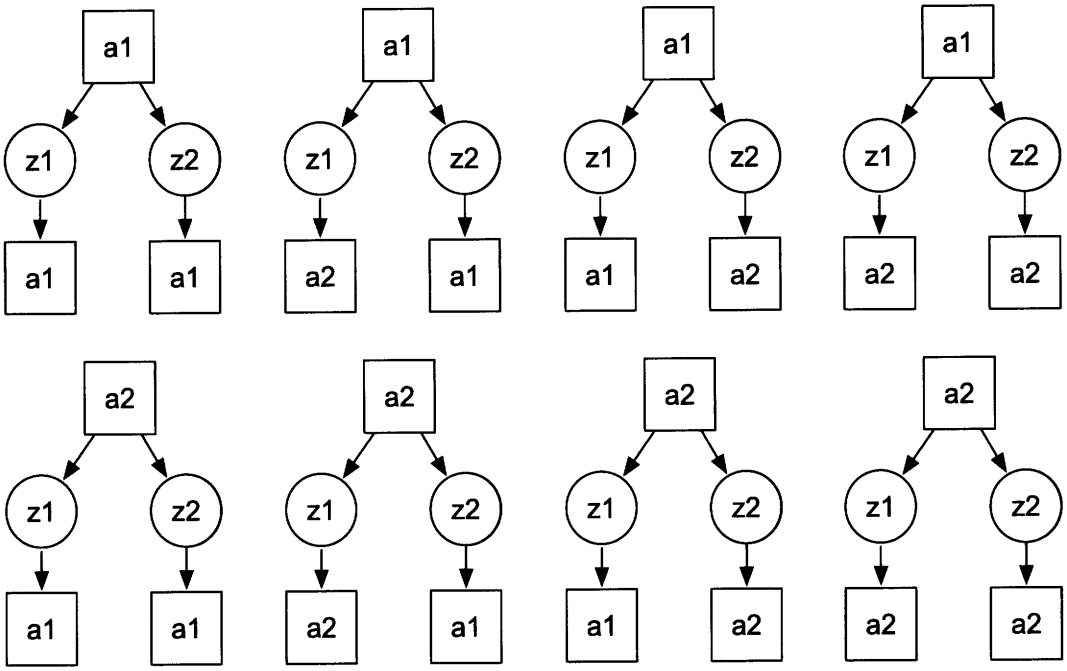


Figure 2-8: The set of potential two-step conditional policy trees for a 2-state, 2-action, 2-observation POMDP. The root node specifies the first action of the policy, and then depending on which observation the agent receives, the agent should then take the action under the corresponding observation's leaf node.

root, then branches on all possible observations. After each observation, a 1-step policy is placed at the leaf node. Figure 2-8 shows all potential 2-step conditional policy trees for this example POMDP. We now need to be able to compute the value of these new conditional policy trees, which will be represented by a new α-vector. Recall that for the 1-step horizon, an α-vector just represented the value under each state of following the particular policy associated with this vector. This explanation still holds for longer horizons: the α-vector associated with a particular conditional policy specifies, for each state s_i, the expected value of following that particular conditional policy, given the agent starts in state s_i. Using the Bellman equation, the value of a 2-step conditional policy is the immediate reward received by taking the root action a, followed by the expected reward at the next time step, given the agent took the action a. We won't know exactly what state the agent will transition to, nor what observation it will receive, and so we must take an expectation over the future state and potential observations received. Taking an expectation over observations is important because different observations will provide different evidence about the new world state: in other words the observation will affect the new belief of the agent, and therefore the expeced reward under that belief.

Consider computing the value of a 2-step conditional policy tree with action a_1 at its root where if the agent then receives observation z_1 then the agent takes action a_2, else if the agent receives observation z_2, the agent takes a_1. The value of this conditional policy tree is

$$\alpha(s) = R(s, a_1) + \gamma \sum_{s'=1}^{2} p(s'|s, a_1)p(z_1|s', a_1)\alpha_2(s') + p(s'|s, a_1)p(z_2|s', a_1)\alpha_1(s').$$

In general, the value of a particular two-step conditional policy tree will depend on the policy and associated α-vector following each observation. Let us denote the particular 1-step policy k chosen after an observation z as α_{zk}. Then we can compute the value of any two-step conditional policy tree as

$$\alpha(s) = R(s, a) + \gamma \sum_{s'} \sum_{z} p(s'|s, a)p(z|s', a)\alpha_{zk}(s')$$

where a is the action taken at the root of the policy tree.

Performing this calculation for all eight=$|A||Z|^{|A|}$ possible conditional policy trees will yield eight new α-vectors. The optimal two-step value function again consists of the supremum of these α-vectors over the belief space.

This whole process can be repeated for 3-step and longer planning horizons. The procedure of computing a new $(t + 1)$-step value function from a set of t-step conditional policies is known as value function backups. The procedure described is an exact method for computing the optimal t-step value function. In general, the number of potential conditional policies will grow in a double-exponential fashion in relation to the time horizon. However, some of these policy trees will not be optimal for any belief state: their value is strictly dominated by other policy trees. In such cases these α-vectors can be pruned without affecting optimality. As discussed by Kaelbling et al. (1998) pruning can be performed using linear programming but this is an expensive procedure and the optimal value function for an infinite-horizon POMDP may still consist of infinitely many α-vectors.

Solving a finite-state, infinite-horizon POMDP with boolean rewards is EXPTIME-hard, Madani et al. (2003) proved that in the case of general bounded rewards this problem is undecidable, and finite horizon POMDPs are known to be PSPACE-hard. All these results suggest that while the POMDP formulation is elegant and flexible, to use it in practice will require significant additional assumptions and approximations. For this reason, though there are a few exact algorithms (e.g. the Witness algorithm (Kaelbling et al., 1998)), the majority of research has focused on approximate solution techniques.

2.4 Approximate POMDP Solvers

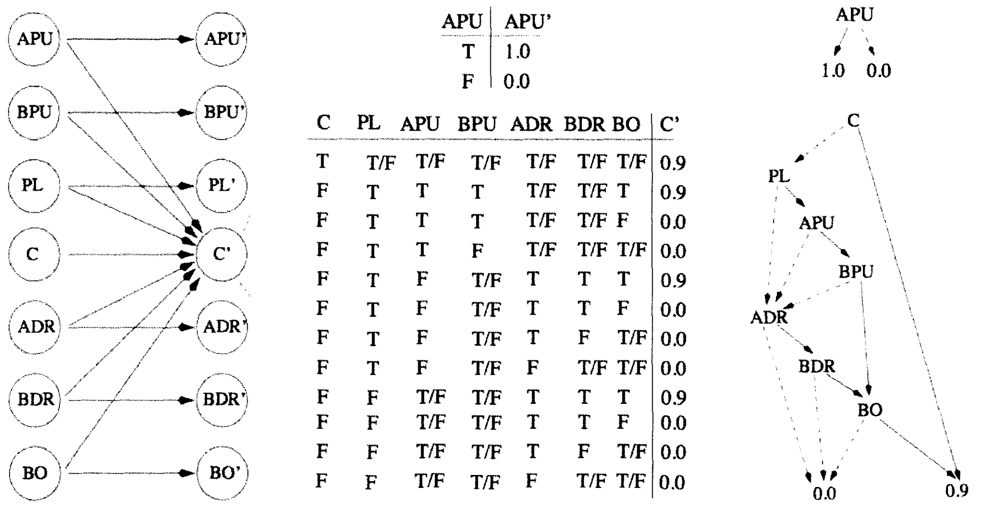
The approaches developed to solving POMDPs can be loosely categorized by two features: the structural assumptions and whether or not the majority of computation is performed before control is commenced.

There have been numerous papers that seek to leverage structural assumptions in order to achieve more computationally tractable POMDP planning. Potential structural assumptions include:

- Model structure
- Belief structure
- Value function structure
- Policy structure

An additional important potential structural assumption is hierarchical structure (Pineau et al., 2003b; Theocharous & Murphy, 2004; Foka & Trahanias, 2007).

Structural assumptions can be thought of as constraining the expressive power of the resulting algorithms, which often results in improvements in algorithmic tractability. We will organize our discussion around these different types of structural assumptions, but it is worth also pointing out that an additional important factor that impacts tractability is whether planning is performed offline or online. An offline planner computes a policy for all possible beliefs that might be encountered before commencing control. There may be a small amount of additional computation time that is required during acting, such as performing belief updating, but the policy is computed in advance. An alternative approach is to compute a policy online by determining the next action to take only for the particular history of observations received and actions selected so far. Here a more significant computational cost will be required at each time step in order to select an action, but no apriori computation is required. There are also approaches that can combine some prior offline planning with additional online computational for further policy improvement. The majority of prior work has been on offline algorithms. Since structural assumptions are largely orthogonal to whether the algorithm is an online or offline algorithm, we will first summarize approximate POMDP planners by their structural assumptions, before discussing some recent work on online POMDP planners.



APU	APU'
T	1.0
F	0.0

C	PL	APU	BPU	ADR	BDR	BO	C'
T	T/F	T/F	T/F	T/F	T/F	T/F	0.9
F	T	T	T	T/F	T/F	T	0.9
F	T	T	T	T/F	T/F	F	0.0
F	T	T	F	T/F	T/F	T/F	0.0
F	T	F	T/F	T	T	T	0.9
F	T	F	T/F	T	T	F	0.0
F	T	F	T/F	T	F	T/F	0.0
F	T	F	T/F	F	T/F	T/F	0.0
F	F	T/F	T/F	T	T	T	0.9
F	F	T/F	T/F	T	T	F	0.0
F	F	T/F	T/F	T	F	T/F	0.0
F	F	T/F	T/F	F	T/F	T/F	0.0

(a) Dynamic Bayes Net

(b) CPTs

(c) Algebraic Decision Diagrams

Figure 2-9: *(Reproduced with permission from Hoey et al. (1998), figure 2).* (a) displays the Dynamic Bayes Net (DBN) which represents the transition model for this particular domain in which a robotic agent is sent to get coffee. Note that only the C variable has a next value C' which depends on the values of all the other variables: all other variables depend only on their prior value. (b) displays the Conditional Probability Table (CPT) for the APU and C variable. CPTs encode how the next value of a variable depends on the prior values of the other variables. (c) presents an Algebraic Decision Diagram (ADD) which is equivalent to the CPTs displayed in (b). ADDs can be far more compact than CPTs when CPTs display a large amount of shared structure.

2.4.1 Model Structure

A number of well motivated problems, including a handwashing assistant for people with dementia (Boger et al., 2005), exhibit significant structure in their environmental models. By structure we refer to the fact that even if the domain is described by a large number of state variables, the transition, observation, and reward dynamics may only depend on a subset of those variables. For example, consider the transition dynamics for a factory example represented in three different ways in Figure 2-9. Subfigure (a) shows the conditional dependency relations between the various state variables at two time steps (the ' indicates a variable at the next time step). The value of variable C at the next time step depends on all variables; however, for all other variables depicted, their value at the next time step is only a function of their prior value. In Figure 2-9b the conditional probability tables to specify the next state values conditioned on the prior values are depicted, which quantify the dependency relations encoded by the dynamic Bayes net. However, it is possible to more compactly encode this table using an algebraic decision diagram structure, as depicted in Figure 2-9(c).

Initial work by Hoey and colleagues (1999) demonstrated that using algebraic decision diagrams (ADDs) to represent the dynamics and reward model of MDPs could allow them to more efficiently solve very large fully-observable problems. Hansen and Feng (2000) then extended this approach to handle POMDPs. These exact approaches can represent all the same domains as representabale by discrete-state POMDPs, and therefore they have high expressive power. However they will only have better computational tractability than exact standard discrete-state POMDP algorithms (such as Witness (Kaelbling et al., 1998)) when the domain exhibits a significant amount of structure that can be leveraged by an ADD representation.

More recently researchers have investigated combining ADD representations with point-based approaches (discussed in Section 2.4.3) with some significant success. Poupart (2005) created Symbolic Perseus, an algorithm which uses ADD operations combined with a Perseus-style point-based POMDP solver. This approach has yielded impressive results on automated handwashing assistant with millions of states (Boger et al., 2005). Last year Sim et al. (2008) combined ADDs with HSVI (Smith & Simmons, 2005), which is a point-based POMDP solver known to outperform Perseus. In addition, researchers have also been investigating more efficient ways to perform the ADD computations and restructuring necessary during value iteration (Shani et al., 2008). When the ADD operations are performed exactly as part of a point-based procedure, the resulting value function is guaranteed to be a lower bound on the true value function (as discussed in Section 2.4.3). However, in order to keep the ADD size bounded some approaches combine nodes with close values, which introduces an additional approximation error into the value function.

These exciting approaches have allowed POMDP solvers to scale to much larger, factored domains, with some real world successes. However, one of their limitations is that they still typically use a discretized representation for the values of each state variable. Therefore, large or infinite-valued domains, such as continuous-valued environments, are often still a significant challenge for these approaches. In Chapter 5 we will present an algorithm that can leverage factoring in the dynamics model when present but compactly represent distributions on state variables that can take on a large or infinite number of values.

An alternate type of model structure, which we will also leverage in later chapters, is assuming that the world models take on a particular compact parametric form. Porta et al. (2006) considered continuous-state POMDP planning where the world models were specified as Gaussians or weighted sums of Gaussians. The weighted sum of Gaussians representation was presented where the transition model is assumed to be linear Gaussian, and the reward and observation models are specified as a weighted sum of Gaussians. These representations enable the belief updates and Bellman backups to be performed analytically and the resulting representations (of the new belief and α-function) stay in closed form. Porta et al. introduce a new point-based POMDP planner (which we will discuss later in this chapter) using these representations. It is interesting that, unlike in discrete-state POMDP planning where the representation required to encode the value of a conditional plan (an α-vector) is fixed at $|S|$ parameters for all time, in the representations examined by Porta et al. the number of parameters required to represent an α-function grows with each backup. Therefore it is often necessary in practice to project the value function down to a compressed representation for computational tractability. This introduces error into the planning process, though no bounds are provided on the optimality of the resulting solution. It is also not clear that this approach is tractable in high-dimensional domains.

In Chapter 3 we will extend this work to consider a broader class of dynamics and also provide a formal analysis of the error introduced when the value function is approximated.

If the models are known to take on very particular forms, namely if the noise model is Gaussian, the dynamics are linear Gaussian, and the reward model is quadratic, then there exists a well known optimal controller, known as the linear quadratic Gaussian (LQG) controller (Burl, 1998). This controller requires computational time that scales polynomially with the state space dimension, meaning that it also has attractive tractability properties. However, most problems do not have linear Gaussian models across the state space, or have more complicated reward models, and in this case this controller can no longer be assumed to be applicable.

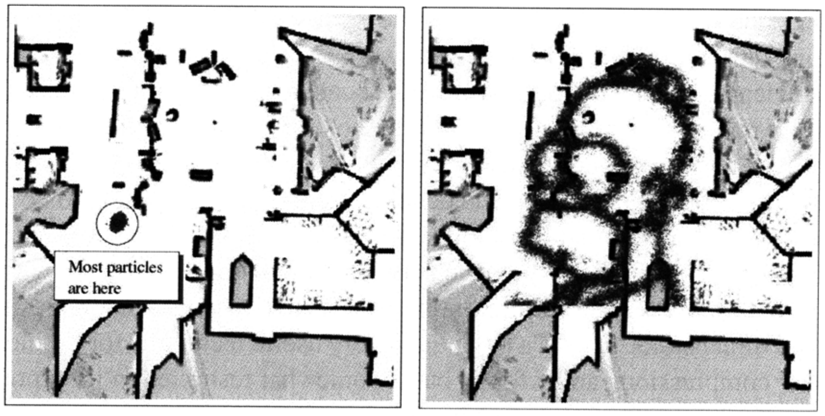
2.4.2 Belief Structure

Exact solutions to POMDPs require computing an optimal policy over the entire belief space. However, the full belief space may be reasonably well approximated for the purposes of planning by a distribution over a much smaller set of parameters. In fact, given a particular observation and dynamics model, some of the beliefs may be extremely improbably or impossible. Figure 2-10 depicts a building in which a robot is performing a navigation task. On the left the figure shows a sample belief in which the robot is fairly well localized. On the right is a very different sort of distribution over the possible location of the robot: it seems highly unlikely if not impossible that the agent could take a series of actions and receive a series of observations that would lead its belief to resemble this, given the typical sorts of dynamics of navigational robots and the type of sensors (such as laser range finders) often used.

If most reachable beliefs can be well approximated by a smaller set of parameters, it may be much easier to plan over this smaller parameter space. This is the intuition between a class of techniques known as belief compression (e.g. Roy and Thrun (2000), Poupart and Boutilier (2002), Roy (2003), Roy et al. (2005)).

The coastal navigation algorithm by Roy and Thrun (2000) approximates beliefs by only the mean and the entropy of the belief state. Planning is performed by discretizing the mean and entropy parameters into a fixed set of bins, and performing fitted value iteration over this discretized space. The results performed well on a mobile robot museum navigation task. However, as the authors point out, the approximation breaks down in environments which require multi-modal beliefs, which are fairly common. This approach focuses on a small class of problems, restricting its expressive power in exchange for good tractability on problems which fit the underlying assumptions of coastal navigation.

A more sophisticated belief compression technique by Roy et al. (2005) uses the exponential principal components analysis (E-PCA) approach developed by Collins et al. (2002). The key idea behind this form of belief compression is that POMDP beliefs may lie on a much lower-dimensional manifold that is embedded in the large $|S|$-dimensional space. If this low-dimensional space can be extracted, then planning can be performed in this reduced dimensionality space. In general the mapping will not be linear between the two spaces, and PCA may not be appropriate. Instead an exponential link function is used to transform the belief space into one which can be used for linear dimensionality reduction: this procedure is known as E-PCA. This link function ensures the transformed beliefs



(a) Reasonable belief

(b) Unlikely or impossible belief

Figure 2-10: *(Reproduced with permission from figure 4 of Roy et al. (2006)).* Both figures depict a building environment, and the dots represent sample robot locations drawn from a particular belief distribution. In (a) the robot's location is well localized to a small region, as is often seen during robot navigational tasks due to the particular motion model and sensors common in navigational robots. Subfigure (b) shows a very strange distribution over potential robot locations, and it is extremely unlikely that the robot would ever find itself in this belief state. This type of example provides intuition focussing planning on a smaller set of beliefs that are likely to be reached and that may exhibit more structure than the generic belief distribution.

remain strictly positive, as probability distributions should. Computing the parameters of the low-order representation involves a non-convex minimization process to maximize the likelihood of a set of sampled belief points, which means the resulting solution is only guaranteed to be a local optimum. Since the value function is unlikely to be piece-wise linear and convex in the transformed space, planning is performed by using fitted value iteration (Gordon, 1995) over the low-dimensional belief space approximation. The algorithm performs well on simulated robot navigation problem and on a person-tagging simulation, motivated by a nursing home application. The algorithm found good solutions by compressing approximately 800 and 2000 states down to 5-6 bases, and performing planning on the 5-6 bases. This suggests that this approach offers a significant potential for scaling up to larger domains.

One of the limitations of the above approaches is that they do not provide bounds on the quality or error of the computed value functions. There are two potential sources of error, the belief compression approximation and the approximate planner used to solve the low-dimensional continuous-state belief MDP, both of which could cause the resulting value function to differ from the optimal value.

Poupart and Boutilier (2002) provide a loose upper bound on the error of the resulting value function as a function of the error in the reward and transition dynamics over the compressed space, relative to the true reward and transition models over the full state space. The authors also consider lossless compressions; however in general to get significant compression it may be necessary to use lossy representations. Though the theoretical contributions are helpful, the initial results were only presented on a 32 state problem, leaving it unclear how well the approach would scale to larger problems. The authors later extend their work and combine it with the use of finite state controllers to form a new POMDP solver named value-direction compression with bounded policy iteration (VDCBPI) (Poupart & Boutilier, 2005). The belief compression is restricted to be linear[2], but a good compression can be found for some factored POMDP problems. The authors demonstrated their approach on a large synthetic network administration problem with millions of states. VDCBPI leverages the factored input representations and compresses the POMDP beliefs to 250 dimensions. This is a potentially very useful technique for problems in which a good linear compression can be found but is somewhat restricted in its expressive power to problems which possess this characteristic.

To try to scale to continuous-state space POMDPs, Brooks et al. (2006) take a belief compression approach that is very similar to the coastal navigation algorithm of Roy and Thrun (2000). Brooks et al. represent the belief as a diagonal Gaussian, which in a two-dimensional robot navigation tasks requires four sufficient statistics to be tracked $(\mu_x, \mu_y, \sigma_x^2, \sigma_y^2)$. The value function will not be piecewise linear and convex (PWLC) in this case, so they must use approximate techniques to estimate the value function. Their parametric POMDP (PPOMDP) algorithm represents the value function at a grid of belief states and use Freudenthal triangulation to interpolate the values in between the represented grid. By using a compact unimodal parametric form for the belief, the authors can finely discretize the belief parameters, as was done by Roy and Thrun (2000) and Roy et al. (2005), This approach has a similar limitation to coastal navigation of being unable to handle multimodal distributions. Multimodal distributions may arise when there is perceptual aliasing, such as two hallways appearing the same. It also does not provide any optimality guarantees on the resulting value function. The authors present experimental results using PPOMDP on a simulated robot navigation task to a goal in an environment with obstacles. Observations of obstacles are assumed to provide information only about one of the two directions (either horizontal or vertical) to make the belief update simple; unfortunately, this causes the robot to become trapped in certain locations. The approach is faster than discrete-representations in large environments since its belief is much more compact. However, as the state space dimension grows, the number of parameters required to represent the belief state will grow polynomially with the dimension. As each parameter is discretized, the number of discrete belief states will still grow exponentially with the dimension. This means that this approach is unlikely to scale well to high-dimensional problems. In Chapter 5 we will present a technique that also uses a Gaussian belief representation, but avoids this problem by using an alternate planning method which does not involve discretization.

Zhou et al.(to appear) extend Brooks et al.'s approach to a more general class of parametric distributions which can also represent multi-modal beliefs. The authors provide a procedure for projecting a new belief back onto a low-dimensional family of parameterized distributions by minimizing the KL-divergence. The low-dimensional parameter space can

[2] Here we use linear to mean that a full belief is expressable as a product of a fixed matrix and the low dimensional belief.

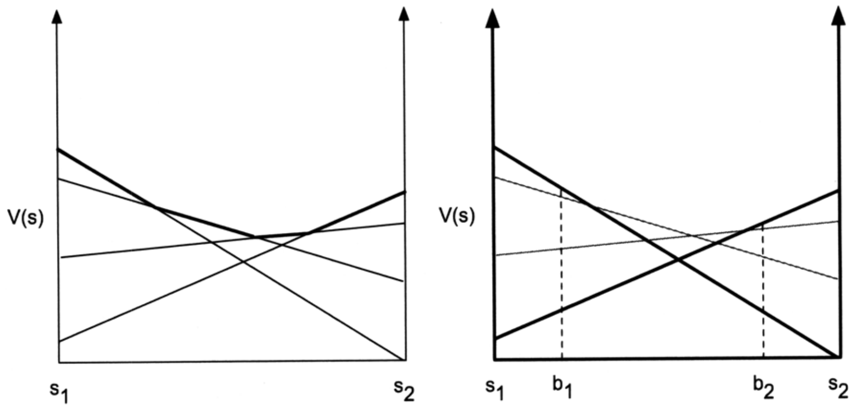
then be discretized and solved using fitted value iteration (FVI), as in coastal navigation or PPOMDP. The authors also provide bounds on the error of the resulting value function given the belief approximation. These bounds are a function of the difference between a belief and its projection, and their one-step evolutions (assuming an action and an observation), and also a smoothness assumption on the value function. In practice there is likely to be a tradeoff between the theoretical guarantees and the computational performance: a larger number of parameters in the low-dimensional belief representation will generally lead to tighter bounds, but will increase the dimension over which FVI must plan, which causes the computational cost to grow. The authors also present results using a Gaussian representation which outperforms PPOMDP. Overall this paper presents some interesting theoretical results, but it is unlikely to scale to problems that require large numbers of modes or many dimensions since each parameter in the low-dimensional representation will be discretized to perform planning, and the number of states in the discretized parameter space will scale exponentially with the number of parameters, leading to a similar exponential increase in the FVI backups. Therefore while this model has more expressive power than the model by Brooks et al. it is still only tractable in low dimensional domains.

2.4.3 Value function structure

An alternate approach is to try to leverage structure in the value function. Consider the value function represented in Figure 2-11a which is the supremum of four α-vectors. Due to the piecewise linear convex (PWLC) aspect of the value function nearby beliefs often have the same best α-vector. Intuitively if one was to only keep the best α-vector for a subset of beliefs, these α-vectors would likely to give close to optimal values for surrounding beliefs, and provide lower bounds for other beliefs. Therefore it may be possible to get a good lower bound estimate of the value function using only a few α-vectors by leveraging the value function's PWLC structure.

This is the insight behind perhaps the most popular approximate POMDP solvers over the last 5-6 years, which are known as point-based approaches. These approaches were first considered by Cheng (1988) and Zhang & Zhang (2001) and later popularized by Pineau et al.'s PBVI (2006), Spaan and Vlassis's Persues (2005a), and Smith and Simmon's HSVI (2005) algorithms. Point-based techniques estimate the value function at only a small set of N chosen belief points $\tilde{B}$, resulting in a value function represented by at most N α-vectors. This eliminates the exponential growth in the number of α-vectors present in exact appropriates. The approximate value function representation is constructed by iteratively computing an approximately optimal t-step value function V_t from the previously-computed $(t-1)$-step value function V_{t-1} by backing up the value function at beliefs $b \in \tilde{B}$ using the Bellman equation (assuming discrete states and actions):

$$\begin{aligned} V_t(b) &= \max_{a\in A} \sum_{s\in S} R(s,a)b(s) + \gamma \sum_{z\in Z} \max_{\alpha_{t-1}\in V_{t-1}} \sum_{s\in S} \sum_{s'\in S} p(s'|s,a)p(z|s')\alpha_{t-1}(s')b(s) \\ &= \max_{a\in A} \left[\sum_{s\in S} R(s,a)b(s) + \gamma \sum_{z\in Z} \max_{\alpha_{azj}} \langle \alpha_{azj}, b\rangle \right] \qquad (2.9) \end{aligned}$$



(a) Complete V

(b) Point-based V

Figure 2-11: Subfigure (a) illustrates an optimal value function which consists of 4 α-vectors: the heavy lines show the maximal value at each belief over a 2-dimensional state space. (b) shows the resulting value function if only the optimal α-vector for 2 selected beliefs, b_1 and b_2 are kept. The resulting value function will be a lower bound on the optimal value function since some α-vectors that might be optimal for other beliefs are missing. However, often the resulting approximation will be sufficient to achieve fairly good performance: note that in this example belief states near b_1 and b_2 will typically have the same optimal α-vector. Point-based value iteration leverages this smoothness by focussing on computing the α-vectors for a subset of belief points.

where α_{azj} is the vector associated with taking action a, receiving observation z and then following the $(t-1)$-step policy associated with α_j. It is efficient to compute the dominant α-vector at a specific b, and those vectors taken together provide an approximation of V_t over entire belief space. This is illustrated in Figure 2-11 which shows in (a) an example optimal value function over the entire belief space and in (b) the value function that if only the α-vectors of two belief points were kept. Note that in general the α-vectors that represent the final point-based value function will not overlap with any of the α-vectors that would constitute the optimal value function, since the point-based backups involve taking the maximum only over N α-vectors rather than an expanding set of all possible α-vectors.

Different point-based techniques include carefully adding new elements to $\tilde{B}$ to improve this lower bound (Pineau et al., 2006) or updating a random subset of $\tilde{B}$ at each iteration (Spaan & Vlassis, 2005b). An overview of the Perseus algorithm is provided in Algorithm 2. One of the most impressive point-based solvers, HSVI, maintains upper and lower bounds on the computed value function, providing a principled approach for when to stop backing up the value function (Smith & Simmons, 2005). A recent and promising approach that has outperformed HSVI on some problems is SARSOP, a point-based approach that focuses on selecting belief points to place in $\tilde{B}$ that are likely to lie within the reachable set of beliefs visited when following the optimal policy (Kurniawati et al., 2008).

These approaches have allowed POMDPs to scale to much larger domains than previously possible, up to problems with tens of thousands of states. However, all of the

Algorithm 2 Point-based POMDP Planner Perseus

```
Gather a set of belief states $\tilde{B}$ by taking actions randomly in the world
Set $\alpha_0$ to be a known lower bound on the value function
$\Gamma = \alpha_0$
while the values of the beliefs change, or the set of $\alpha$s change do
  Compute the value $V_{old}(b)$ of all beliefs $b \in \tilde{B}$ under the current set of $\alpha$-vectors $\Gamma$
  Set $Unupdated = \tilde{B}$
  while $Unupdated$ is non-empty do
    Randomly select a belief state $b_j \in \tilde{B} \cap Unupdated$
    Perform a value-function backup for belief $b_j$ to generate a new $\alpha$-vector $a_j$
    if the value of $b_j$ has not increased under $\alpha_j$ then
      Set $\alpha_j$ to be the prior $\alpha$ which maximized the value of $b_j$
    end if
    add $\alpha_j$ to the new set of $\alpha$-vectors $\Gamma'$
    Find all $b \in Unupdated$ whose value under the new $\alpha_j$ $\sum_s \alpha_j(s)b(s) \geq V_{old}(b)$ and remove them from $Unpdated$
  end while
end while
```

approaches discussed involve computing a value function backup on at least a subset of the beliefs in $\tilde{B}$ which creates new α-vectors defined over the entire state space. Recall Equation 2.9 which showed how to compute the value of a new α-vector corresponding to a new one-step-longer conditional policy tree. This backup operation involves at least $O(|S|^2|Z|)$ operations. Consider a mobile robot navigating in a large environment, where its state is described by an (x,y) tuple. If the environment is larger than 1 kilometer squared, then even a coarse 1m discretization will yield a state space of 10^6. This problem is only exaggerated in higher-dimensional, continuous-valued environments. In such scenarios it will be extremely computationally demanding to even perform a single backup using a standard discrete approach.

Some POMDP planners attempt to amelierote this difficulty by working directly with continuous states and assuming a particular structure of the value function. To our knowledge, the first such POMDP planner is the Monte Carlo POMDP (MC-POMDP) algorithm by Thrun (2000). In MC-POMDP the belief is represented by a weighted set of particles, and planning is performed by Q-learning as the agent explores in the world. The value of a belief outside of the existing set is a linear function of the k nearest neighboring beliefs within the existing set. MC-POMDP dynamically grows the existing set of beliefs whenever less than k nearest neighbors can be found for existing states. The results present experiments on a small two-dimensional simulation and on a 3D robot grasping task. This approach helped pave the way for considering continuous-state and continuous-action POMDP planners, but offered no theoretical guarantees on the resulting quality of the plans achieved. In addition, the number of particles necessary to achieve a good approximation of the belief will typically scale exponentially with the dimension of the state space, so this approach may struggle in high-dimensional environments.

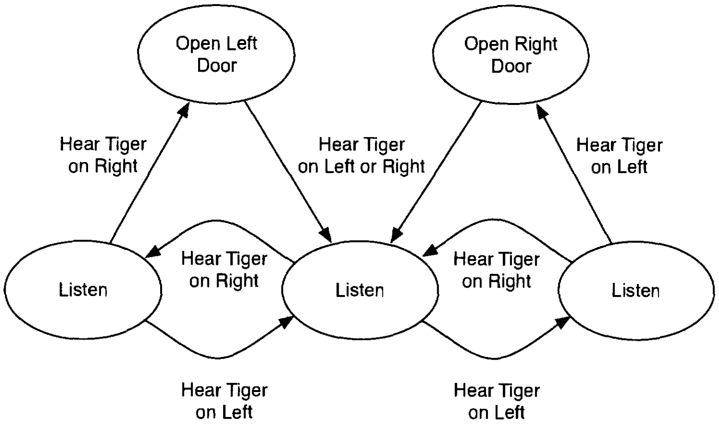


Figure 2-12: Finite state controller for tiger POMDP problem (Kaelbling et al., 1998) given an initial belief of $(0.5, 0.5)$ for whether the tiger is behind the left or right door, and the $Listen$ action returns the correct door of the tiger 85% of the time and the wrong door the remaining 15% of the time. This representation avoids having to explicitly perform belief updating: instead control proceeds by taking the action in the current node, and then transitioning to the connected node according to which observation was received.

2.4.4 Policy structure

Rather than computing a value function over the space of beliefs and then later extracting a policy from that value function, an alternate approach is to try to directly construct the policy. There are a number of different approaches for making structural assumptions on the POMDP policy form or POMDP policy class and using this information during policy search (for example see Kearns et al. (2000) or Ng and Jordan (2000)). Here we will outline some approaches that leverage a particular form of encoding the policy which avoids belief filtering.

As discussed by Kaelbling et al. (1998), the optimal finite-horizon POMDP policy can be represented by a finite-state controller (FSC) by constructing a set of nodes, each of which is associated with a particular action. Nodes are connected with edges labeled by observations. Given an initial node location, control is performed by selecting the action of the current node, and then following the edge labeled with the observation received: see Figure 2-12 for an example. FSCs offer the advantage that they eliminate the need for belief updating: implicitly each node is associated with a collection of beliefs that have the same best conditional policy tree, but this tree or corresponding value is not explicitly represented. FSC representations essentially collect equivalent (for the purposes of planning) beliefs, aggregating beliefs into a set of belief clusters with the identical best next action.

An early approach by Hansen (1998) demonstrated that a policy iteration approach could be used to compute good policies using a FSC representation; however this exact approach was limited to very small problems. Later Hansen and Zhou (2003) extended this work to show how programmer-specified hierarchies of FSCs could be used to solve larger POMDPs problems. Charlin et al. (2008) provided an approach which would automatically learn the hierarchical structure, and also allowed for recursive FSCs, which can potentially lead to an exponentially more compact hierarchical FSC. Unfortunately this approach was limited in practice due to the very large number of parameters it has to optimize over, and using a non-linear solver package allowed this approach to only scale to problems with approximately tens of states. However, this practical limitation was recently addressed with some significant success by an elegant approach by Toussaint and colleagues that treats planning as inference (Toussaint et al., 2008). The authors present a way to compute the parameters of a hierarchical FSC by representing the associated POMDP as a Dynamic Bayes Net with a special reward formulation: performing expectation-maximization (EM) on the network generates the FSC parameters. Though EM is only guaranteed to converge to a local solution, the empirical quality of performance on a number of large benchmark finite-state POMDP problems was competitive with one of the state of the art point-based planners, HSVI.

It seems likely that it may be possible to extend Toussaint et al.'s recent work to continuous-valued domains and this is something we are interested in exploring in the future. One of the attractive aspects of this new framework is that it can leverage the considerable advances made in the inference community. However, most of these approaches do not provide optimality guarantees on the produced plans.

2.4.5 Online planning

The majority of approaches discussed so far have focused on offline planning by using the Bellman equation to iteratively compute a value function over the entire belief space. In some cases, such as exact methods or in point-based approaches, offline planning involved repeatedly computing the value of successively longer conditional plans. In other cases, such as in many of the belief compression techniques, planning was performed using Bellman backups to estimate the value over an approximate parameter space.

An alternate approach to offline methods is to instead compute an action only for the current belief. After taking the selected action, receiving an observation, and updating the resulting belief state, one can repeat the computation to select the best next action. Such methods are known as *online* planners since they only plan for the beliefs reached during acting in the world as each belief is encountered, rather than constructing a plan in advance for all possible beliefs that might be reached.

One recently popular online approach is forward search, which has been used with impressive success on very large problems such as RoboCup Rescue (Paquet et al., 2005), and offers a number of attractive features. The first is that the majority of computational effort is spent on selecting actions for beliefs which are actually reached during execution[3] whereas even the best offline approaches sometimes spend time computing α-vectors for beliefs that are extremely unlikely to be reached during acting.

Second, forward search approaches can be used in highly time constrained situations, or in scenarios when a significant amount of time is reasonable between each action: in general the solutions provided will continue to improve as more time is allowed, but forward search can still be used to provide some answer in almost any allotted time. In this

[3] Some computational effort is typically spent on other beliefs that are reached in the forward search process, which may or may not be representative of the beliefs experienced during actual execution.

sense forward search is an *anytime* method. A number of offline approaches can also be considered anytime methods, since their computed α-vectors provide an expected value for any potential belief, and so halting planning only after a few backups will still yield a plan.

However, a third additional benefit of forward search approaches is that they can use an existing offline plan, and are guaranteed to equal or improve upon its expected value (see Ross et al. (2008b) for one discussion of this). Intuitively this is reasonable: essentially forward search allows one to look even further into the future, for a particular root belief, and use this additional information plus the prior policies to select an action. If the provided policy is not optimal, then looking further into the future and effectively performing additional Bellman backups for a particular belief, will only improve the value of the action selected for this belief. In addition, even loose lower or upper bounds on the value function computed offline can be used in forward search techniques to guide the tree expansion.

Finally, a fourth strong additional benefit is that in forward search techniques the action and observation space (or sampling method) can be adapted during forward search. This can be extremely helpful when the action or observation space is very large. Hsaio et al. (2008) use this strategy successfully to automatically perform robotic manipulation given only noisy sensors, by altering the actions considered based on the current belief state.

One challenge in forward search POMDP planning that has been addressed relatively little in past research is considering the computational complexity of performing forward search in large or infinite-valued, high dimensional state spaces. In such domains even performing some of the standard operations of forward search can become expensive. In Chapter 5 we will further discuss forward search POMDP methods, and describe an algorithm to quickly perform the necessary forward search operations in structured, continuous-valued, high-dimensional environments.

2.5 Summary

Despite the important advances made by the research discussed in this chapter, there remain many open challenges to scaling up to large, high-dimensional, uncertain domains and to providing bounds on the plan quality relative to the optimal value where possible. In the following chapters we will discuss several algorithms we developed to try to address these challenges.

Information is the resolution of uncertainty.
Claude Shannon

3 Switching Mode Planning for Continuous Domains

The previous chapter surveyed past research on sequential making under uncertainty. Though there has been much important work in this field, most approaches were clustered at one or two limits of the three-way spectrum of tractability, expressive power, and optimality. There was particularly a dearth of approaches applicable for the large, high-dimensional, uncertain domains of interest of this thesis, such as autonomous navigation through traffic to reach a destination location. In this chapter we present a new algorithm for making decisions in large uncertain partially observable environments. The main contribution will be the Switching Modes POMDP (SM-POMDP) algorithm which leverages parametric representations of the world models in order to perform closed form planning in continuous-valued environments. SM-POMDP generalizes prior work by Porta et al. (2006) to be applicable to a more general class of dynamics models that are likely to be necessary in real-world environments. SM-POMDP has greater expressive power with similar tractability to prior approaches. See Figure 3 for a graphical illustration of how SM-POMDP relates to some other POMDP planning approaches.

The SM-POMDP algorithm falls into the class of POMDP approaches that seek to leverage structure in the world models (see Chapter 2.4.1 for a prior discussion). To review, the main hope of structured models is that they may be more compact, or make certain computations more efficient, and therefore enable planning algorithms that use such representations to be faster and/or scale to larger domains. In SM-POMP we follow in the footsteps of Porta et al. (2006) and represent the world models using a weighted sum of Gaussians.

One of the important benefits to this representation is that it results in a dynamic variable resolution representation of the value function. It is always possible to discretize a continuous-valued problem into a finite set of states. However, it is often challenging to select the granularity of the resulting finite representation. If the discretization chosen is too coarse, it will be impossible to represent the correct value function well, and therefore it is likely that a poor policy will be computed: indeed an example of this phenomenon will be seen later in this chapter. However, since the computation time for a single discrete-state backup is a quadratic function of the number of states, then an excessively fine discretization will result in a significantly longer computation time than necessary. Therefore there is a significant incentive to choose the granularity at precisely the right level. Unfortunately, this correct granularity is typically not known in advance. In addition, it is often the case

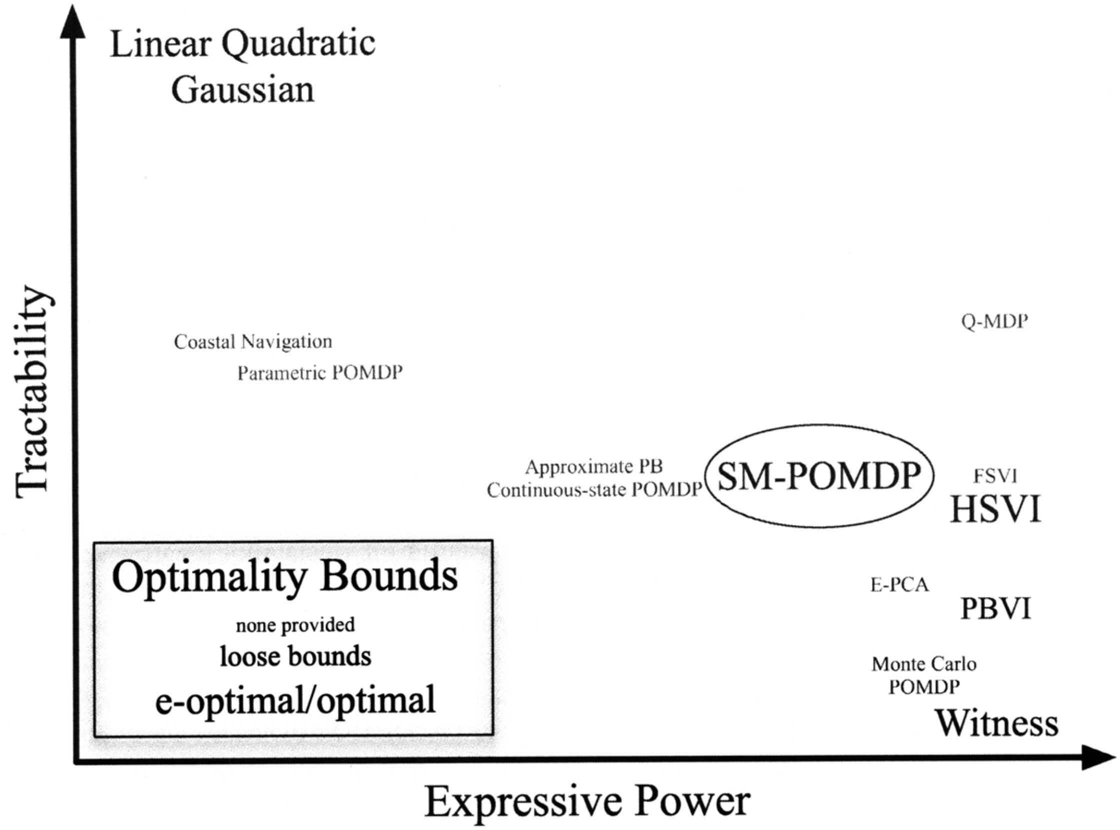


Figure 3-1: Illustration of the optimality, tractability and expressive power of SM-POMDP in comparison to several other POMDP planning techniques.

that the value function may not require a uniform level of discretization: work by Munos and Moore (1999), and by Feng et al. (2004) on continuous-state MDPs (which may be used to solve the continuous belief-state MDP defined by a discrete-state POMDP) has demonstrated that the value function may be well approximated by a variable-resolution grid. However, the approach of Munos and Moore involved an iterative refinement of the grid where the MDP (in our case, POMDP) is repeatedly solved, which is not feasible in our case, and Feng et al.'s work made restricted assumptions on the dynamics model.

Instead, as we will shortly see, the representation we propose allows closed form backups of the value function and the belief state. In doing so, it will implicitly adapt the value function representation at each backup, introducing new components to the representation as the structure of the value function changes. In this sense it can be considered a variable resolution approach, but one that dynamically determines the resolution as part of the standard value function computation. This can be a significant advantage over standard uniform resolution discrete-state POMDP planners, as Porta et al. (Porta et al., 2006) demonstrated by showing their linear Gaussian POMDP planner was faster than the discrete-state POMDP planner Perseus (Spaan & Vlassis, 2005a) in a simulation experiment.

However, if parametric functions are used to represent the world models and value function, it is also critical to ensure that the chosen functions are sufficiently expressive to adequately represent the problems of interest. Previous work on planning for continuous-state POMDPs has typically modeled the world dynamics using a single linear Gaussian model[1] to describe the effects of an action (Brooks et al., 2006; Porta et al., 2006; Thrun, 2000)). However, the dynamics of robotic grasping, autonomous navigation over varying terrain, and many other problems of interest are highly complex and nonlinear. For example, an autonomous car crossing varied terrain will exhibit different dynamics depending on whether the ground underneath is sand or rocks. Though such dynamics are easily represented in discrete-state environments using the standard transition matrices, a single linear Gaussian continuous-state model will be insufficient to adequately model these multi-modal state-dependent dynamics.

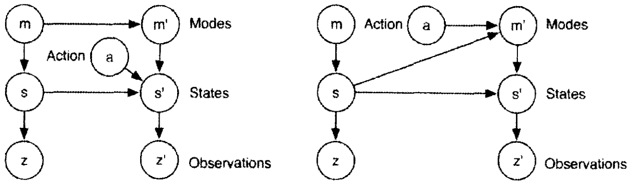
Our Switching Modes POMDP (SM-POMDP) algorithm uses a switching state space representation of the world dynamics to perform approximate closed form planning in continuous-valued environments. This model can both represent actions that result in multimodal stochastic distributions over the state space, and succinctly represent any shared dynamics among states. SM-POMDP is a new point-based POMDP solver that uses this new structured representation of the dynamics model in order to generalize to a larger class of POMDP problems than past related approaches, while still maintaining the strengths of parametric representations in continuous-state POMDP planning. We demonstrate the advantage of SM-POMDP over prior parametric planners on an unmanned aerial vehicle collisions avoidance simulation, and a robot navigation simulation where the robot has faulty actuators.

In contrast to prior approximate continuous-state POMDP research (Thrun, 2000; Porta et al., 2006; Brooks et al., 2006), we also provide a theoretical analysis of our several variants of our SM-POMDP planner, bounding where possible the error between the resulting SM-POMDP value function, and the optimal value function. We also analyze the computational complexity of planning using SM-POMDP. As is often the case, we show there is a tradeoff between computational tractability and guarantees on the resulting optimality of the produced plan. Our analysis could aid researchers interested in using SM-POMDP in selecting the right SM-POMDP variant appropriate for the tractability and optimality constraints of their own application.

3.1 Switching State-space Dynamics Models

Switching state-space models (SSM) (also known as hybrid models and jump-linear systems) are a popular model in the control community for approximating systems with complex dynamics (Ghahramani & Hinton, 2000). Typically an SSM consists of a set of linear state transition models. At each time step a hidden discrete mode state indexes a particular transition model which is used to update the hidden continuous-state vector. Frequently, the transition dynamics of the mode states are modeled as a Markov process (Ghahramani & Hinton, 2000) (see Figure 3.1a for an illustration). SSMs have been used to approximate the dynamics of a diverse set of complex systems, including planetary rover operation (Black-

[1] In some prior work (Brooks et al., 2006; Thrun, 2000) a special exception is included to encode boundary conditions such as obstacles in a robotic task.



(a) Typical SSM (b) Our Dynamics Model

Figure 3-2: Switching State-Space Models

more et al., 2007), honeybee dances (Oh et al., 2005) and the IBOVESPA stock index (Fox et al., 2009). The bulk of prior similar work on SSMs has focused on model parameter learning, and we are not aware of any prior work on using these models for POMDP planning tasks.

In order to model systems involving multi-modal, state-dependent dynamics (such as on sand vs concrete), we create a particular variant of an SSM that conditions the mode transitions on the previous continuous-state, similar to (Blackmore et al., 2007). Figure 3.1b displays a graphical model of the dynamics model used in this paper, which can be expressed as

$$p(s'|s,a) = \sum_h p(s'|s,m'=h)p(m'=h|s,a) \tag{3.1}$$

where s, s' are the continuous states at time t and $t+1$ respectively, a is the discrete action taken at time t, and m' is the discrete mode at time $t+1$. We assume that for each action a the hidden mode m can take on one of H values. Each mode value h and action a is associated with a linear Gaussian model $\mathcal{N}(s'; \zeta_{ha}s + \beta_{ha}, \sigma^2_{ha})$.

For mathematical convenience we model the conditional probability of a mode taking on a particular value h given the previous continuous-state s and action a using a weighted sum of F Gaussians

$$p(m'=h|s,a) = \sum_{f=1}^{F} w_{fha}\mathcal{N}(s; \mu_{fha}, \sigma^2_{fha}). \tag{3.2}$$

This representation is slightly unusual; we are expressing the probability of a discrete variable m conditioned on a continuous variable s. See Figure 3-3 for an example mode model $p(m|s)$. Note that for finite F it is impossible to select the parameters $w_{fha}, \mu_{fha}, \sigma^2_{fha}$ such that the sum of probabilities of the next mode state m' taking on any value for a given state s, $\sum_h p(m'=h|s,a)$, equals 1 for all states s. Therefore in practice we will choose models that approximately sum to 1 over all the states of interest in a particular experimental domain. We choose to make this alternate representational choice rather than using a softmax function over the state space because the mixture of Gaussians representation will allow closed form updates of the belief state and value function, as will be shown in the following sections.

Substituting equation 3.2 into equation 3.1, the full dynamics model is a sum of Gaus-

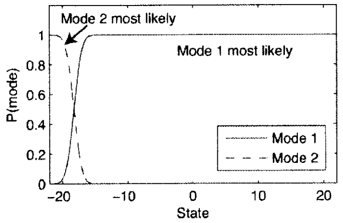


Figure 3-3: Example of a mode model $p(m|s)$

sian products:

$$p(s'|s,a)=\sum_{h=1}^{H}\mathcal{N}(s';\zeta_{ha}s+\beta_{ha},\sigma_{ha}^2)\sum_{f=1}^{F}w_{fha}\mathcal{N}(s;\mu_{fha},\sigma_{fha}^2).$$

An added benefit of this model is that it can flexibly represent relative transitions (transitions that are an offset from the current state, by setting $\zeta \neq 0$) and absolute transitions (transitions that go to some arbitrary global state, by setting $\zeta = 0$ and $\beta \neq 0$). This allows the model to compactly represent domains in which many states share the same relative or absolute transition dynamics. In contrast to discrete approaches which must enumerate over all discrete states the probability of each state transitioning to each other state, even if many states share the same dynamics (such as "go 2 grid cells to the left"), our model can represent such dynamics by a single linear Gaussian $\mathcal{N}(s'; s-2)$. This reduces the number of model parameters and should reduce the time needed to learn these models. We explore this issue further in Chapter 4. Indeed, existing work (Leffler et al., 2007a) on generalizing dynamics across states in a reinforcement learning setting has demonstrated that sharing transition mdoel parameters does speed learning in discrete state environments.

3.2 SM-POMDP

We now describe a new planning algorithm for POMDPs with hybrid dynamics models. Recently Porta et al. (2006) showed that for a continuous-state space S and discrete actions A and observations Z, the optimal finite horizon value function is piecewise linear and convex and may be represented by a finite set of α-functions[2]. Therefore point-based approaches to continuous state POMDPs that represent α-functions at a finite set of beliefs will also provide a lower bound on the value function. Porta et al.'s algorithm provides an approximation of a lower bound on the value function: our algorithm is inspired by theirs and handles multi-modal state-dependent dynamics.

For clarity we will explain the mathematics for a one-dimensional state space, but it can also be extended to higher dimensions. We will assume that the reward model $r(s,a)$ and observation model $p(z|s)$ are represented by a weighted sum of Gaussians. To be precise,

[2] The expectation operator $\langle f, b\rangle$ is a linear function in the belief space and the value function can be expressed as the maximum of a set of these expectations: for details see Porta et al. (2006).

we assume the reward function $r(s,a)$ is expressed as a sum of G Gaussian components for each action a, $r(s,a) = \sum_{g=1}^{G} w_{ag}\mathcal{N}(s; \mu_{ag}, \sigma_{ag}^2)$, and each discrete observation $z \in Z$ is expressed as a sum of L Gaussian components $p(z|s) = \sum_{l=1}^{L} w_{zl}\mathcal{N}(s; \mu_{zl}, \sigma_{zl}^2)$ such that $\forall s \sum_z p(z|s) = 1$. Here we have assumed an observation model very similar to the mode representation (equation 3.2) and the same comments made for that choice apply here to the observation model.

We also choose to represent the belief states b and α-functions using weighted sums of Gaussians. Each belief state b is a sum of D Gaussians $b(s) = \sum_{d=1}^{D} w_d\mathcal{N}(s; \mu_d, \sigma_d^2)$, and each α_j, the value function of policy tree j, is represented by a set of K Gaussians $\alpha_j(s) = \sum_k w_k\mathcal{N}(s; \mu_k, \sigma_k^2)$. Recall that for each action a, there are H modes and F Gaussian components per mode.

We do not lose expressive power by choosing this representation because a weighted sum of a sufficient number of Gaussians can approximate any continuous function on a compact interval (and our domains of interest are closed and bounded and therefore fulfill the criteria of being compact) (see e.g. Park and Sandberg 1991). But, of course, we will be effectively limited in the number of components we can employ, and so in practice our models and solutions will both be approximations.

Point-based POMDP planners must include a method for backing up the value function at a particular belief b (as in equation 2.9) and for updating the belief state b after a new action a is taken and a new observation z is received. Representing all parameters of the value function as a weighted sum of Gaussians allows both computations to be performed in closed form.

The belief state is updated using a Bayesian filter:

$$\begin{aligned} b^{a,z=i}(s) &= p(s'|z=i,a,b) \\ &\propto p(z=i|s',a,b)p(s'|a,b) \\ &= p(z=i|s')p(s'|a,b) \end{aligned}$$

where the last equality holds due to the Markov assumption. We compute the update by substituting in the dynamics and observation models:

$$b^{a,z=i}(s) \propto \sum_{l=1}^{L} w_l\mathcal{N}(s'; \mu_l, \sigma_l^2) \int_s \sum_{h=1}^{H} \mathcal{N}(s'; \zeta_{ha}s + \beta_{ha}, \sigma_{ha}^2) \sum_{f=1}^{F} w_{fha}\mathcal{N}(s; \mu_{fha}, \sigma_{fha}^2) \sum_{d=1}^{D} w_d\mathcal{N}(s; \mu_d, \sigma_d^2) ds.$$

After pulling the sums outside the integral, it is necessary to integrate the product of three Gaussians within the integral:

$$b^{a,z=i}(s) \propto \sum_{d,f,h,l} w_d w_{fha} w_d \mathcal{N}(s'; \mu_l, \sigma_l^2) \int_s \mathcal{N}(s'; \zeta_{ha}s + \beta_{ha}, \sigma_{ha}^2)\mathcal{N}(s; \mu_{fha}, \sigma_{fha}^2)\mathcal{N}(s; \mu_d, \sigma_d^2) ds.$$

This integration can be analytically computed by repeatedly applying the closed formula for the product of two Gaussians which is included for completeness as Equation A.1 in the Appendix. To apply this operator we also re-express the $\mathcal{N}(s'; \zeta_{ha}s + \beta_{ha}, \sigma_{ha}^2)$ as a function of s by manipulating the exponent. Performing these operations, then integrating, yields the final expression for the new belief:

$$b^{a,z=i}(s) = \sum_{dfhl} c_{dfhl} N(s|\mu_{dfhl}, \sigma^2_{dfhl})$$

where

$$\begin{aligned}
C_1 &= \frac{\sigma^2_{fs}\sigma^2_d}{\sigma^2_{fa} + \sigma^2)d} \\
c_1 &= \frac{\mu_d\sigma^2_{fa} + \mu_{fa}\sigma^2_d}{\sigma^2_{fa} + \sigma^2_d} \\
C_2 &= C_1 + \frac{\sigma^2_{zl}}{\zeta_{ha}} \\
c_2 &= \beta_{ha} + \zeta_{ha}c_1 \\
\sigma^2_{dfhl} &= ((\sigma^2_{zl})^{-1} + C_2^{-1})^{-1} \\
\mu_{dfhl} &= \sigma^2_{dfhl}(\mu_{zl}(\sigma^2_{zl})^{-1} + c_2C_2^{-1}) \\
c_{dfhl} &\propto w_d w_{fha} w_d \mathcal{N}(\mu_{zl}|c_2, \sigma^2_{zl} + C_2)
\end{aligned}$$

where the weights are scaled so the belief is normalized such that $\int_s b^{a,z}(s)ds = 1$. Hence the representation of the belief as a mixture of Gaussians is closed under belief updating.

The other key computation is backing up the value function for a particular belief. Since we also use discrete actions and observations, the Bellman equations can be expressed as a slight modification to equation 2.9, replacing sums with integrals and writing out the expectation:

$$\begin{aligned}
V_t(b) =& \max_{a \in A} \int_{s \in S} R(s,a)b(s)ds + \gamma \sum_{z \in Z} \max_{\alpha_{azj}} \int_s \alpha_{azj} b(s)ds \\
=& \langle \max_{a \in A} R(s,a) + \gamma \sum_{z \in Z} \arg\max_{\alpha_{azj}} \alpha_{azj}, b \rangle
\end{aligned}$$

where we have used the inner-product operator $\langle f, b \rangle$ as shorthand for expectation to obtain the second equality. As stated previously, α_{azj} is the α-function for the conditional policy corresponding to taking action a, receiving observation z and then following the previous $(t-1)$-step policy tree α_j, and can be expressed as

$$\alpha_{azj}(s) = \int_{s'} \alpha_{j,t-1}(s')p(z|s')p(s'|s,a)ds'.$$

Substituting in all the parametric models yields

$$\begin{aligned}\alpha_{azj}(s) &= \int_{s'} \sum_{k=1}^{K} w_k \mathcal{N}(s'; \mu_k, \sigma_k^2) \sum_{l=1}^{L} w_l \mathcal{N}(s'; \mu_l, \sigma_l^2) \sum_{h=1}^{H} \mathcal{N}(s': \zeta_{ha} s + \beta_{ha}, \sigma_{ha}^2) \sum_{f=1}^{F} w_{fha} \mathcal{N}(s; \mu_{fha}, \sigma_{fha}^2) ds' \\ &= \sum_{f=1}^{F} \sum_{h=1}^{H} \sum_{k=1}^{K} \sum_{l=1}^{L} w_{fha} w_k w_l \mathcal{N}(s; \mu_{fha}, \sigma_{fha}^2) \int_{s'} \mathcal{N}(s'; \mu_k, \sigma_k^2) \mathcal{N}(s'; \mu_l, \sigma_l^2) \mathcal{N}(s'; \zeta_{ha} s + \beta_{ha}, \sigma_{ha}^2) ds\end{aligned}$$

We combine the three Gaussians inside the integrand into a single Gaussian by repeatedly using the formula for the product of two Gaussians (Equation A.1). This creates a function of s' and other terms that are independent of s'. Integrating over s' yields

$$\alpha_{azj} = \sum_{f=1}^{F} \sum_{h=1}^{H} \sum_{k=1}^{K} \sum_{l=1}^{L} w_{fhkl} \mathcal{N}(s; \mu_{fhkl}, \sigma_{fhkl}^2) \tag{3.3}$$

where

$$\begin{aligned}w_{fhkl} &= w_{fha} w_k w_l \mathcal{N}(s_l; s_k, \sigma_k^2 + \sigma_l^2) \mathcal{N}(\mu_{fha}; \frac{c - \beta_{ha}}{\gamma_{ha}}, \sigma_{fha}^2 + \frac{C + \sigma_{ha}^2}{\gamma_{ha}^2}), \\ C &= \frac{\sigma_l^2 \sigma_k^2}{\sigma_l^2 \sigma_k^2}, \\ \mu_{fhkl} &= \frac{(C + \sigma_{ha}^2)\mu_{fha} + \sigma_{fha}^2 \gamma_{ha}(c - \beta_{ha})}{\sigma_{fha}^2 (C + \sigma_{ha}^2)}, \\ \sigma_{fhkl}^2 &= \frac{\sigma_{fha}^2 (C + \sigma_{ha}^2)}{C + \sigma_h^2 + \sigma_{fha}^2 \gamma_{ha}^2}, \\ c &= \frac{\mu_k \sigma_l^2 + \sigma_k^2 \mu_l}{\sigma_l^2 \sigma_k^2}.\end{aligned}$$

The new α_{azj} now has $F \times H \times K \times L$ components, compared to the α_j for the $(t-1)$-step policy tree, which only had K components. To finish the value function backup, we substitute α_{azj} back into equation 3.3 and choose the α-function that maximizes the future expected reward $\langle \alpha, b \rangle$ of belief b

$$\begin{aligned}\alpha(s) &= \max_a R(s, a) + \gamma \sum_{z=1}^{|Z|} \max_{\alpha_{azj}} \alpha_{azj} \\ &= \max_a \left[\sum_{g=1}^{G} N(s|\mu_g, \sigma_g^2) + \gamma \sum_{z=1}^{|Z|} \max_{\alpha_{azj}} \alpha_{azj} \right]\end{aligned}$$

Since all elements in this result are weighted sums of Gaussians, the α-function stays in closed form. Note that had we utilized a softmax distribution for the observation model or mode probabilities that it would not be possible to perform the integrals in closed form. Figure 3-4 displays an example value function, showing both its Gaussian components and

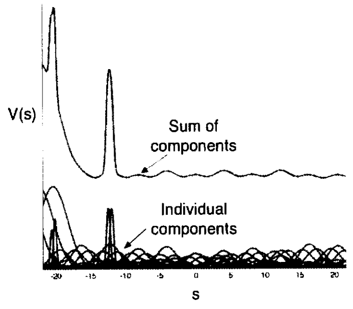


Figure 3-4: A sample α-function, showing both its total value at each state, as well as the individual Gaussian components that are summed to create its final value.

its final (summed) representation.

Note that the number of Gaussian components in a single α-function has increased: from K to $G + |Z|FHKL$ components.

3.2.1 Computational complexity of Exact value function backups

Before proceeding, it is important to consider the computational complexity of an exact value function backup for a particular belief state. As just described, the first step is to create the set of new α_{azj} functions which consist of considering each possible action, then receiving each possible observation, and then following one of the initial α-functions, α_j. From Equation 3.3 we know that creating each new α_{azj} components, involves $F \times H \times K \times L$ matrix operations (including a matrix inverse from combining two Gaussians, an O(D^3) computation where D is the number of state dimensions D). This procedure is repeated for each of the $N_\alpha|A||Z|$ new α_{azj} functions, yielding a cost of $O(FHKLN_\alpha|A||Z|D^3)$. It is then necessary to compute the expected value of the current belief state under each of these new functions in order to select the α_{azj} which maximizes the expected future reward of the current belief state under each possible action and observation. This operation involves the integral of the product of two weighted sums of Gaussians, an operation that can be done analytically and whose computational complexity is $O(D^3)$ for each of the combined Gaussians (to perform the necessary matrix inverse) multiplied by the number of components in each original expression. This operation will take $O(FHKLN_\alpha|A||Z|O(D^3)N_B)$ time.

Completing the backup requires summing over all observations for each action which is a simple summation over all the previously computed $\langle\alpha_{azj}, b\rangle$ values. Finally the immediate reward is added in which simply adds in G more Gaussian components. To select the best action requires also estimating the expected immediate reward, an $O(GD^3N_B)$ calculation for each action. In total the computational complexity for a single backup is $O((FHKLN_\alpha|Z| + G)|A|D^3N_B)$.

However, as multiple backups are performed, K, which represents the number of components used to represent each α-function, will increase by a factor of $FHL|Z|$ after each backup.[3] Therefore, when performing backups at horizon T, the α-functions will have about $K(FHL|Z|)^T$ components, and the computational cost of a single further backup will be $O(((FHL|Z|)^T KN_\alpha|Z| + G)|A|D^3 N_B)$. Therefore the computational cost of performing a single backup will grow exponentially with the time step of the backup performed.

3.2.2 Approximating the α-functions

It is not computationally feasible to maintain all components over multiple backups. Instead, by carefully combining the components generated after each backup, we maintain a bounded set of α-functions. Since α-functions represent the value of executing a particular policy tree over the entire belief space, it is important to make the approximation as close as possible throughout the belief space.[4] In Porta et al. (Porta et al., 2006)'s work they faced a similar (though smaller) expansion in the number of components. In order to reduce the number of components used to represent the belief states and α-functions, they used a slight variant of Goldberger and Roweis method (Goldberger & Roweis, 2005) that minimizes the Kullback-Leibler (KL) distance between an original model $f(x)$ and the approximation $\tilde{f}(x)$

$$D_{KL}(f||\tilde{f}) = \int_x f(x) \log \frac{f(x)}{\tilde{f}(x)} dx \tag{3.4}$$

where f and $\tilde{f}$ are weighted mixtures of Gaussians. However, the KL distance is not particularly appropriate as a distance measure for the α-functions since they are not probability distributions. For example, the KL divergence can result in poor approximations in parts of the space where the original function has small values: if $f(x)$ is zero then regardless of the value of $\tilde{f}(x)$ the distance for that x is always zero.

We consider three alternate projection operators for computing an approximation of the α-function: highest peaks, minimizing the L2 norm, and minimizing the empirical max norm.

3.2.3 Highest Peaks Projection

A simple projection operator is to keep the N_C components with the highest peaks and discard all other components. The peak value of the i-th Gaussian component is simply

$$\frac{w_i}{(2\pi)^{D/2}|\Sigma_i|^{1/2}|}$$

where D is the state-space dimension, Σ_i is the variance of the i-th Gaussian, and w_i is its weight. It is simple and fast to compute all these values, and select the N_C largest components. Figure 3-5 shows an example of an original α-function, and the approximations obtained by keeping different numbers of components.

[3] To be precise, it will the new number of components is $FHL|Z|K + G$ but the main factor is due to the multiplication with $FHL|Z|$.

[4] Ideally we could restrict this approximation to the reachable belief space; however analyzing the reachable belief space in continuous-state POMDPs will be an area of future work.

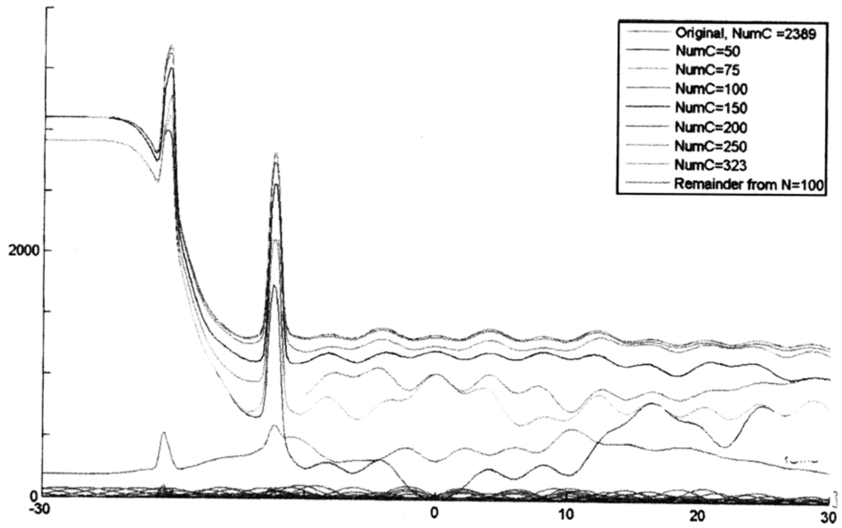
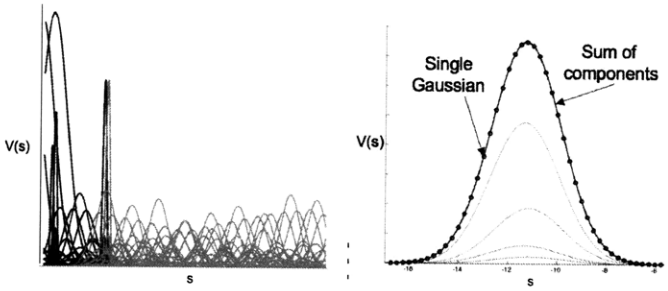


Figure 3-5: Highest peak projection. Here the Gaussian components composing the original function are sorted by their maximum value and the highest N_C of these are kept. This figure displays the resulting approximations as more and more components are kept, along with the resulting residual for when $N_C = 100$. Though this is a simple approximation to compute, it is clear from the figure that the resulting approximation can be quite poor when N_C is small.



(a) Assign clusters

(b) Fit cluster

Figure 3-6: L2 clustering for approximating the α-functions. We iteratively cluster components and assign them to one of the components in the approximate α function (see (a)). Then given all the components assigned to the same cluster, we refit the cluster parameters (see (b)). This process is then iterated until a convergence criteria is met.

3.2.4 L2 minimization

Another alternate distance measure without the shortcomings of the KL-divergence is the L2 norm: a small value means a good approximation $\tilde{\alpha}$ of the value function over the entire belief space.

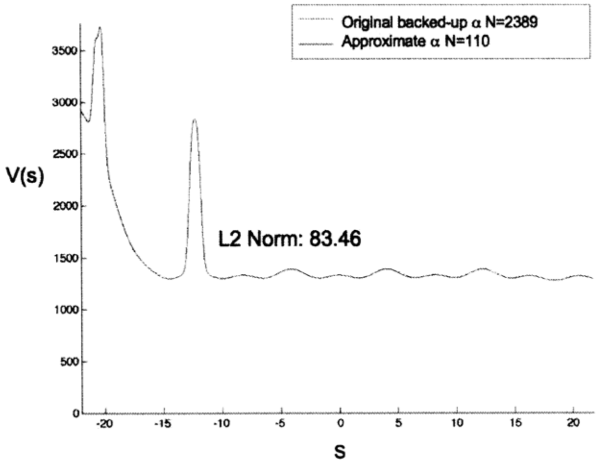


Figure 3-7: The original (in green) and approximated α-function (in blue). The original α-function was composed of 2,389 components. The approximate α only uses 110 components, yet in the plotted function is indistinguishable from the original α.

The L2 norm, or sum squared error, between two weighted sums of Gaussians is

$$\begin{aligned} &\int_s \left[\sum_i^{N_C} w_i \mathcal{N}(s;\mu_i,\sigma_i^2) - \sum_j^{N} w_j \mathcal{N}(s;\mu_j,\sigma_j^2)\right]^2 ds \\ = \;& \sum_i^{N_C} \sum_{i'}^{N_C} w_i w_i' \mathcal{N}(\mu_{i'};\mu_i,\sigma_i^2+\sigma_{i'}^2) + \sum_j^{N} \sum_j^{\prime N} w_j w_j' \mathcal{N}(\mu_{j'};\mu_j,\sigma_j^2+\sigma_{j'}^2) - \\ & 2\sum_j^{N} \sum_i^{N_C} w_j w_i \mathcal{N}(\mu_i;\mu_j,\sigma_j^2+\sigma_i^2). \end{aligned}$$

While there is no analytic solution for the parameters w_i, μ_i, σ_i^2 that minimizes this expression, we can find an approximation to the optimal solution using Zhang and Kwok's recent work on reducing the number components in kernel function mixture models (Zhang & Kwok, 2006). This work minimizes an upper bound on the L2 norm by clustering the original components into small groups, and fitting a single weighted Gaussian to each group. More precisely, the L2 norm can be upper bounded by a function of the L2 error of each cluster:

$$L2 \leq N_C \sum_{i=1}^{N_C} \int \left[w_i \mathcal{N}(s;\mu_i,\sigma_i^2) - \sum_{j \in S_i} w_j \mathcal{N}(s;\mu_j,\sigma_j^2) \right]^2 ds$$

where S_i is the i-th cluster and N_C is the total number of components in the approximation $\tilde{\alpha}$. The parameter fitting procedure is simplified by the clustering procedure. The complete procedure can be performed iteratively, by creating clusters through assigning all components in the original function to their closest (in the L2 norm sense) component in the approximation, refitting a single component for each cluster, and repeating (see Figure 3-6).

Though we can use this procedure to optimize an upper bound to the L2 norm, we would also like to constrain the exact L2 norm error to be within some threshold. To do this, we can formulate the approximation step as a constrained optimization problem, using the objective function from Zhang and Kwok, but requiring that the final approximation $\tilde{\alpha}$ both lies below a maximal L2 norm threshold, and contains no more than a fixed maximum number of components, N_{Cmax}. This can be stated mathematically as follows:

$$\arg \min_{w_i, \mu_i, \sigma_i^2, N_C} N_C \sum_{i=1}^{N_C} \int \left[w_i \mathcal{N}(s; \mu_i, \sigma_i^2) - \sum_{j \in S_i} w_j \mathcal{N}(s; \mu_j, \sigma_j^2) \right]^2 ds$$
$$s.t. ||\tilde{\alpha} - \alpha||_2 \leq t \, \& \, N_C \leq N_{Cmax}$$

where t is L2 norm threshold.

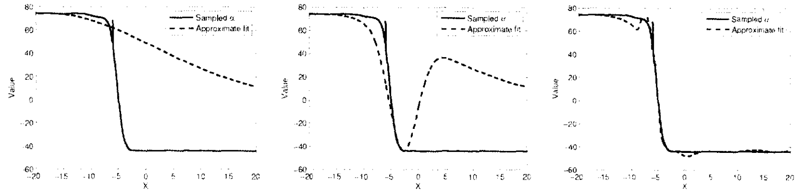
We initialize the components of the approximation with a random subset of the original components. The remaining original components are then clustered to their closest (in the L2 norm) component in the approximation, and then a single component is refit for each cluster. In order to ensure the parameter initialization lies within the feasible region spanned by the constraints, we compute the L2 norm of this initialization, and discard solutions that do not satisfy the L2 norm constraint. Note discarding soluations that do not satisfy the L2 norm constraint provides a principled mechanism for selecting the number of components constituting the approximation: the number of components N_C in the approximation is increased until either an initialization is found that lies within the feasible region of the constraints, or N_C no longer satisfies the second constraint. If both constraints cannot be satisfied, N_C is set to N_{Cmax}.

Once the parameters are initialized and M is fixed, we follow Zhang and Kwok's procedure for optimizing. Experimentally this approach was found to produce reasonable results: Figure 3-7 displays an example where the L2 norm is still fairly large, the original and approximate α-functions are indistinguishable to the naked eye.

3.2.5 Point-based minimization

One of the limitations of the L2-based approximation method presented is that it may be computationally too slow in practice. An alternate choice is a faster heuristic method which greedily minimizes the max norm at a subset of points along the value function.

In this method, the α-function is first sampled at regular intervals along the state space. These samples are considered potential residuals r. First the largest magnitude residual (either positive or negative) is found and a Gaussian is placed at this point, with a height equal to the residual and variance approximated by estimating the nearby slope of the function. This Gaussian is added to the set of new components. Then a new set of residuals is computed by estimating this Gaussian along the same intervals of state space, and subtracting its value from the original residuals: $r = r - w_i \mathcal{N}(s|\mu_i, \Sigma_i)$. This creates a new set of residuals. We now find the maximum magnitude sample of this set of residuals and fit a second Gaussian in the same manner as before. This process is repeated until the number of new Gaussians reaches the fixed size of our representation for the α-functions. See Figure 3-8 for an illustration of this process. Note that at each round during this process we are adding a Gaussian component at the location which currently has the max norm error at the sampled points. Therefore this algorithm takes a greedy approach to minimizing the max norm between the approximate $\tilde{\alpha}$-function and the original α-function along the set of



(a) 1 component (b) 2 components (c) 15 components

Figure 3-8: Approximating the α-functions. We use a simple residual fitting technique. First the function is sampled. Then a Gaussian component is added to reduce the highest component. This process is repeated until the maximum number of Gaussians is reached.

sampled points.

3.2.6 Planning

We now have the major components necessary to apply a point-based approach to POMDP planning: specifically, a belief update operator, a value function backup procedure, and a projection operator to keep the number of parameters used to represent the value function bounded. Algorithm 3 outlines the SM-POMDP algorithm. Inspired by results in discrete-state planners (Smith & Simmons, 2004; Shani et al., 2007) we create a belief set B by sampling belief states in a set of short trajectories, though in our case we simply perform random actions from an initial starting belief state b_0. We initialize the value function as a single α function in a domain-dependent manner. Starting with this initial α function, we iteratively perform value function backups for the chosen belief set B. At each round, we select a trajectory from our belief set and back up the beliefs in it in reverse order. In problems where a number of steps must be executed in order to reach a high reward state this approach was found to be superior to randomly backing up belief points.

3.3 Analysis

We now provide an analysis of the SM-POMDP algorithm. We are interested in bounding the potential difference between the optimal value function, denoted by V^* and the value function computed by the SM-POMDP algorithm, denoted by V^{SM}. Let H represent an exact value function backup operation, H^{PB} a point-based backup operation, and A the projection operator which maps the value function to a value function with a smaller number of components. Our goal is a bound on the error between the n-step optimal value function V_n^* and the SM-POMDP value function V_n^{SM}. The total error $||V^* - V_n^{SM}||_\infty$ is bounded by the n-step error plus the error between the optimal n-step and infinite horizon value functions, $||V^* - V_n^*||_\infty$ which is bounded by $\gamma^n||V^* - V_0^*||_\infty$. We will show the n-step error can be decomposed into a term due to the error from point-based planning and

Algorithm 3 SM-POMDP

```
Input: $N_{traj}$ (number of trajectories), $N_T$ (number of beliefs per trajectory), POMDP
for i=1:$N_{traj}$ do {Generate belief set $B$}
    $b = b_0$
    Draw an initial state $s$ from $b_0$
    for j=1:$N_T$ do
        Select an action $a$
        Sample a transition given $s$,$a$ and dynamics model
        Sample an observation $z$
        $b^{a,z} = BeliefUpdate(b, a, z)$
        Add belief to set: $B[j, i].b = b^{a,z}$
        $b = b^{a,z}$
    end for
end for
Initialize value function
loop {Perform planning until termination condition}
    Select a trajectory $r$ at random
    for t=T:-1:1 do
        $\alpha_{new}$ = ValueFunctionBackup(B[r,t].b)
        Project to smaller rep: $\alpha'_{new} = ProjectDownToN_CComponents(\alpha_{new})$
        if Value of $B[r, t].b$ is better under new $\alpha'_{new}$ than it was before then
            Add $\alpha_{new}$ to value function
        end if
    end for
end loop
```

a term due to the error from the projection to a fixed number of components. [5]

Theorem 3.3.1. *Let the error introduced by performing a point-based backup instead of an exact value backup be $\epsilon_{PB} = ||HV - H^{PB}V||_\infty$ and let the error introduced by projecting the value function down to a smaller number of components be $\epsilon_{proj} = ||V - AV||_\infty$. Then the error between the optimal n-step value function V_n^* and the value function computed by the SM-POMDP algorithm is bounded by*

$$||V_n^* - V_n^{SM}||_\infty \leq \frac{\epsilon_{PB} + \epsilon_{proj}}{1 - \gamma}$$

Proof. We proceed by first re-expressing the error in terms of the backup operators, and then add and subtract a new term in order to separate out the error due to performing

[5] In this analysis we ignore possible error due to the approximate normalization of the observation and mode models, assuming that enough Gaussians are used to represent these functions that they can be made to sum arbitrarily close to 1 and therefore contribute negligible error.

backups on different value functions, versus the different backup operators themselves:

$$\begin{aligned}||V_n^* - V_n^{SM}||_\infty &= ||HV_{n-1}^* - AH^{PB}V_{n-1}^{SM}|| \\ &\leq ||HV_{n-1}^* - HV_{n-1}^{SM}||_\infty + ||HV_{n-1}^{SM} - AH^{PB}V_{n-1}^{SM}||_\infty \\ &\leq \gamma||V_{n-1}^* - V_{n-1}^{SM}||_\infty + ||HV_{n-1}^{SM} - AH^{PB}V_{n-1}^{SM}||_\infty\end{aligned}$$

where the result follows by the use of the triangle inequality.

We then again add and subtract the same term in order to separate the error introduced by performing a point-based backup, from the error introduced from the projection operator, and again use the triangle inequality:

$$\begin{aligned}||V_n^* - V_n^{SM}||_\infty &\leq \gamma||V_{n-1}^* - V_{n-1}^{SM}||_\infty + ||HV_{n-1}^{SM} - H^{PB}V_{n-1}^{SM}||_\infty + \dots \\ &\quad + ||H^{PB}V_{n-1}^{SM} - AH^{PB}V_{n-1}^{SM}||_\infty \\ &\leq \gamma||V_{n-1}^* - V_{n-1}^{SM}||_\infty + \epsilon_{PB} + \epsilon_{proj}.\end{aligned}$$

This is a geometric series, therefore

$$\begin{aligned}||V_n^* - V_n^{SM}||_\infty &\leq \frac{(1-\gamma^{n+1})(\epsilon_{PB} + \epsilon_{proj})}{1-\gamma} \\ &\leq \frac{\epsilon_{PB} + \epsilon_{proj}}{1-\gamma}.\end{aligned}$$

□ □

In the following sections we will discuss bounds on ϵ_{proj}, but the remainder of this section will focus on ϵ_{PB}.

Pineau, Gordon and Thrun (Pineau et al., 2003a) previously provided a proof for bounding the error induced by performing a point-based backup, ϵ_{PB} for discrete-state POMDPs. We now provide a similar proof for continuous-state POMDPs. $d_{KL}(p, q)$ will denote the Kullback-Leibler divergence between two probability distributions p and q. B is the set of points in the belief set used to perform point-based backups and Δ is the full belief space.

Lemma 1. *Let w_{min} be the lowest weight and w_{max} be the highest weight of the weights used to specify the original weighted sum of Gaussians reward function. Let $|\Sigma_{min}|$ be the smallest determinant of the variance of the Gaussians, and let N_C be the maximum number of components used to represent the weighted sum of Gaussians. Then the error introduced by performing a point-based backup is at most $\epsilon_{PB} = \frac{N_C(w_{max}-w_{min})\max_{b'\in\Delta}\min_{b\in B} 2\sqrt{2d_{KL}(b,b')}}{(1-\gamma)(2\pi)^{D/2}|\Sigma_{min}|^{1/2}}$.*

Proof. Following Pineau et al. (2003a) let $b' \in \Delta$ be the belief at which the point-based value function has its largest error compared to the optimal value function, and let $b \in B$ be the closest sampled point in the belief set B to b' where close is defined in terms of the KL-divergence. Let α' and α be the two functions which respectively maximize the expected value of b' and b. By not including α' the point-based operation makes an error of at most $\int_s \alpha'(s)b'(s) - \alpha(s)b'(s)ds$.

To bound this error, we first add and subtract the same term:

$$\begin{aligned}\epsilon_{PB} &\leq \int_s \alpha'(s)b(s) - \alpha(s)b'(s)ds \\ &= \int_s \alpha'(s)b(s) - \alpha(s)b(s) + \alpha(s)b(s) - \alpha(s)b'(s)ds.\end{aligned}$$

The value of α' at b must be lower than α since α is defined to give the maximal value. Therefore

$$\begin{aligned}\epsilon_{PB} &\leq \int_s \alpha'(s)b(s) - \alpha'(s)b(s) + \alpha(s)b(s) - \alpha(s)b'(s)ds \\ &= \int_s (\alpha'(s) - \alpha(s))(b(s) - b'(s))ds \\ &\leq \int_s |(\alpha'(s) - \alpha(s))(b(s) - b'(s))|ds.\end{aligned}$$

Hölder's inequality can be used to bound this expression as follows

$$\epsilon_{PB} \leq \left(\int_s |\alpha'(s) - \alpha(s)|^{\infty} ds\right)^{1/\infty} \left(\int_s |b(s) - b'(s)|ds\right)$$

where the first term is the max norm of $\alpha' - \alpha$ and the second term is the 1-norm between b and b'. Since there is no analytical method to compute the 1-norm between two Gaussians or weighted sums of Gaussians, we now relate this 1-norm to the KL-divergence using the results from Kullback (Kullback, 1967) (included for completeness as Lemma 11 in the Appendix):

$$\epsilon_{PB} \leq ||\alpha'(s) - \alpha(s)||_{\infty} 2\sqrt{2d_{KL}(b, b')}.$$

The worst error in immediate reward between two value functions occurs when all N_C components have the same mean and the smallest possible variance Σ_{min}, and where one value function has all weights set at w_{max} and the other value function has all weights set at w_{min}. This yields an error of $\frac{N_C(w_{max} - w_{min})}{(2\pi)^{D/2}|\Sigma_{min}|^{1/2}}$. The infinite horizon values in this worst case would magnify these errors by at most $1/(1 - \gamma)$. Therefore the max norm error between two value functions is bounded by

$$||\alpha' - \alpha||_{\infty} \leq \frac{N_C(w_{max} - w_{min})}{(1 - \gamma)(2\pi)^{D/2}|\Sigma_{min}|^{1/2}}$$

and so the full point-based error is bounded as follows:

$$\epsilon_{PB} \leq \frac{N_C(w_{max} - w_{min})2\sqrt{2d_{KL}(b, b')}}{(1 - \gamma)(2\pi)^{D/2}|\Sigma_{min}|^{1/2}}.$$

□

Belief points can be chosen to reduce this error, such as in Pineau's PBVI (Pineau et al., 2003a) algorithm which adds belief points that have the largest error (according the 1-norm) along some trajectory relative to the existing beliefs in B. One slight complication is how to estimate the KL-divergence between the belief points. If the belief points are represented as single Gaussians then there exists a closed form expression for the KL-divergence between two beliefs. There exists no general closed form representation for the KL-divergence between two weighted sums of Gaussians. However, a recent paper by Hershey and Olsen (2007) investigates a number of approximations to this quantity, including an upper bound that is fast to compute, which can be utilized to ensure that the error estimated is an upper bound to the true error.

The second cause of error in the final value function is due to the projection step, ϵ_{proj}, and will depend on the particular projection operator used. We next consider four different operator choices and discuss where possible their associated ϵ_{proj}. We also consider the computational complexity of the resulting SM-POMDP algorithm under each of these choices, considering the cost of the projection operator and the backup operator. The results of this section are summarized in Table 3.1.

3.3.1 Exact value function backups

If the value function backups are performed exactly then ϵ_{proj} will equal zero. Therefore exact value backups offer some appealing performance guarantees. Unfortunately, as discussed in Section 3.2.1, there is a significant computational cost to these guarantees. This cost will often become infeasible when the backup is iteratively performed several times and F, H, L, or $|Z|$ are large.

We next consider the three projection operators.

3.3.2 Highest peak projection analysis

Take the projection method described in Section 3-5 and assume that prior to the backup each α consists of N_C Gaussians. Then the backup procedure creates a new α with $FHL|Z|N_C+G$ components. Therefore, assuming the Gaussians are sorted in descending order of peak value, the error introduced by projecting to N_C components is the value of the discarded $FHL|Z|N_C+G-N_C$ Gaussians,

$$\epsilon_{proj} = \sum_{j=N_C+1}^{FHL|Z|N_C+G} w_j \mathcal{N}(s|\mu_j, \Sigma_j).$$

In the worst case all of these Gaussians would be centered at the same location, yielding an error of

$$\epsilon_{proj} \leq \frac{(FHL|Z|N_C+G-N_C)\max_j w_j}{(2\pi)^{D/2}\min_j |\Sigma_j|^{1/2}}. \tag{3.5}$$

Equation 3.5 an upper bound on the projection error which can be used to bound the overall error between the SM-POMDP value function and the optimal value function. In addition, the computational cost of this procedure is fairly cheap. It requires calculating the determinant of each component, which is an $O(D^3)$ operation where D is the number of state dimensions, and a sorting algorithm to identify the N_C components with the largest peak. Quicksort or a similar algorithm can be used which has a worst case complexity of the number of items squared. Therefore the computational complexity of a highest peak projection is

$$O\left((FHL|Z|N_C + G)D^3 + (FHL|Z|N_C + G)^2\right).$$

However, despite its appealing theoretical guarantees, unless a large number of components is used, the empirical quality of the approximation may not be as good as the other two projection methods (as is illustrated by comparing Figure 3-5 to Figures 3-7 and 3-8).

3.3.3 L2 minimization analysis

Section 3.2.4 outlined a projection method where the objective is to minimize the L2 norm between the original and projected value function. A strength of this approach is that it is possible to analytically evaluate the final L2 norm. However, in continuous spaces L2 is not necessarily an upper bound to the max norm. This can be seen by considering the case of a delta function: the L2 norm of this function is zero, but the max norm is infinity. Though this approach does not provide a direct estimate of ϵ_{proj}, it does try to closely approximate the value function over the entire state space, rather than focusing on the worst error.

The algorithm presented in Section 3.2.4 is an iterative procedure. It requires repeatedly computing the L2 norm between two weighted sets of Gaussians, one with $FHL|Z|N_C+G$ components, and one with N_C components. This involves computing the determinant of D-dimensional matrices, and the full computational complexity of this projection operator is $O((FHL|Z|N_C+G)MD^3)$. This calculation is repeated many times during the procedure.

3.3.4 Point-based minimization analysis

In Section 3.2.5 we discussed an approach that greedily minimizes the empirical max norm of a set of Q points sampled along the value function. Let $\tilde{\epsilon}_{proj}$ represent the empirical max norm computed. As the number of points sampled goes to infinity, this error will converge to the true projection error ϵ_{proj}:

$$\lim_{Q\to\infty} \tilde{\epsilon}_{proj} = \epsilon_{proj}.$$

Unfortunately we do not know of any finite Q guarantees on the relationship between the empirical max norm and the true max norm error. However, in practice, a small max norm error on the sampled points often indicates that there is a small max norm between α and $\tilde{\alpha}$.

This method requires computing the value function at a set of Q points which is an $O(Q(FHL|Z|N_C + G)D^3)$ operation due to the need to compute the inverse and determinant of the Gaussian variances. However, the number of points Q necessary to compute a good estimate of the value function will increase exponentially with the state dimension:

essentially this procedure requires computing a discretized estimate of the value function, and then refits this estimate with a new function. If Q_1 is the number of points used to estimate a one-dimensional value function, then Q_1^D will be the number of points needed to estimate a D-dimensional function. Therefore we can re-express the computational complexity as

$$O(Q_1^D(FHL|Z|N_C + G)D^3).$$

Generally this projection operator will work best in low dimensional domains.

Before summarizing these approaches in Table 3.1, we need to also briefly provide an analysis of the computational complexity of performing backups when the value function is projected to a smaller number of components.

3.3.5 Computational complexity of backup operator when use a projection operator

The computational cost analysis is similar to the exact value function case presented in Section 3.2.1, with one important alteration: the number of components in the resulting α-functions is now restricted. As before, let N_C be the maximum number of components in an estimated α-function during planning: N_C can either be specified in advance, or computed during planning if the number of components is dynamically adjusted in order to prevent the approximation step from exceeding a maximum desired error. In this case the computational complexity of performing a single backup never exceeds $O(FHLN_C N_\alpha|A||Z|D^3 N_B)$, regardless of the horizon at which the backup is performed (or in other words, regardless of the number of prior backups). This means that the computational cost is bounded (assuming that N_C does not grow as a direct function of the horizon T). The choice of N_C provides the above mentioned parameter for tuning between the quality of the resulting solution and the computational cost required to compute that solution. In our experiments we show it is possible to achieve good experimental results by fixing the value of N_C in advance.

3.3.6 Analysis Summary

Table 3.1 summarizes the computational complexity of performing a value function backup, and the additional error and computational overhead introduced by projecting the resulting α-function down to a smaller number of components. In general, it will be computationally necessary to perform a projection if more than a couple value function backups are required (that is, if T is greater than 2 or 3). To provide some intuition for this, we provide a simple example. Assume the following problem setup:

- There are 2 observations ($||Z|| = 2$) each represented by 5 components ($L = 5$)
- There are 2 actions ($|A| = 2$) each with 3 modes ($H = 3$) and each mode is represented using 5 components ($F = 5$)
- Each reward is represented with 3 components ($G = 3$)

able 3.1: Summary of the different projection operators in the SM-POMDP algorithm, and estimates of the error they introduce, and ne associated computational complexity of using them as part of a planning algorithm. D is the number of state dimensions, $|A|$ is the umber of actions, $|Z|$ is the number of observations, N_B is the number of components in the belief state, K is the number of components 1 the original α-functions, and the remaining symbols are defined in the text.

Projection Algorithm	ϵ_{proj}	Computational Complexity of the T-th Step		Empirical Performance
		Projection Operator	Backup Operator	
No projection	0	0	$O(((FHL\|Z\|)^T KN_\alpha\|Z\| + G)\|A\|D^3 N_B)$	Computationally intractable
Highest Peak	$\frac{(FHL\|Z\|N_C+G-M)\max_j w_j}{(2\pi)^{D/2}\min_j \|\Sigma_j\|^{1/2}}$	$O((FHL\|Z\|N_C + G)D^3 + (FHL\|Z\|N_C + G)^2)$	$O((FHL\|Z\|N_C N_\alpha\|Z\| + G)\|A\|D^3 N_B)$	Poor quality approximation
Point-based	$\lim_{Q\to\infty} \tilde{\epsilon}_{proj} \quad \to \quad \epsilon_{proj}$. The empirical norm $\tilde{\epsilon}_{proj}$ can quickly become quite small after a few components.	$O(Q_1^D(FHL\|Z\|N_C + G)D^3)$	$O((FHL\|Z\|N_C N_\alpha + G)\|A\|D^3 N_B)$	Good quality approximation and faster than L2 minimization in low-dimensional domains used
L2 minimization	Computes ϵ_{L2} which is not guaranteed to dbe an upper bound to ϵ_{proj}. However it tries to fit a good approximation to the value function over the entire state space.	$O(N_C(FHL\|Z\|N_C + G)D^3)$	$O((FHL\|Z\|N_C N_\alpha + G)\|A\|D^3 N_B)$	Good quality approximation but slower than point-based

- $N_\alpha = 1$
- The original α-function is represented as a re-scaled reward function with 3 components ($K = 3$)
- $N_B = 1$
- $N_C = 50$

Then the number of components after a single point-based backup is

$$FHL|Z|K + G = 5 * 3 * 5 * 2 * 3 + 3 = 453.$$

The number of components after the second exact point-based backup can then be

$$FHL|Z|453 + G = 5 * 3 * 5 * 2 * 453 + 3 = 67503 = (FHL|Z|)^2K + FHL|Z|KG + G^2,$$

and on the third round

$$FHL|Z|67503 + G = 10125453 \approx (FHL|Z|)^3K.$$

In contrast, if the α-function is projected to N_C components after each backup, the number of new components created during a backup is fixed at

$$FHL|Z|N_C + G = 5 * 3 * 5 * 2 * 50 + 3 = 7503.$$

Therefore, as long as the projection operator is cheaper than manipulating and updating a set of components that grows as $(FHL|Z|)^T$ where T is the number of backups, then projecting will be significantly faster than maintaining all components.

Empirically we found that the max-norm projection method was faster than the L2 norm projection method, and that both gave significantly higher quality results than the highest peaks method. Therefore we used the max-norm projection method in all experiments.

The key advantage of SM-POMDP over prior techniques is its ability to handle a broader class of dynamics models. Therefore we examined its performance on three simulated examples that involve switching and/or multi-modal dynamics. The first example is a navigation domain where a robot must locate a hidden power supply. This problem exhibits switching dynamics because the robot's transitions depend on whether it is currently in the free space of the hallway, or near one of the walls. In the second example we consider a faulty robot trying to hop across treacherous terrain to reach a goal destination. This problem exhibits multi-modal dynamics since the robot's attempts to jump only sometimes succeed. In the final example we consider a two-dimensional unmanned aerial vehicle simulation where one agent must avoid another agent.

Except where otherwise noted, we focus on comparing SM-POMDP to a linear-model parametric continuous-state planner, such as the planner developed by Porta et al. (Porta et al., 2006). This is the most similar planner to SM-POMDP as only the dynamics models are changed, and is therefore an important benchmark.

Brooks et al. (2006) took an alternate approach to continuous-state planning by restricting the belief state to be a unimodal Gaussian, and then planned by discretizing the

Gaussian parameters and using Fitted Value Iteration over that representation. However, this approach will be limited in any domain that requires multi-modal beliefs to achieve good performance. In addition, the resulting plan quality depends on the resolution of the discretization of the parameter space, which is hard to know in advance and has a direct impact on computational cost. Thrun's Monte Carlo POMDP (Thrun, 2000) approach can handle arbitrary dynamics models but requires sampling to perform the belief updates and to compute the value function estimate. This approach will generally be more expensive than analytic computations, so this approach is not investigated here.

In all examples SM-POMDP outperformed a linear-model continuous-state planner. α-functions were projected down to at most 50 components using the empirical max norm projection method. Beliefs were restricted to have four components, and were projected down using the KL-divergence technique from Goldberger and Roweis (2005) which is more appropriate for beliefs than value functions since beliefs represent a probability distribution over states.

Porta et al. had previously demonstrated that their planner was faster than a standard discrete-state planner Perseus (Spaan & Vlassis, 2005a) on a simulation problem, demonstrating the benefits of parametric planners that automatically adjust the parameterization of the value function during planning. In the first experiment we confirm that this advantage holds for SM-POMDP which can handle domain dynamics that are poorly approximated by a single linear Gaussian model. In addition we show this advantage when comparing to a state-of-the-art discrete-state planner HSVI2 (Smith & Simmons, 2005) which is known to outperform Perseus (Spaan & Vlassis, 2005a). We then focus our experimental efforts on demonstrating the additional flexibility and modeling power of SM-POMDP that allows it to find better plans that linear Gaussian POMDP planners.

3.3.7 Finding Power: Variable Resolution Example

In the first experiment a robot must navigate a long corridor ($s \in [-21, 21]$) to find a power socket which is located at -16.2. The robot can move left or right using small or large steps that transition it 0.1 or 5.0 over from its current location plus Gaussian noise of standard deviation 0.01. If the robot tries to move too close to the left or right wall it will bump into the wall. This causes the dynamics to exhibit switching: if the robot is in the middle of the hallway it will have different dynamics than near the walls. The robot can also try to plug itself in, which leaves the robot at the same location. All movement actions receive a reward of 0.05. If the robot plugs itself in at the right location it receives a reward of 12.6 (modeled by a highly peaked Gaussian); otherwise it receives a lesser reward of 5.8. Therefore it is least preferable to move around, and it is best to plug in at the correct location. The power supply lies beneath the robot sensors so the robot is effectively blind. However, if the robot knows the true world dynamics, it can localize itself by travelling to one end of the hallway, reasoning that it will eventually reach a wall.

In addition to the switching dynamics, this problem is also interesting because the resulting value function is smooth for large parts of the state space, intermixed with small sections that are rapidly changing. This domain provides another demonstration of where continuous-state parametric planners' dynamically adjustable value function representation may have an advantage over discrete-state planners. We compare to HSVI2 (Smith &

Simmons, 2005), a highly optimized, state-of-the-art discrete-state POMDP planner.

We present results for the discrete-state HSVI2 planner, a linear continuous-state planner and SM-POMDP. The two continuous-state planners were run on 100 trajectories gathered by starting in a random state $s \in [-21, 21]$ with a Gaussian approximately uniform belief and acting randomly for episodes of 10 steps (leading to a total set of 1000 belief states). The robot can always achieve at least the reward associated with only executing $PlugIn$. Therefore we used the $PlugIn$ action reward function, scaled to account for infinite actions and discounting (multiplied by $1/(1-\gamma)$) as the initial lower bound value function.

All algorithms require a stopping threshold. We set our continuous-hybrid approach to do at least 10 rounds, and halt when $\sum_b V_t(b) - \sum_b V_{t-1}(b)| \leq 0.001$. HSVI2 halts when the upper and lower bounds on the value of the initial belief state falls below a certain threshold. The default setting of ϵ is 0.001. We ran experiments using this value and also a higher value of 1.5 which corresponded roughly to a ratio measure we originally used.

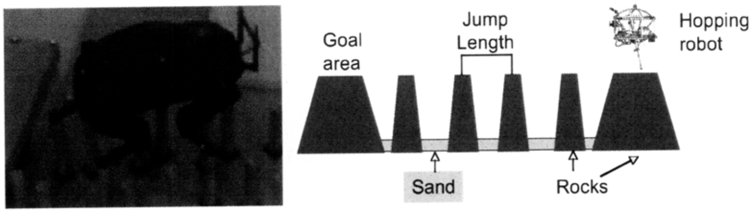
The value functions produced by each planner were tested by computing the average reward received over 100 episodes of 50 steps/episode using the policy associated with the α-functions/vectors in the value function. At the start of each episode the robot was placed at a state s that was chosen randomly from the uniform distribution spanning [-19,19]. The robot's belief state is initialized to be a set of 4 high variance Gaussians spanning the state space. See Table 3.2 for results.

Our hybrid model found a good policy that takes big and small actions to first localize the belief (at the wall) and then taking three more steps to reach the power socket. The linear model continuous-state POMDP planner was faster but failed to find a good policy since it could not model the state-dependent dynamics near the wall edges, and there are no unique observations. Therefore the linear planner believed localization was hopeless.

The discrete-state solution performed poorly at coarse granularities since the $PlugIn$ reward gets washed out by averaging over the width of a too wide state. At fine state granularities the discrete approach found a good policy but required more time: our continuous-state planner found a solution significantly faster than the coarsest discrete-state planner that can find a good solution. This result shows that Porta et al.'s previous demonstration of the benefit of parametric representations over discrete-state representations (Porta et al., 2006) also can hold in some more domains with more complex dynamics.

Table 3.2: Power Supply Experiment Results. There was no significant difference between the rewards received by the good policies (marked by a *).

Models	Continuous-state		HSVI2: Discretized (Number of States)				
	Linear	SM-POMDP	840	1050	1155	1260	2100
Time(s) $\epsilon = 0.001$	30.6	**548**	577	28906	86828	**23262**	75165
Time(s) $\epsilon = 1.5$	n/a	**n/a**	-	9897	34120	**5935**	23342
Reward	290	**465***	290	290	290	**484***	484*



(a) Tricky navigation

(b) Observation Model

Figure 3-9: (a) displays a motivating example for this experiment: navigation over a challenging set of poles by a quadruped robot. (b) shows the particular experimental domain used. A robot must try to reach the other large rock by hopping across the obstacles and then signalling it has reached the end.

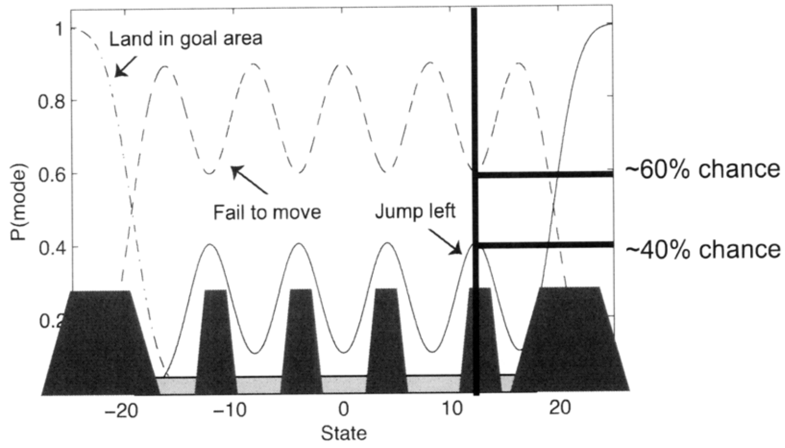
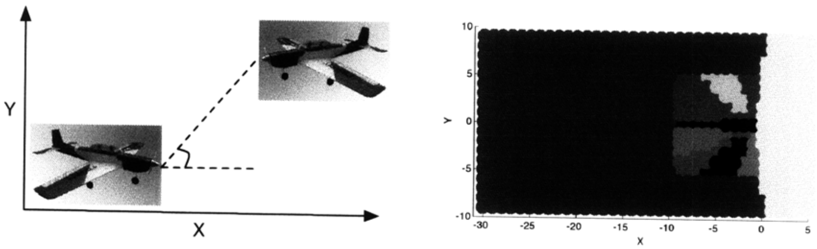


Figure 3-10: Multi-modal state-dependent dynamics of *Jump*. A given mode's probability (such as step left) varies over the state space, and more than one mode is possible for a state.

3.3.8 Locomotion over Rough Terrain: Bimodal Dynamics

In robotic legged locomotion over rough terrain (such as DARPA's LittleDog project) a robot may need to traverse an uneven rocky surface to reach a goal location (see Figure 3-9a). Our example is inspired by this problem. The robot starts on an initial flat surface and must traverse an area with 4 rocks separated by sand to reach a goal location (see Figure 3-9b). At each step the robot can attempt to step forward or signal it is done. The robot is faulty and works best on hard surfaces: at each step it may succeed in executing a step or stay in the same place. Figure 3-10 displays the multi-modal state-dependent nature of the dynamics. The robot is receives a penalty of -59 for stepping into the sand, and -1.1 for each step on the rocks. A $Signal$ action results in a reward of ≈ 19.5 if the robot has reached the final location, or -0.1. All rewards are specified using a weighted sum of Gaussians. The observation model provides a noisy estimate of where the robot is (sand, rock 1-4, start or finish).

We tested SM-POMDP and a linear model that averages the bimodal distributions. The



(a) Plane Setup

(b) Observation model

Figure 3-11: UAV experiment. (a) Shows the basic problem setup: an agent must avoid an incoming UAV. (b) Shows the observation model. Different colors represent the placement of the centers of the Gaussians associated with different observations. When the agent is close to the incoming UAV it receives a fairly high resolution estimate of the relative bearing to the incoming UAV, otherwise the agent receives only coarse observations.

models were tested in a manner similar to the prior experiment (except using 100 beliefs). The agent can always perform at least as well as performing the $Signal$ action forever so the $Signal$ action reward was used as an initial value function, scaled to account for discounting and performing the action indefinitely. The results are displayed in the following table:

	Average Total Reward Received
SM-POMDP	302.8
Linear Model	-4.7

Since the hybrid model could correctly represent that a step from the initial platform would keep the robot on the platform or move it to the first rock, it could find a good policy of repeatedly trying to step and eventually reached the goal platform. In contrast, the linear model performed poorly because its dynamics model led it to believe that stepping from the platform would result in landing in a relatively undesirable and hard to escape sandpit. Instead the linear model policy simply signaled immediately.

A belief compression technique that assumes a unimodal belief state, such as Brooks et al. (2006) approach, cannot perform optimally in this domain. This is because the belief state resulting from an action is inherently bimodal due to the faulty dynamics model, and a unimodal approach must, for example, average, or select one peak of the belief. When we evaluated our SM-POMDP approach with only a single component per belief, the resulting average reward was 255, significantly lower than the average reward of 302.8 received by SM-POMDP.

3.3.9 UAV avoidance

In our final experiment we investigated the performance of SM-POMDP on a two-dimensional collision avoidance problem. The challenge is for the agent, an unmanned aerial vehicle, to successfully avoid crashing with an incoming aerial vehicle. The state is represented by the relative x and y between the two planes (see Figure 3-11a). The agent has 6 possible actions: it can go slowly forward ($x = +1, y = +1$), slowly up and forward ($x = +1, y = +0$), slowly down and forward ($x = +1, y = -1$), fast forward ($x = +5, y = +0$), fast forward and up ($x = +5, y = +1$), or fast forward and down ($x = +5, y = -1$). The incoming UAV always moves forward (from its perspective, so that it comes closer to the agent) but it can also stochastically choose between going straight, up and down. Therefore the dynamics model has 3 modes which are equally likely and depend on the (unknown) action of the incoming UAV. The agent incurs a small cost (of 3) for each forward action, a slightly larger cost (5) for going up or down and a large cost of 200 for getting within 1 unit of the incoming UAV. However, after the agent has passed the incoming UAV, it is rewarded by having the cost of actions drop to zero. The agent receives one of 11 discrete observations. If the agent is close to the incoming UAV, the agent receives a discretized bearing estimate to the incoming UAV. Otherwise the agent is too far away and receives a broad range estimate that essentially specifies whether the agent is in front or behind the other UAV. As with all the world models, these observation models are represented by a weighted set of Gaussians. The centers of those Gaussians are shown in Figure 3-11b.

We compared SM-POMDP to the linear model POMDP planner. Both approaches used 1000 belief points during planning. The resulting approaches were evaluated on 200 episodes, each of length 30 steps. The agent is always started between $(-30, -28)$ in the x direction, and $(-4, 4)$ in the y direction. The results are displayed in Table 3.3. This task is fairly easy, but SM-POMDP still outperformed the linear model. SM-POMDP yielded a policy with a lower average cost per episode of 47.2, whereas the linear model incurred the greater cost of 50.5. The difference was statistically significant (t-test, p < 0.001). This improvement in performance is likely to be due to the fact that SM-POMDP had a better model of the incoming UAV's dynamics: in order to model the UAV as having a single dynamics model, the linear dynamics used a dynamics model with a higher variance than is really present. Using a higher variance linear model made it appear that the incoming UAV could transition to some states which the true model cannot reach, and therefore the linear model was overly conservative.

Table 3.3: UAV avoidance average cost results: SPOMDP outperforms the linear model.

Algorithm	Average Cost/Episode
SM-POMDP	47.2
Linear Model	50.5

3.4 Limitations

Although our experimental results demonstrated that SM-POMDP can outperform discrete-state approaches and prior parametric approaches in certain domains, it has a number of significant weaknesses.

The first is due to the combinatorial explosion of the number of components required to exactly represent the belief or value function as belief updates or value function backups are performed. In all domains except those with an extremely small number of components per world model, this requires a projection operator to keep the number of components at a manageable level. However this projection steps adds a significant additional computational cost to the algorithm running time. Typically the number of components chosen for the projected representation will be decided by computational constraints, rather than using theoretical bounds on the approximation quality to guide the number of components selected. This means that often the potential error incurred during this projection step may be high even when the cost of performing the projection is still significant. An alternative approach employed by Poupart et al. (2006) to handle a similar situation was to select a small fixed set of components to project onto, and essentially simply change the weights on this basis set after each value backup. This allows the problem to be reframed so that the old value function is mapped immediately to the projected new value function without ever representing the new value function with a larger number of components. Using a fixed set of bases for the value function has the potential to greatly speed up the computational cost of SM-POMDP. However using a fixed set of bases introduces the challenge of how to select a good set of bases. In addition, if only the set of fixed bases is used, and the full backed up value function is not represented, then it will typically be hard to get any bounds on the error of the resulting solution. However, this could be a useful approach for practical applications.

An independent challenge is compactly expressing the world models themselves as the state dimension increases. Recall that the probability of a discrete mode was conditioned on a continuous-valued state variable $p(m|s,a)$, and a weighted set of Gaussians were used to express this discrete-conditioned-on-continuous distribution. This representational choice was made in order to achieve closed-form belief updates and value function backups. However, it is challenging to specify a weighted sum of Gaussians such that the mode distribution is a proper probability, namely that

$$\sum_m p(m|s,a) = 1$$

for any state variable s. It is possible to get a distribution that approximately sums to one by tiling the space with uniformly spaced Gaussians. However, this is essentially a form of discretization, and the number of Gaussians required increased exponentially with the number of state dimensions when a uniform tiling is used. This problem arises for both the mode probabilities and the observation probabilities, which suggests that this challenge will similarly affect the parametric representation of Porta et al. (2006). At this time it is unclear how these models could be specified more compactly in order to avoid this issue. This growth in the number of components merely to represent the world models is concerning because the number of new value function components created in a single backup is directly related to the number of components in the world models. If we choose to use an alternate fixed bases representation, as discussed in the prior paragraph, then these large models would get combined and re-expressed in order to perform the updates over the fixed basis, which might ameliorate the challenge of operating with models with a number of parameters which is an exponential function of the dimension. However, without modification the current representation presented in this chapter will severely in high dimensions.

Both of these issues apply to belief updates as well, though less severely as a belief update does not take an expectation over observations, and therefore an update does not increase the number of components as significantly as when a value backup is performed.

3.5 Summary

This chapter introduced SM-POMDP, an algorithm for planning in continuous-state domains characterized by switching-state, multi-modal dynamics. The parametric representation of the dynamics model allowed all POMDP planning operators (belief updating and value function backups) to be performed in closed form. SM-POMDP demonstrated the benefit of operating directly with continuous states by outperforming state of the art discrete-state planners HSVI2 and Perseus on a simulated example. SM-POMDP can handle more general dynamics models than the prior work it builds on by Porta et al. (2006) and the benefit of its increased expressive power was demonstrated in 3 simulated experiments.

Though SM-POMDP is a useful approach for handling a broad class of large (infinite-valued) domains, it has several important limitations which restrict its use to somewhat toy experimental setups in its current framework. It is particularly limited in its ability to scale to higher dimensions and in Chapter 5 we will present an alternative POMDP planning approach that scales more gracefully to higher dimensional problems. However, SM-POMDP still demonstrates the useful potential of parametric models for planning in large, low-dimensional, uncertain, partially-observable environments.

To teach how to live with uncertainty, yet without being paralyzed by hesitation, is perhaps the chief thing that philosophy can do.

Bertrand Russell

4 Learning in Switching High-dimensional Continuous Domains

The work described in this chapter is a result of a collaboration with two other graduate students, Bethany Leffler and Lihong Li, in addition to professors Nicholas Roy and Michael Littman. In particular, the theoretical analysis was developed in a close collaboration with Lihong Li and he was responsible for some of the central insights behind Lemmas 1-8.

Let's consider again the challenge first posed in the introduction of an agent learning to drive a car, or learning to drive in a new city, where the agent is aiming to reach some given destination. This is a challenge of making sequential decisions in an uncertain, large, high-dimensional environment when the state is fully observed, but the dynamics model is unknown. This problem is one instance of reinforcement learning, and in this chapter we will present evidence that parametric models can provide a bias that helps accelerate learning in certain large, high-dimensional, uncertain environments.

In selecting a parametric class to represent the world models, it is important to consider both expressive power and tractability (see Figure 4). With a sufficiently fine discretization, standard tabular representations that store the probability of each state transitioning to any other state, can express any possible form of the transition dynamics. However, tabular representations often have intractably slow learning: since each discrete state can have a different dynamics model, the experience gained from making decisions in one state cannot be shared with other states. This means that a separate dynamics model must be learned per state, which can take a prohibitively long time. The time until a good dynamics model is learned for all states becomes even more significant in high-dimensional domain since the number of discrete states typically scales exponentially with the state space dimension. This can cause an algorithm to make poor decisions for a long time as it explicitly or implicitly learns a model for each state. At the opposite extreme of expressive power is making the assumption that all states share the same dynamics, such as in a linear Gaussian controller. This allows experience gained from one state to be instantly shared across all other states. Prior work by Strehl and Littman (2008) demonstrated that if the dynamics are linear Gaussian, and the variance is known, then there exist algorithms which will select close-to-optimal actions on all but a number of samples which is polynomial in the state dimension, in contrast to tabular representations that typically scale exponentially. However, assuming all states share the same dynamics severely restricts the expressive power of these models.

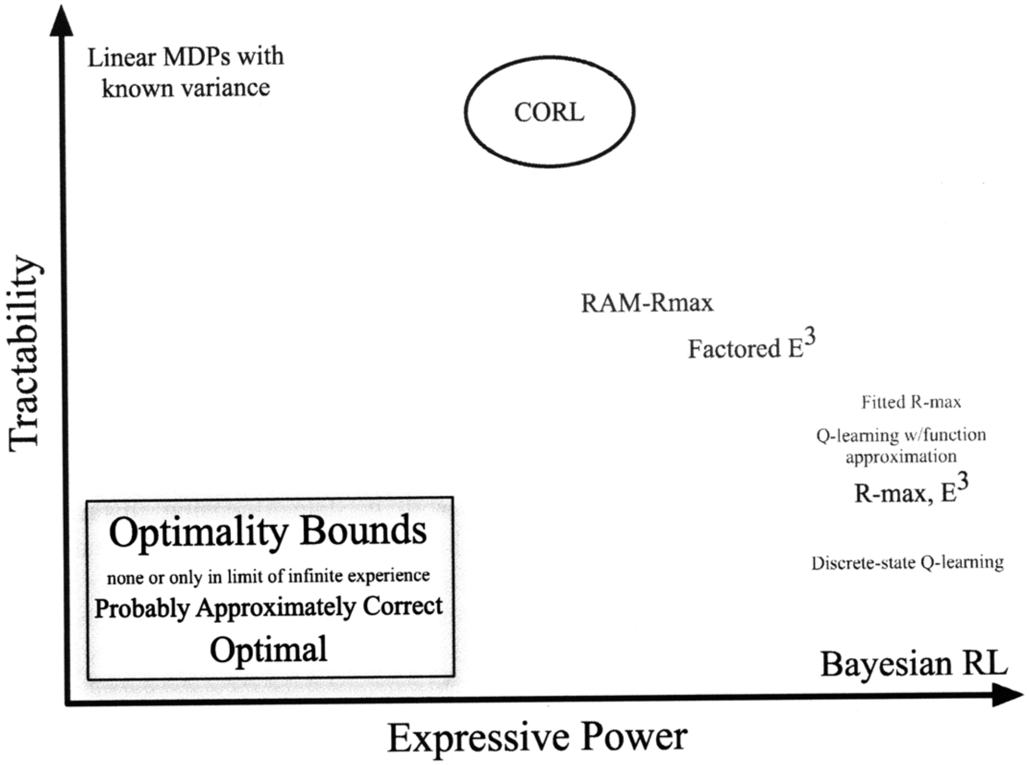

Tractability
Linear MDPs with known variance
CORL
RAM-Rmax
Factored E^3
Fitted R-max
Q-learning w/function approximation
R-max, E^3
Discrete-state Q-learning
Optimality Bounds
none or only in limit of infinite experience
Probably Approximately Correct
Optimal
Bayesian RL
Expressive Power

Figure 4-1: Illustrating the trade off between expressive power, optimality guarantees, and expressive power of reinforcement learning approaches. Tabular representations can represent almost any model, but assume that each state can have a different transition function, resulting in limited generalization between experiences. Linear Gaussian models assume that all states share the same dynamics model, which allows experience from any state to be generalized to all other states, but is a fairly strict assumption. In the middle are approaches which posit the existence of a few different dynamics models, named by types, where all states within the same type share the same dynamics.

This chapter will demonstrate that a broader class of dynamics models, termed "typed noisy-offset" will offer similar guarantees. This parametric representation substantially reduces the amount of experience necessary to make probabilistically guaranteed good decisions in continuous-valued, high-dimensional domains, but is expressive enough to capture a variety of domains of interest, such as navigation over varying terrain, or driving on highways versus small side streets.

In particular, the dynamics representation chosen is a simple noisy-offset model, where the next state is presumed to be a function of the prior state, plus an offset and some Gaussian distributed noise. The offset and Gaussian parameters are assumed to be specified by the type t of the state and action a, thereby allowing all states of the same type to share dynamics parameters. More formally,

$$s' = s + \beta_{at} + \varepsilon_{at}, \tag{4.1}$$

where s is the current state, s' is the next state, $\varepsilon_{at} \sim \mathcal{N}(0, \Sigma_{at})$ is drawn from a zero-mean Gaussian with covariance Σ_{at}, and β_{at} is the offset. Typed noisy-offset models are subclass of the switching state space models present in Chapter 3: in noisy-offset models the dynamics model for a particular type is restricted to be unimodal.

After outlining a general algorithm for decision making in these domains, titled CORL for "Continuous-Offset Reinforcement Learning", we will provide an analysis of a particular instantiation of the CORL algorithm. Rather than frame the decision-making problem within a generic POMDP framework, where the dynamics parameters are treated as hidden state variables, and attempt to solve the POMDP exactly, we will instead focus on a less ambitious objective for CORL. In doing so we will gain the opportunity to make precise claims about how often, and with what probability, it may be possible to make near optimal decisions. The analysis presented here will demonstrate that variants of CORL lie within the general class of Probably Approximately Correct (PAC) reinforcement learning(RL). The most important theoretical contribution of this chapter will be to prove that CORL can have a sample complexity that scales *polynomially* with the state dimension: this is in contrast to typical discrete-state representations which have a sample complexity that has an *exponential* dependence on the state dimension. In contrast to prior work, our proof will explicitly incorporate any potential error that arises from only approximately solving the estimated Markov Decision Process: this error will typically be non-zero since no generic, optimal, continuous-state MDP planners exist.

At a high level, this work falls into the category of model-based reinforcement-learning algorithms in which the MDP model (Equation 4.1) can be *KWIK-learned* (Li et al., 2008), and thus it is efficient in exploring the world. That framework provides another justification of the soundness and effectiveness of the CORL algorithm.

4.1 Background

We first briefly describe the basic framework. The world is characterized by a continuous-state discounted MDP $M = \langle S, A, p(s'|s,a), R, \gamma \rangle$ where $S \subseteq \mathbb{R}^N$ is the N-dimensional state space, A is a set of discrete actions, $p(s'|s,a)$ is the unknown transition dynamics that satisfy the parametric form of Equation 4.1, $\gamma \in [0,1)$ is the discount factor and $R : S \times A \rightarrow [0,1]$ is the known reward function. In addition to the standard MDP formulation, each state s is associated with a single observable type $t \in T$. The total number of types is N_T and the mapping from states to types $S \rightarrow T$ is assumed to be known. The dynamics of the environment are determined by the current state type t and action a taken:

$$p(s'|s,a) = \mathcal{N}(s'; s + \beta_{at}, \Sigma_{at}) \tag{4.2}$$

where s' denotes the successor state of state s. Therefore, types partition the state space into regions, and each region is associated with a particular pair of dynamics parameters.

In this work, we focus on the known reward model and unknown dynamics model situation. The parameters of the dynamics model, β_{at} and Σ_{at}, are assumed to be unknown for all types t and actions a at the start of learning. This model is a departure from prior related work (Abbeel & Ng, 2005; Strehl & Littman, 2008), which focuses on a more general linear dynamics model but assumes a single type and that the variance of the noise

Algorithm 4 CORL

```
Input: $N$ (dimension of the state space),$|A|$ (number of actions), $N_T$ (number of types), $R$ (reward model), $\gamma$ (discount factor), $N_{at}$ (minimum number of samples per state-action pair)
Set all type–action tuples $\langle t, a \rangle$ to be unknown and initialize the dynamics models (see text) to create an empirical known-state MDP model $\hat{M}_K$.
Start in a state $s_0$.
loop
    Solve MDP $\hat{M}_K$ using approximate solver and denote its optimal value function by $Q_t$.
    Select action $a = \mathrm{argmax}_a Q_t(s, a)$.
    Increment the appropriate $n_{at}$ count (where $t$ is the type of state $s$).
    Observe transition to the next state $s'$.
    If $n_{at}$ exceeds $N_{at}$ where $N_{at}$ is specified according to the analysis, then mark $\langle a, t \rangle$ as "known" and estimate the dynamics model parameters for this tuple.
end loop
```

Σ_{at} is known. We argue that in many interesting problems, the variance of the noise is unknown and estimating this noise may provide the key distinction between the dynamics models of different types.

The objective of the agent will be as in standard reinforcement learning, which is, the agent's goal is to learn a policy $\pi : S \rightarrow A$ that allows the agent to choose actions to maximize the expected total reward it will receive. As is standard, let π^* be the optimal policy, and its associated value function be $V^*(s)$. For simplicity, the reward is assumed to be only a function of state in this chapter, but as long as the reward can be KWIK-learned (Li et al., 2008) then the results are easily extended to when the reward is unknown. KWIK-learnable reward functions include, for instance, Gaussian, linear and tabular rewards.

4.1.1 Algorithm

The CORL algorithm (*c.f.*, Algorithm 4) is derived from the R-max algorithm of Brafman and Tennenholtz (2002). We first form a set of $\langle t, a \rangle$ tuples, one for each type–action pair. Note that each tuple corresponds to a particular pair of dynamics model parameters, $\langle \beta_{at}, \Sigma_{at} \rangle$. A tuple is considered to be "known" if the agent has been in type t and taken action a a number N_{at} times. At each time step, we construct a new MDP $\hat{M}_K$ as follows, using the same state space, action space, and discount factor as the original MDP. If the number of times a tuple has been experienced, n_{at}, is greater than or equal to N_{at}, then we estimate the parameters for this dynamics model using maximum-likelihood estimation:

$$\tilde{\beta}_{at} = \frac{\sum_{i=1}^{n_{at}} (s_i' - s_i)}{n_{at}} \qquad (4.3)$$

$$\tilde{\Sigma}_{at} = \frac{\sum_{i=1}^{n_{at}} (s_i' - s_i - \tilde{\beta}_{at})(s_i' - s_i - \tilde{\beta}_{at})^T}{n_{at}} \qquad (4.4)$$

where the sum ranges over all state–action pairs experienced for which the type of s_i was t and the action taken was a. Note that while Equation 4.4 is a biased estimator, it is also popular and consistent, and becomes extremely close to the unbiased estimate when the number of samples n_{at} is large. We choose it because it makes our later analysis simpler.

Otherwise, we set the dynamics model for all states and the action associated with this type–action tuple to be a transition with probability 1 back to the same state. We also modify the reward function for all states associated with an unknown type–action tuple $\langle t_u, a_u \rangle$ so that all state–action values $Q(s_{t_u}, a_u)$ have a value of $V_{\max}$ (the maximum value possible, $1/(1-\gamma)$). We then seek to solve $\hat{M}_K$. This MDP includes switching dynamics with continuous states, and we are aware of no planners guaranteed to return the optimal policy for such MDPs in general. Therefore CORL assumes the use of an approximate solver to provide a solution for a MDP. There are a variety of existing MDP planners, such as discretizing or using a linear function approximation, and we will consider particular planner choices in the following sections. At each time step, the agent chooses the action that maximizes the estimate of its current approximate value according to Q_t: $a = \mathrm{argmax}_a Q_t(s, a)$. The complete algorithm is shown in Algorithm 4.

4.2 Learning complexity

4.2.1 Preliminaries and framework

When analyzing the performance of an RL algorithm $\mathcal{A}$, there are many potential criteria to use. In our work, we will focus predominantly on sample complexity with a brief mention of computational complexity. Computational complexity refers to the number of operations executed by the algorithm for each step taken by the agent in the environment. We will follow Kakade (2003) and use *sample complexity* as shorthand for the *sample complexity of exploration*. It is the number of time steps at which the algorithm, when viewed as a non-stationary policy π, is not ϵ-optimal at the current state; that is, $Q^*(s, a) - Q^\pi(s, a) > \epsilon$ where Q^* is the optimal state–action value function and Q^π is the state–action value function of the non-stationary policy π. Following Strehl, Li and Littman (2006), we are interested in showing, for a given ϵ and δ, that with probability at least $1 - \delta$ the sample complexity of the algorithm is less than or equal to a polynomial function of MDP parameters. Note that we only consider the number of samples to ensure the algorithm will learn and execute a near-optimal policy with high probability. As the agent acts in the world, it may be unlucky and experience a series of state transitions that poorly reflect the true dynamics due to noise.

Prior work by Strehl, Li and Littman (2006) provided a framework for analyzing the sample complexity of R-max-style RL algorithms. This framework has since been used in several other papers (Leffler et al., 2007a; Strehl & Littman, 2008) and we will also adopt the same approach. Therefore we first briefly discuss the structure of this framework.

Strehl, Li and Littman (2006) defined an RL algorithm to be greedy if it chooses its action to be the one that maximizes the value of the current state s ($a = \mathrm{argmax}_{a \in A} Q(s, a)$). Their main result goes as follows: let $\mathcal{A}(\epsilon, \delta)$ denote a greedy learning algorithm. Maintain a list K of "known" state–action pairs. At each new time step, this list stays the same unless during that time step a new state–action pair becomes known. MDP $\hat{M}_K$ is the agent's current estimated MDP, consisting of the agent's estimated models for the known state–action pairs, and self loops and optimistic rewards (as in our construction described in the prior section) for unknown state–action pairs. MDP M_K is an MDP which consists of the true (underlying) reward and dynamics models for the known state–action pairs, and again self loops and optimistic rewards for the unknown state–action pairs. To be clear, the only difference between MDP $\hat{M}_K$ and MDP M_K is that the first uses the agent's experience to generate estimated models for the known state–action pairs, and the second uses the true model parameters. π is the greedy policy with respect to the current state-action values $Q_{\hat{M}_K}$ obtained by solving MDP $\hat{M}_K$: $V_{\hat{M}_K}$ is the associated value function for $Q_{\hat{M}_K}$ and may equivalently be viewed as the value of policy π computed using the estimated model parameters. $V^{\pi}_{M_K}$ is the value of policy π computed using the true model parameters. Assume that ϵ and δ are given and the following 3 conditions hold for all states, actions and time steps:

1. $Q^*(s,a) - Q_{\hat{M}_K}(s,a) \leq \epsilon$.
2. $V^{\pi}_{\hat{M}_K}(s) - V^{\pi}_{M_K}(s) \leq \epsilon$.
3. The total number of times the agent visits a state–action tuple that is not in K is bounded by $\zeta(\epsilon, \delta)$ (the *learning complexity*).

Then, Strehl, Li and Littman (2006) show for any MDP M, $\mathcal{A}(\epsilon, \delta)$ will follow a 4ϵ-optimal policy from its initial state on all but N_{total} time steps with probability at least $1-2\delta$, where N_{total} is polynomial in the problem's parameters $(\zeta(\epsilon,\delta), \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma})$.

The majority of our analysis will focus on showing that our algorithm fulfills these three criteria. In our approach, we will define the known state–action pairs to be all those state–actions for which the type–action pair $\langle t(s), a\rangle$ is known. We will assume that the absolute values of the components in Σ_{at} are upper bounded by a known constant, B_σ which is, without loss of generality, assumed to be greater than or equal to 1. This assumption is often true in practice. We denote the determinant of matrix D by $\det D$, the trace of a matrix D by $\operatorname{tr}(A)$, the absolute value of a scalar d by $|d|$ and the p-norm of a vector v by $\|v\|_p$. Full proofs, when omitted, can be found in the Appendix.

4.2.2 Analysis

Our analysis will serve to prove the main result:

Theorem 4.2.1. *For any given δ and ϵ in a continuous-state noisy-offset dynamics MDP with N_T types where the covariance along each dimension of all the dynamics models is bounded by $[-B_\sigma, B_\sigma]$, on all but N_{total} time steps, our algorithm will follow a 4ϵ-optimal policy from its current state with probability at least $1-2\delta$, where N_{total} is polynomial in the problem parameters $(N, |A|, N_T, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma}, \lambda_N, B_\sigma)$ where λ_N is the smallest eigenvalue of the dynamics covariance matrices.*

Proof. To prove this, we need to demonstrate that the three criteria of Strehl, Li, and Littman (2006) hold. Our efforts will focus on the second criterion. This criterion states the value of states under the estimated known-state MDP $\hat{M}_K$ must be very close to the value of states under the known-state MDP M_K that uses the true model parameters for all known type–action pairs. To prove this we must bound how far away the model parameters estimated from the agent's experience can be from the true underlying parameters, and how this relates to error in the resulting value function. We must also consider the error induced by approximately solving the estimated MDP $\hat{M}_K$. Achieving a given accuracy level in the final value function creates constraints on how close the estimated model parameters must be to the true model parameters. We will illustrate how these constraints relate to the amount of experience required to achieve these constraints, and this in turn will give us an expression for the number of samples required for a type–action pair to be known, or the learning complexity for our algorithm. Once we have proved the second criterion the first criterion follows which states that all state–action values in our estimated known-state MDP must be at least the optimal true value of that state–action pair minus ϵ. This holds immediately for all unknown state–action values which have value V_{max} due to the optimism under uncertainty bias of our approach. If all state–action tuples are known, then from the second criterion the resulting value is guaranteed to be within ϵ of the true optimal value. When only some state–action tuples are known, then, due to the optimistic overestimate of the unknown state–action tuples, the value of the known state–action tuples cannot be ϵ-worse than their optimal value.

Therefore we commence by formally relating how the amount of experience (number of transitions) of the agent corresponds to the accuracy in the estimated dynamics model parameters.

Lemma 2. *Given any $\epsilon, \delta > 0$, then after $T = \frac{12N^2B_\sigma^2}{\epsilon^2\delta}$ transition samples (s, a, s') with probability at least $1 - \frac{\delta}{3}$, the estimated offset parameter $\tilde{\beta}$, computed by Equation 4.3, and estimated covariance parameters $\tilde{\sigma}_{ij}$, computed by Equation 4.4, will deviate from the true parameters β and σ_{ij} by at most ϵ: $\Pr(\left\|\tilde{\beta} - \beta\right\|_2 \leq \epsilon) \geq 1 - \frac{\delta}{3}$ and $\Pr(\max_i |\tilde{\sigma}_{ij} - \sigma_{ij}| \leq \epsilon) \geq 1 - \frac{\delta}{3}$.*

Proof. T will be the maximum of the number of samples to guarantee the above bounds for the offset parameter β and the number of samples needed for a good estimate of the variance parameter. We first examine the offset parameter:

Lemma 3. *Given any $\epsilon, \delta > 0$, define $T_\beta = \frac{3N^2B_\sigma}{\epsilon^2\delta}$. If there are T_β transition samples (s, a, s'), then with probability at least $1 - \frac{\delta}{3}$, the estimated offset parameter $\tilde{\beta}$, computed by Equation 4.3, will deviate from the true offset parameter β by no more than ϵ along any dimension d; formally,* $\Pr(\max_d \|\tilde{\beta}_d - \beta_d\| \geq \frac{\epsilon}{\sqrt{N}}) \leq \frac{\delta}{3N}$.

Proof. From Chebyshev's inequality we know that the probability of a single offset $s'_{id} - s_{id}$ (where s_{id} is the value of the i-th state along dimension d) deviating by more than ε from the true offset parameter β_d along dimension d is bounded by the variance of the offset parameter:

$$P(|s'_{id} - s_{id} - \beta_d| \geq \frac{\epsilon}{\sqrt{N}}) \leq \frac{\sigma_d^2 N}{\epsilon^2}$$

where σ_d^2 is the variance of the offset along dimension d. Using the fact that the variance of a sum of T_β i.i.d. variables is just T_β multiplied by the variance of a single variable, we obtain

$$\begin{aligned} P(|\sum_{i=1}^{T_\beta} s'_{id} - s_{id} - T_\beta\beta_d| \geq T_\beta \frac{\epsilon}{\sqrt{N}}) &\leq \frac{T_\beta \sigma_d^2 N}{T_\beta^2 \epsilon^2} \\ P(|\tilde{\beta}_d - \beta_d| \geq \frac{\epsilon}{\sqrt{N}}) &\leq \frac{\sigma_d^2 N}{T_\beta \epsilon^2}. \end{aligned}$$

We require this to hold with probability at most $\frac{\delta}{3N}$ and solve for T_β:

$$T_\beta = \frac{3\sigma_d^2 N^2}{\delta\epsilon^2}$$

We know that the variance along any dimension is bounded above by B_σ so we can substitute this in the above expression to derive a bound on the number of samples required.

$$T_\beta \geq \frac{3B_\sigma N^2}{\delta\epsilon^2}.$$

□

Lemma 3 immediately implies a bound on the L_2 norm between the estimated offset parameter vector and the true offset parameter vector, as follows:

Lemma 4. *Given any* $\epsilon, \delta > 0$, *if* $\Pr(\max_d \|\tilde{\beta}_d - \beta_d\|_2 \geq \frac{\epsilon}{\sqrt{N}}) \leq \frac{\delta}{3N}$, *then* $\Pr(\|\tilde{\beta} - \beta\|_2 \geq \epsilon) \leq \frac{\delta}{3}$.

Proof. By the union bound, the probability that any of the dimensions exceed an estimation error of at most $\frac{\epsilon}{\sqrt{N}}$ is at most δ. Therefore with probability at least $1 - \delta$ all dimensions will simultaneously have an estimation error of less than $\frac{\epsilon}{\sqrt{N}}$ and from the definition of the L2 norm this immediately implies that $|\tilde{\beta} - \beta|_2 \leq \epsilon$. □

Lemma 5. *Assume* $\max_d \|\tilde{\beta}_d - \beta_d\|_2 \leq \epsilon$ *for* $\epsilon < 1/3$. *Given any* $\delta > 0$, *define* $T_\sigma = \frac{12N^2B_\sigma^2}{\delta\epsilon^2}$. *If there are* T_σ *transition samples* (s, a, s'), *then with probability at most* $\frac{\delta}{3}$, *the estimated covariance parameter* $\tilde{\sigma}_{ij}$, *computed by Equation 4.4, deviates from the true covariance parameter* σ_{ij} *by more than* ϵ *over all entries* ij; *formally,* $\Pr(\max_{i,j} |\tilde{\sigma}_{ij} - \sigma_{ij}| \geq \epsilon) \leq \frac{\delta}{3}$.

Proof. First recall that σ_{ij} represents the covariance between dimensions i and j. We are interested in the probability that the estimated covariance $\tilde{\sigma}_{ij}$ differs from the true parameter σ_{ij}: $P(|\tilde{\sigma}_{ij} - \sigma_{ij}| \geq \epsilon)$. From Chebyshev's inequality, we can bound this expression as

$$P(|\tilde{\sigma}_{ij} - \sigma_{ij}| \geq \epsilon) \leq \frac{Var(\tilde{\sigma}_{ij})}{\epsilon^2} \tag{4.5}$$

where $Var(\tilde{\sigma}_{ij})$ is the variance of the sample variance.

We therefore require an upper bound on the variance of the sample covariance. We will derive a bound on this below in the general case of the covariance between two variables x and y both of which are Gaussian distributed.

$$\begin{aligned}
Var(\tilde{\sigma}_{xy}) &= E[(\tilde{\sigma}_{xy} - \sigma_{xy})^2] \\
&= E\left[\left(\frac{1}{T_\sigma}\sum_{k=1}^{T_\sigma}(x_k - \bar{x})(y_k - \bar{y}) - \sigma_{xy}^2\right)^2\right]
\end{aligned}$$

where in the second line we have written out the definition of the sample covariance. We can then use the linearity of expectation to derive

$$\begin{aligned}
Var(\tilde{\sigma}_{xy}) &= \frac{1}{T_\Sigma^2}\sum_{k=1}^{T_\sigma}\sum_{m=1}^{T_\sigma} E[(x_i - \bar{x})(x_m - \bar{x})(y_k - \bar{y})(y_m - \bar{y})] \\
&\quad -2\sigma_{xy}\frac{1}{T_\sigma}\sum_{k=1}^{T_\sigma} E[(x_m - \bar{x})(y_m - \bar{y})] + E[(\sigma_{xy})^2] \\
&= \frac{1}{T_\Sigma^2}\sum_{k=1}^{T_\sigma}\sum_{m=1}^{T_\sigma} E[(x_i - \bar{x})(x_m - \bar{x})(y_k - \bar{y})(y_m - \bar{y})] - (\sigma_{xy})^2
\end{aligned}$$

where the second line follows from the definition of the variance σ_{xy}. We next divide the summation into two expressions, when $m = k$ and when $m \neq k$, and use the property that the expectation of independent variables is the product of their expectations:

$$\begin{aligned}
Var(\tilde{\sigma}_{xy}) &= \frac{1}{T_\sigma}E[(x_i - \bar{x})^2(y_k - \bar{y})^2] + \frac{T_\sigma(T_\sigma - 1)}{T_\Sigma^2}E[(x_k - \bar{x})(y_k - \mu_k)]E[(x_m - \bar{x})(y_m - \bar{y})] - (\sigma_{xy})^2 \\
&= \frac{1}{T_\sigma}E[(x_i - \bar{x})^2(y_k - \bar{y})^2] + \frac{T_\sigma(T_\sigma - 1)}{T_\Sigma^2}(\sigma_{xy})^2 - (\sigma_{xy})^2.
\end{aligned}$$

We can now use the Cauchy-Schwarz inequality on the first term

$$\begin{aligned}
Var(\tilde{\sigma}_{xy}) &\leq \frac{1}{T_\sigma}\sqrt{E[(x - \bar{x})^4]E[(y_i - \bar{y})^4]} + \frac{T_\sigma(T_\sigma - 1)}{T_\Sigma^2}(\sigma_{xy})^2 - (\sigma_{xy})^2 \\
&= \frac{1}{T_\sigma}\sqrt{E[(x + \mu_x - \mu_x - \bar{x})^4]E[(y_i + \mu_y - \mu_y - \bar{y})^4]} + \frac{T_\sigma(T_\sigma - 1)}{T_\Sigma^2}(\sigma_{xy}^2)^2 - (\sigma_{xy}^2)^2 \\
&= \frac{1}{T_\sigma}\sqrt{(3\sigma_{xx}^4 + 6\sigma_{xx}^2(\bar{x} - \mu_x)^2 + (\bar{x} - \mu_x)^4)(3\sigma_{yy}^4 + 6\sigma_{yy}^2(\bar{y} - \mu_y)^2 + (\bar{y} - \mu_y)^4)} + \\
&\quad \frac{T_\sigma(T_\sigma - 1)}{T_\Sigma^2}(\sigma_{xy})^2 - (\sigma_{xy})^2
\end{aligned}$$

where we have used the fact that the fourth central moment of a Gaussian distribution is

$3\sigma_{xx}^2$ in the final line. Next we make use of the assumptions that B_σ is an upper bound to all covariance matrix elements and the bound on the maximum error in the parameter offset estimates:

$$\begin{aligned} Var(\tilde{\sigma}_{xy}^2) &\le \frac{(\epsilon^4 + 6\epsilon^2 B_\sigma + 3B_\sigma^2)}{T_\sigma} + \frac{T_\sigma(T_\sigma - 1)}{T_\Sigma^2}(\sigma_{xy})^2 - (\sigma_{xy})^2 \\ &\le \frac{4B_\sigma^2}{T_\sigma} \end{aligned}$$

where the last line follows because $\epsilon < 1/4$ and $B \ge 1$. We can then substitute this result into Equation 4.5

$$P(|\tilde{\sigma}_{ij} - \sigma_{ij}| \ge \epsilon) \;\le\; \frac{4B_\sigma^2}{\epsilon^2 T_\sigma}. \tag{4.6}$$

To ensure that this bound holds simultaneously with probability $\frac{\delta}{3}$ for all N^2 covariance matrix elements it suffices by the union bound to require that each covariance entry exceeds its expected value by more than ϵ with probability at most $\frac{\delta}{3N^2}$:

$$\frac{4B_\sigma^2}{\epsilon^2 T_\sigma} \le \frac{\delta}{3N^2}. \tag{4.7}$$

Re-arranging yields the bound for the number of samples required:

$$T_\sigma \;\ge\; \frac{12N^2 B_\sigma^2}{\delta\epsilon^2}.$$

□

Combining Lemmas 4 and 5 gives a condition on the minimum number of samples necessary to ensure, with high probability, that the estimated parameters of a particular type–action dynamics model are close to the true parameters. Then

$$T = \max\{T_\beta, T_\sigma\} = \max\left\{\frac{3N^2 B_\sigma}{\epsilon^2\delta}, \frac{12N^2 B_\sigma^2}{\epsilon^2\delta}\right\} = \frac{12N^2 B_\sigma^2}{\epsilon^2\delta}. \tag{4.8}$$

□

From Lemma 2 we now have an expression that relates how much experience the agent needs in order to have precise estimates of each model parameter. We next need to establish the distance between two dynamics models which have different offset and covariance parameters: this will later be important for bounding the value function difference between the estimated model MDP $\hat{M}_K$ and the true model MDP M_K.

Following Abbeel and Ng (2005), we choose to use the variational distance between two dynamics models P and Q:

$$d_{var}(P(x), Q(x)) = \frac{1}{2}\int_{\mathcal{X}} |P(x) - Q(x)| dx. \tag{4.9}$$

In our algorithm, β_1 and Σ_1 are the true offset parameter and covariance matrix of the Gaussian distribution, and β_2 and Σ_2 are the offset parameter and covariance matrix estimated from data. Since we can guarantee that they can be made arbitrarily close (element-wise), we will be able to bound the variational distance between two Gaussians, one defined with the true parameters and the other with the estimated parameters. The real-valued, positive eigenvalues of Σ_1 and Σ_2 are denoted by $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N > 0$ and $\lambda'_1 \geq \lambda'_2 \geq \cdots \geq \lambda'_N > 0$, respectively. Because of symmetry and positive definiteness of Σ_1 and Σ_2, λ_i and λ'_i must be real as well as positive. Since all eigenvalues are positive, they are also the singular values of their respective matrices, as singular values are simply the absolute value of the matrix eigenvalues.

Lemma 6. *Assume* $\max_{i,j} |\Sigma_1(i,j) - \Sigma_2(i,j)| \leq \epsilon$, *and* $N \left\|\Sigma_1^{-1}\right\|_\infty \epsilon < 1$, *then,*

$$d_{var}(\mathcal{N}(s'-s|\beta_1,\Sigma_1),\mathcal{N}(s'-s|\beta_2,\Sigma_2)) \leq \frac{||\beta_1-\beta_2||_2}{\sqrt{\lambda_N}} + \sqrt{\frac{N^2\epsilon}{\lambda_N} + \frac{2N^3B_\sigma\epsilon}{\lambda_N^2-(N)^{1.5}\epsilon\lambda_N}}.$$

Proof. We will use $\mathcal{N}(\beta,\Sigma)$ as an abbreviation for $\mathcal{N}(s'-s|\beta,\Sigma)$. Then

$$\begin{aligned} d_{var}(\mathcal{N}(\beta_1,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_2)) &\leq d_{var}(\mathcal{N}(\beta_1,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_1)) + d_{var}(\mathcal{N}(\beta_2,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_2)) \\ &= \frac{||(\mathcal{N}(\beta_1,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_1))||_1}{2} + \frac{||(\mathcal{N}(\beta_2,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_2))||_1}{2} \\ &\leq \sqrt{2d_{KL}(\mathcal{N}(\beta_1,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_1))} + \sqrt{2d_{KL}(\mathcal{N}(\beta_2,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_2))} \end{aligned}$$

where d_{KL} is the Kullback-Leibler divergence. The first step follows from the triangle inequality and the last step follows from Kullback (1967) (included for completeness in Lemma 11 in the appendix).

The KL divergence between two N-variate Gaussians has the closed form expression

$$\mathrm{d_{KL}}\left(\mathcal{N}(\beta_1,\Sigma_1) \parallel \mathcal{N}(\beta_2,\Sigma_2)\right) = \frac{1}{2}\left((\beta_1-\beta_2)^T\Sigma_1^{-1}(\beta_1-\beta_2) + \ln\frac{\det\Sigma_2}{\det\Sigma_1} + \operatorname{tr}\left(\Sigma_2^{-1}\Sigma_1\right) - N\right). \tag{4.10}$$

Substituting this expression into the above bound on d_{var} we get

$$d_{var}(\mathcal{N}(\beta_1,\Sigma_1),\mathcal{N}(\beta_2,\Sigma_2)) \leq \sqrt{(\beta_1-\beta_2)^T\Sigma_1^{-1}(\beta_1-\beta_2)} + \sqrt{\ln\left(\frac{\det\Sigma_2}{\det\Sigma_1}\right) + \operatorname{tr}\left(\Sigma_2^{-1}\Sigma_1\right) - N}$$

Our proof relies on bounding each term of the KL divergence. Note that the above expression reduces (up to a constant) to the bound proved by Abbeel and Ng (2005) when the variance is known.

To simplify the final expression we have also bounded some characteristics of the covariance matrix: these bounds may be found in Lemma 12 in the appendix. We now start with the first term:

Lemma 7.

$$(\beta_1 - \beta_2)^T \Sigma_1^{-1} (\beta_1 - \beta_2) \leq \frac{1}{\lambda_N} ||\beta_1 - \beta_2||_2^2.$$

Proof. First note that since Σ_1^{-1} is a Hermitian matrix then

$$\frac{(\beta_1 - \beta_2)^T \Sigma_1^{-1} (\beta_1 - \beta_2)}{||\beta_1 - \beta_2)||_2^2}$$

is a Rayleigh quotient which is bounded by the maximum eigenvalue of Σ_1^{-1}. The eigenvalues of Σ_1^{-1} are precisely the reciprocals of the eigenvalues of Σ_1. Therefore the Rayleigh quotient is at most $\frac{1}{\lambda_{\min} \Sigma_1}$ which is simply $\frac{1}{\lambda_N}$. Therefore

$$(\beta_1 - \beta_2)^T \Sigma_1^{-1} (\beta_1 - \beta_2) \leq \frac{||\beta_1 - \beta_2||_2^2}{\lambda_N}.$$

□

We next bound the second term.

Lemma 8. *If* $|\Sigma_1(i,j) - \Sigma_2(i,j)| \leq \epsilon$ *for any* $1 \leq i, j \leq N$, *then*

$$\left| \ln \frac{\det \Sigma_2}{\det \Sigma_1} \right| \leq 2N\epsilon \left(\frac{1}{\lambda_1} + \frac{1}{\lambda_2} + \cdots + \frac{1}{\lambda_N} \right) \leq \frac{N^2 \epsilon}{\lambda_N}.$$

Proof. Define $E = \Sigma_2 - \Sigma_1$. Clearly, E is symmetric since both Σ_1 and Σ_2 are symmetric. Its eigenvalues are denoted by $\psi_1 \geq \psi_2 \geq \cdots \geq \psi_N$, which are real (but can be negative or positive). First, it is known that

$$\det \Sigma_1 = \prod_{i=1}^{N} \lambda_i \quad \& \quad \det \Sigma_2 = \prod_{i=1}^{N} \lambda_i'.$$

Therefore,

$$\ln \frac{\det \Sigma_2}{\det \Sigma_1} = \ln \prod_{i=1}^{N} \frac{\lambda_i'}{\lambda_i} = \sum_{i=1}^{N} \ln \frac{\lambda_i'}{\lambda_i}.$$

From Geršgorin's theorem (Horn & Johnson, 1986, Theorem 6.1.1), the eigenvalues of E must be small as the elements of E are small. Specifically, any ψ_i must lie in one of the n Geršgorin discs D_j:

$$\forall 1 \leq j \leq N : D_j = \{x \in R \mid |x - E(j,j)| \leq \sum_{j' \neq j} |E(j,j')|\}.$$

It follows immediately that

$$|\psi_i| \leq \sum_{j=1}^{N} |E(i,j)| \leq N\epsilon$$

as every component in E lies in $[-\epsilon, \epsilon]$.

On the other hand, from Weyl's theorem (Horn & Johnson, 1986, Theorem 4.3.1), we have

$$\psi_1 \geq \lambda_i' - \lambda_i \geq \psi_N.$$

We have just proved that both $|\psi_1|$ and $|\psi_N|$ are at most $N\epsilon$, and thus

$$|\lambda_i' - \lambda_i| \leq N\epsilon.$$

Consequently,

$$\frac{\lambda_i'}{\lambda_i} \leq \frac{\lambda_i + N\epsilon}{\lambda_i} = 1 + \frac{N\epsilon}{\lambda_i}.$$

Therefore, we have

$$\ln \frac{\det \Sigma_2}{\det \Sigma_1} = \sum_{i=1}^{N} \ln \frac{\lambda_i'}{\lambda_i} \leq \sum_{i=1}^{N} \ln\left(1 + \frac{N\epsilon}{\lambda_i}\right) \leq \sum_{i=1}^{N} \frac{N\epsilon}{\lambda_i} \leq \frac{(N)^2\epsilon}{\lambda_N}$$

where the second to last inequality uses the inequality $\ln(1+x) \leq x$ for $x \geq 0$. □

Finally we bound the third term.

Lemma 9. *If* $\max_{i,j} |\Sigma_1(i,j) - \Sigma_2(i,j)| \leq \epsilon$ *and* $N\epsilon \left\|\Sigma_1^{-1}\right\|_1 < 1$, *then*

$$\operatorname{tr}\left(\Sigma_2^{-1}\Sigma_1\right) - N \leq \frac{2N^3\epsilon B_\sigma}{\lambda_N^2 - (N)^{1.5}\lambda_N\epsilon}.$$

Proof. The i-th row (or column) of Σ_1^{-1} is the solution to the system of linear equations: $\Sigma_1 x = e_i$ where e_i has $N-1$ zero components except a 1 in the i-th component. Similarly, the i-th row (or column) of Σ_2^{-1} is the solution to $\Sigma_2 y = e_i$. Since Σ_1 and Σ_2 differ by at most ϵ in every component, we have

$$\frac{\|\Sigma_1 - \Sigma_2\|_1}{\|\Sigma_1\|_1} \leq \frac{N\epsilon}{\|\Sigma_1\|_1}.$$

For simplicity, denote the right-hand side by ϵ'. It follows from Lemma 13 (provided in the Appendix) that

$$\|x - y\|_1 \leq \frac{2\epsilon'\kappa_1(\Sigma_1)\|x\|_1}{1 - \epsilon'\kappa_1(\Sigma_1)}.$$

The above inequality holds for all N possible e values. Note that $\|x-y\|_1$ is the absolute sum of the i-th row (or column) of $\Sigma_1^{-1} - \Sigma_2^{-1}$ for e_i (if e has a 1 in its i-th component). Let $\psi_1 \geq \psi_2 \geq \cdots \geq \psi_N \geq 0$ be the singular values of $\Sigma_1^{-1} - \Sigma_2^{-1}$. From Geršgorin's theorem, it follows that for all i,

$$\lambda_i \leq \max_b \|x - y\|_1 \leq \frac{2\epsilon'\kappa_1(\Sigma_1)}{1 - \epsilon'\kappa_1(\Sigma_1)} \max_b \|x\|_1 = \frac{2\epsilon'\kappa_1(\Sigma_1)}{1 - \epsilon'\kappa_1(\Sigma_1)} \left\|\Sigma_1^{-1}\right\|_1$$

where $\kappa(\Sigma_1) = \|\Sigma_1\| \left\|\Sigma_1^{-1}\right\|$ the condition number of Σ_1. We can now finish the proof:

$$\begin{aligned}
\operatorname{tr}\left(\Sigma_2^{-1}\Sigma_1\right) - N &= \operatorname{tr}\left((\Sigma_2^{-1} - \Sigma_1^{-1})\Sigma_1\right) && (4.11)\\
&\leq \sum_{i=1}^{N} \psi_i \lambda_i && (4.12)\\
&\leq \frac{2\epsilon' \kappa_1(\Sigma_1) \left\|\Sigma_1^{-1}\right\|_1}{1 - \epsilon' \kappa_1(\Sigma_1)} \sum_{i=1}^{N} \lambda_i && (4.13)\\
&= \frac{2\epsilon' \kappa_1(\Sigma_1) \left\|\Sigma_1^{-1}\right\|_1}{1 - \epsilon' \kappa_1(\Sigma_1)} \operatorname{tr}(\Sigma_1) && (4.14)\\
&= \frac{2N\epsilon \left\|\Sigma_1^{-1}\right\|_1^2}{1 - N\epsilon \left\|\Sigma_1^{-1}\right\|_1} \operatorname{tr}(\Sigma_1) && (4.15)\\
&= \frac{2N^2 B_\sigma \epsilon \left\|\Sigma_1^{-1}\right\|_1^2}{1 - N\epsilon \left\|\Sigma_1^{-1}\right\|_1}, && (4.16)\\
&\leq \frac{2N^3 \epsilon B_\sigma}{\lambda_N^2 - N^{1.5} \lambda_N \epsilon}. && (4.17)
\end{aligned}$$

where the first equality (Equation 4.11) is due to the identity $\operatorname{tr}\left(\Sigma_1^{-1}\Sigma_1\right) = \operatorname{tr}(I) = N$, and the first inequality (Equation 4.12) is a direct application of von Neumann's inequality (Lemma 14) which can be used since the eigenvalues equal the singular values in this case. The second inequality (Equation 4.13) makes use of what we have proved earlier, the second equality (Equation 4.14) follows by the definition of matrix traces, and the third equality (Equation 4.15) is obtained by noting that $\kappa_1(\Sigma_1) = \|\Sigma_1\|_1 \left\|\Sigma_1^{-1}\right\|_1$. Since each term in the covariance matrix is known to be bounded by B_σ then the trace is bounded by NB_σ which allows us to generate the fourth equality (Equation 4.16). The final result is obtained using the result of Lemma 12. □

Combining the results of Lemmas 7, 8, and 9 completes the proof of Lemma 6. □

Note this bound is tight when the means and the variances are the same.

At this point we can relate the number of experiences (samples) of the agent to a distance measure between the estimated dynamics model (for a particular type–action) and the true dynamics model.

As a function of this, we now bound the error between the state-action values of the true MDP model M_K solved exactly and the approximate state-action values of our estimated model MDP $\hat{M}_K$ obtained using an approximate planner. This is a departure from most related PAC-MDP work which typically assumes the existence of a planning oracle for choosing actions given the estimated model.

Lemma 10. ***(Simulation Lemma)*** *Let $M_1 = \langle S, A, p_1(\cdot|\cdot,\cdot), R, \gamma\rangle$ and $M_2 = \langle S, A, p_2(\cdot|\cdot,\cdot), R, \gamma\rangle$ be two MDPs*[1] *with dynamics as characterized in Equation 1 and*

[1] For simplicity we present the results here without reference to types. In practice, each dynamics parameter would be subscripted by its associated MDP, type, and action.

non-negative rewards bounded above by 1. Given an ϵ (where $0 < \epsilon \leq V_{\max}$), assume that $d_{var}(p_1, p_2) \leq (1-\gamma)^2\epsilon/(2\gamma)$ and the error incurred by approximately solving a MDP, defined as ε_{plan} is also at most $(1-\gamma)^2\epsilon/(2\gamma)$ (to be precise, $\varepsilon_{plan} = ||V^* - \tilde{V}^*||_\infty \leq (1-\gamma)^2\epsilon/(2\gamma)$ where $\tilde{V}^*$ is the value computed by the approximate solver). Let π be a policy that can be applied to both M_1 and M_2. Then, for any and stationary policy π, for all states s and actions a, $|Q_1^\pi(s,a) - \tilde{Q}_2^\pi(s,a)| \leq \epsilon$, where $\tilde{Q}_2^\pi$ denotes the state–action value obtained by using an approximate MDP solver on MDP M_2 and Q_1^π denotes the true state–action value for MDP M_1 for policy π.

Proof. Let $\Delta_Q = \max_{s,a} |Q_1^\pi(s,a) - \tilde{Q}_2^\pi(s,a)|$ and define $\tilde{V}_2^\pi$ be the approximate value of policy π computed using an approximate MDP solver on MDP M_2, and V_1^π be the exact value of policy π on MDP M_1. Note that since we are taking the max over all actions, Δ_Q is also equal to or greater than $\max_s |V_1^\pi(s) - \tilde{V}_2^\pi(s)|$. Let $Lp_2(s'|s,a)$ denotes an approximate backup of FVI for MDP M_2: L denotes the approximation induced by using FVI.

Since these value functions are the fixed-point solutions to their respective Bellman operators, we have for every (s,a) that

$$\begin{aligned}
&|Q_1^\pi(s,a) - \tilde{Q}_2^\pi(s,a)| \\
&= \left|\left(R(s,a) + \gamma \int_{s' \in S} p_1(s'|s,a) V_1^\pi(s')ds'\right) - \left(R(s,a) + \gamma \int_{s' \in S} Lp_2(s'|s,a)\tilde{V}_2^\pi(s')ds'\right)\right| \\
&\leq \gamma \left|\int_{s' \in S} \left[p_1(s'|s,a)V_1^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')\right] ds'\right| \\
&\leq \gamma \left|\int_{s' \in S} \left[p_1(s'|s,a)V_1^\pi(s') - p_1(s'|s,a)\tilde{V}_2^\pi(s') + p_1(s'|s,a)\tilde{V}_2^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')\right] ds'\right| \\
&\leq \gamma \left|\int_{s' \in S} \left[p_1(s'|s,a)(V_1^\pi(s') - \tilde{V}_2^\pi(s')) + p_1(s'|s,a)\tilde{V}_2^\pi(s') - p_2(s'|s,a)\tilde{V}_2^\pi(s')\right.\right. \\
&\qquad \left.\left. + p_2(s'|s,a)\tilde{V}_2^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')\right] ds'\right| \\
&\leq \gamma \left|\int_{s' \in S} p_1(s'|s,a)(V_1^\pi(s') - \tilde{V}_2^\pi(s'))ds'\right| + \gamma \left|\int_{s' \in S} (p_1(s'|s,a) - p_2(s'|s,a))\tilde{V}_2^\pi(s')ds'\right| \\
&\qquad + \gamma \left|\int_{s' \in S} p_2(s'|s,a)\tilde{V}_2^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')ds'\right|
\end{aligned}$$

where the final expression was obtained by repeatedly adding and subtracting identical terms and using the triangle inequality. This expression must hold for all states s and actions a, so it must also hold for the maximum error over all states and actions:

$$\begin{aligned}
\max_s \max_a |Q_1^\pi(s,a) - \tilde{Q}_2^\pi(s,a) &\leq \gamma \int_{s' \in S} p_1(s'|s,a)\Delta_Q ds' + \gamma \left| \int_{s' \in S} (p_1(s'|s,a) - p_2(s'|s,a))\tilde{V}_2^\pi(s')ds' \right| \\
&\quad + \gamma \left| \int_{s' \in S} p_2(s'|s,a)\tilde{V}_2^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')ds' \right| \\
\Delta_Q &\leq \gamma\Delta_Q + \gamma \left| \int_{s' \in S} (p_1(s'|s,a) - p_2(s'|s,a))\tilde{V}_2^\pi(s')ds' \right| \\
&\quad + \gamma \left| \int_{s' \in S} p_2(s'|s,a)\tilde{V}_2^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')ds' \right| \\
&\leq \gamma\Delta_Q + \gamma V_{max} \left| \int_{s' \in S} p_1(s'|s,a) - p_2(s'|s,a)ds' \right| \\
&\quad + \gamma \left| \int_{s' \in S} p_2(s'|s,a)\tilde{V}_2^\pi(s') - Lp_2(s'|s,a)\tilde{V}_2^\pi(s')ds' \right| \\
&\leq \gamma\Delta_Q + \gamma V_{max} d_{var}(p_1(s'|s,a), p_2(s'|s,a)) + \gamma\varepsilon_{plan}
\end{aligned}$$

where we have again used the triangle inequality. Therefore

$$\begin{aligned}
\Delta_Q &\leq \gamma\Delta_Q + \gamma V_{\max} d_{var} + \gamma\varepsilon_{plan} \\
&= \frac{\gamma\frac{dvar}{1-\gamma}}{1-\gamma} + \frac{\gamma\varepsilon_{plan}}{1-\gamma}.
\end{aligned}$$

We have now expressed the error in the value function as the sum of the error due to the model approximation and the error due to using an approximate MDP planner. Using the assumptions in the lemma, the result immediately follows. □

We can now use the prior lemmas to prove Theorem 1. First we need to examine under what conditions the two assumptions of the Simulation Lemma hold. The first assumption requires that $d_{var} \leq (1-\gamma)^2\epsilon/(2\gamma)$. From Lemma 6 this holds if

$$\frac{||\beta_2 - \beta_1||_2^2}{\sqrt{\lambda_N}} + \sqrt{\frac{N^2\varepsilon}{\lambda_N} + \frac{2N^3 B_\sigma \varepsilon}{\lambda_N^2 - N^{1.5}\max_{ij}|\tilde{\sigma}_{ij} - \sigma_{ij}|\lambda_N}} \leq \frac{(1-\gamma)^2\epsilon}{2\gamma} \tag{4.18}$$

and

$$\max_{ij} |\tilde{\sigma}_{ij} - \sigma_{ij}| \leq \varepsilon. \tag{4.19}$$

We can ensure Equation 4.18 holds by splitting the error into three terms:

$$\begin{aligned} \frac{||\beta_2 - \beta_1||_2}{\sqrt{\lambda_N}} &\leq \frac{(1-\gamma)^2\epsilon}{4\gamma} \\ \frac{N^2\varepsilon}{\lambda_N} &\leq \frac{(1-\gamma)^4\epsilon^2}{32\gamma^2} \\ \frac{2N^3B_\sigma\varepsilon}{\lambda_N^2 - N^{1.5}\max_{ij}|\tilde{\sigma}_{ij} - \sigma_{ij}|\lambda_N} &\leq \frac{(1-\gamma)^4\epsilon^2}{32\gamma^2}. \end{aligned}$$

Given these three equations, and Equation 4.19, we can obtain bounds on the error in the dynamics parameter estimates:

$$||\tilde{\beta} - \beta||_2 \leq \frac{(1-\gamma)^2\epsilon\lambda_N^{0.5}}{4\gamma} \tag{4.20}$$

$$\max_{ij}|\tilde{\sigma}_{ij} - \sigma_{ij}| \leq \frac{(1-\gamma)^4\epsilon^2\lambda_N}{32\gamma^2N^2} \tag{4.21}$$

$$\max_{ij}|\tilde{\sigma}_{ij} - \sigma_{ij}| \leq \frac{(1-\gamma)^4\epsilon^2\lambda_N^2}{16\gamma^2N^3B_\sigma + (1-\gamma)^4\epsilon^2N^{1.5}\lambda_N}. \tag{4.22}$$

Assume[2] that $\lambda_N \leq 1$, $N > 1$ and $B_\sigma \geq 1$. In this case the upper bound in Equation 4.22 will be at least as small as the upper bounds in Equations 4.20 and 4.21.

We therefore require that the error ε in the model parameters be bounded by

$$\varepsilon \leq \frac{(1-\gamma)^4\epsilon^2\lambda_N^2}{16\gamma^2N^3B_\sigma + (1-\gamma)^4\epsilon^2N^{1.5}\lambda_N}$$

(from Equation 4.22). Lemma 2 provides a guarantee on the number of samples $T = \frac{12N^2B_\sigma^2}{\varepsilon^2 g}$ required to ensure with probability at least $1 - g$ that all the model parameters have error of most ε. In order to ensure that the model parameters for all actions and types simultaneously fulfill this criteria with probability δ, it is sufficient from the union bound to require that $g = \delta/(|A|N_T)$. We can then substitute this expression for g and Equation 4.23 into the expression for the number of samples T:

$$T = \frac{12N^2B_\sigma^2}{\left(\frac{(1-\gamma)^4\epsilon^2\lambda_N^2}{16\gamma^2N^3B_\sigma+(1-\gamma)^4\epsilon^2N^{1.5}\lambda_N}\right)\frac{\delta}{|A|N_T}} = \frac{12N^2|A|N_TB_\sigma^2(16\gamma^2N^3B_\sigma + N^1.5\lambda_N(1-\gamma)^4\epsilon^2)^2}{(1-\gamma)^8\epsilon^2\delta\lambda_N^4}.$$

Therefore the first assumption of the Simulation Lemma holds with probability at least $1-\delta$ after

$$O\left(\frac{N^8|A|N_TB_\sigma^4}{(1-\gamma)^8\epsilon^4\delta(\lambda_N)^4}\right)$$

samples.

[2] This is just a simplifying assumption and it is trivial to show the bounds will have a similar polynomial dependent on the parameters if the assumptions do not hold.

The second assumption in the Simulation Lemma requires that we have access to an MDP planner than can produce an approximate solution to our typed-offset-dynamics continuous-state MDP. At least one such planner exists if the reward model is Lipschitz continuous: under a set of four conditions, Chow and Tsititsiklis (1991) proved that the optimal value function V_ε of a discrete-state MDP formed by discretizing a continuous-state MDP into $\Theta(\varepsilon)$-length (per dimension)[3] grid cells is an ε-close approximation of the optimal continuous-state MDP value function, denoted by V^*:

$$||V_\varepsilon - V^*||_\infty \leq \varepsilon.$$

The first condition used to prove the above result is that the reward function is Lipschitz-continuous: in our work, the reward function is assumed to be given so this condition is a prior condition on the problem specification. The second condition is that the transition function is piecewise Lipschitz continuous, namely that the transition model is Lipschitz-continuous over each of a set of finite subsets that cover the state space, and that the boundary between each subset region is piecewise smooth. For each type and action our transition model is a Gaussian distribution which is Lipschitz-continuous, and there are a finite number of different types so it is piecewise Lipschitz-continuous. Therefore as long as our domain fulfills our earlier stated assumption that there are a finite number of different type regions, and the boundaries between each are piecewise smooth, then Chow and Tsitsiklis's second assumption is satisfied. The third condition is that the dynamics probabilities represent a true probability measure that sums to 1 ($\int_{s'} p(s'|s,a) = 1$), though the authors show that this assumption can be relaxed to $\int_s' p(s'|s,a) \leq 1$ and the main results still hold. In our work, our dynamics models are defined to be true probability models. Chow and Tsitsiklis's final condition is that there is a bounded difference between any two controls: in our case we consider only finite controls, so this property holds directly. Therefore, assuming the reward model fulfills the first condition, our framework satisfies all four conditions made by Chow and Tsitsiklis and we can use their result.

By selecting fixed grid points at a regular spacing of $\frac{(1-\gamma)^2\epsilon}{2\gamma}$, and requiring that there is at least one grid point placed in each contiguous single-type region, we can ensure that the maximum error in the approximate value function compared to the exactly solved value function is at most $\frac{(1-\gamma)^2\epsilon}{2\gamma}$, which provides a mechanism for ensuring the second assumption of the Simulation Lemma holds. An alternate phrasing of this requirement is that if the grid width used is the minimum of $\frac{(1-\gamma)^2\epsilon}{2\gamma}$ and the minimum contiguous length of a single-type region, then the resulting value function using this discrete approximate planner is no more than $\frac{(1-\gamma)^2\epsilon}{2\gamma}$-worse in value than the optimal exact planner for the continuous-state MDP.[4]

Therefore, given at least T samples, the Simulation Lemma (Lemma 10) guarantees that the approximate value of our known state MDP $\hat{M}_K$ is ϵ-close to the optimal value of the known state MDP with the true dynamics parameters M_K: $||\tilde{V}^{\pi}_{\hat{M}_K} - V^{\pi}_{M_K}||_\infty \leq \epsilon$. All

[3] More specifically, the grid spacing h_g must satisfy $h_g \leq \frac{(1-\gamma)^2\varepsilon}{K_1+2KK_2}$ and $h_g \leq \frac{1}{2K}$ where K is the larger of the Lipschitz constants arising from the assumptions discussed in the text, and K_1 and K_2 are constants discussed in (Chow & Tsitsiklis, 1991). For small ε any h_g satisfying the first condition will automatically satisfy the second condition.

[4] The condition of the extent of a typed region is a requirement in order to ensure that the discrete-representation doesn't skip over a smaller region of a different type, that may have a different optimal policy.

unknown type–action pairs that have not yet been experienced N_{at} times are considered to be unknown and their value is set to $V_{\max}$. So, conditions (1) and (2) of Strehl, Li and Littman (2006) hold. The third condition limits the number of times the algorithm may experience an unknown type–action tuple. Since there are a finite number of types and actions, this quantity is bounded above by $N_{at}N_T|A|$, which is a polynomial in the problem parameters $(N, |A|, N_T, \frac{1}{\epsilon}, \frac{1}{\delta}, \frac{1}{1-\gamma}, \lambda_N, B_\sigma)$. Therefore, our algorithm fulfills the three criteria laid out and the result follows. □

4.2.3 Discussion

Prior PAC-MDP work has focused predominantly on discrete-state, discrete-action environments. The sample complexity of the R-max algorithm by Brafman and Tennenholtz (2002) scales with the number of actions and the square of the number of discrete states since a different dynamics model is learned for each discrete state–action tuple. In environments where states of the same type share the same dynamics, Leffler, Littman and Edmunds (2007a) proved that the sample complexity scales with the number of actions, number of discrete states, and number of types: assuming the number of types is typically much less than the number of states, this can result in significantly faster learning, as Leffler et al. demonstrate empirically. However, a naïve application of either technique to a continuous-state domain involves uniformly discretizing the continuous-state space, a procedure that results in a number of states that grows exponentially with the dimension of the state space. In this scenario the approaches of both Brafman and Tennenholtz and Leffler et al. will have a sample complexity that scales exponentially with the state space dimension, though the approach of Leffler, Littman and Edmunds (2007a) will scale better if there are a small number of types. In contrast, the sample complexity of our approach scales polynomially in the numbers of actions and types as well as state space dimension, suggesting that it is more suitable for high dimensional environments. Our results follow the results of Strehl and Littman (2008) who gave an algorithm for learning in continuous-state and continuous-action domains that has a sample complexity that is polynomial in the state space dimension and the action space dimension. Our work demonstrates that we can get similar bounds when we use a more powerful dynamics representation (allowing states to have different dynamics, but sharing dynamics within the same types), learn from experience the variance of the dynamics models, and incorporate the error due to approximate planning.

The analysis presented considers the discounted, infinite-horizon learning setting. However, our results can be extended to the H-step finite horizon case fairly directly using the results of Kakade (2003). Briefly, Kakade considers the scenario where a learning algorithm $\mathcal{A}$ is evaluated in a cycling H-step periods. The H-step normalized undiscounted value U of an algorithm is defined to be the sum of the rewards received during a particular H-step cycle, divided by $1/H$. Kakade defines $\mathcal{A}$ to be ϵ-optimal if the value over a state-action trajectory until the end of the current H-step period within ϵ of the optimal value U. Using this alternate definition of the reward requires a modification of the Simulation Lemma which results in a bound on Δ_Q of $(H-1)V_{max}d_{var} + (H-1)\varepsilon_{plan}$: in short, the $\gamma/(1-\gamma)$ terms has been replaced with $H-1$. The earlier results on the number of samples required to obtain good estimates of the model parameters are unchanged, and the final result follows largely as before, except we now have a polynomial dependence on the horizon H of the undiscounted problem, compared to a polynomial dependence on the discount factor $1/\gamma$.

Finally, though our focus is on sample complexity, it is also important to briefly consider computational complexity. To ensure the approximate planner produces highly accurate results, our algorithm's worst-case computational complexity is exponential in the number of state dimensions. While this fact prevents it from being theoretically computationally efficient, our experimental results presented in the following section demonstrate that our algorithm performs well empirically compared to a related approach in a real-life robot problem.

4.3 Experiments

We first demonstrate the benefit of utilizing domain knowledge about the structure of the dynamics model on the standard reinforcement learning benchmark, PuddleWorld (Boyan & Moore, 1995). We then illustrate the importance of explicitly learning the variance parameters of the dynamics world (in contrast to some past approaches which assume it to be given, such as Strehl and Littman (2008)) in a simulated "catching a plane" experiment.

In addition, our approach presupposes that offset dynamics are reasonable approximation in several real world scenarios. In our second experiment we applied our algorithm to a simulated problem in which an agent needs to learn the best route to drive to work. In our final experiment we applied our algorithm to a real-world robot car task in which the car most cross varying terrain (carpet and rocks) to reach a goal location.

CORL requires a planner to solve the estimated continuous-state MDP. Planning in continuous-state MDPs is an active area of research in its own right, and is known to be provably hard (Chow & Tsitsiklis, 1989). In all our experiments we used a standard technique, Fitted Value Iteration (FVI), to approximately solve the current MDP. In FVI, the value function is represented explicitly at only a fixed set of states that are (in our experiments) uniformly spaced in a grid over the state space. Planning requires performing Bellman backups for each grid point: the value function over points not in this set is computed by function interpolation. In our experiments we used Gaussian kernel functions as the interpolation method. Using sufficiently small kernel widths relative to the spacing of the grid points will make the approach equivalent to using a nearest neighbour standard discretization but there are some practical advantages in coarse grids to more smooth methods of interpolation. We discuss this issue in more depth in the next section.

In each experiment N_{at} was tuned based on informal experimentation.

4.3.1 Puddle World

Puddle world is a standard continuous-state reinforcement learning problem introduced by Boyan and Moore (1995). The domain is a two-dimensional square of width 1 with two oval puddles, which consist of the area of radius 0.1 around two line segments, one from $(0.1, 0.75)$ to $(0.45, 0.75)$ and the other from $(0.45, 0.4)$ to $(0.45, 0.8)$. The action space consists of the four cardinal directions, and upon taking an action the agent moves 0.05

in the specified cardinal direction with added Gaussian noise $\mathcal{N}(0, 0.001 * I)$ where I is a two-by-two identity matrix. The episode terminates when the agent reaches the goal region which is defined as the area in which $x + y \geq 1.9$. All actions receive a reward of -1 unless the agent is inside a puddle: it then receives a reward of -400 times the distance inside the puddle.

We expect CORL will outperform prior approaches on this problem for two reasons. The first is that we assume the reward is given.[5] However even if the reward model is provided, most past work still has to learn the dynamics model for this world, which is a significant undertaking. The second reason is a feature of the CORL algorithm: its dynamics model uses the additional information that the dynamics for one action for all states of the same type are identical. Here there is only a single type and so CORL must learn only 4 sets of transition parameters, one for each action. Our point here is simply to demonstrate this additional information can lead to significantly faster learning over other more general techniques.

Puddle world has previously been used in reinforcement learning competitions and our evaluation follows the procedure of the second bakeoff (Dutech et al., 2005). We initially generated 50 starting locations, and cycled through these when initializing each episode. Each episode goes until the agent reaches the goal, or has taken 300 actions. We report results from taking the average reward within fifty sequential episodes (so the first point is the average reward of episodes 1-50, the second point is the average of episodes 51-100, etc.). We set N_{at} to be 15 and solved the approximate MDP model using Fitted Value Iteration with Gaussian kernel functions. The kernel means were spaced uniformly in a 20x20 grid across the state space (every 0.05 units), and their standard deviation was set to 0.01. Note that in the limit as the standard deviation goes to 0 the function interpolation becomes equivalent to nearest-neighbour, which is the interpolation method used in the approximate continuous-state MDP solver by Chow and Tsitsiklis (1991) which provides guarantees on the resulting value function approximation. However, since computational complexity scales exponentially with the grid discretization, practical application can require the use of coarse grids. In this case we found that the empirical performance of using a smoother method of function interpolation outperformed a nearest neighbour approach. In particular, we found that as we varied the kernel width (the standard deviation) from very small (much smaller than the variance of the dynamics models and the grid spacing) to large (such as 0.1, larger than the grid spacing and dynamics models) that a middle kernel standard deviation of 0.01 gave the best empirical performance.

We compare our results with the reported results of Jong and Stone (2007)'s Fitted R-max, which is an instance-based approach that smoothly interpolates the dynamics of unknown states with previously observed transitions, and takes a similar approach for modeling the reward function. On the first 50 episodes Fitted R-max had an average reward of approximately -130, though on the next 50 episodes it had learnt sufficiently good models that it reached its asymptotic performance of an average reward of approximately -20. In contrast, CORL learns a good model of the world dynamics within the very first episode

[5] If the reward is not provided, it can be learned with a slightly extended version of the framework presented in this chapter: however, if the reward does not similarly exhibit a typed structure then the theoretical guarantees of CORL will be significantly weakened.

Algorithm	Number of episodes			
	50	100	200	400
CORL	-19.9	-18.4	-20.3	-18.7
Fitted R-max	-130	-20	20	-20
LSPI	-500	-330	-310	-80

Table 4.1: PuddleWorld results. Each entry is the average reward/episode received during the prior 50 episodes. Results for the other algorithms are as reported by Jong and Stone(Jong & Stone, 2007).

(since it has the advantage of knowing that the dynamics across the entire state space are the same) which means that CORL performs well on all subsequent episodes, leading to an average reward on the first 50 episodes of -19.9. Jong and Stone also compared to Least Squares Policy Iteration (Lagoudakis & Parr, 2003) which learns slower than Fitted R-max. Results are summarized in Table 1.

In summary, given the reward function, CORL can extremely quickly learn a good dynamics model in puddle world, since the dynamics are typed-offset with a single type. This additional information enables CORL to learn a good policy for puddle world much faster than Fitted R-max and LSPI. This experiment illustrates the advantage of CORL when the dynamics model is known to be identical over regions of the state space.

4.3.2 Catching a Plane

We next consider some examples with multiple types. For instance, thousands of people now rely on internet maps or GPS units to do efficient trip planning, and better traffic prediction is an area of recent research (Horvitz et al., 2005). In many street systems there exist a number of different classes of road types according to the designated speed limits associated with these roads. These different road types are often also associated with different variances. For example, while highways have mean speeds that are faster than small side streets, highways typically have a very large variance: rush hour highway speeds may often be slower than smaller side streets.

In our first experiment we considered a scenario in which learning this variance is critical: an agent must learn the best way to drive to an airport in time to catch a plane. A similar real-world environment is depicted in Figure 1: the agent starts from home and can either drive directly along a small side street, or cross over to the left or right to reach a main highway. The agent goes forward more quickly on the highway (with a mean offset of 2 units) but the highway also has a high variance of 0.49. In contrast, on the small side streets the agent goes forward more slowly (a mean offset of 1 unit) but with very small variance (0.00001). The state space is four dimensional, consisting of the agent's current x and y location, its orientation, and the amount of time that has passed. At each step 5 minutes pass. The region the agent is allowed to drive in is 14 by 19: outside of this is considered to be too far away.[6] If the agent exits this region it receives a reward of -1.

[6] Note that this could be a reasonable assumption in some cases, such as when doctors had to stay within

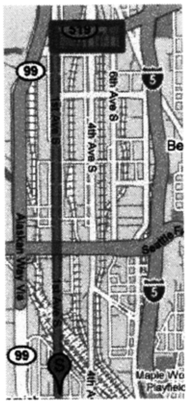


Figure 4-2: Motivating example where an agent must drive from a start location (designated with an S) to a goal area (shown with a highlighted rectangle) in time for a deadline, and can choose to take large high speed and variance roads, or slower small variance roads.

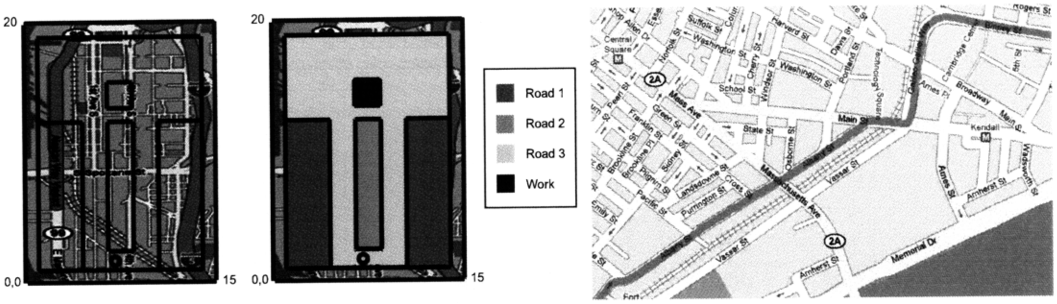
The cost for each step is -0.05, the reward for reaching the airport in time for check in is $+1$ and the cost for not making the airport in time is -1. The discount factor is set to 1.0. N_{at} was set to 10. An episode starts when the agent takes its first step and lasts until the agent reaches the goal, exits the allowed region or reaches the time at which check in for the plane closes. The agent starts at location $7, 3$ facing north, and must figure out a way to reach the goal region which spans $[6.5 - 8.5, 15.5\text{-}17.5]$. To solve the underlying MDP, fitted value iteration was used with regularly spaced grid points with 15 across the x dimension, 20 across the y dimension, 4 orientation angles, and 21 time intervals: this yielded over 25,000 fixed points.

The correct path to take depends on the amount of time left. Here we explored three different deadline scenarios: when the agent has 60 minutes remaining ($RunningLate$), 70 minutes remaining ($JustEnough$), or 90 minutes remaining until check in closes for the plane ($RunningEarly$). In all three scenarios within 15 episodes the agent learned a good enough model of the dynamics to compute a good policy: note that using a naïve discretization of the space with the FVI fixed points as states and applying R-max would be completely intractable, as a different model would have to be learned for each of over 25,000 states. Results for all three scenarios are displayed in Table 2. In the $RunningLate$ scenario the agent learned that the side streets are too slow to enable it to ever reach the airport in time. Instead it took the higher variance highway which enables it to reach the

a certain radius of a hospital when on call.

Scenario	Deadline (min)	Average Reward/Episode	Average Time to Reach Goal (min)
$RunningLate$	60	-0.4833	56.62
$JustEnough$	70	0.45	60
$RunningEarly$	90	0.4629	58.6

Table 4.2: Catching a plane: results after N_{at} has been reached for all types.



(a) (b) (c)

Figure 4-3: (a) Simulated world map showing example type of environment. (b) World divided into a set of discrete types. Agent starts at driveway (black circle) and tries to learn a good route to reach work. (c) An example section of a car trajectory (constructed from a set of GPS locations).

airport in time in over half the episodes after N_{at} is reached for all types: its average reward is -0.4833 and it takes on average 56.62 minutes for it to reach the airport. In $JustEnough$ the agent learned that the side streets speed is sufficiently fast for the agent to reach the airport consistently in time, whereas the higher speed and variance highway would result in the agent failing to reach the check in time in some cases. Here the agent always reached the goal and receives an average reward of 0.45. In the $RunningEarly$ scenario the agent has enough time so that it can take either route and reliably reach the airport in time. In this scenario it learned to always take the highway, since in expectation that route will be faster. The average reward here was 0.4629.

This simulation serves to illustrate that our algorithm can quickly learn to perform well in situations where learning the variance is critical to ensuring good performance.

4.3.3 Driving to Work

In our second experiment we again consider a simulated trip routing problem, but we now generate transitions in the simulator by sampling from real traffic data distributions. Here an agent must learn the best series of actions to drive from home to work in a small simulated world (see Figure 4.3.3a & b). The state consists of the current coordinates (x, y) and the orientation of the agent. There are three road types and each road type is associated with a different distribution of speeds. The distributions were obtained from the CarTel project (Eriksson et al., 2008), which consists of a set of car trajectories from the Boston, Massachusetts area. GPS locations and time stamps are stored approximately every second from a fleet of 27 cars.[7] A section from one car trajectory is shown in Figure 4.3.3c. Using this dataset we extracted car trajectories on an interstate highway, small side streets and a local highway: these constitute types 1, 2 and 3 respectively in the simulation world. Each car trajectory consisted of a set of D GPS+time data points, which was converted into a set of $D - 1$ transitions. Each transition in the simulation was sampled from these real-world transitions: for example, transitions in the simulator on road type 2 were sampled from

[7] See more information about the project at http://cartel.csail.mit.edu/.

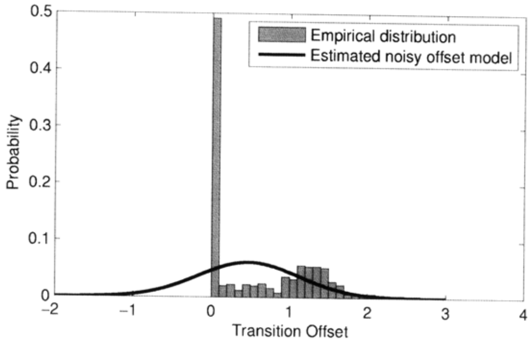
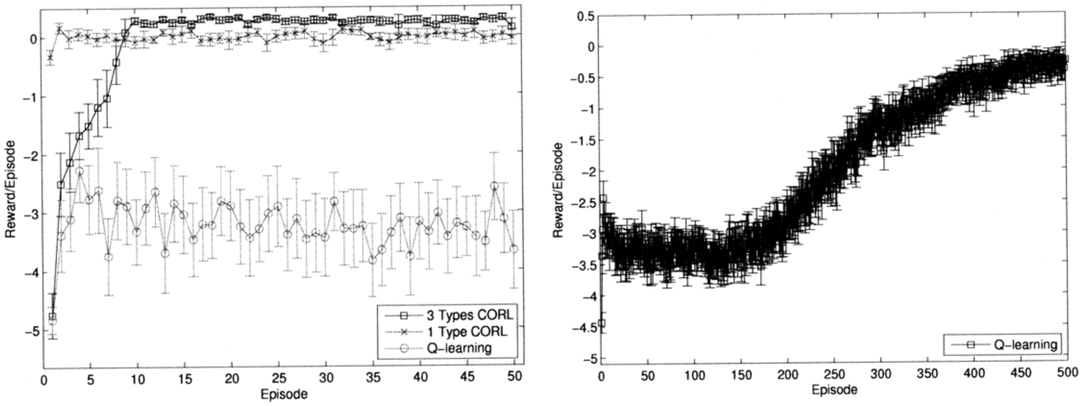


Figure 4-4: A histogram of car speeds on small roads that is used to generate transitions on road type 2 and the estimated dynamics model parameters found during the experiment

real-world transitions on small side streets. Transitions from all three road types were all rescaled by the same constant in order to make the distances reasonable for the simulated world.[8]. Figure 3 displays a histogram of rescaled transitions associated with small side streets. This figure shows that the speed distribution for small side streets was not perfectly Gaussian: the speed distribution for the other two street types was also not perfectly Gaussian. In particular, in no trajectories used does the car ever go backwards, whereas in some Gaussian models there will be small probability of this occurring. In this experiment we sought to investigate how well a noisy-offset model could function in this environment, and the benefit of directly modelling different types of roads. Each transition in the simulated environment was sampled from the histogram of speeds associated with the road type at the agent's current position. Therefore, the data from the simulator is closer to the real environment than to the Gaussian distributions assumed by the learning algorithm.

The agent received a reward of 1 for reaching the work parking lot, -0.05 for each step, and -1 if it left the local area. Each episode finished when the agent either reached the goal, left the local area, or had taken 100 steps. An agent can go left, right or straight at each step. The transition induced by a straight action was determined by the road type as specified in the prior paragraph, and going left or right changed the orientation of the agent by 90 degrees with a very small amount of noise. The number of samples needed until a type–action tuple is known, N_{at}, was set to be 20. The discount factor was 1. The agent was always started in the same location and allowed to learn across a set of 50 episodes. Results were averaged across 20 rounds of 50 episodes per round. In one experiment the agent was given full knowledge of the three world types, and learned a different dynamics model for each type and action. In the second experiment the agent assumed there was only a single type and learned a dynamics model for each action. We also compared our approach to Q-learning over a uniformly-spaced discrete grid over the environment with an ϵ-greedy policy. We used a discretization that was identical to the fixed points used in the fitted value iteration planner of CORL. Points were mapped to their nearest neighbors. Q-learning requires specifying two parameters: the learning rate α which determines how much to adjust the state-action value estimates after each update, and ϵ which specifies how

[8] We also removed outlier transitions, as extremely fast speeds/transitions were likely to be errors in the log file.



(a)

(b)

Figure 4-5: Reward versus episode. (a) Compares CORL with 3 types and 1 type to Q-learning. Results are averaged over 20 rounds (50 episodes per round). Error bars show 95% confidence intervals. (b) Shows Q-learning with 500 episodes per round, averaged over 100 rounds.

often to take a random action instead of the action that maximizes the current Q values. In this experiment α was set to 1.0 and decreased by multiplying by a factor of 0.9999 at each step.[9] We set ϵ to be 0.1.

The CORL results are displayed in Figure 4-5a. This figure displays three encouraging results. The first is that in both CORL algorithms the agent learned to consistently reach the goal: the only way that the agent can receive a reward greater than -1 is to reach the goal, and all confidence intervals lie above -1 for all episodes after 10, indicating that the agent in both cases was successfully reaching the goal. This is promising because even though the underlying dynamics models were not exactly Gaussian noisy-offset dynamics, a noisy-offset model approximation was sufficient for the agent to learn a good policy in this environment. The estimated parameters computed for one type and action are displayed in Figure 3.

The second result is that the policy found by the agent that models all three types differently resulted in significantly higher reward than modeling the world with a single type: its performance suffered initially because it takes longer to learn a model of the world dynamics, but from about episode 10-50 modelling all types separately resulted in significantly higher reward per episode than modelling all types as the same. Table 3 displays the average reward of both approaches on episodes 10-50. These results demonstrate that traffic data does exhibit different speed distributions on different types of roads, and that by considering such differences CORL can improve route directions even in a small simulated example.

The third result is that both CORL algorithms significantly outperformed Q-learning: again see Table 3 for comparing the short term performance of Q-learning to the CORL algorithm. This is not surprising since Q-learning is a model-free approach that trades

[9] We tried different decay factors for the α parameter but found that this worked better than decaying α more rapidly.

Algorithm	Average reward/episode
CORL with 3 types	0.27
CORL with 1 type	0.00
Q-learning	-3.2485

Table 4.3: Average reward on episodes 10-50 for the driving to work example.

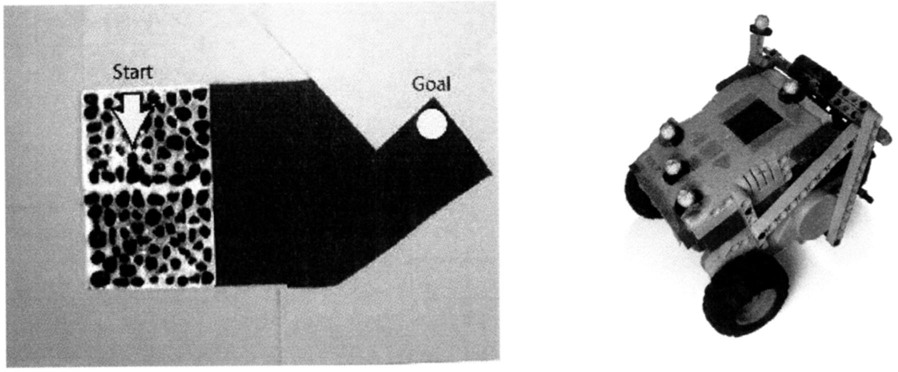
off speed of computation per step in return for not requiring consistency between its state values through a learned model. Here in particular there is a large amount of structure in the domain that Q-learning cannot use. Q-learning does eventually begin to consistently reach the goal but this is only after about 500 episodes, more than an order of magnitude longer than the CORL algorithms took to find a good policy. These results are displayed in Figure 4-5b. Such results argue that in situations where data is costly to gather, using a model can be extremely helpful.

4.3.4 Robot navigation over varying terrain

We also tried our algorithm in a real-life robotic environment involving a navigation task where a robotic car must traverse multiple surface types to reach a goal location. This experiment is a second exemplar of where noisy-offset dynamics might provide a sufficiently good representation of real-world dynamics to allow our algorithm to learn good policies. We compared to Leffler, Littman and Edmunds (2007a)'s RAM-Rmax algorithm, a provably efficient RL algorithm for learning in discrete-state worlds with types. The authors demonstrated that, by explicitly representing the types, they could get a significant learning speedup compared to R-max, which learns a separate dynamics model for each state. The RAM-Rmax algorithm represents the dynamics model using a list of possible next outcomes for a given type. CORL works directly with continuous-valued states, resulting in the improved sample complexity discussed earlier. This is achieved through assuming a fixed parametric representation of the dynamics, which is a less flexible model than the one used in RAM-Rmax. In this experiment we were interested in whether our representation was still rich enough to capture the real world dynamics involved in varying terrain traversal. We also investigated the computational load of CORL compared to RAM-Rmax, since by restricting our representation size we hoped to also achieve computational savings.

In this experiment we ran a LEGO® Mindstorms NXT robot (see Figure 4-6b) on a multi-surface environment. A tracking pattern was placed on the top of the robot and an overhead camera was used to determine the robot's current position and orientation. The domain, shown in Figure 4-6a, consisted of two types: rocks embedded in wax and a carpeted area. The goal was for the agent to begin in the start location (indicated in the figure by an arrow) and end in the goal without going outside the environmental boundaries. The rewards were -1 for going out of bounds, $+1$ for reaching the goal, and -0.01 for taking an action. Reaching the goal and going out of bounds ended the episode and resulted in the agent getting moved back to the start location.[10]

[10] A video of the task can be seen at http://people.csail.mit.edu/emma/corl/SuccessfulRun.mov and



(a)

(b)

Figure 4-6: (a) Image of the environment. The start location and orientation is marked with an arrow. The goal location is indicated by the circle.(b) LEGO® robot.

Due to the close proximity of the goal to the boundary, the agent needs an accurate dynamics model to reliably reach the goal. Part of the difficult of this task is that the actions were going forward, turning left, and turning right. Without the ability to move backwards, the robot needed to approach the goal accurately to avoid falling out of bounds.

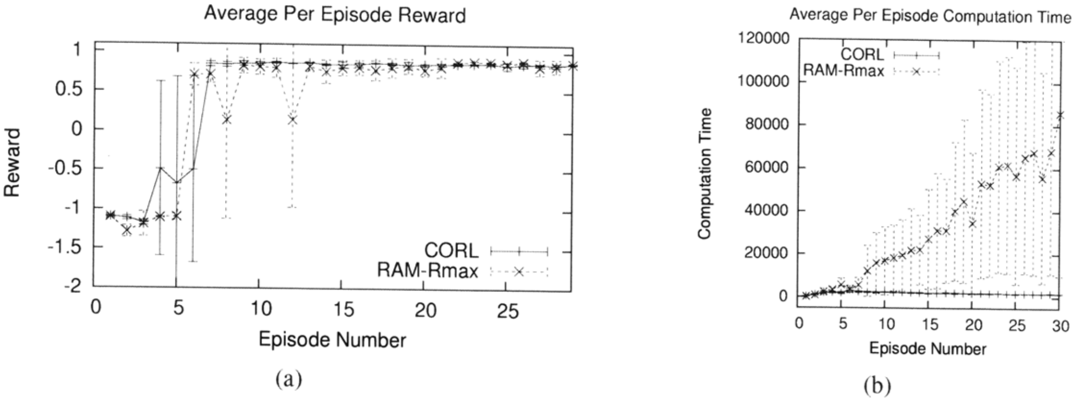
For the experiments, we compared our algorithm ("CORL") and the RAM-Rmax algorithm ("RAM"). The fixed points for the fitted value iteration portion of our algorithm were set to the discretized points of the RAM-Rmax algorithm. Both algorithms used an EDISON image segmentation system to uniquely identify the current surface type. The reward function was provided to both algorithms.

The state space is three dimensional: x, y, and orientation. Our algorithm implementation for this domain used a full covariance matrix to model the dynamics variance model. For the RAM-Rmax agent, the world was discretized to a forty-by-thirty-by-ten state space. In our algorithm, we used a function approximator of a weighted sum of Gaussians, as described in Section 4.1.1. We used the same number of Gaussians to represent the value function as the size of the state space used in the discretized algorithm, and placed these fixed Gaussians at the same locations. The variance over the x and y variables was independent of each other and of orientation, and was set to be 16. To average orientation vectors correctly (so that $-180°$ degrees and $180°$ do not average to 0) we converted orientations θ to a Cartesian coordinate representation $x_\theta = \cos(\theta), y_\theta = \sin(\theta)$. The variance over these two was set to be 9 for each variable (with zero covariance). For our algorithm and the RAM-Rmax algorithm, the value of N_{at} was set to four and five, respectively, which was determined after informal experimentation. The discount factor was set to 1.

Figure 4.3.4a shows the average reward with standard deviation for each of the algorithms over three runs. Both algorithms are able to receive near-optimal reward on a consistent basis, choosing similar paths to the goal. Our dynamics representation is sufficient to allow our algorithm to learn well in this real-life environment.

In addition, by using a fixed size (parametric) dynamics representation, the computational time per episode of our algorithm is roughly constant (Figure 4.3.4b). In the im-

http://people.csail.mit.edu/emma/corl/SuccessfulRun.wmv.



(a)

(b)

Figure 4-7: (a) Reward received by algorithms averaged over three runs. Error bars show one standard deviation. (b) Total time taken by algorithms averaged over three runs. Error bars show one standard deviation.

plementation of RAM-Rmax, the computational time grew with the number of episodes due to its dynamics model representation. This suggests that using a fixed size dynamics representation can have significant computation benefits. Overall CORL performed well in this domain, both in terms of reward achieved and computation required.

4.4 Limitations & Discussion

Beyond those already discussed previously in this chapter, there are a few other limitations of the CORL algorithm it is important to explicitly discuss. The first is that the CORL algorithm assumes the reward model is known. In some situations this may be reasonable: for example, in the case of a driver navigating to reach a destination, her insurance may have told her the cost of crashing with another car, and if her destination is the airport where a plane is waiting for her, she may know the value of the airplane ticket should she not reach her destination in time. However, in other scenarios, and in the standard RL setup, the reward model is assumed to be unknown. Though we have demonstrated in this chapter that there are some real-world examples where assuming a region of states share the same dynamics, this does not imply that all states within the same region share the same reward. Indeed, several of our experiments involves instances where of states with the same type, a small subset of those states had a different reward model. In such examples we might be able to similarly think of the states as having reward types, and apply a similar approach to learning the reward models.

In general, the advantage of CORL will depend on having a fairly small number of dynamics types (plus reward types if the reward is unknown): if the number of types scales exponentially with the state dimension, as would be the case if the underlying world is essentially different for each discrete grid cell, then CORL will not have an advantage over learners that assume a discrete state space.

A perhaps more significant concern could come from learning the typed models. The underlying assumption of CORL is that the dynamics model for all states are the same.

However, if the states in a single type violate this assumption, such as if some states within a single type transition to another type (such as from carpet to rocks) and other states within the same type transition to another type (from carpet to carpet) then this assumption will be violated. This problem is not unique to CORL: a similar problem faces the RAM-Rmax representation of Leffler et al. (2007a). Indeed, this problem even affects learners that assume a discrete environment (such as R-max) since they assume that the dynamics model for all scenarios labeled as a particular discrete-state share the same dynamics model. In practice this is unlikely to be true unless the number of types is extremely large or (equivalently) the resolution of the discretization is extremely small. For example, in our robotic experiment, the dynamics models of rocks right next to the carpet is likely to be different than the dynamics model of the agent when it is on rocks and can only reach other rocks in a single action. It would be interesting to consider trying to learn the right partitioning of the state space into types in order to limit the severity of this problem: Leffler (2008) has recently investigated this issue, mostly in discrete-state environments. However, learning the types as well as learning the dynamics model could in the worst case eliminate the improvement in sample complexity gained by assuming types, as was proved by counter-example in the discrete-state setting by Leffler et al. (2005).

Finally, CORL currently uses a MDP planner which essentially discretizes the continuous-state known MDP. This means that the computational complexity of CORL is an exponential function of the state space dimension. However, it may be possible to instead use forward search MDP planning approaches such as those developed by Kearns, Mansour and Ng (2002). These approaches compute ϵ-close estimates of the state-action values for the current state only, and do so with a computational cost which is independent of the size of the state space. These approaches achieve this through a computational cost which is an exponential function of the horizon of the forward search, with often a significant constant. Still it would be quite interesting to try extend the CORL bounds to when a forward search planner is used for solving the known MDP, and also to examine the practical performance compared to using a discretized MDP planner. We will consider forward search planners in more depth in Chapter 5 in the context of partially observable planning, and demonstrate how we can get a significant speedup there by using parametric representations.

4.5 Summary

This chapter presented CORL, an algorithm for efficiently learning to act in typed, continuous-state environments. CORL has a sample complexity that scales polynomially with the state space dimension and the number of types: this bound also directly incorporates the error due to approximate planning. Experiments on a simulated driving example using real world car data, and a small robot navigation task, suggest that noisy-offset dynamics are a sufficiently rich representation to allow CORL to perform well in some real-world environments. CORL demonstrates that parametric models can significantly speed up learning in continuous-valued, high-dimensional environments.

The necessity for action compels us ... to behave as if we had behind us ... a series of ... advantages and disadvantages, each multiplied by its appropriate probability, waiting to be summed.

John Keynes

5 Parametric Representations in Forward Search Planning

We now again consider the challenge of planning in large, high-dimensional, partially observed, uncertain environments. We will continue to use our running example of an experienced driver who is making a series of decisions about how to reach her final destination as quickly as possible, while avoiding surrounding cars. The driver must rely only on local sensor observations, such as the view through her right side mirror. In this chapter our focus will be on producing an algorithm that is tractable in this very large and high-dimensional domain, where the state variables can include the location of the agent's car, the location of all other cars, the current weather, whether the driver of each other car is talking on the phone, the current gas prices, and a range of other variables.

Though our prior algorithm for planning in large environments, the SM-POMDP algorithm (Chapter 3), was empirically faster than discrete-state planners on a continuous-valued problem, it is limited in scale by the challenge of compactly specifying the dynamics model. The switching mode dynamics model will typically require an exponential (in the number of dimensions) Gaussians in order to create a dynamics model that was approximately normalized. The SM-POMDP backup operation means that the number of parameters necessary to represent the value function increases by a factor that is exponential in the number of problem dimensions at each time step, which is intractable in problems with more than a small number of dimensions.

This chapter will present the Low-Order Parametric POMDP (LOP-POMDP) planner which scales more gracefully to high-dimensional, continuous-valued environments: Figure 5-1 displays graphically how this approach relates to the space of POMDP planning techniques along the axes of optimality, tractability and expressive power. LOP-POMDP has two key characteristics. First, instead of computing an explicit representation of the value function over the full state space, planning will be performed in an online, forward-search manner, computing values and corresponding decisions only for the situations encountered. Second, we will use parametric models that allow the computational complexity of the forward search operations to be a polynomial function of the number of domain dimensions: this enables the approach to scale to much higher-dimensional problems. The majority of this chapter will focus on using unimodal parametric models which have limited expressive power, but we also discuss alternate representations which may incur larger computational cost in Sections 5.2.3 and 5.6. We also demonstrate how this approach can be used to solve a large, high-dimensional simulated driving problem such as the example

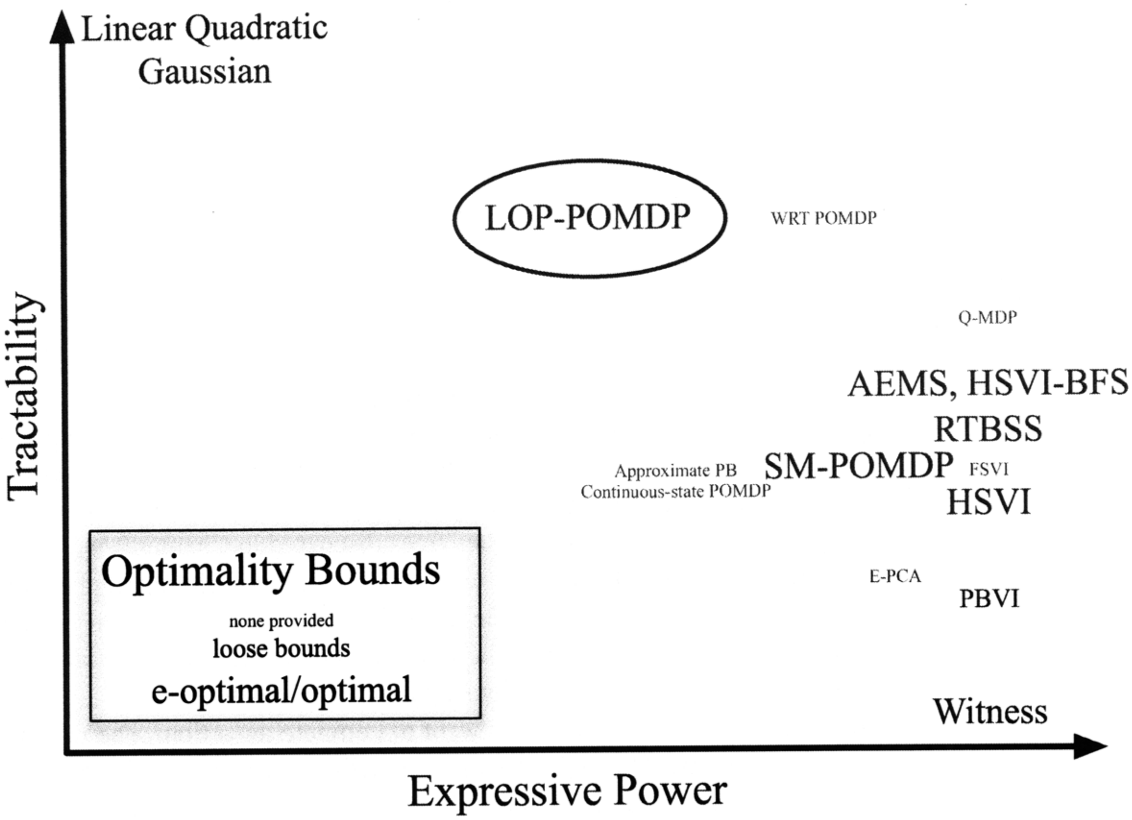


Figure 5-1: Optimality, tractability and expressive power of a representative set of POMDP planning techniques.

we have used throughout this thesis.

5.1 Forward Search POMDP Planning

Very efficient online planners using forward search have demonstrated impressive performance in many high-dimensional fully observable domains such as game-playing, robotics and computational biology (e.g. LaValle and Kuffner, 2001). This makes the idea of using forward search for stochastic, partially observable planning attractive, and due to the argument just described, it may allow good solutions to be found in environments in which it is computationally intractable to represent and compute the value function explicitly.

We first outline how forward search could be applied to produce a POMDP planning algorithm. The key idea is that rather than computing a value function or policy in advance of acting, instead forward search will be used at each step to estimate the expected value of taking any of the potential actions from the current belief state. The action with the best value will be taken. The belief state will then be updated given the action taken and the received observation, and the whole process will begin again. Algorithm 5 outlines the

Algorithm 5 Online POMDP Planning with Forward Search

```
nput: Initial belief state $b_0$
$b = b_0$
loop
    $\forall a Q(a, b)$ = ComputeBeliefQValuesWithForwardSearch(b,a)
    Take action $a = \text{argmax}\, Q(a)$
    Receive observation $z$ and reward $r$
    $b = BeliefUpdate(b, a, z)$
end loop
```

main steps in using forward search in an online POMDP planner.

The key step in Algorithm 5 is *ComputeBeliefQValuesWithForwardSearch*, which is an algorithm for using forward search to estimate the expected future value of the current belief, given any of the potential actions are taken. These values can be computed by constructing a tree similar to the conditional policy trees discussed in Chapter 2.

To estimate each belief-action value $Q(b, a)$ we construct a tree by alternating between considering all possible actions, and all possible observations, out to some fixed horizon length. This forward search, or rollout, technique effectively considers all potential future trajectories of alternating actions and observations up to a finite length.

At the tree root is the belief b for which we want to compute an action. The tree first branches on all possible actions. Then underneath each action node, the tree branches on all possible observations. Each observation node z_i is associated with a new belief state, the belief state that results from taking the parent action a_k and the receiving the observation z_i: $b^{a_k z_i}$. The observation node is also associated with the expected reward for this belief $b^{a_k z_i}$

$$R(b^{a_k z_i}) = \int_s R(s) b^{a_k z_i}(s) ds$$

where for simplicity we have assumed that the reward is a function only of the state. Each observation node also stores the probability of receiving this particular observation z_i given the parent belief b and action a_k, denoted by $p(z_i|a_k, b)$, which is calculated as

$$p(z_j|a_i, b) = \int_{s'} p(z_j|s', a_i) \left(\int_s p(s'|s, a) b(s) ds \right) ds'.$$

Figure 5-2 displays this first step of the forward search tree construction process. We then repeatedly branch on all actions, then all observations, updating the beliefs at each observation node and calculating the associated expected reward, until we reach a certain horizon H. The choice of horizon is typically chosen either to ensure certain optimality guarantees on the resulting state-action value estimates, or according to computational constraints.

After the tree is constructed, we can then use it to calculate the belief-action value estimates at the root node. We start at the leaf observation nodes and take an expectation over all expected rewards across all observations, weighted by the probability of that observa-

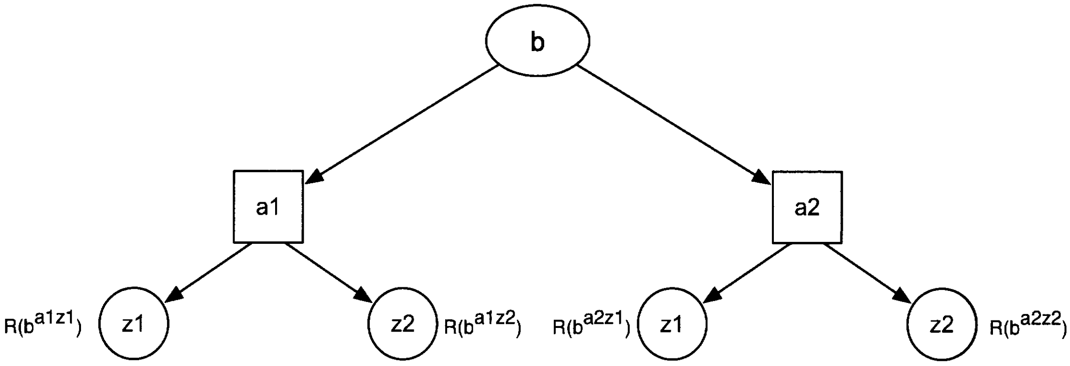


Figure 5-2: The first step of constructing a forward search tree for POMDP planning.

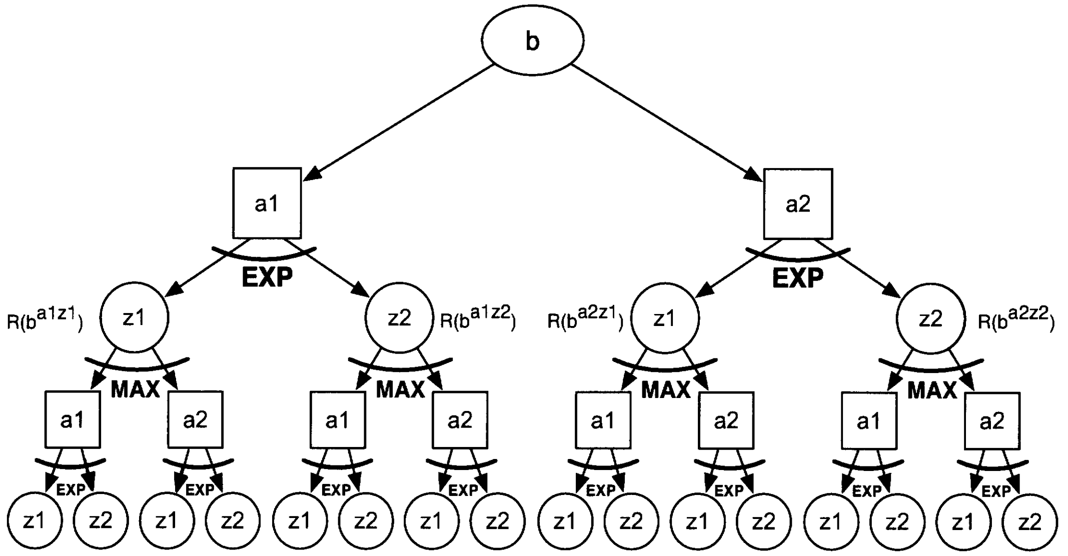


Figure 5-3: The process of computing the state-action values at the root belief involves repeatedly computing the expectation over observation node rewards and the maximum over action nodes. As the value is propagated up the tree, additional rewards from higher level nodes are added to the sum.

tion occurring given its parent action and belief. We then choose the highest expected value across all actions (since an agent can always choose its action, but must accept whatever observation it receives). This process is repeated all the way up the tree, essentially computing the H-step horizon expected value.[1] Figure 5-3 displays this process for a $H = 2$ forward search tree. The full algorithm for computing the expected state-action values is displayed in Algorithm 6.

Though the online procedure presented so far assumes a finite number of observations that are enumerated in the tree, in many situations this set will be extremely large or infinite.

[1] Values from lower down in the tree are multiplied by the discount factor γ as the sum is computed.

Algorithm 6 ComputeNodeValue()

```
input: Node $n$
output: Value $V$
if $n$ is a leaf node then
  return immediate expected reward $R_n$
else if $n$ is an observation node then
  $bestActionValue = minValue$
  for all children action nodes do
    $aValue = ComputeNodeValue(ChildActionNode)$
    if $aValue \geq bestActionValue$ then
      $bestActionValue = aValue$
    end if
  end for
  return $R_n + \gamma bestActionValue$
else {action node}
  sumZValue = $0$
  for all child observation nodes $z$ do
    sumZValue = sumZValue+$p(z|a, currentbelief)$ComputeNodeValue(Child $z$)
  end for
  return $R_n + \gamma sumZValue$
end if
```

In such situations it will be necessary to approximate the expectation over the observation nodes by sampling a finite number of observations.

Therefore, in order to use forward search for POMDP planning, we require methods for three key operators:

1. Belief updating
2. Observation sampling
3. Computing the immediate expected reward for a belief state

While forward search for POMDP planning has been considered previously (see Ross et al. (2008b) for a recent survey), with few exceptions (e.g. Ross et al. (2008a)), the prior work has focused on discrete-state and discrete-observation planning. Discrete-state representations offer impressive expressive power, because with a fine enough discretization they can represent to a high degree of precision almost any dynamics, observation or reward model. However, their expressive nature means that they require as many parameters to represent a regular shape, such as a Gaussian or rectangle, as they do to represent a highly unusual and irregular model.[2]

The expressive power of discrete-state representations can significantly limit their tractability in large, high-dimensional environments. Consider the computational complexity of the above three operators: if the state space is represented as a discrete set of

[2] Naturally if ones knows that the belief state will be sparse or of a particular form, that knowledge can be exploited. The point here is that if this assumption is not applied, then standard discrete approaches can fail.

states with cardinality $|S|$, then the computational complexity of the belief update operation is $O(|S|^2)$. Updating the belief based on the action taken must occur before sampling from the set of observations, and so this operation is also $O(|S|^2)$. Finally, computing the immediate expected reward involves integrating over the state space, yielding a $O(|S|)$ operation.

Therefore it is important to consider the resulting size of a discrete-state representation that results from a standard discretization[3] of a high-dimensional, continuous-valued domain. In this procedure two factors influence the resulting size of the state space: the number of dimensions D, and each dimension d's range r_d. A uniform discretization of each dimension into grid cells of length u will result in a state space size of approximately $(r_d/u)^D$. Therefore, the state space will increase exponentially with the number of dimensions, with a base that depends on the range of each dimension. In large (high r_d) high-dimensional (large D) domains this will cause the three forward search operators described to take $O((r_d/u)^D)$ time.

Therefore planning time will scale *exponentially* with the dimension of the domain, making planning computationally intractable in large environments. In order to search efficiently in the belief space of large, high dimensional problems, we require compact representations and fast algorithms for performing the three forward-search operations.

5.2 Model Representation

Here we use compact parametric representations to encode the world observation and dynamics models. In particular the models chosen will have the property that belief updates can be performed, either approximately or exactly, in closed form with a number of parameters which is a polynomial function of the state space dimension. For example, one of the two representations we will examine in more detail using a Gaussian model to describe the observation and dynamics model. To specify a Gaussian model in a D-dimensional state space requires $D + D^2$ parameters: D parameters to represent the mean vector and D^2 parameters to represent the covariance matrix.[4] If the original belief state is a Gaussian $\mathcal{N}(s|\mu_b, \Sigma_b)$, and the observation and dynamics model are Gaussian models, $\mathcal{N}(z|s, \Sigma_z)$ and $\mathcal{N}(s'|s + \mu_a, \Sigma_a)$ respectively, then the new belief after taking a particular action and

[3] There are variable resolution approaches to discretize continuous POMDPs, but to our knowledge none of these are trivial to apply directly without either significant domain knowledge or additional computational investment.

[4] The covariance matrix can be expressed slightly more compactly by storing only the diagonal entries and half the other terms, since the off-diagonal terms are symmetric, but the number of terms is still $O(D^2)$.

receiving a specific observation is also a Gaussian:

$$\begin{aligned} b^{a,z}(s') &= \frac{p(z|s',a)\int_s p(s'|s,a)b(s)ds}{\int_{s'} p(z|s',a)\left(\int_s p(s'|s,a)b(s)ds\right)ds'} \\ &= \frac{\mathcal{N}(z|s',\Sigma_z)\int_s \mathcal{N}(s'|s+\mu_a,\Sigma_a)\mathcal{N}(s|\mu_b,\Sigma_b)ds}{\int_{s'} \mathcal{N}(z|s',\Sigma_z)\left(\int_s \mathcal{N}(s'|s+\mu_a,\Sigma_a)\mathcal{N}(s|\mu_b,\Sigma_b)ds\right)ds'} \\ &= \frac{\mathcal{N}(z|s',\Sigma_z)\mathcal{N}(s'|\mu_a+\mu_b,\Sigma_a+\Sigma_b)}{\int_{s'} \mathcal{N}(z|s',\Sigma_z)\mathcal{N}(s'|\mu_a+\mu_b,\Sigma_a+\Sigma_b)ds'} \\ &= \mathcal{N}(s'|c,C) \end{aligned}$$

where $C = ((\Sigma_z)^{-1} + (\Sigma_a + \Sigma_b)^{-1})^{-1}$ and $c = C\,(\mu_z\Sigma_z^{-1} + (\mu_a + \mu_b)(\Sigma_a + \Sigma_b)^{-1}))$. Therefore the belief update stays in closed form. More generally, linear dynamical systems observed with Gaussian noise are well known to involve a number of parameters which is a polynomial function of the state dimension, and the Kalman filter is a standard procedure for performing belief updating in such problems (Kalman, 1960). This filter also has a computational complexity which is polynomial in the number of model parameters, and therefore is a polynomial function of the state dimension.

The representation of the model dynamics as a weighted sum of Gaussians employed in Chapter 3 also keeps the updated belief in the same parameter family. However it has the unfortunate property that the number of Gaussians used to represent the belief state increases with each belief update. While the number of parameters is still only a polynomial function of the state dimension, the number of components can become an exponential function of the number of time steps past. We will discuss this issue further in Section 5.6.

Indeed, even if the number of parameters required to represent the belief state stays constant after a belief update, the number required to represent the value function typically increases as the value function is computed using the back up operator. Recall that a value function back up involves integrating over all possible observations, and this integration will in general cause an increase in the number of parameters needed to represent the value function. This is clear when we consider how to compute the best $(t+1)$-step α-function that starts with action a for a particular belief b:

$$\alpha_a^{t+1} = R(s,a) + \gamma \int_z \int_s p(s'|s,a)p(z|s',a) \operatorname*{argmax}_{\alpha_i^t} \left(\int_{s'} \alpha_i^t(s')b(s')ds' \right) dsdz.$$

For the selected action a and each possible observation z, we must choose the subsequent t-step conditional policy which will lead to the highest expected value for the belief state b. Typically the t-step policy chosen (and its associated α-function) will be different for different observations, and so the new α_a^{t+1} will be a weighted sum of a set of functions. Therefore, if the value function is computed explicitly then the number of parameters needed to represent it will grow at least by a factor of the number of observations summed over during the backup operation.[5] If the observations are continuous-valued, then the number

[5] As pointed out by Hoey and Poupart (2005), the observation space can be partitioned into the set of all observations that share the next best $(t-1)$-step policy. However unless all observations share the same $(t-1)$-step policy, the number of components will increase by the number of distinct sets of observations

of parameters will grow by the number of observations sampled to approximate the integral over observations in the backup. With multiple backups the number of components required to represent the value function will grow very quickly.

Instead, we will avoid representing the value function explicitly and only compute the state-action values for a particular belief using a forward search. If efficient operations can be developed for all forward search operations, then this approach is computationally more tractable for large domains.

We have already discussed that we will choose to represent our world models such that the number of parameters to represent the dynamics and observation model is a polynomial function of the number of dimensions, and such that the belief update can be performed in closed form. This fulfills the requirement of enabling a fast belief update operation during forward search. The second operator, observation sampling, can be constrained to be computationally practical by limiting the number of observations sampled to a polynomial function of the state dimension.

Finally, if the reward model is of a complementary form to the dynamics and observation models, then the expected reward (the third operator) computation can be performed analytically. For example, if the belief state is represented using an exponential family distribution, then if the reward is specified as a weighted sum of functions which are the conjugate prior to the belief distribution, then the expectation can be computed analytically. This operation is similar to computing the posterior distribution in an exponential family, and has a computational complexity that is a direct function of the number of parameters needed to represent the distributions. Since the representations employed have only a polynomial in the state dimension number of parameters, the reward expectation operation will have a computational complexity that scales polynomially with the number of state dimensions.

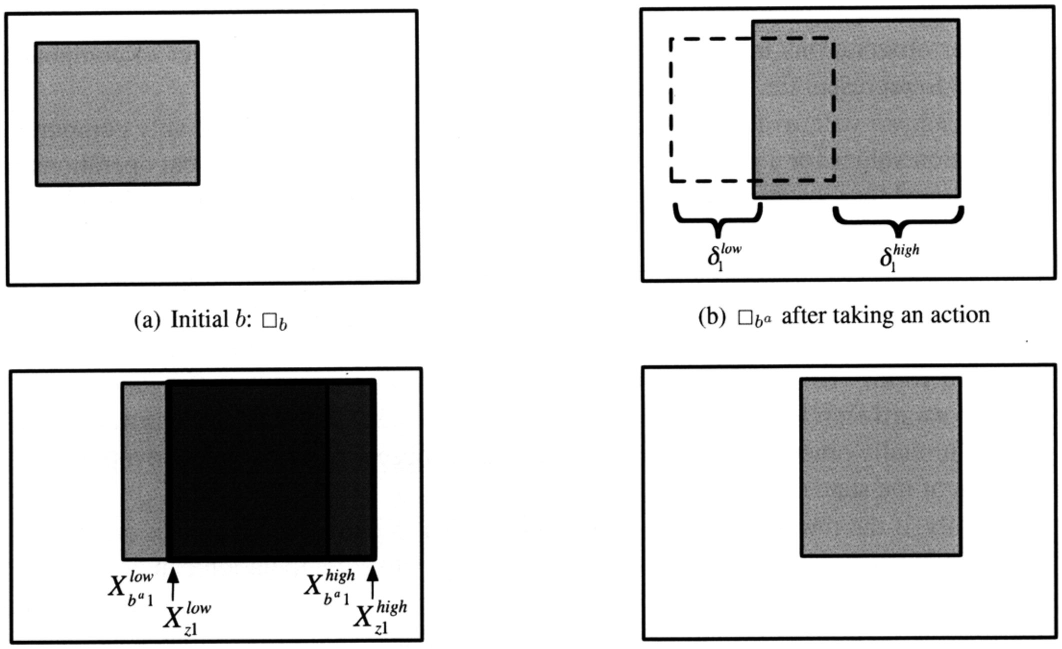
We now proceed to discuss two particular representations that enable these three operations to be approximately performed with a computational complexity that scales polynomially with the number of dimensions.

5.2.1 Hyper-rectangle POMDPs

The first of our two parametric representations are hyper-rectangular functions: hyper-rectangles are essentially the multi-dimension generalization of uniform distributions. Though similar representations have been used to solve continuous-state MDPs (e.g. Feng (2004)) to our knowledge this is the first extension to partially-observable environments. The key attribute of this representation that allows all the forward search operators to have a computational complexity that depends polynomially on the state dimension is that the state variables are assumed to be factored, which allows a separate parametric distribution to be placed on each dimension.

Formally, consider a D-dimensional continuous-valued state space. Let $\Box = \otimes_{i=1}^{D}[X_i^{low}, X_i^{high}]$ denote a D-dimensional hyper-rectangle that spans X_1^{low} to X_1^{high} in the first dimension, X_2^{low} to X_2^{high} in the second dimension, $\ldots$ and X_D^{low} to X_D^{high} in the D-th dimension. To represent the dynamics of an action a that uniformly shifts each state

sharing the same next best policy.



(a) Initial b: $\Box_b$

(b) $\Box_{b^a}$ after taking an action

(c) Observation intersection

(d) New $\Box_{b^{az}}$

Figure 5-4: Hyper-rectangle belief update. The dotted line in (b) denotes the initial belief, the dark rectangle in (c) denotes the possible states given the received observation, and the light gray rectangle denotes the belief as it is transformed during the update process.

dimension i over an amount between δ_{ai}^{low} and δ_{ai}^{high} we write $\Box_a = \otimes_{i=1}^{D}[\delta_{ai}^{low}, \delta_{ai}^{high}]$. The representation can also support absolute transitions but without loss of generality we focus on relative dynamics. The observation model is similarly specified as hyper-rectangles of uniform probability over particular areas $\Box_z$. The reward model is specified by a set of (potentially overlapping) hyper-rectangle $(\Box_r, r)$ tuples where $\Box_r$ specifies the region over which the reward applies, and r specifies the associated value of the reward for that region. The belief over the state space will also be specified as a hyper-rectangle $\Box_b$. We will shortly see that this choice of world models allows the number of parameters for the belief state to stay bounded when approximate updates are performed: using a number of hyper-rectangles to represent the reward model increases the expressive power of the reward model without increasing the number of belief components within the tree.

$ComputeExpectedReward(\Box_b, a)$ The expected reward is calculated by first computing the area of the intersection between the belief $\Box_b$ and each of the hyper-rectangles constituting the reward for a particular action a (with associated reward r_{ai}). Each intersection area is normalized by the area of the belief state, yielding the fraction of the belief overlapping with each particular reward hyper-rectangle. The total reward is a weighted sum over all N_{ra} reward rectangles associated with a particular action:

$$R(b,a) \quad = \quad \int_s b(s)r(s,a)ds = \sum_{i=1}^{N_{ra}} r_{ai}\frac{Area(\Box_b \cap \Box_{r_{ai}})}{Area(\Box_b)}.$$

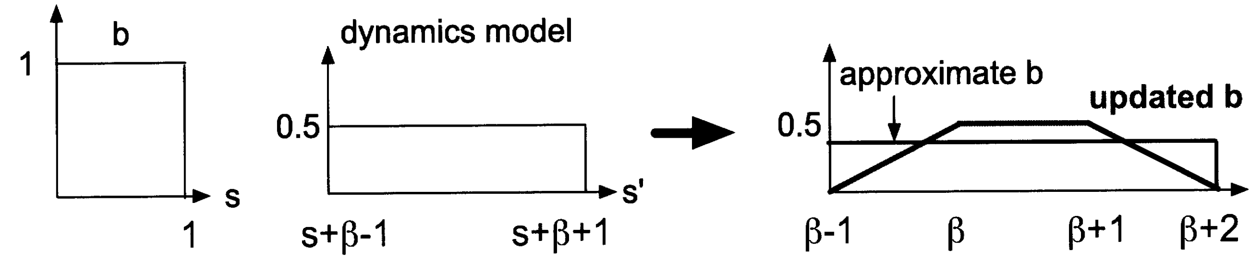


Figure 5-5: Illustration of updating a belief after an action is taken using a hyper-rectangle representation. The figure on the right shows the true updated belief: due to the convolution it becomes piecewise linear. The approximated belief stays piecewise constant.

This computation requires computing rectangular intersections, which can be done in time linear in the number of dimensions $O(D)$.

For domains with discrete observations, we can either expand all observations or sample observations according to their probability given the current belief node. However, in domains with continuous observation spaces, observations will often consist of a single hyper-rectangle $\Box_z$ centered at the true state. [6] In this case we proceed by computing the intersection of $\Box_z$ with the current belief state to compute the overlap region $\Box_{overlap} = \Box_b \cap \Box_z$. If the overlap is non-empty, then n_z observations are sampled from within this overlap, where n_z is a parameter for the particular problem. In addition, if the belief state only overlaps partially with the observation region, there is also a "null" observation that corresponds to no observation being received: this assigns positive probability to the event that the true state lies in the belief state region outside of the visible observation area. If there is no overlap, then only a null observation is used. Note that if there are only a discrete set of observations, then this situation will not occur: one observation will always be received. The computational cost of performing observation sampling is dominated by hyper-rectangle intersection, which can be done in $O(D)$.

Finally, we can efficiently perform approximate belief updates. An exact belief update given a particular action a, $b^a(s) = \int_s p(s'|s,a)b(s)ds$, requires convolving two uniform distributions. However, this will result in a new belief that is piece-wise linear instead of piece-wise constant, and will create at least 2 separate piece-wise linear components per dimension. See Figure 5-5 for an illustration. In general, each time a new belief update is performed, the order of the representation will increase (constant to linear to quadratic ...), and the number of components will increase exponentially with the number of backups. To maintain computational efficiency, we keep the belief represented as a single uniform distribution after the update (see Figure 5-5). Specifically:

$$\Box_{b^a} = \otimes_{i=1}^{D}[X_{bi}^{low} + \delta_{ai}^{low}, X_{bi}^{high} + \delta_{ai}^{high}].$$

This procedure of approximating the updated belief as a single rectangle essentially keeps the distribution in terms of set-based uncertainty: the true state has a uniform probability

[6] Note that the location of $\Box_z$ could change at different points in the tree.

of lying anywhere within a single rectangle, and zero probability of being outside this area. This idea draws from by prior work by Li and Littman (2005) on continuous-state MDP planning using piecewise linear functions: here the authors repeatedly projected the value function representation to a piecewise linear representation to prevent the functional form of the value function from increasing to piecewise linear, quadratic, etc. during planning. Li and Littman also provided principled approaches for computing the number of piecewise linear components used in order to bound the error in the resulting value function.

Next, incorporating information from an observation $\Box_z$ involves a simple intersection between the belief $\Box_{b^a}$ and the observation $\Box_z$:

$$\begin{aligned}\Box_{b^{az}} &= \Box_{b^a} \cap \Box_z \\ &= \otimes_{i=1}^{D}[\max(X_{b^a i}^{low}, X_{zi}^{low}), \min(X_{b^a i}^{high}, X_{zi}^{high})].\end{aligned}$$

A diagram of the belief update process is in Figure 5-4. Since each step of the belief update can be done over each dimension independently, the computational complexity of $UpdateBelief$ is also a linear function of the number of dimensions, $O(D)$.

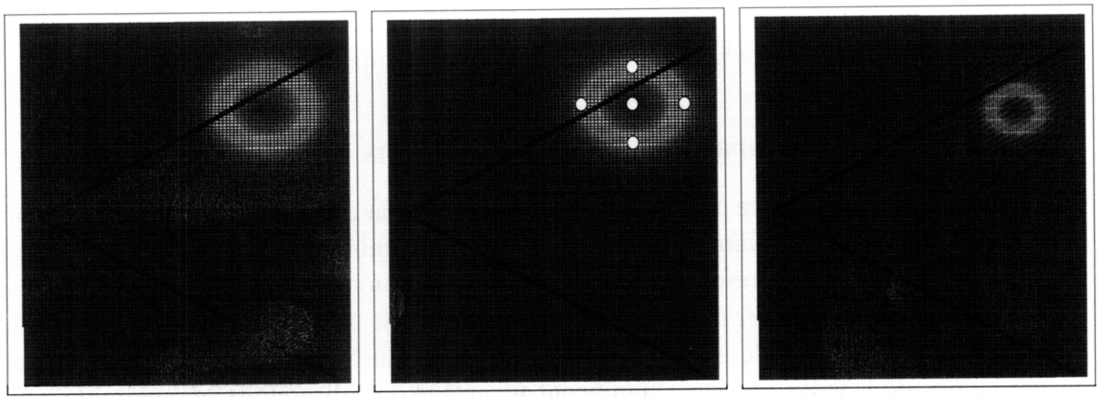
In addition to its attractive computational benefits, this representation is likely to be well suited to domains which have sharp cutoffs in their transition, dynamics, or reward models. For example, domains involving meeting times or scheduling, where successive tasks may take a range of times that accumulate on top of previous times, may be easily represented in this framework.

5.2.2 Gaussian POMDPs

In some domains, such as navigational robotics and controls, Gaussians or truncated Gaussians provide a much better approximation of the dynamics and sensor models than uniform distributions. Therefore our second parameterization focuses on Gaussian representations. For example, the range measurements from a laser sensor are usually distributed as a Gaussian around the true range for objects within the sensor's field of view and maximum and minimum range measurements. Though linear quadratic Gaussian (LQG) controllers are known to be optimal when the dynamics and observation models are linear Gaussian functions and the reward model is quadratic (Burl, 1998), the LQG solution technique does not generalize to other rewards or sensor models. However, Gaussian models allow us to efficiently implement the operators needed for forward search.

If the reward model is a weighted sum of Gaussians, then $ComputeExpectedReward$ can be computed analytically. Let $r(s,a)$ consist of a weighted sum of I Gaussians: $r(s,a) = \sum_i^I w_i \mathcal{N}(s;\mu_i,\Sigma_i)$ where $\mathcal{N}(s;\mu_i,\Sigma_i)$ is the probability density assigned to a state s by a Gaussian distribution with parameters μ_i and Σ_i. Let $b = \mathcal{N}(s;\mu_b,\Sigma_b)$. Then

$$\begin{aligned}R(b,a) &= \int_s \sum_i^I w_i \mathcal{N}(s;\mu_i,\Sigma_i)\mathcal{N}(s;\mu_b,\Sigma_b)ds \\ &= \sum_i^I w_i \mathcal{N}(\mu_i;\mu_b,\Sigma_b+\Sigma_i).\end{aligned}$$



(a) Belief updated after action, b^a (b) Sigma Points (c) Re-fit Expected Observation

Figure 5-6: An example of how to incorporate a truncated Gaussian observation model. (a) The rainbow covariance shows the belief state probability after incorporating the action update (red is highest probability). The black lines show the range in which an observation is possible (the smaller angle between the two black lines). Subfigure (b) shows the sigma points computed from the b^a distribution. Points outside the sensor range are then discarded, and only the remaining ones are used, in conjunction with the Gaussian observation model within the sensor range, to generate a new expected observation mean and covariance, illustrated in (c).

The computational complexity $O(ID^3)$ is dominated by inverting the summed covariance matrices.

To implement $SampleObservations$ for Gaussian beliefs with non-linear truncated observation models, we first analytically compute $2D + 1$ "sigma points" (see Julier and Uhlmann (1997)) given the current belief's mean and covariance. These are then propagated through the nonlinear observation model to compute the expected observation that would be seen at each point. Points that lie outside the visible range of the sensor are removed, and the remaining points are refit to a Gaussian. Potential observations are sampled from this Gaussian: in experimental results we typically simply select the expected mean observation. The computational complexity of this operator is dominated by the Cholesky decomposition ($O(D^3)$) used to compute the sigma points: since this is done for $2D$ points the compuational complexity of $SampleObservations$ is $O(D^4)$.

We modified the unscented filter (Julier & Uhlmann, 1997), which extends Kalman filtering to non-linear domains, to perform $UpdateBelief$. Briefly, in an unscented filter a set of $2D + 1$ sigma points are computed as in $SampleObservations$. These points are propagated through the dynamics model and observation model, and used to recompute a new mean and covariance. This represents the new b^{az}. Either step can be replaced by a Kalman filter update if the dynamics or observation model are linear Gaussian. In domains where the observation model is truncated we remove the sigma points that fall outside the sensor visibility range, reweight the remaining points to renormalize the sum of weights to 1, and then re-fit an expected mean and covariance using the points. This process is illustrated in a two-dimensional example in Figure 5-6. This is a fast heuristic that performed well empirically. In general the overall $UpdateBelief$ complexity is $O(D^4)$ though for factored, untruncated, linear domains the complexity reduces to $O(D)$.

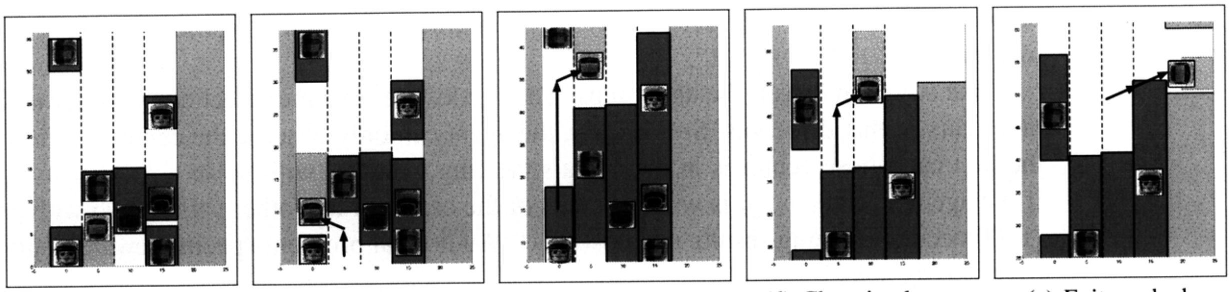
5.2.3 Alternate Parametric Representations

There are several alternate parametric representations that could be employed which would yield fast, tractable forward search operations. As demonstrated by the hyper-rectangle model, any factored representation will result in operations that depend polynomially on the state dimension, rather than exponentially, since each state variable can be treated individually. In order to scale well to large environments (such as continuous domains), parametric functions will still offer a significant advantage over discrete-state representation, as we will demonstrate in the experimental section. In order to perform the belief update and expected reward calculations in closed form, parametric models which have known conjugate distributions should be vused to represent the world dynamics, observation and reward models across each dimension. Some such distributions include the exponential, gamma and poisson distribution: more generally the exponential family distributions can be used, with one model for each separate dimension.

In some domains there will be some correlation between the state variables, such as when a ship navigates by turning the rudder and accelerating the motor: both the x and y location of the ship will change. In such cases one cannot assume that the state space can be factored into D independent variables. One model which does not assume that the state space is factored is a Gaussian model: the covariance matrix models any covariance between different state dimensions. For more complicated environments, a switching state model (SSMs) consisting of a finite number of a Gaussian models can be employed, such as the models used in the prior chapters. As noted before, SSMs have been used to successfully to perform state estimation of a variety of complex domains, include honeybee dancing (Oh et al., 2005) and the IBOVESPA stock index (Fox et al., 2009). Therefore it is seems likely that there exist compact parametric representations that can model a wide range of problems and still enable forward search operations that have a computational complexity that scales linearly with the number of state dimensions. SSMs can cause the number of state components to increase, and we will discuss this and some other potential limitations of our models in Section 5.6.

5.3 Planning

The prior sections described how two particular parametric representations could lead to fast methods for the three POMDP forward search operators. We can use either of these representations as the basis of the generic online forward search POMDP planner outlined at the start of this chapter. One significant limitation of forward search techniques is that the size of the tree (the number of nodes) grows exponentially with the horizon: for a horizon of T steps (a sequence of T action-observation updates) there will be $(|A|N_z)^T$ leaf nodes where N_z is the number of observations sampled per action. Therefore a full search can typically be done only for short horizons. To help ameliorate this problem, we use two simple modifications to the standard forward search procedure. First, we use the



(a) Start (b) Moving over (c) Passing cars (d) Changing lanes (e) Exit reached

Figure 5-7: Boston driving simulation. In (a) the agent is the car at the bottom of the second lane to the left. The agent's viewing region is shown in light gray. The belief state over the possible extent of each other car is shown by a dark gray rectangle. The goal is to reach the right hand exit. A video of the simulation can be viewed at http://people.csail.mit.edu/emma/car.avi

popular addition of a static evaluation function at the leaf nodes to provide an estimate of the remaining expected value of that belief state. Second, we truncate branches whose expected reward has fallen below the minimal value known to be achievable in this domain. In general this will be helpful when the minimal value is high enough to help prune some branches: for example, when there may be catastrophic actions, such as jumping off a cliff, as well as safe "no-op" actions, then this may help prevent exploring branches after the catastrophic action is taken. Both the static evaluation function and the branch truncation conditions were specified by hand for each experimental domain.

5.4 Experiments

We now consider the performance of our algorithm on two large POMDPs.

5.4.1 Boston Driving Simulation

Consider as before an agent driving in traffic: the agent needs to reach a specific location, which renders local collision avoidance insufficient, and requires the agent to solve a global planning problem (see Figure 5-7). The location of each car on the road must be represented with additional dimensions, and realistic models of sensing limitations (e.g., occlusions, limited field of view) lead to considerable state uncertainty. In this experiment, the world consists of a 20x60 road area. There are 7 other cars, each of which has its center located at (x_i, y_i), where i ranges from 1 to 7. The agent's fully observable dimensions are represented by (x_t, y_t, f_t) where f_t specifies one of 8 possible viewing angles: looking directly in front, directly in back, or three possible each side. The agent can choose from 7 actions which cause the agent to move forward (quickly or slowly), to the left or right, stop, or change its gaze: $[x_t + 0, y_t + 2, f_t]$, $[x_t + 0, y_t + 5, f_t]$, $[x_t - 5, y_t + 2, f_t = 0]$,$[x_t + 5, y_t + 2, f_t = 0]$,$[x_t + 0, y_t + 0, f_t]$, $[x_t + 0, y_t + 2, f_t + 1]$, or $[x_t + 0, y_t + 2, f_t - 1]$, respectively. At each time step the other agents move forward 0 to 2 units: $[x_i + 0, y_i + [0, 2]]$.

Like a human driver, the agent must actively look in specific directions in order to see other cars in that direction. Specifically, the agent can only observe the center location of the first car whose center is in the agent's viewing window (with noise $+/-0.1$ in x/y. If the agent's center x_t, y_t is within 4 units of the center of any other car, a crash with cost -1000 occurs. Turning results in a cost of -1. Additional problem-specific reward is given for reaching a goal area.

We use a hyper-rectangle depth-5 forward-search planner. The belief state over the location of each other car is maintained independently. The static evaluation function used at the leaves is a function of the distance to the goal and the minimum value used to prune branches is 0.

We present results on 5, 8 and 20 step problems. A naive representation of the state space would be to discretize the world into 1 by 1 units, yielding 1200 grid cells. To represent the joint, non-factored state space over all 7 agents would yield a state space of over 10^{21}, which is unrepresentable by existing non-forward-search planners. We instead constructed new similar possible planners that might be reasonable alternatives: a discrete-belief partially-observable (DBPO) forward-search planner that uses different granularities of discretization, and a fully observable forward-search planner that ignores the uncertainty of the belief and plans using the mean of the belief state (MBS). Results are presented in Table 5.1 (in each case, the best performer is in bold). In all problems our LOP-POMDP planner achieved the highest reward we know to be possible; however, we cannot guarantee that this is optimal due to the approximations performed.

In the 5-step problem, if too coarse of a discretization was used, the DBPO planner could not find a good policy since its overly coarse representation made certain actions appear to result in collisions when they were in fact safe. When a finer discretization was used, the DBPO planner could find a better policy but was much slower. LOP-POMDP was faster than both approaches and found a better policy, illustrating the computational advantage of our approach over even factored discrete representations. This advantage will increase in problems with dimensions with a large range of values. Our LOP-POMDP also avoids the challenge of selecting the right discretization granularity.

In the 8-step problem we also compared our algorithm to the MBS fully observable planner for two problems. In *8 Step Problem A*, fully observable forward search found a decent policy due to the original car locations; however, it could not do as well as LOP-POMDP. In *8 Step Problem B* the initial car locations resulted in a crash if the agent was not careful, and here MSB performed poorly. This performance could be improved by sampling more states for each action branch; however, sampling more states incurred a very large computational cost and still performed worse than LOP-POMDP. LOP-POMDP was faster than both MSB approaches here, we suspect due to its ability to prune further exploring branches where a crash event occurs. In general, fully observable search requires estimating the expected reward by sampling over next states, in contrast to LOP-POMDP which involves sampling over next observations: in continuous-valued state spaces with only a limited number of observations, this can give a significant computational advantage to partially observable planners.

Finally, we also tried a 20 step problem (shown in Figure 5-7) where the agent's goal

	Avg. Total Reward	Avg. Total Planning & Execution Time (s)
5 Step Problem		
2x2 DBPO	0	9.2
1x1 DBPO	349.5	13.7
LOP-POMDP	**699**	**7.4**
8 Step Problem A		
MBS1	198	**100.2**
LOP-POMDP	**398**	162.9
8 Step Problem B		
MBS1	-822	97.7
MBS3	-232	6527.4
LOP-POMDP	**0**	**13.7**
20 Step Problem		
LOP-POMDP	95	507.7

Table 5.1: Results for Boston driving domain. DBPO= discrete belief partially observable. MBS1 = mean belief state fully observable forward search, 1 indicates how many states were sampled for each action. MBS3 is the same as MLS1 except 3 states are sampled per action. All results are averaged across 10 episodes. In 8-Step Problem A the car on the immediate left of the agent starts at the highest edge of the belief rectangle, and in B it starts at the lowest. The goal reward is 100 except in the 5-step problem where it is 350.

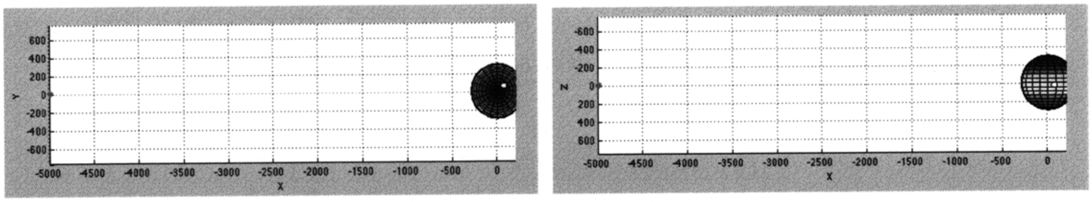
was to make a right turn: though only a 5-depth forward search was employed, the agent consistently reached the goal. At the end the agent did not check its blind spot, reasoning the other car was unlikely to be there, inspiring us to call our planner a "Boston driver."

Note that in contrast to past highway driving domains (e.g. Forbes et al. (1995), McCallum (1995), and Abbeel and Ng (2004)) which provide reward based only on local collision avoidance, to perform well in our new third domain requires balancing local collision avoidance and global goals (reaching the destination). LOP-POMDP can perform well in this new more complicated simulated driving task.

5.4.2 Unmanned Aerial Vehicle Avoidance

In our second experiment, we consider the challenge of having unmanned aerial vehicles (UAVs) autonomously reach goals and avoid other aircraft that may not be explicitly watching for UAVs. Here we employ our Gaussian POMDP representation.

In our simulation there is an agent and an opponent. The agent's state is fully known and represented by $\langle x_a, y_a, z_a, \theta_a, p_a, v_{xa}, v_{ya}, v_{za}\rangle$ where θ_a represents the yaw of the agent (in the xy-plane), p_a represents the pitch angle of the agent, x,y,z represent the agent's location and v represents the agent's velocity. The opponent's state is hidden and is represented as a position relative to the agent (x_o, y_o, z_o) and a velocity relative to the agent (v_{xo}, v_{yo}, v_{zo}). The agent can take 6 acceleration actions to increase or decrease its forward, yaw or heading (pitch) velocity. It can also change nothing. The opponent is assumed to move along the



(a) x-y

(b) x-z axis

Figure 5-8: Unmanned aerial vehicle domain. The agent is represented by a red dot. The belief state over the other opponent's location is shown as an ellipsoid over 3 standard deviations. The white circle represents the true opponent's location. For clarity (a) shows the x-y viewpoint and (b) shows the x-z viewpoint. The velocity estimates are also estimated but are omitted here for clarity.

negative x-axis with a large amount of Gaussian noise.

The agent has a radar sensor with a range of 5 nautical miles, and a view range of +/– 15 degrees in elevation, and +/– 110 degrees in azimuth. The cost of turning is -1. If the agent is within 500 ft in the x or y direction and within 100 ft in the z direction of the opponent, a cost of -20000 is received for crashing. For the purposes of planning, this reward model is approximated by a weighted sum of 25 Gaussians. Additionally, the agent receives a cost of $(\delta_z/50)^2 + (\delta_y/50)^2$ at each time step where δ_y is the amount the agent has deviated from its original y coordinate, and δ_z is the amount the agent has deviated from its original z coordinate. Example beliefs are shown in Figure 5-8.

To focus attention on hard scenarios, the agent's starting location was set at (x, y, z)=$(-2000, 0, 0)$ ft. The opponent's initial location was drawn from a Gaussian about (0,0,0). Forward search was run for 5 steps, and the problem horizon length was 8. Results averaged over 25 episodes are reported below:

Avg. Total Reward	Avg. Time (s)
-9733	228.8

The algorithm performed well, avoiding collisions in all episodes. The negative reward is due to the swerving action taken to avoid the opponent.

5.5 Analysis

We now provide an analysis of under what circumstances the LOP-POMDP algorithm provides formal guarantees on the quality of the produced policy, as well as an analysis of the computational complexity of LOP-POMDP. We will consider two cases: when the belief update operator and the expected reward operator can be computed exactly and when one or both of these operators is performed approximately.

5.5.1 Exact Operators

We first consider the performance of LOP-POMDP when the belief update and expected reward operations are performed exactly. There are two separate cases to consider: when there is a small number of observations and when the observation space is large or continuous-valued.

If there are a very small number of observations then all observations can be expanded at each action node. Assuming there are H levels in the tree (where a level corresponds to branching on the actions and then branching on observations) then the state-action values at the root belief will be an exact computation of the H-step expected return:

$$Q(b,a) = E[\sum_{i=0}^{H-1} \gamma^i R_i]$$

where the expectation is taken over all possible observations that could be experienced in the H-step trajectory. This value can be at most $\gamma^H R_{max}/(1-\gamma)$ from the optimal value, due to the known bounds on the difference between the H-step and infinite-horizon values (see e.g. Putterman (1994)). If there is an admissible estimate of the future value placed at the leaves then this estimate can be further improved.

If instead the observation space is large or continuous-valued then it will be intractable to expand all possible observations for each action node. Instead we can sample observations. Prior work by Kearns, Mansour and Ng (2002) proved that near-optimal state-action values could be computed for a particular state in a MDP by building a forward search tree where the number of next-states sampled at each action node was independent of the size of the state space.[7] These results also apply to POMDPs with large observation spaces: as discussed by McAllester and Singh (1999), for a given ϵ, if the number of observations sampled at each action node is a polynomial function of ϵ, and the forward-search horizon H is sufficient large, then the state-action estimates at the root belief will be ϵ-optimal. To be more precise, we repeat here the theorem from Kearns, Mansour and Ng:

Theorem 5.5.1. *If*

$$\begin{aligned} H &= \lceil \frac{1}{1-\gamma} \ln \frac{4R_{max}}{\epsilon(1-\gamma)^3} \rceil \\ C &= \frac{4R^2_{max}}{\epsilon^2(1-\gamma)^6}(2H \log \frac{4|A|HR^2_{max}}{1}\epsilon^2(1-\gamma)^4 + \log \frac{4R_{max}}{\epsilon(1-\gamma)}) \end{aligned}$$

then $|V^{fs}(b) - V^*(b)| \leq \epsilon$ *where* V^{fs} *is the value computed by forward search,* V^* *is the optimal value and* C *is the number of observations sampled per action node.*

The intuition behind this result is that only a bounded number of values is required until the sample mean is close to the true expected value with high probability: for example, Hoeffding's inequality guarantees this for all bounded distributions. Therefore if LOP-

[7] Their result holds as long as rewards are bounded, which we assume throughout this document.

POMDP fulfills these requirements, its values will also be ϵ-close to the optimal state-action values.

There are a variety of parametric functions which will support exact forward search operations, including Gaussian models and weighted sums of Gaussian models.

The computational complexity of LOP-POMDP will depend on the particular parametric representation used to represent the world models, and the observation branching factor. However, LOP-POMDP assumes that each of the forward search operators requires a computational cost that scales polynomially with the state dimension D. This cost will be incurred at each node, and for a H horizon tree with an observation branching factor of C, the total computational complexity of building the tree and computing the belief-action values at the root will be

$$O((|A|poly(\frac{1}{\epsilon}))^H poly(D)).$$

In summary, when the belief updates and expected reward computations can be done precisely, it is possible to obtain ϵ-close estimates of the state-action values used to choose the action for the current belief by building the tree using approaches from the research literature. However, this accuracy parameter directly relates to the observation branching factor, and highly accurate models will typically require an intractably large number of observations to be sampled for each action node.

5.5.2 Approximate operators

In Section 5.2.1 we presented an approximate belief backup operator for working with beliefs represented as hyper-rectangles. An update of hyper-rectangle beliefs involves a convolution operator and would cause the belief to change from piecewise linear, to piecewise quadratic, and on to higher order polynomials, while simultaneously resulting in an exponentially increasing number of components to exactly represent the belief state. Instead the belief state is approximated as a single uniform distribution over the region where the belief state has non-zero probability.

This approximation, and many other belief state approximations, can cause the belief state to be arbitrarily far from the original exact belief state. It is possible to specify reward distributions that will result in the expected value using the approximate belief state to be particularly poor approximations of the expected reward under the true value, that is $|\int_s R(s,a)b(s)ds - \int_s R(s,a)\tilde{b}(s)ds|$ can be very large, where $\tilde{b}$ is the approximate belief representation.

Initially it seems that there may be some hope in certain domains since even if the error between a belief b and its approximate representation $\tilde{b}$, $|b(s) - \tilde{b}(s)|_\infty$, is large, the error in the estimated rewards $|\int_s R(s,a)b(s)ds - \int_s R(s,a)\tilde{b}(s)ds|$ may be small. This will occur whenever the reward model is constant over the region spanned by the belief state. Unfortunately, even if the expected reward of the beliefs, $R(b,a)$, are correct at each node, the observation probabilities, $p(z|b,a)$ may be altered since the belief state is an approximation. More formally, for arbitrary b and $\tilde{b}$, in general

$$p(z|b,a) \neq p(z|\tilde{b},a).$$

This means that as the expected immediate rewards are summed over observations, these sums will generally differ from the sum over nodes with exact belief updates. Therefore despite good empirical performance, at the present time we do not have any theoretical guarantees that can ensure that the resulting approximate value function is close to optimal.

The computational complexity of selecting an action at the root belief using approximate representations is similar to the complexity of using exact operators, and is again a function of the size of the tree and the cost of performing the node operations. As analyzed previously, the cost of performing the hyper-rectangle node operations is linear in the number of state dimensions, $O(D)$. Therefore the total computational complexity of planning is $O((|A|N_Z)^H D)$ where N_Z is the number of observations expanded at each action.

5.6 Limitations & Discussion

There are a number of limitations with the LOP-POMDP approach as described so far. Regardless of whether the state space is factored or not, one challenge when using parametric representations is incorporating negative information. For example, consider the unmanned autonomous vehicle simulation and assume the agent's current belief over the location of the other aircraft is a single Gaussian with a wide variance, centered in front of the agent. Now consider what happens if the agent moves forward and no longer receives an observation about the other aircraft's location. Since the agent can sense a conic section in front of it, this lack of observation indicates with high probability the other aircraft is nowhere inside that particular cone. However, this negative information essentially splits the prior Gaussian belief distribution over the other aircraft's location into two separate disconnected distributions, neither of which has a Gaussian shape. Updating the belief distribution to include this non-observation results in a new belief which has double the number of components as before (going from unimodal to bimodal) and each of those components are no longer in the same (in this case, Gaussian) parametric family. A similar issue can occur in the hyper-rectangle example: a driver who checks her right hand mirror and doesn't view a car where she thought there might be, is likely to have a new belief distribution which is either non-hyper-rectangular (it may be a rectangle with a small rectangular piece missing from it) and/or becomes multiple rectangles. To generalize, the problem with incorporating non-observations (or negative information) is that it may either increase the number of parameters required to represent the belief state, and/or cause the updated family to no longer belong to the same parametric family. In the experiments performed we either ignore such negative information (in the case of the hyper-rectangular representation) or can refit a new distribution within the same parametric family to the updated belief (in the unmanned vehicles example we refit a single Gaussian after the observation truncation is performed). While we could allow negative information to cause the belief state to split into multiple components, it is always a significant limitation, at least to the theoretical algorithm properties, whenever the belief is no longer in the same parametric family after an update is performed.

Splitting the belief into several components can arise when the lack of a sensor reading is incorporated, but it can also occur if the dynamics model is multi-modal. For example, if the dynamics model is a combination of M linear Gaussian models, then the number of components required to represent the belief state increases by a factor of M upon each belief update. In general this means the number of of components required to represent the belief state will be $O(M^T)$ for the T-th step belief state, or $O(M^H)$ for the belief state at the H-level horizon of the forward search tree. The challenge here is that even if one can constrain the expansion of the forward search tree to a fixed small horizon, the number of components in the belief will grow with each action taken in the real environment, and so the initial belief state at the forward search tree root will have a number of components that is steadily increasing. Though the number of components is still only a polynomial function of the state dimension, the factor in front of this polynomial function will become an exponential function of the number of time steps and forward search horizon. This has the potential to cause significant computational difficulties if the agent must act for a large number of steps, or if the forward search horizon is long. Some hope may come if the domain observation model is such that some observations collapse the belief down to only a small number of components: such situations may occur when an agent receives an observation of a landmark or other observation which is strongly associated with one world state and unlikely to be aliased with any other world states. These observations can effectively cause the belief state to return to a unimodal belief state, thereby reseting to a small number the parameters required to specify the distribution.

In the future it would be interesting to try to enumerate benchmark POMDP problems which have observations that reset the belief state to be unimodal, or keep the belief state fairly well localized during the planning process. For example, the coastal navigation algorithm of Roy and Thrun (2000) will plan paths that keep the belief state essentially unimodal where possible, through choosing actions that keep the belief state well localized.

An additional limitation of this work is it is will be intractable in domains which require a long horizon lookahead in order to select the correct action. This challenge may be ameliorated by selectively expanding actions and observation branches, instead of the current method of building a symmetric tree. Selective tree construction has been shown to make a significant difference in large discrete-state domains, as Ross et al. (2008b) recently discussed in their survey paper. Some of the most promising techniques make use of upper and lower bounds on the value function in order to selectively expand branches which contribute most to the uncertainty in the value estimates of the root belief node, and which have a high probability of being reached.[8]

In this chapter we are interested in planning in infinite (continuous), high-dimensional environments. Therefore we require bounds that can be specified over continuous-valued states. A trivial lower bound comes from finding the minimum value of the reward model and then scaling this to its infinite horizon value, if this minimum value were to be received at every step:

$$V_{min} = \min_{a,s} \frac{R(s,a)}{1-\gamma}.$$

[8] This second characteristic can be a little tricky since really one would like to expand branches that have a high probability of being reached under the optimal policy. As the optimal policy is unknown, a substitute must be used when evaluating the likelihood of reaching a particular node, which specifies the likelihood of taking those actions and receiving that trajectory of observations.

This is the minimum expected sum of future rewards that could be received by an agent. If the models are weighted sums of Gaussians, then the SM-POMDP described in Chapter 3 could be used to compute a lower bound on the value function by performing a few exact value function backups (maintaining all components in the resulting α-functions). This would provide a tighter value function but would be more computationally expensive.

A simple upper bound would be to consider the best possible reward and compute the associated value if this reward was received at every time step:

$$V_{max} = \max_{a,s} \frac{R(s,a)}{1-\gamma}.$$

A slightly tighter upper bound would be to take the above upper bound, and use it to perform of a step of MDP value iteration, since the MDP value is known to be an upper bound on the POMDP value function (Hauskrecht, 2000):

$$Q_{upper}(s,a) = R(s,a) + \gamma \int_{s'} p(s'|s,a) V_{max}(s') ds.$$

In general it will be non-trivial to repeatedly perform MDP value iteration, as standard value iteration involves maximizing over the action set, and this operation can cause the resulting value function to be no longer lie within the same parametric family class. However, at least a single MDP backup is computable in closed form.

Ross et al. (2008b) showed that even loose upper and lower bounds could yield impressive results when used as part of a smart node expansion heuristic in a forward search planner. It is possible that these weak bounds could be used to significant benefit as part of a selective forward search. To use some of the heuristics for node expansion presented by Ross et al. it is necessary to be able to update the upper and lower bounds for particular belief states within the tree to reflect information gained as the tree is updated and modified. This operation essentially involves taking the expectation over observations or maximizing over prior upper and lower bounds. One additional technical difficulty when the state and observation space is continuous-valued is that it will not be possible to expand all possible observations for an action node, as there are an infinite number of potential observations. This means that only a few observations will be sampled to approximate the expectation. In order to compute correct upper and lower bounds, it is necessary to include the weight of the non-sampled observations as part of the computation. For example, consider updating the lower bound for a belief state when only a set of $z_s \in Z$ have been sampled below the action node a. The lower bound $L_T(b,a)$ can be updated from its previous value as follows

$$L_T(b,a) = R(b,a) + \gamma \sum_{z_s} p(z_s|b,a) L_T(b^{a,z_s}) + \gamma(1 - \sum_{z_s} p(z_s|b,a)) L_T(b^a)$$

where we can use the updated lower bounds for the expanded observation nodes (corresponding to a set of beliefs b^{a,z_s}, one for each sampled observation z_s) and the prior lower bound for all the remaining, unsampled observations (which have a relative weight of $1 - \sum_{z_s} p(z_s|b,a)$). A similar procedure can be performed for the upper bound.

Therefore we believe that it should be possible to use some of the heuristic node expansion techniques presented by Ross et al. (2008b) to do selective tree construction, and therefore choose better actions, than with a standard forward search approach. This should also help our LOP-POMDP scale to longer horizon domains. Another interesting direction for helping scale LOP-POMDP to longer horizon problems would be to extend or embed it into a hierarchical method. Prior hierarchical POMDP solvers, such as by Pineau et al. (2003b), have shown some significant advantage in scaling up POMDPs to large domains with long time horizons.

Finally, we also believe that it would be helpful to compare to some existing alternative POMDP planners. Perhaps the most suitable alternative approach would be Symbolic Perseus (Poupart, 2005), a factored discrete-state offline planner that uses algebraic decision diagrams to represent the world models. For example, this approach could be used in the driving domain by representing the agent's location in the global space, and all the other cars in a space relative to the agent. Symbolic Perseus still requires a discrete-state representation of each variable, and has only been used on significantly smaller problems than our car simulation. Therefore we anticipate that LOP-POMDP will be significantly faster in the car simulation and other very high-dimensional domains. However this intuition can only be verified experimentally.

5.7 Summary

In this chapter we have presented two different representations that enable the three key operations of standard POMDP forward-search planning to be approximately computed with a computational complexity that scales polynomially with the number of state dimensions. This allowed our planner to scale to high-dimensional, continuous-state problems outside the reach of prior approaches. The representations presented severely restrict their expressive power in return for significant gains in computational tractability. We have also discussed how alternate parametric representations might be employed which would increase the expressive power of this approach, and what computational trade offs these representations might require.

[T]he original . . . problem that started my research is still outstanding - namely the problem of ... planning dynamically under uncertainty. If such a problem could be successfully solved it could ... contribute to the well-being and stability of the world.

George Dantzig

6 Conclusions

We commenced this document with an example of a driver navigating traffic in order to reach a final destination. This seemingly simple domain is large, high-dimensional, uncertain and often can only be sensed through local observations. As such this problem exhibits a number of characteristics that prior algorithms which make sequential decisions find challenging. The focus of this thesis has been on learning when the environment is fully observable, and planning when the environment is partially observable, in domains which are large, high-dimensional and uncertain.

Our criteria for evaluating potential solutions to these problems has been to consider an algorithm's expressive power, tractability and optimality guarantees on the produced policy. Expressive power loosely corresponds to the number of domains that can be well modeled by the algorithm's assumed representation. Tractability is a measure of how well the approach scales in large, high-dimensional environments, and refers to computational complexity for planning algorithms, and computational and sample complexity for learning algorithms. Finally, we are also concerned with what, if any, type of guarantee is provided on the resulting policy quality.

Prior approaches to large, high-dimensional sequential decision making under uncertainty have mostly either had high tractability and limited expressive power, or low tractability and a large amount of expressive power: only a subset of either set of approaches also make formal guarantees on the resulting algorithm performance. In this thesis we have presented three algorithms, SM-POMDP, CORL, and LOP-POMDP, which reside in the middle region of expressive power and tractability. In doing so these approaches achieve a substantial advantage in tractability over algorithms that assume more expressive discrete-state models, but are still significantly more expressive than, for example, the famous linear quadratic Gaussian controller (Burl, 1998). This combination of expressive power and tractability was achieved through the use of compact parametric models, which also enable formal optimality guarantees to be placed on variants of the resulting algorithms.

In our first contribution we presented the SM-POMDP algorithm for planning in low-dimensional large (continuous) environments. This approach extended prior parametric work to a more general class of environments by using a weighted sum of Gaussians to represent the world models. We also provided an analysis of the optimality of this approximate planning approach which has not, to our knowledge, been performed previously for approximate continuous-state POMDP planners.

In our second contribution we showed we could get significant improvements in the sample complexity of learning in large, high-dimensional environments which had noisy-offset typed dynamics models. We presented the CORL algorithm for learning in such domains, and proved that the sample complexity of CORL was a polynomial function of the state dimension. This was a significant improvement over past discrete-state approaches that scale exponentially with the state dimension, and CORL's sample complexity is only a linear increase over prior approaches that restrict their expressive power to assuming a single dynamics model for all states (see e.g. Strehl and Littman (2008)). Our theoretical bounds also consider that at this time general continuous-state MDPs can only be solved approximately, and therefore MDP planning will only be approximately optimal. We demonstrated that noisy-offset dynamics were adequate approximations of real world dynamics through a robot experiment and a simulation that used real car trajectory data.

In our final thesis contribution (Chapter 5) we presented a new approach to planning in partially observable environments that is tractable in large, high-dimensional environments. The LOP-POMDP planner is a forward search technique that represents the world models using compact parametric functions which have a number of parameters that scales polynomially with the state dimension. We presented two particular parametric representations that allowed approximate versions of the forward search operations to be performed with a computational complexity that scales polynomially with the state dimension. This is a significant advance over discrete-state representations that result in forward search operations with an exponential computational dependence on the state dimension. LOP-POMDP scaled to a high-dimensional global driving navigation simulation, where an agent must perform local collision avoidance while trying to reach a particular location.

6.1 Future Research Possibilities

This section will briefly outline some potential avenues for future work.

6.1.1 Planning in CORL

Due to the approximate MDP planning used in the CORL algorithm, we cannot currently guarantee both polynomial sample complexity and polynomial computational complexity for CORL. There are a number of recent advances in continuous-state MDP planning (Kocsis & Szepesvári, 2006; Kveton & Hauskrecht, 2006; Marecki & Tambe, 2008). However perhaps the most promising approach is to draw from the forward search planning techniques first outlined by Kearns, Mansour and Ng (2002). As discussed in Sections 4.4 and 5.5, these approaches can yield an ϵ-close estimate of the state-action values for a single state with a computational complexity that is independent of the size of the state space. This impressive result comes at the cost of an exponential dependence on the horizon of the forward search, with a large base in the exponential dependence. However, it would be interesting to try to incorporate an ϵ-optimal forward search planner as an alternate planner for solving the known MDP inside the CORL algorithm, and extend our existing theoretical bounds to this new case. It would also be interesting to empirically compare our current approach to using a forward search planner inside CORL.

6.1.2 Learning Types

More generally, it would be interesting to consider learning the types in a CORL representation. Types could be considered states that share a sufficiently similar dynamics model. It would be interesting to explore how to precisely quantify the similarity between parametric dynamics models in order to make formal guarantees on the resulting learner's performance.

6.1.3 Increasing the horizon of search in LOP-POMDP

The current version of LOP-POMDP expands all action nodes at every level of the tree, and samples the same number of observation nodes for every action. As recently surveyed by Ross et al. (2008b), forward search planning in discrete-state POMDPs has substantially benefited from approaches that selectively expand the forward search tree, using upper and lower bounds on the value function. As outlined in Section 5.6, we believe that it should be possible to provide loose initial bounds on the value function associated with world models represented by compact parametric functions. We hope to explore this in the future as we believe this may be a significant step towards allowing LOP-POMDP to consider problems which require a long horizon lookahead to produce good policies.

A more ambitious step would be to consider integrating LOP-POMDP with a hierarchical planner (e.g. Marthi, Russell and Wolfe (2008)). Hierarchy presents the potential to allow an agent to consider making decisions at very different levels of resolution, which seems almost definitely necessary when one considers how to make decisions over the course of a lifetime. LOP-POMDP could be used as a subcontroller in part of a hierarchical planner to provide good low level control when handling input sensors that are commonly continuous-valued.

6.1.4 Model assumptions

It would also be interesting to conduct a more thorough investigation of how robust these models are in environments that fail to satisfy their assumptions. This issue was considered to some degree for the CORL algorithm in Chapter 4 by using CORL on a robotics domain, and with data sampled from real car trajectories, that were unlikely to have dynamics that precisely matched the CORL dynamics model assumptions. It would be useful to perform a similar analysis for the PFS-POMDP algorithm. It may be the case that even if the world does not fulfill the model assumptions, that the solutions obtained by assuming it does are still quite reasonable.

6.1.5 Applications

High-dimensional problems often arise when an agent is operating among other agents that it does not have control over, but must consider when making decisions, such as navigating through a crowd during a search and rescue effort. It would be interesting to try to use LOP-POMDP on such a robot application. One possibility is to use it on the autonomous helicopter platform being developed in Professor Roy's laboratory.

6.2 Conclusion

This thesis has examined using compact parametric functions to represent the world models in large, high-dimensional Markov decision processes. Though these representations are not sufficiently powerful to represent all domains, we have demonstrated these models have sufficient expressive power to represent several real-world and simulated applications of interest. Through the use of theoretical analysis and empirical experiments we have demonstrated that algorithms which use compact parametric representations can offer an attractive tradeoff between tractability, expressive power, and optimality when making sequential decisions in high-dimensional, continuous-valued, stochastic domains.

A

Theorems from Outside Sources Used in Thesis Results

Formula for product of 2 Gaussians, as referenced in Porta et al. pg.15 (2006)

$$\mathcal{N}(s|a,A)\mathcal{N}(s|b,B) = \mathcal{N}(a|b,A+B)\mathcal{N}(s|c,C) \tag{A.1}$$

where

$$\begin{aligned} C &= (A^{-1}+B^{-1})^{-1} \\ c &= C(A^{-1}a+B^{-1}b) \end{aligned}$$

Lemma 11. ***(Theorem from Kullback, 1967)*** *Let p_1 and p_2 be two probability density functions defined over $\mathcal{X}$. Define*

$$\Omega = \{x \in \mathcal{X} \mid p_1(x) \geq p_2(x)\}.$$

If p_1 and p_2 are both measurable (integrable) over Ω, then

$$\mathrm{d_{KL}}(p_1 \parallel p_2) \geq \frac{1}{8}\,||p_1 - p_2||_1^2\,.$$

Lemma 12. *Suppose the covariance matrix Σ_1 is non-singular; that is its eigenvalues $\lambda_1 : \lambda_N > 0$. Then*

$$\begin{aligned} \mathrm{tr}\left(\Sigma_1^{-1}\right) &= \sum_{i=1}^{N}\frac{1}{\lambda_i} \leq \frac{N}{\lambda_N} \\ \max_{ij}|\Sigma_1^{-1}(i,j)| &\leq ||\Sigma_1^{-1}||_1 \\ ||\Sigma_1^{-1}||_1 \leq \sqrt{N}||\Sigma_1^{-1}||_2 &= \frac{\sqrt{N}}{\lambda_N} \end{aligned}$$

Proof. We prove the three upper bounds one by one:

1. It is a known fact that the trace of a matrix equals the sum of its eigenvalues. The first equality follows from the observation that the eigenvalues of Σ_1^{-1} are $\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \ldots, \frac{1}{\lambda_N}$.

2. This inequality follows from the definition of $||\Sigma_1^{-1}||_1$: it is the maximum absolute

row sum of the matrix Σ_1^{-1}, and therefore is not less than the largest absolute component of the matrix.

3. It is known that $||A||_1 \leq \sqrt{N}||A||_2$ for any $N \times N$ matrix A (see, eg. theorem 5.6.18 in Horn and Johnson (1986)). On the other hand, $||\Sigma_1^{-1}||_2$ equals the largest eigenvalue of Σ_1^{-1}, which is $\frac{1}{\lambda_N}$.

□

Lemma 13. ***(Lemma 2.7.1 and Theorem 2.7.2 from Golub (1996))*** *Suppose $Ax = b$ and $(A + \Delta A)y = b + \Delta b$ with $||\Delta A|| \leq \epsilon\, ||A||$ and $||\Delta b|| \leq \epsilon\, ||b||$. If $\epsilon\kappa(A) < 1$, then $A + \Delta A$ is nonsingular, and*

$$\frac{||y - x||}{||x||} \leq \frac{2\epsilon\kappa(A)}{1 - \epsilon\kappa(A)},$$

where $||\cdot||$ can be any ℓ_p matrix/vector norm, and $\kappa(A) = ||A||\, ||A^{-1}||$ is the corresponding condition number.

Lemma 14. ***(A trace inequality of von Neumann (1937))*** *Let A and B be two symmetric matrices of order n, whose singular values are $\xi_1 \geq \xi_2 \geq \cdots \geq \xi_n \geq 0$ and $\zeta_1 \geq \zeta_2 \geq \cdots \geq \zeta_n \geq 0$, respectively. Then*

$$|\mathrm{tr}\,(AB)| \leq \sum_{i=1}^{n} \xi_i \zeta_i.$$

Bibliography

Abbeel, P., & Ng, A. (2004). Apprenticeship learning via inverse reinforcement learning. *Proceedings of the 21st International Conference on Machine Learning (ICML).*

Abbeel, P., & Ng, A. Y. (2005). Exploration and apprenticeship learning in reinforcement learning. *Proceedings of the 22nd International Conference on Machine Learning (ICML)* (pp. 1–8).

Altman, E. (2002). *Handbook of markov decision processes:methods and applications*, chapter 16.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. *Proceedings of the Twelfth International Conference on Machine Learning (ICML).*

Bellman, R. (1957). *Dynamic programming.* Princeton University Press.

Blackmore, L., Gil, S., Chung, S., & Williams, B. (2007). Model learning for switching linear systems with autonomous mode transitions. *IEEE Conference on Decision and Control.*

Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., & Mihailidis, A. (2005). A decision-theoretic approach to task assistance for persons with dementia. *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).*

Boyan, J., & Moore, A. (1995). Generalization in reinforcement learning: Safely approximating the value function. *Advances in Neural Information Processing Systems (NIPS) 7* (pp. 369–376). Cambridge, MA: The MIT Press.

Brafman, R. I., & Tennenholtz, M. (2002). R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Bresina, J., Dearden, R., Meuleau, R., Ramakrishnan, S., Smith, D., & Washington, R. (2002). Planning under continuous time and resource uncertainty: a challenge for ai. *Uncertainty in Artificial Intelligence.*

Brooks, A., Makarenko, A., Williams, S., & Durrant-Whyte, H. (2006). Parametric POMDPs for planning in continuous state spaces. *Robotics and Autonomous Systems.*

Burl, J. B. (1998). *Linear optimal control.* Prentice Hall.

Castro, P., & Precup, D. (2007). Using linear programming for Bayesian exploration in Markov decision processes. *20th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 2437–2442).

Charlin, L., Poupart, P., & Shioda, R. (2008). Automated hierarchy discovery for planning in partially observable environments. *Advances in Neural Information Processing Systems (NIPS) 21.*

Cheng, H. (1988). *Algorithms for partially observable markov decision processes*. Doctoral dissertation, University of British Columbia.

Chow, C.-S., & Tsitsiklis, J. N. (1989). The complexity of dynamic programming. *Journal of Complexity*, *5*, 466–488.

Chow, C.-S., & Tsitsiklis, J. N. (1991). An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, *36*, 898–914.

Collins, M., Dasgupta, S., & Schapire, R. (2002). A generalization of principal components analysis to the exponential family. *Advances in Neural Information Processing Systems 14 (NIPS)*.

Deisenroth, M. P., Rasmussen, C. E., & Peters, J. (2008). Model-based reinforcement learning with continuous states and actions. *Proceedings of the 16th European Symposium on Articial Neural Networks* (pp. 19–24).

Doshi, F., Pineau, J., & Roy, N. (2008). Reinforcement learning with limited reinforcement: Using bayes risk for active learning in pomdps. *Proceedings of the 25th International Conference on Machine Learning (ICML)*.

Duff, M. (2002). *Optimal learning: Computational proceduresfor Bayes-adaptive Markov decision processes*. Doctoral dissertation, University of Massassachusetts Amherst.

Dutech, A., T.Edmunds, Kok, J., M.Lagoudakis, Littman, M., M.Riedmiller, Russell, B., B.Scherrer, R.Sutton, S.Timmer, N.Vlassis, A.White, & S.Whiteson (2005). Reinforcement learning benchmarks and bake-offs ii. *Advances in Neural Information Processing Systems*.

Engel, Y., Mannor, S., & Meir, R. (2003). Bayes meets bellman: The gaussian process approach to temporal difference learning. *Proceedings of the 20th International Conference on Machine Learning (ICML)*.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with gaussian processes. *Proceedings of the 22nd International Conference on Machine Learning (ICML)*.

Eriksson, J., Balakrishnan, H., & Madden, S. (2008). Cabernet: A WiFi-Based Vehicular Content Delivery Network. *14th ACM MOBICOM*. San Francisco, CA.

Feng, Z., Dearden, R., Meuleau, N., & Washington, R. (2004). Dynamic programming for structured continuous Markov decision problems. *UAI*.

Foka, A., & Trahanias, P. (2007). Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*, *55*, 561–571.

Forbes, J., Huang, T., Kanazawa, K., & Russell, S. (1995). The BATmobile: Towards a Bayesian automated taxi. *IJCAI*.

Fox, E. B., Sudderth, E. B., Jordan, M. I., & Willsky, A. S. (2009). Nonparametric Bayesian learning of switching linear dynamical systems. *Advances in Neural Information Processing Systems (NIPS).*

Ghahramani, Z., & Hinton, G. (2000). Variational learning for switching state-space models. *Neural Computation, 12,* 831–864.

Goldberger, J., & Roweis, S. (2005). Hierarchical clustering of a mixture model. *NIPS.*

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations.* The Johns Hopkins University Press. 3rd edition.

Gordon, G. (1995). Stable function approximation in dynamic programming. *Proceedings of the 12th International Conference on Machine Learning (ICML)* (pp. 261–268).

Guez, A., Vincent, R., Avoli, M., & Pineau, J. (2008). Adaptive treatment of epilepsy via batch-mode reinforcement learning. *IAAI.*

Hansen, E. (1998). Solvings POMDPs by searching in policy space. *Uncertainty in Artificial Intelligence.*

Hansen, E., & Zhou, R. (2003). Synthesis of hierarchical finite state controllers for pomdps. *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS).*

Hansen, E. A., & Feng, Z. (2000). Dynamic programming for POMDPs using a factored state representation. *Artificial Intelligence Planning Systems.*

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research, 13,* 33–94.

Hauskrecht, M., & Fraser, H. (2000). Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine, 18,* 331–244.

Hershey, J., & Olsen, P. (2007). Approximating the kullback leibler divergence between gaussian mixture models. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP).*

Hoey, J., & Poupart, P. (2005). Solving pomdps with continuous or large discrete observation spaces. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1332–1338).

Hoey, J. amd St-Aubin, R., Hu, A., & Boutilier, C. (1999). Spudd: stochastic planning using decision diagrams. *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)* (pp. 279–288). Stockholm.

Horn, R. A., & Johnson, C. R. (1986). *Matrix analysis.* Cambridge University Press.

Horvitz, E., Apacible, J., Sarin, R., & Liao, L. (2005). Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI).*

Howard, R. (1960). *Dynamic programming and markov processes.* M.I.T. Press.

Hsiao, K., Lozano-Pérez, T., & Kaelbling, L. (2008). Robust belief-based execution of manipulation programs. *the Eighth International Workshop on the Algorithmic Foundations of Robotics (WAFR).*

Jong, N. K., & Stone, P. (2007). Model-based function approximation in reinforcement learning. *Sixth International Conference on Autonomous Agents and Multiagent Systems (AAMAS).*

Julier, S., & Uhlmann, J. (1997). A new extension of the Kalman filter to nonlinear systems. *International Symposium on AeroSense.*

Kaelbling, L., Littman, M., & Cassandra, A. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kakade, S. (2003). *On the sample complexity of reinforcement learning.* Doctoral dissertation, University College London.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, *82*, 35–45.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 740–747).

Kearns, M., Mansour, Y., & Ng, A. (2000). Approximate planning in large pomdps vis reusable trajectories. *Advances in Neural Information Processing Systems 12 (NIPS).*

Kearns, M., Mansour, Y., & Ng, A. (2002). A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, *49*, 193–209.

Kearns, M. J., & Singh, S. P. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, *49*, 209–232.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. *Proceedings of the Seventeenth European Conference on Machine Learning (ECML-06)* (pp. 282–293).

Kullback, S. (1967). A lower bound for discrimination in terms of variation. *IEEE Transactions on Information Theory*, *13*, 126–127.

Kurniawati, H., Hsu, D., & W.S., L. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. *Robotics: Science and Systems.*

Kveton, B., & Hauskrecht, M. (2006). Solving factored MDPs with exponential-family transition models. *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS).*

Lagoudakis, M., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research, 4,* 1107–1149.

Leffler, B. (2008). *Perception-based generalization in model-based reinforcement learning.* Doctoral dissertation, Rutgers, The State University of New Jersey.

Leffler, B., Littman, M., & Edmunds, T. (2007a). Efficient reinforcement learning with relocatable action models. *Proceedings of the AAAI National Conference on Artificial Intelligence.*

Leffler, B. R., Littman, M. L., & Edmunds, T. (2007b). Efficient reinforcement learning with relocatable action models. *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)* (pp. 572–577).

Leffler, B. R., Littman, M. L., Strehl, A. L., & Walsh, T. J. (2005). Efficient exploration with latent structure. *Proceedings of Robotics: Science and Systems.* Cambridge, USA.

Li, L., & Littman, M. L. (2005). Lazy approximation for solving continuous finite-horizon mdps. .

Li, L., Littman, M. L., & Walsh, T. J. (2008). Knows what it knows: A framework for self-aware learning. *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)* (pp. 568–575).

Madani, O., Hanks, S., & Condon, A. (2003). On the undecidability of probabilistc planning and related stochastic optimization problems. *Artifcial Intelligence, 147,* 5–34.

Marecki, J., & Tambe, M. (2008). Towards faster planning with continuous resources in stochastic domains. *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI).*

Marthi, B., Russell, S., & Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS).*

Martin-Guerrero, J., Gomez, F., Soria-Olivas, E., Schmidhuber, J., Climente-Marti, M., & Jimenez-Torres, N. (2009). A reinforcement learning approach for individualizing erythropoietin dosages in hemodialysis patients. *Expert Systems with Applications, 36,* 9737–9742.

McAllester, D., & Singh, S. (1999). Approximate planning for factored pomdps using belief state simplification. *Uncertainty in Artificial Intelligence.*

McCallum, A. (1995). *Reinforcement learning with selective perception and hidden state.* Doctoral dissertation, University of Rochester.

Munos, R., & Moore, A. (1999). Variable resolution discretization for high-accuracy solutions of optimal control problems. *International Joint Conference on Artificial Intelligence*.

Ng, A. Y., & Jordan, M. (2000). Pegasus: A policy search method for large mdps and pomdps. *Uncertainty in Artificial Intelligence (UAI)*.

Oh, S., Rehg, J., Balch, T., & Dellaert, F. (2005). Data-driven mcmc for learning and inference in switching linear dynamic systems. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Paquet, S., Tobin, L., & Chaibdraa, B. (2005). An online POMDP algorithm for complex multiagent environments. *AAMAS*.

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, *21*, 682–697.

Pineau, J., Gordan, G., & Thrun, S. (2003a). Point-based value iteration: an anytime algorithm for pomdps. *International Joint Conference on Artificial Intelligence* (pp. 1025–1032).

Pineau, J., Gordon, G., & S., T. (2003b). Policy-contingent abstraction for robust robot control. *Conference on Uncertainty in Articifical Intelligence (UAI)*.

Pineau, J., Gordon, G., & Thrun, S. (2006). Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, *27*, 335–380.

Porta, J., Spaan, M., Vlassis, N., & Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research*, *7*, 2329–2367.

Poupart, P. (2005). *Exploiting structure to efficiently solve large scale partially observable markov decision processes*. Doctoral dissertation, University of Toronto.

Poupart, P., & Boutilier, C. (2002). Value-directed compression of POMDPs. *Advances in Neural Information Processing Systems (NIPS) 15* (pp. 1547–1554).

Poupart, P., & Boutilier, C. (2005). VDCBPI: an approximate scalable algorithm for large scale POMDPs. *In Advances in Neural Information Processing Systems 17 (NIPS)* (pp. 1081–1088).

Poupart, P., Vlassis, N., Hoey, J., & Regan, K. (2006). An analytic solution to discrete bayesian reinforcement learning. *International Conference on Machine Learning*.

Puterman, M. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. Wiley Series in Probability and Mathematical Sciences. John Wiley and Sons, Inc.

Rasmussen, C. E., & Kuss, M. (2004). Gaussian processes in reinforcement learning. *Advances in Neural Information Processing Systems 16 (NIPS)*.

Ross, S., Chaib-draa, B., & Pineau, J. (2008a). Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. *ICRA*.

Ross, S., Pineau, J., Paquet, S., & Chaib-draa, B. (2008b). Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 663–704.

Roy, N. (2003). *Finding approximate belief solutions through belief compression*. Doctoral dissertation, Carnegie Mellon Unversity.

Roy, N., Gordon, G., & Thrun, S. (2005). Finding approximate pomdp solutions through belief compression. *Journal of Artificial Intelligence Research*, *23*, 1–40.

Roy, N., & Thrun, S. (2000). Coastal navigation with mobile robots. *Advances in Neural Information Processing Systems (NIPS 1999)* (pp. 1043–1049).

Schäl, M. (2002). *Handbook of markov decision processes:methods and applications*, chapter 15.

Shani, G., Brafman, R., & Shimony, S. (2007). Forward search value iteration for POMDPs. *IJCAI*.

Shani, G., Poupart, P., Brafman, R., & S., S. (2008). Efficient add operations for point-based algorithms. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Sim, H., Kim, K., Kim, J., Chang, D., & Koo, M. (2008). Symbolic heuristic search value iteration for factored pomdps. *Proceedings of the 23rd Conference on Artificial Intelligence (AAAI)*.

Smith, T. (2007). *Probabilistic planning for robotic exploration*. Doctoral dissertation, Carnegie Mellon University.

Smith, T., & Simmons, R. (2004). Heuristic search value iteration for POMDPs. *UAI*.

Smith, T., & Simmons, R. (2005). Point-based pomdp algorithms: Improved analysis and implementation. *UAI*.

Sondik, E. J. (1971). *The optimal control of partially observable markov processes*. Doctoral dissertation, Stanford University.

Spaan, M., & Vlassis, N. (2005a). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

Spaan, M., & Vlassis, N. (2005b). Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, *24*, 195–220.

Strehl, A. L., Li, L., & Littman, M. L. (2006). Incremental model-based learners with formal learning-time guarantees. *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI)*.

Strehl, A. L., & Littman, M. L. (2008). Online linear regression and its application to model-based reinforcement learning. *Advances in Neural Information Processing Systems (NIPS) 20.*

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Theocharous, G., & Murphy, K. andk Kaelbling, L. (2004). Representing hierarchical pomdps as dbns for multi-scale robot localization. *International Conference on Robotics and Automation.*

Thrun, S. (2000). Monte carlo POMDPs. *Advances in Neural Information Processing Systems 12 (NIPS).*

Thrun, S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Hähnel, D., Rosenberg, C., Roy, N., Schulte, J., & Schulz, D. (2000). Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research, 19*, 972–999.

Toussaint, M., Charlin, L., & Poupart, P. (2008). Hierarchical POMDP controller optimization by likelihood maximization. *Uncertainty in Artificial Intelligence.*

Tsitsiklis, J., & Van Roy, B. (1996). Feature-based methods for large scale dynamic programming. *Machine Learning, 22*, 59–94.

von Neumann, J. (1937). Some matrix-inequalities and metrization of matrix-space. *Tomsk University Review, 1*, 286–300.

Wang, T., Lizotte, D., Bowling, M., & Schuurmans, D. (2005). Bayesian sparse sampling for on-line reward optimization. *ICML.*

Watkins, C. J. (1989). *Learning from delayed rewards.* Doctoral dissertation, King's College, University of Cambridge, UK.

Zhang, K., & Kwok, J. (2006). Simplifying mixture models through function approximation. *NIPS.*

Zhang, N. L., & Zhang, W. (2001). Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artcial Intelligence Research, 14*, 1–28.

Zhou, E., Fu, M., & Marcus, S. (to appear). Solving continuous-state POMDPs vis density projection. *IEEE Transactions on Automatic Control.*