Decision Making in the Presence of Complex Dynamics from Limited, Batch Data

by

Joshua Mason Joseph

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2014

Author . . . .
Department of Aeronautics and Astronautics
May 22, 2014

Certified by . . . .
Nicholas Roy
Associate Professor of Aeronautics and Astronautics
Chairman, Thesis Committee

Certified by . . . .
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Member, Thesis Committee

Certified by . . . .
J. Andrew Bagnell
Associate Professor, Robotics Institute
Member, Thesis Committee

Certified by . . . .
Emilio Frazzoli
Professor of Aeronautics and Astronautics
Member, Thesis Committee

Accepted by . . . .
Paulo C. Lozano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Decision Making in the Presence of Complex Dynamics from Limited, Batch Data

by

Joshua Mason Joseph

Submitted to the Department of Aeronautics and Astronautics
on May 22, 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Robot decision making in real-world domains can be extremely difficult when the robot has to interact with a complex, poorly understood environment. In these environments, a data-driven approach is commonly taken where a model is first learned and then used for decision making since expert knowledge is rarely sufficient for specifying the world's dynamics. Unfortunately, learning a model for a complex environment often involves fitting a large number of parameters which can require an unobtainable amount of data. In real-world domains we are also typically confronted with fitting a model that is only an approximation of the true dynamics, causing difficulties for standard learning approaches.

In this thesis we explore two core methodologies for learning a model for decision making in the presence of complex dynamics: explicitly selecting the model which achieves the highest estimated performance and allowing the model class to grow as more data is seen. We show that our approach for explicitly selecting the model with the highest estimated performance has desirable theoretical properties and outperforms standard minimum error fitting techniques on benchmark and real-world problems.

To grow the size of model class with the amount of data, we first show how this can be accomplished by using Bayesian nonparametric statistics to model the dynamics, which can then be used for planning. We then present an alternative approach which grows the policy class using the principle of *structural risk minimization*, for which the resulting algorithm has provable performance bounds with weak assumptions on the true world's dynamics.

Acknowledgments

I cannot imagine ever again being surrounded by a collection of such awe-inspiring people as I have been during the past eight years at MIT, and there are a few that I'd like to specifically thank.

First and foremost, my thanks go my advisor, Nick. While I could spend (at least) a page of text listing the things I've learned from Nick – everything from how to tell the story of a paper to the technical minutia of many planning algorithms – there is one thing above all that I am most thankful for: our arguments. The hours we spent arguing about research were the most challenging and educational times of my life. It never once felt like it mattered if Nick or I were "right" (even though Nick usually was[1]), it was always about the science, and the science alone.

I am also so gratetful to have spent these years with my RRG lab-mates, especially the fantastic and inevitable push-back during group presentations. I'd like to thank a few friends who have been incredibly influential on me over the years. Finale, I learned so much about nonparametric (and parametric) statistics from you and without you I'm fairly sure I never would have gotten my first couple papers published. Alborz, I owe the vast majority of my RL knowledge to our model-based vs value-based (among other) discussions. Stefanie, whenever we happen to be in a conversation about AI, it didn't just feel like the right conversation to have but the conversation everyone should be having. Javier, who in the same week, had started (multi-hour) conversations with "Look at what I found in an obscure Scandinavian math journal" and "There shouldn't be money". No longer sitting ten feet apart is the thing I will miss most from my time here. Mike, to providing much of the inspiration behind my finally finishing up this document and the uncanny sense that when we attempt insane things, the most likely outcome is that the insane thing will actually happen.

I also thank my committee: Jon How, Drew Bagnell, and Emilio Frazzoli. The work contained in this document is much stronger thanks to your insightful input and feedback. Last but not least, I also thank my thesis readers, Marek and Stefanie, as

[1]Remember the Emotiv headset?

well as Russ Tedrake and John Roberts from the Robot Locomotion Group for the use of and help operating the hydrodynamic cartpole.

This research was sponsored by iRobot (with special thanks to Michael Rosenstein), the Army Research Office[2], and the Office of Naval Research[3]. I would like to extend an additional thanks to the Charles Stark Draper Laboratory for its support through my Master's studies.

[2]Under the Nostra project, STTR W911NF-08-C-0066

[3]Under Behzad Kamgar-Parsi, MURI N00014-11-1-0688

Contents

List of Figures

Chapter 1

Introduction

Real-world robots commonly have to act in complex, poorly understood environments where the true world dynamics are unknown. For example, consider the task of intercepting and tracking a car driving around the greater Boston area with an autonomous helicopter (described in detail in Section 3). The success of interception and tracking tasks often hinges on the quality of the motion models our agent has for predicting the target's future locations. These predictions are especially important when our agent's sensor range is limited. Unfortunately, the motion patterns of targets are often difficult to specify from expert knowledge alone. For example, suppose that our agent is a helicopter that must intercept and track a car or several cars in a large region such as a city. The necessary model of traffic patterns may be hard to specify. Even determining what parameters are important to model the target's behavior—and how they should interact—can be a challenging task.

A data-driven approach to learning the target's motion patterns avoids the need for an expert to fully specify the model. Instead, the agent simply uses previously observed trajectories of the target to predict the target's future locations, where these predictions may depend on both the target's current position and past position history. Using a data-driven approach also side-steps the need to understand the target's motivations, which may appear irrational to an outside observer. For example, drivers rarely take the minimum-time route to a location [Letchner et al., 2006]; an expert model that assumes that optimizing travel time is the driver's primary objective will likely make poor predictions about a car's future locations. Our approach focuses on the features our agent needs to make good predictions of the targets' future locations.

While a data-driven approach reduces the need for expert knowledge, we still need to specify the class of models to which we expect the target's motion patterns to belong. For example, we may choose to model the target's motion as a series of straight-line segments, higher-order splines, or even cylindrical trajectories. When considering real-world data, the correct class of motion models is not always obvious. One solution is to consider sophisticated model classes with parameters governing the forms of all the motion patterns we expect to occur. While such a flexible model class may be able to model any observable motion pattern, large amounts of data will be needed to train the many parameters. Collecting sufficient data to train a large number of parameters may be prohibitively expensive.

We first approach decision making in complex environments using Bayesian nonparametric statistics to model the dynamics. Nonparametric approaches are well-suited for poorly-understood environments because they let the data determine the sophistication of the model. The Bayesian aspect helps the model generalize to unseen data and make inferences from noisy data. These techniques are representationally powerful and generalize well with relatively little training data due to their ability to appropriately fit the complexity of the model [Rasmussen and Ghahramani, 2002]. We present two Bayesian nonparametric models and apply them to real-world data: a model of mobile targets and a model of battery health. We use these models in Chapters 3 and 4 for decision making with GPS data from taxis in the greater Boston area and iRobot Roomba battery data, respectively.

Although Bayesian nonparametric models work well for some real-world problems, it is sometimes not clear if the mathematical sophistication required to use these models results in a performance gain compared to a parametric model which roughly approximates the true dynamics. It is also often difficult to understand if enough data is available to grow the model to sufficiently approximate the true dynamics. To continue studying model learning for decision making in complex environments we then turn to parametric modeling with the assumption that the model class is *misspecified*, meaning it cannot represent true environment dynamics.

The key problem caused by misspecified parametric model classes is that choosing a model using the standard metrics for "closeness" between the true model and a model in our class can result in arbitrarily poor performance, even in the limit of infinite data [Baxter and Bartlett, 2001]. These standard metrics are typically either minimum squared error or maximum likelihood, both of which measure the error between the data and the models in the model class to determine which model is closest to the truth. We develop an algorithm called Reward Based Model Search (RBMS), a batch Reinforcement Learning (RL) technique that overcomes this problem. Our approach improves upon the standard minimum error model selection techniques by explicitly selecting the model which achieves the highest expected reward, rather than the model which has lowest prediction error. An evaluation of RBMS compared to standard minimum error techniques is shown on benchmark RL problems and the real-world hydrodynamic cart-pole, a domain whose complex dynamics cannot be known exactly. We then extend RBMS from parametric to Bayesian nonparametric models to show how to overcome misspecification with these far more sophisticated classes of models.

A motivation behind using misspecified models is that they will generally require less data, on account of their being "small". Unfortunately, RBMS with parametric models can still over-fit the misspecified model class with limited data and Bayesian nonparametric models cannot provide any performance guarantees with a misspecified model class. We introduce an algorithm called Structural Return Maximization (SRM) which solves both of these difficulties. SRM was developed by mapping the principle of *structural risk minimization* from the statistical learning theory literature to RL. The resulting algorithm prevents over-fitting to an overly large policy class by appropriately choosing the size of the policy class to the amount of data available. SRM penalizes policy classes based on the Rademacher complexity, a measure of the size of the class of policies. We present analysis which shows that the return of a policy produced by SRM has provable bounds on performance with extremely weak assumptions on the true underlying dynamics.

1.1 Contributions and Document Outline

Reinforcement Learning with Bayesian Nonparametric Models The contribution of Chapter 3 is a Bayesian nonparametric model of driver behavior and remaining battery health. We show how Bayesian nonparametric models are well-suited for modeling complex, poorly understood environments and how these models can then be used in decision making.

We first apply Bayesian nonparametric models to modeling motion patterns (Section 3.1). We show that these models are well-suited for poorly understood environments because they let the training data determine the sophistication of the model—we no longer need to specify which parameters are important. Moreover, the Bayesian aspect allows the model generalize to unseen data and make inferences from noisy data. Specifically, we model a target's motion patterns with a Dirichlet process mixture model over Gaussian process target trajectories (DPGP). Using this nonparametric model boosts learning rates by generalizing quickly from small amounts of data continuing to increase in sophistication as more trajectories are observed. We applied this DPGP model to applications tracking a single target whose current position was always observed (Section 3.2) on both synthetic and real GPS data.

We then no longer assume that the target's position is directly available to the agent (Section 3.3). Instead, we consider scenarios in which the agent can only observe the target if it is nearby; now the agent's goal is to first intercept and then track the target. Adapting our approach to make predictions about unseen targets using only partial information is one of the main contributions of Section 3.3. Second, we also consider scenarios where multiple targets must be intercepted and tracked. Modeling multiple targets fits seamlessly into our DPGP model, demonstrating both the quality and versatility of our approach.

In Chapter 4, we turn our attention to battery health modeling and present a Bayesian nonparametric model which we use to make battery replacement decisions. We do this by considering how the trajectories of voltage and temperature change as the battery cools after charging. The learned model allows us to predict time-to-battery-death which we train and evaluate on iRobot Roomba data.

Reward Based Model Search In real-world systems, the model is often not fully known and practitioners often rely on domain knowledge to choose a dynamics model class that reasonably trades off sample complexity for expressive power. In these systems, this trade-off means the chosen representation will often be misspecified (*i.e.*, the approximate representation cannot exactly capture the true dynamics). Such models can introduce representational bias, the difference between the performances of the true optimal policy and the best policy found based on the misspecified model class. Furthermore the learning algorithm using such model classes can introduce learning bias, which is the difference between the performances of the best possible policy in the model class and the policy that is produced by the learner [Kalyanakrishnan and Stone, 2011].

The focus of Chapter 5 is on identifying the cause of learning bias in Reinforcement Learning (RL) and presenting an algorithm to overcome it. Our contribution, Reward Based Model Search (RBMS), is a batch RL algorithm that estimates the performance of models in the model class and explicitly searches for the model that achieves the highest performance (Section 5.1). Additionally, we present a theoretical analysis of RBMS (Section 5.2) as well as an empirical validation on benchmark problems and the real-world hydrodynamic cart-pole system (Section 5.3). We conclude the chapter by applying RBMS, to Bayesian nonparametric models (Section 5.4).

Structural Return Maximization The main contribution of Chapter 6 is the Structural Return Maximization (SRM) algorithm. The algorithm was developed by applying the principle of Structural Risk Minimization from Statistical Learning Theory to RL. We first reframe RL as a classification problem (Section 6.1.1), allowing us to transfer generalization bounds developed for classification. Our analysis, based on Rademacher complexity [Bartlett and Mendelson, 2003], results in a bound on the return of any policy from a policy class (Section 6.1.2). Given a structure of policy classes, we then apply the principle of SRM to find the highest performing policy from the family of policy classes (Section 6.1.2). Lastly, we present an empirical validation of SRM on simulation domains in Section 6.2.

Chapter 2

Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a framework for solving sequential decision making problems under uncertainty. In RL, our problem is typically formulated as a Markov decision process (MDP) Sutton and Barto [1998]. A finite horizon MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, s^{start}, m, \rho, T)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, and s^{start} is the starting state. The dynamics model is defined as, $m : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, where $m(s, a, s') = p(s'|s, a)$, the probability of transitioning to state s' by taking action a in state s. The reward function[1], $\rho : \mathcal{S} \mapsto \mathbb{R}$, describes the reward the agent receives for being in state s. An episode of data is a sequence $s_0, a_0, s_1, a_1, \cdots, s_{T-1}, a_{T-1}$, where T is the episode length, a_t is the action taken at state s_t, and $s_{t+1} \sim m(s_t, a_t, \cdot)$.

The solution to a finite time MDP is the time-varying policy, $\pi_t : \mathcal{S} \to \mathcal{A}$, where, at each time step t, the policy maximizes return, such that

$$\pi^*_{0:T-1} = \arg\max_{\pi_{0:T-1}} V_{\pi_{0:T-1}}(s_0) \tag{2.1}$$

[1]The work contained in this thesis focuses on this simpler form of ρ due to ease of exposition but can be straightforwardly extended to the more general $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$.

where

$$V_{\pi_{0:T-1}}(s_0) = E_{s_1,\ldots,s_T}\left[\sum_{t=0}^{T-1}\rho(s_t)\middle| s_{\tau+1} \sim m(s_\tau, \pi_\tau(s_\tau), \cdot), s^{start} = s_0\right] \tag{2.2}$$

and $\pi_{0:T-1} = \{\pi_0, \pi_1, \ldots, \pi_{T-1}\}$. In this thesis, we do not assume m is known, which prevents us from explicitly calculating $V_{\pi_{0:T-1}}(s)$ using Equation 2.2. Instead, we estimate $V_{\pi_{0:T-1}}(s)$ from data.

Section 2.1.1 introduces methods for computing the return of a policy (Equation 2.2) with unknown m. Section 2.1.2 then introduces methods of solving Equation 2.1 based on minimizing different measures of error.

2.1.1 Policy Evaluation

To compute the return of a policy (Equation 2.2) without knowing the true dynamics model, m, we collect episodes of data from interactions with the world and use this data to approximately compute Equation 2.2. Policy evaluation can be run in two modes: *on-policy* and *off-policy*. In on-policy policy evaluation, data is collected only using the policy we are attempting to evaluate, which is opposed to off-policy policy evaluation, in which data may be collected under any policy.

A straightforward method for estimating $V_{\pi_{0:T-1}}(s_0)$ from on-policy data is using Monte Carlo (MC) policy evaluation Barto and Duff [1994], where $\pi_{0:T-1}$ is executed for N episodes starting from s_0. The estimate is computed using

$$V^{emp}_{\pi_{0:T-1}}(s_0) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T-1}\rho(s^n_t), \tag{2.3}$$

where s^n_t is the agent's state at time t of episode n. Off-policy policy evaluation, on the other hand, is far more complex and is the subject of Section 5.1.1.

2.1.2 Minimum Error Reinforcement Learning

Methods for solving Equation 2.1 generally fall into one of three categories: policy search, value-based, and model-based. In this section we review approaches which learns policies by minimizing a measure of error.

Policy Search Given a method for evaluating Equation 2.2, policy search (PS) methods parameterize the set of policies and search over the parameter space to solve for the optimal policy in Equation 2.1 using

$$\theta^* = \arg\max_{\theta \in \Theta} V_{\pi^{\theta}_{0:T-1}}(s_0) \tag{2.4}$$

where $\theta \in \Theta$ is the policy's parametrization Peshkin [2000]. A naive PS technique to find θ^* is to enumerate all parameter values $\theta_1, ..., \theta_{|\Theta|}$, compute $\hat{V}_{\pi^{\theta_1}_{0:T-1}}(s_0), ..., \hat{V}_{\pi^{\theta_{|\Theta|}}_{0:T-1}}(s_0)$, and select the parameter which achieves the highest return. However, this approach is impractical for many real-world domains as the parameter space is often large or continuous, preventing Θ from being directly searched over or even enumerated. Large policy spaces can be overcome using policy gradient Williams [1992], Sutton et al. [2000], where the parameter estimate, $\hat{\theta}$, is found using a gradient step

$$\hat{\theta}_{i+1} = \hat{\theta}_i + c\frac{\partial V_{\pi^{\hat{\theta}_i}_{0:T-1}}(s_0)}{\partial \hat{\theta}_i}, \tag{2.5}$$

for iteration i where $c > 0$ and $\hat{\theta}_i$ is updated until convergence. Equation 2.5 allows us to change $\hat{\theta}$ proportional to the gradient of return, which can be estimated using Equation 2.3 with data generated from the policy we wish to evaluate (*i.e.*, on-policy data). Therefore each $\hat{\theta}_i$ must be simulated a sufficient number of times to accurately estimate $V_{\pi^{\hat{\theta}_i}_{0:T-1}}(s_0)$.

Model-Based Model-based (MB) methods solve Equation 2.2 by explicitly learning a dynamics model, $m(s, a, s'; \theta)$ from data, which commonly results in a significant amount of data efficiency by generalizing in the space of transitions[2]. Maximum likelihood (ML) is a common method for inferring the dynamics model which maximizes the likelihood of the model conditioned on the data, where

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \prod_{n=1}^{N} \prod_{t=0}^{T-1} m(s_t^n, a_t^n, s_{t+1}^n; \theta). \tag{2.6}$$

A MB solver first uses Equation 2.6 to compute $\hat{\theta}$ and then uses Equations 2.2 and 2.1 to compute the policy optimal with respect to $\hat{\theta}$. The implicit assumption made by selecting a representation "closest" to the true representation (using minimum error) is that the policy which is optimal with respect to the minimum error representation will perform well in the true world. If, however, the representation for the dynamics model is not expressive enough to capture the true m, using Equation 2.6 can result in a poorly performing policy [Baxter and Bartlett, 2001].

Value-Based A class of approaches which aim to reduce the sample complexity of solving equation 2.4 are value-based (VB) methods. In VB we directly parametrize the value function $V^{\theta}_{\pi_{t:T-1}}(s_t)$ such that

$$V^{\theta}_{\pi_{t+1:T-1}}(s_{t+1}) = \rho(s_t) + \sum_{s \in S} m(s_t, \pi_t(s_t), s) V_{\pi_{t:T-1}}(s) \tag{2.7}$$

and

$$\theta^* = \arg\max_{\theta} V^{\theta}_{\pi_{0:T-1}}(s_0) \tag{2.8}$$

Solving Equations 2.7 and 2.8 is most commonly used for infinite time, discounted problems where either a generative model is known [Bellman, 1957, Howard, 1960] or on-policy data is available [Watkins and Dayan, 1992, Sutton and Barto, 1998].

[2]We purposely use θ for both the policy and model parametrization to illustrate that a dynamics model is an indirect policy representation.

2.2 Dirichlet Processes

In this section[3] we review Dirichlet processes (DP) [Teh, 2010, Antoniak, 1974], the mathematical object that is at the core of allowing the models to grow in Chapters 3 and 4. DPs define not just a measure over any measure but over measures that have Dirichlet distributed (Section A.2) finite marginals. This is identical to how a GP has Gaussian distributed finite marginals. Formally, for a random measure G over space Θ, it is said that $G \sim DP(\alpha, H)$ is Dirichlet process distributed with base distribution H and concentration parameter α if

$$(G(A_1), ..., G(A_r)) \sim \text{Dir}(\alpha H(A_1), ..., \alpha H(A_r))$$

for every finite partition $A_1, ..., A_r$ of Θ. A partition of Θ is a collection of disjoint sets that contain all elements of Θ. $A_1 = [0, 0.4)$ and $A_2 = [0.4, 1]$ is an example of a partition of Θ when Θ is the unit interval. To further clarify the notation, if H is a uniform distribution over the unit interval, $H(A_1) = \int_0^{0.4} dx = 0.4$.

The base distribution, H, can be understood as the "mean" of the DP. The concentration parameter, α, controls how closely a draw from this DP resembles its "mean." This is similar to the precision describing how closely a draw from a Gaussian distribution resembles its mean. In Figure 2-1 the top plot is the base distribution, $H = f_G(2, 1/2)$ (Section A.1), and the three plots below are for $\alpha = \{1, 10, 10000\}$. Immediately, what may strike the reader is that the draws of this process are discrete probability distributions even though the base measure is continuous. Something less obvious from the picture is that, while it is true they are discrete, there is a probability mass associated with every point of Θ (in this example $\Theta = [0, \infty)$). These plots show how the base measure can be thought of as the mean of a DP and α can be thought of as precision. The higher α is, the more closely the draw resembles H.

While there are many related processes, there are two that are particularly useful to this thesis: the Chinese restaurant process and the GEM process. Note that the Pólya urn scheme, the original process used to prove the existence of the DP, is not

[3]The text of this section originally appeared in Joseph [2008]

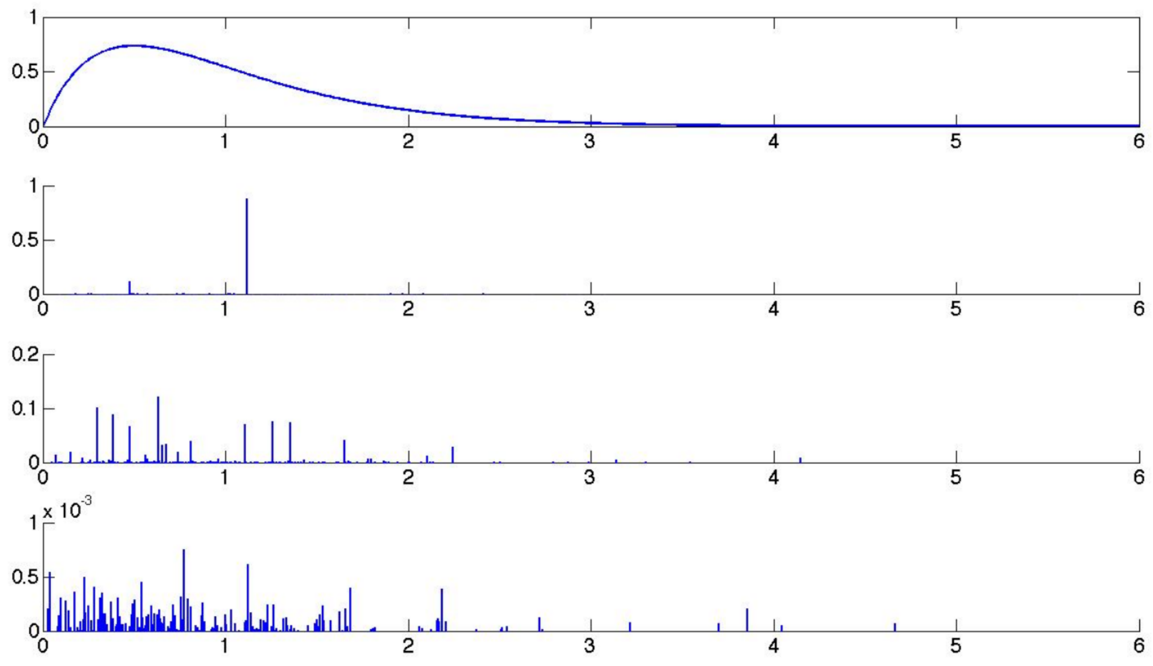


Figure 2-1: Three sample draws (bottom three plots) from the base measure (top) with $\alpha = 1$ (second from the top), $\alpha = 10$ (second from the bottom), $\alpha = 10000$ (bottom).

covered here but an interested reader can refer to Blackwell and Macqueen [1973].

The Posterior Dirichlet Process

The Dirichlet distribution is the conjugate prior of the multinomial distribution. That relation allows for the posterior distribution (the DP conditioned on previously observed $\theta_1, ..., \theta_n$) to itself be DP distributed when a DP is used as the prior. Formally, conditioned on $\theta_1, ..., \theta_n$ drawn from $G \sim DP(\alpha, H)$, for every finite partition,

$$(G(A_1), ..., G(A_r))|\theta_1, ..., \theta_n \sim \mathrm{Dir}(\alpha H(A_1) + n_1, ..., \alpha H(A_r) + n_r)$$

where n_i is the number of $\theta_1, ..., \theta_n$ that fall in the partition A_i. Therefore the posterior DP can be written as

$$G|\theta_1, ..., \theta_n \sim DP\left(\alpha + n, \frac{\alpha}{\alpha + n} H + \frac{1}{\alpha + n} \sum_{i=1}^{n} \delta_{\theta_i}\right)$$

where δ_{θ_i} is a point mass centered at θ_i (i.e., the contribution of $\frac{1}{\alpha+n}$ is only added to $\frac{\alpha}{\alpha+n}H$ at θ_i for $i = 1, ..., n$).

Chinese Restaurant Process

The Chinese restaurant process (CRP), initially named in Aldous [1985], comes from that author's observation that Chinese restaurants appear to have infinitely many tables. The generative model for the CRP is as follows. Consider an empty restaurant with countably infinite tables. The first customer walks in and sits at the first table deterministically. The second customer sits at the first table with probability $\frac{1}{1+\alpha}$ and the second table with probability $\frac{\alpha}{1+\alpha}$. It then continues such that the nth customer sits at a previously occupied table proportional to the number of customers already seated at it and a new table proportional to α.

As described thus far, this process is a measure on partitions of the integers where each customer is an integer and each table is a partition. This process can be extended to produce a draw from a DP with base distribution H. When a customer sits at a new table, a parameter for that table is drawn from the base measure H. A customer sitting at an already occupied table inherits the parameter of that table. Looking back at Figure 2-1, the bottom three plots show the next customer sitting at a table with parameter θ proportional to the height of the bar at θ. Therefore, the distribution of $\theta_1 \in \Theta$ as the first customer walks in is $\theta_1|\alpha, H \sim H$, and after n people have been seated is

$$\theta_{n+1}|\theta_1, ..., \theta_n, \alpha, H \sim \sum_{i=1}^{n} \frac{1}{n+\alpha}\delta(\theta_{n+1}, \theta_i) + \frac{\alpha}{n+\alpha}H. \tag{2.9}$$

where δ is the Kronecker-delta function ($\delta(a, b) = 1$ if and only if $a = b$ and zero otherwise). The purpose of explaining this process is that n samples from the CRP are equivalent to sampling from a draw from a DP n times. Figure 2-2 shows the graphical model for n data points, x_i for $i = 1$, ..., n, generated from this process. It is important to note that it is very likely that many of the θ_i values are the same (all people at the same table have the same θ_i) and while it is not obvious yet, this formulation makes inference about these θ_is difficult.

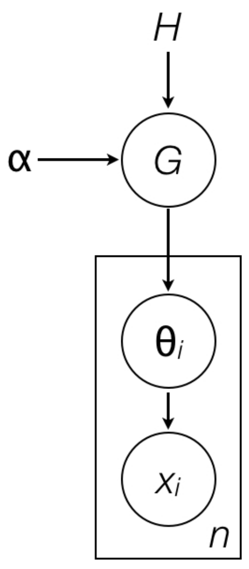


Figure 2-2: The graphical model of the CRP.

The purpose of explaining processes that are related to DPs is to enable inference procedures of the latent variables θ_i. These variables are referred to as latent since they are not directly observable from the data. Instead, the variables must be inferred from their relationship with variables that are observable. The next section describes another process related to DPs that make inference significantly easier. As a final remark on CRPs, if H is a continuous distribution it is impossible for two tables to have the same parameter, but if H is a discrete distribution there is a non-zero probability that two tables have the same parameter.

GEM Process

One of the simplest constructions of the DP was shown in Sethuraman [1994] by the use of "stick breaking." This construction is similar to that of the Chinese restaurant if there are infinite customers. The result of stick breaking is a discrete probability distribution over all natural numbers. These stick breaking weights, $\pi = (\pi_n)_{n=1}^{\infty}$ such that $\sum_{n=1}^{\infty} \pi_i = 1$, are said to be drawn from a GEM process Pitman [2002], standing

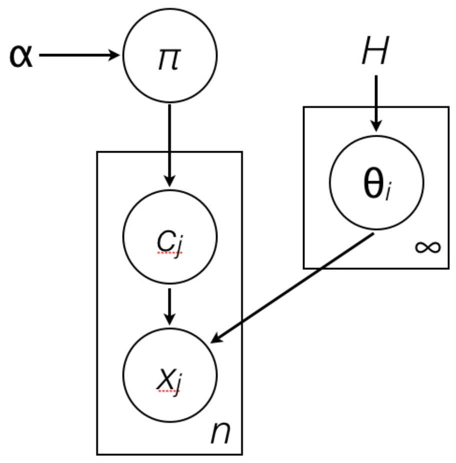


Figure 2-3: The graphical model of the GEM process.

for Griffiths, Engen, and McCloskey. More concretely,

$$\pi'_n|\alpha \sim \text{Beta}(1, \alpha)$$

$$\pi_n = \pi'_n \sum_{i=1}^{n-1}(1 - \pi'_i).$$

For Dirichlet process distributed $G \sim DP(\alpha, H)$, $G = \sum_{n=1}^{\infty} \pi_n \delta_{\theta_n}$ where $\delta_{\theta_n} = 1$ at $\theta_n \sim H$ and zero otherwise. Figure 2-3 shows the graphical model for n data points drawn from this process where c_j indexes the θ_i from which x_j was generated. Due to its simplicity, this is the exact method that was used to generate Figure 2-1.

It was mentioned in Section 2.2 that this formulation, while slightly more difficult to understand than the CRP, is far easier on which to perform inference. To make this algorithm clear, it is going to be described using the same terms as the CRP formulation (customers, tables, etc) but it is important to understand that this procedure was enabled by understanding a DP as Figure 2-3 and not as Figure 2-2. The reason the CRP terminology is still used is that it is extremely intuitive.

2.3 Statistical Learning Theory

In this section we review the necessary background of Statistical Learning Theory to build up to Structural Risk Minimization [Vapnik, 1995].

2.3.1 Classification

Classification is the problem of deciding on an output, $y \in \mathcal{Y}$, for a given input, $x \in \mathcal{X}$. The performance of a decision rule $f : \mathcal{X} \to \mathcal{Y}$ is measured using risk, $\mathcal{R}$, defined as

$$\mathcal{R}(f) = \int \mathcal{L}(y, f(x)) p(x,y) dx\ dy \tag{2.10}$$

where $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function and $p(x,y)$ is distribution which generated the data. For a class of decision rules, $\mathcal{F}$, the objective of classification is to select the decision rule which minimizes risk or, more formally,

$$f^* = \arg\min_{f \in \mathcal{F}} \mathcal{R}(f). \tag{2.11}$$

2.3.2 Empirical Risk Minimization

Commonly, the distribution $p(x,y)$ in Equation 2.10 is unknown, and we therefore are unable solve Equation 2.11 using Equation 2.10. Given a dataset $\mathcal{D} = \{(x^1,y^1),(x^2,y^2),\ldots,(x^N,y^N)\}$ where (x^n,y^n) is drawn i.i.d. from $p(x,y)$, Equation 2.10 can be approximated by empirical risk

$$R_{emp}(f;\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}(y^n, f(x^n)). \tag{2.12}$$

Empirical risk gives us an estimate of risk and therefore we can attempt to solve Equation 2.11 using the principle of Empirical Risk Minimization (ERM) [Vapnik, 1998] where

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_{emp}(f;\mathcal{D}). \tag{2.13}$$

2.3.3 Bounding the Risk of a Classifier

We can bound risk (Equation 2.10) using empirical risk (Equation 2.12) with a straightforward application of Hoeffding's inequality

$$\mathcal{R}(f) \leq R_{emp}(f; \mathcal{D}) + \sqrt{\frac{-\ln \delta}{2N}} \tag{2.14}$$

which holds with probability $1 - \delta$. Since Equation 2.13 is used to choose $f \in \mathcal{F}$, we need to ensure that $\mathcal{R}(f)$ is bounded for *all* $f \in \mathcal{F}$ (not just for a single f as Equation 2.14 guarantees). Bounds of this form ($\forall f \in \mathcal{F}$) can be written as

$$\mathcal{R}(f) \leq R_{emp}(f; \mathcal{D}) + \Omega(\mathcal{L} \circ \mathcal{F}, \mathcal{D}, \delta) \tag{2.15}$$

where Ω can be thought of as a complexity penalty on the size of $\mathcal{L} \circ \mathcal{F} = \{\mathcal{L}(\cdot, f(\cdot)) : f \in \mathcal{F}\}$ and the bound holds with probability $1 - \delta$.

The remainder of this section describes two specific bounds on $\mathcal{R}(f)$ $\forall f \in \mathcal{F}$, based on these two different measures of the size of $\mathcal{L} \circ \mathcal{F}$. These measures are Vapnik-Chervonenkis (VC) dimension and Rademacher complexity, although many additional bounds exist in the literature using, *e.g.*, maximum discrepancy [Bartlett et al., 2002a], local Rademacher complexity [Bartlett et al., 2002b], Gaussian complexity [Bartlett and Mendelson, 2003] and for loss sensitive bounds [Bartlett et al., 2006, Ben-David et al., 2012, Avila Pires et al., 2013].

Vapnik-Chervonenkis Dimension

A well-studied bound from Vapnik [1995] that takes the form of Equation 2.15 uses

$$\Omega(\mathcal{L} \circ \mathcal{F}, \mathcal{D}, \delta) = \frac{B - A}{2} \sqrt{4 \frac{h \left(\ln \left(\frac{2N}{h} \right) + 1 \right) - \ln(\delta/4)}{N}} \tag{2.16}$$

where $A \leq \mathcal{L} \circ \mathcal{F} \leq B$ and h is the Vapnik-Chervonenkis (VC) dimension of $\mathcal{L} \circ \mathcal{F}$ (see Vapnik [1998] for a thorough description of VC dimension).

Rademacher Complexity

In contrast to the well-studied bound from Vapnik [1995] which depends on the Vapnik-Chervonenkis (VC) dimension of $\mathcal{L} \circ \mathcal{F}$, Bartlett and Mendelson [2003] provided a bound based on the Rademacher complexity of $\mathcal{L} \circ \mathcal{F}$, a quantity that can be straightforwardly estimated from the dataset. Their bound, which takes the form of Equation 2.15, uses

$$\Omega(\mathcal{L} \circ \mathcal{F}, \mathcal{D}, \delta) = R^N(\mathcal{L} \circ \mathcal{F}) + \sqrt{\frac{-8\ln(2\delta)}{N}} \tag{2.17}$$

where $0 \leq \mathcal{L} \circ \mathcal{F} \leq 1$, and σ_n is a uniform random variable over $\{-1, +1\}$. $R^N(\mathcal{F})$, the Rademacher complexity of $\mathcal{F}$, can be estimated using

$$R^N(\mathcal{F}) = E_{\sigma_{1:N}, x^{1:N}} \left[\sup_{f \in \mathcal{F}} \frac{2}{N} \left| \sum_{n=1}^{N} \sigma_n f(x^n) \right| \right] \approx E_{\sigma_{1:N}} \left[\sup_{f \in \mathcal{F}} \frac{2}{N} \left| \sum_{n=1}^{N} \sigma_n f(x^n) \right| \, \middle| \, x^{1:N} \right], \tag{2.18}$$

Bartlett and Mendelson [2003] also showed that the error from estimating Rademacher complexity using the right hand side of Equation 2.18 is bounded with probability $1 - \delta$ by

$$R^N(\mathcal{F}) \leq \hat{R}^N(\mathcal{F}; \mathcal{D}) + \sqrt{\frac{-8\ln\delta}{N}}. \tag{2.19}$$

2.3.4 Structural Risk Minimization

As discussed in Vapnik [1995], the principle of ERM is only intended to be used with a large amount of data (relative to the size of $\mathcal{F}$). With a small data set, a small value of $R_{emp}(f; \mathcal{D})$ does not guarantee that $\mathcal{R}(f)$ will be small and therefore solving Equation 2.13 says little about the generalization of f. The principle of Structural Risk Minimization (SRM) states that since we cannot guarantee the generalization of ERM under limited data we should explicitly minimize the bound on generalization (Equation 2.15) by using a *structure* of function classes.

A structure of function classes is defined as a collection of nested subsets of functions $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_k \subseteq \cdots$ where $S_k = \mathcal{L} \circ \mathcal{F}_k$. For example, a structure of radial basis functions created by placing increasing limits on the magnitude of the basis functions. SRM then treats the capacity of $\mathcal{L} \circ \mathcal{F}_k$ as a controlling variable and minimizes Equation 2.15 for each S_k such that

$$\hat{k} = \arg\min_{k} R_{emp}(\hat{f}_k; \mathcal{D}) + \Omega(\mathcal{L} \circ \mathcal{F}_k, \mathcal{D}, \delta), \quad \hat{f}_k = \arg\min_{f \in \mathcal{F}_k} R_{emp}(f; \mathcal{D}). \tag{2.20}$$

To solve Equation 2.20 we must solve both equations jointly. One can imagine enumerating k, finding $\hat{f}_k$ for each k, and choosing the corresponding $\hat{f}_k$ which minimizes $R_{emp}(\hat{f}_k; \mathcal{D}) + \Omega(\mathcal{L} \circ \mathcal{F}_k, \mathcal{D}, \delta)$.

Chapter 3

A Bayesian Nonparametric Approach to Modeling Motion Patterns

In this section, we show that Bayesian nonparametric approaches to modeling motion patterns are well-suited for poorly understood environments because they let the data determine the sophistication of the model—we no longer need to specify which parameters are important. Moreover, the Bayesian aspect helps the model generalize to unseen data and make inferences from noisy data. Specifically, will we show we can model a target's motion patterns with a Dirichlet process mixture model over Gaussian process target trajectories (DPGP). Using this nonparametric model boosts learning rates by generalizing quickly from small amounts of data but continuing to increase in sophistication as more trajectories are observed. We applied this DPGP model to applications tracking a single target whose current position was always observed (imagine having a GPS tracker on the target but not knowing where the target will go).

3.1 Mobility Model

We represent a target's trajectory t^i as a set of xy-locations $\{(x_1^i, y_1^i), (x_n^i, y_n^i), \ldots, (x_{L^i}^i, y_{L^i}^i)\}$, where L^i is the length of trajectory t^i. Depending on how the trajectory data is collected, these locations may come at irregular intervals: for example, the distance between (x_t^i, y_t^i) and (x_{t+1}^i, y_{t+1}^i) may not be the same as the distance between (x_{t+1}^i, y_{t+1}^i) and (x_{t+2}^i, y_{t+2}^i). Trajectories may also be of different lengths both because some trajectories may be physically longer than others and because some trajectories may have a larger number of observed locations along the route.

We use time-stamped GPS coordinates of greater Boston taxis from the CarTel project as our motivating dataset.[1] Figure 3-1 plots some of the trajectories (red points) on a map of Boston[2], emphasizing the discrete nature of our observations. One sample trajectory is highlighted in green, showing how the discrete observations are irregularly spaced along the trajectory. Working with these types of trajectories is one of the challenges of this dataset, which we address by using Gaussian processes to learn a trajectory model.

The technical details of our motion model are described in Section 3.1.1, but we first outline the two key elements of our motion model and describe how they are combined. Specifically, each motion model is a *mixture* of *motion patterns*. A motion pattern represents a class of similar trajectories. A mixture model over different motion patterns defines the probability of each particular motion pattern.

3.1.1 Dirichlet Process Gaussian Process Motion Model

Motion Patterns

Many ways exist to describe a class of trajectories: for example, one could use a set of piecewise linear segments or a spline. We define a *motion pattern* as a mapping from locations (x, y) to a distribution over trajectory derivatives $(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$ indicating

[1]CarTel project, `http://cartel.csail.mit.edu`. The data was down-sampled to a rate of 1 reading per minute and pre-processed into trajectories. If the car stayed in the same place for five minutes, this indicated the end of a trajectory.

[2]`http://maps.google.com`

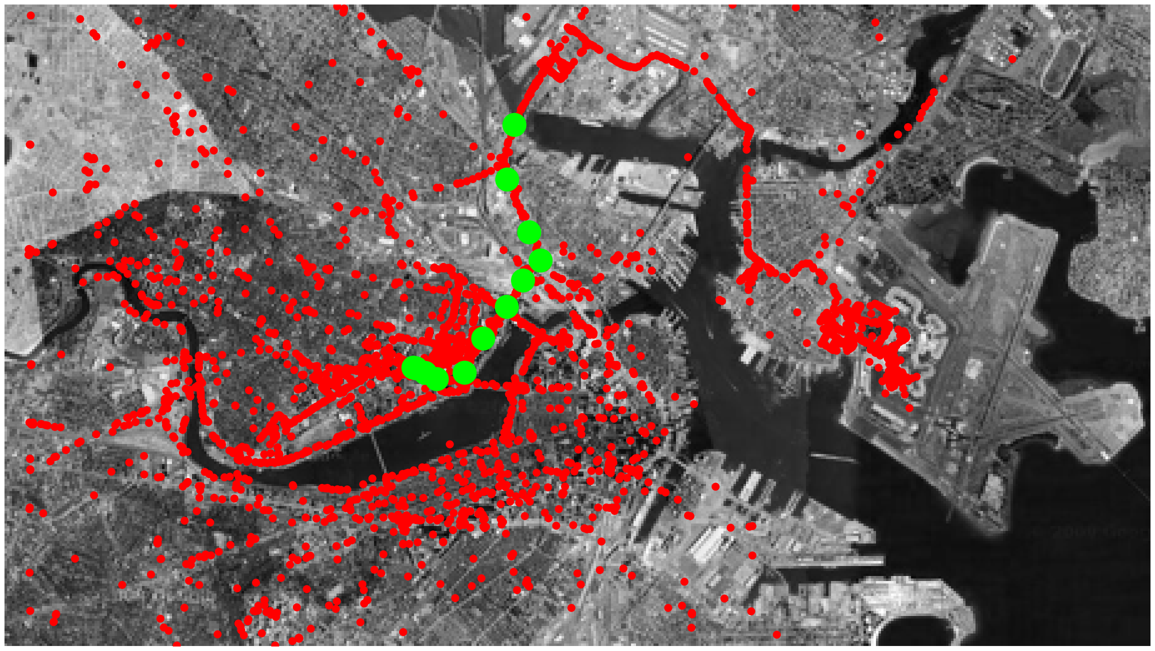


Figure 3-1: A small set of the raw GPS data points (red) and a single trajectory (green) used to learn our model.

the agent's future motion.[3] Thus, a motion pattern is a flow-field of trajectory derivatives in x-y space. Modeling motion patterns as flow fields rather than single paths allows us to group target trajectories sharing key characteristics: for example, a single motion pattern can capture all the paths that a target might take from different starting points to a single ending location. Using trajectory derivatives also makes the representation blind to the lengths and discretizations of the trajectories.

We use a Gaussian process (GP) to model the mapping of positions to velocities. The GP allows us to learn a distribution over trajectory derivatives (velocities) at each location (details in Section 3.1.1). Given the target's current position (x_t, y_t) and a trajectory derivative $(\frac{\Delta x_t}{\Delta t}, \frac{\Delta y_t}{\Delta t})$, its predicted next position (x_{t+1}, y_{t+1}) is given by

$$x_{t+1} = x_t + \frac{\Delta x_t}{\Delta t}\Delta t, \qquad y_{t+1} = y_t + \frac{\Delta y_t}{\Delta t}\Delta t.$$

Thus, the trajectories are easily generated by integrating the trajectory derivatives.

[3]The choice of Δt determines the scales we can expect to predict the target's next position well, making the trajectory derivative more useful than instantaneous velocity.

Mixtures of Motion Patterns

We expect to encounter trajectories with qualitatively different behaviors and using trajectory-derivative flow fields as motion patterns helps group together trajectories with certain characteristics. For example, different trajectories may share some segments but then branch off in different directions. Returning to the CarTel taxi dataset, we see that scenarios with overlapping paths are common. Figure 3-2 shows just one example of two routes that share a common corridor, but the red trajectory travels east and the green trajectory travels north. These motion patterns are not well modeled by traditional techniques such as Markov chain models that simply try to predict a target's future location based on its current position (and ignore its previous history), nor can they be modeled by a single trajectory-derivative flow field. We address this issue by using mixture models over motion patterns.

Formally, a finite mixture model with M motion patterns $\{b_1, b_2, \ldots, b_M\}$ first assigns a prior probability for each pattern $\{p(b_1), p(b_2), \ldots, p(b_M)\}$. Given these prior probabilities, the probability of the i^{th} observed trajectory t^i under the mixture model[4] is

$$p(t^i) = \sum_{j}^{M} p(b_j) p(t^i|\theta_j) \tag{3.1}$$

where θ_j contains the parameters for motion pattern b_j.

The primary complication with a simple finite mixture model is that M is not known in advance, and may need to grow as more data is observed. In Section 3.1.1, we detail how we use a Dirichlet process (DP) (Section 2.2) mixture model to create an infinite mixture of motion patterns. An important property of the DP model is that it places a prior over an infinite number of motion patterns such that the prior probabilities $p(b_1), p(b_2), p(b_3), \ldots$ still sum to one, such that the probability of a trajectory is

$$p(t^i) = \sum_{j}^{\infty} p(b_j) p(t^i|\theta_j). \tag{3.2}$$

These probabilities, $p(b_j)$, and the number of different motion patterns in a given

[4]Note that throughout the section a t with a superscript, such as t^i, refers to a trajectory and a t without a superscript is a time value.

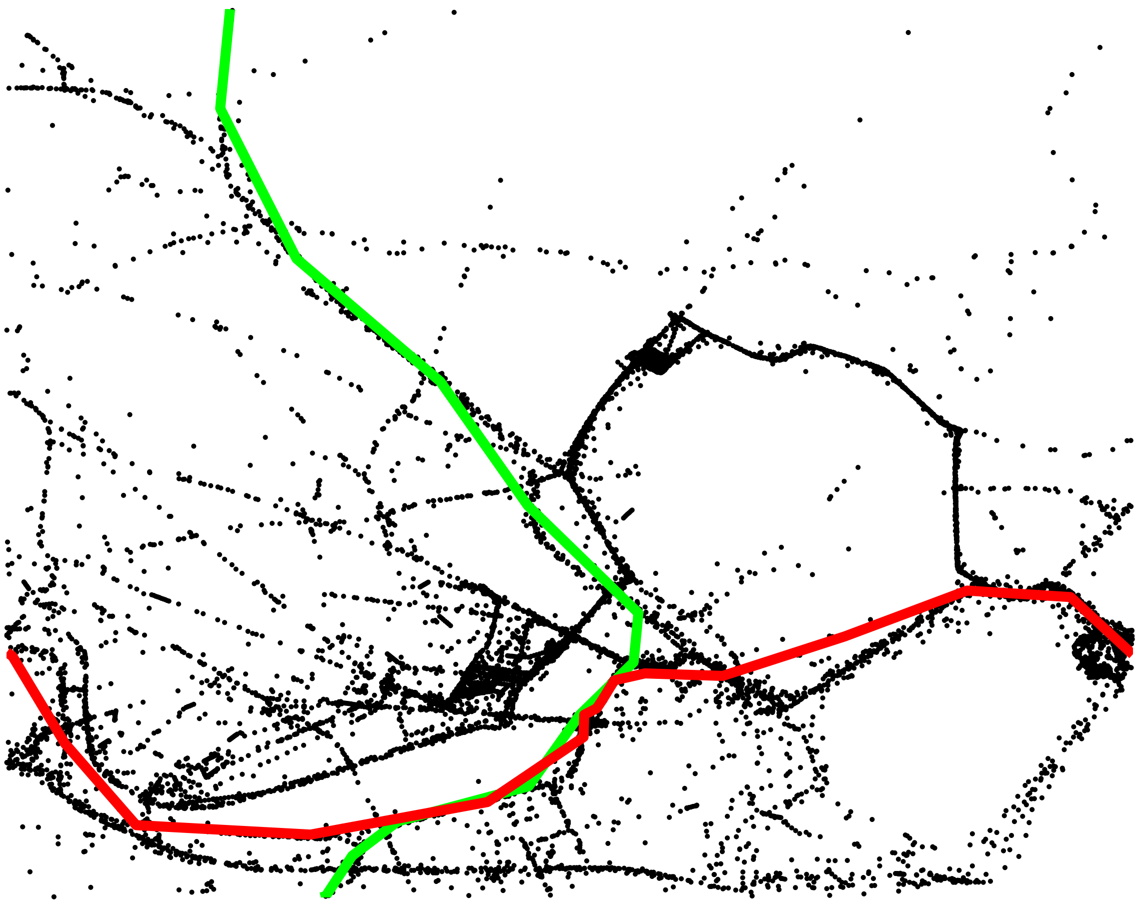


Figure 3-2: An example of two trajectories that share a road segment. The red trajectory travels east and the green trajectory travels north. The Markov model cannot distinguish the two trajectories once they cross, but the DP model classifies them as two different paths.

dataset are determined during at inference time.

We define the *motion model* as a mixture of weighted motion patterns. Each motion pattern is weighted by its probability (Section 3.1.1) and place a Dirichlet process prior over mixture weights (Section 3.1.1).[5]

Under our DPGP model, the prior probability of motion pattern b_j is given by its DP mixture weight $p(b_j)$. The posterior probability of b_j given a target trajectory t^i is proportional to $p(b_j) \cdot l(b_j; t^i)$, where $l(b_j; t^i)$ describes the likelihood of motion

[5]This model is similar to models described by Rasmussen and Ghahramani [2002] and Meeds and Osindero [2006]; however, unlike these previous works, our goal is to cluster trajectories of varying lengths, not just partition single points.

pattern b_j under trajectory t^i:

$$l(b_j; t^i) = \prod_t^{L^i} p\left(\frac{\Delta x_t}{\Delta t}\middle| x_{1:t}^i, y_{1:t}^i, \{t^k : z_k = j\}, \theta_{x,j}^{GP}\right) \cdot \prod_t^{L^i} p\left(\frac{\Delta y_t}{\Delta t}\middle| x_{1:t}^i, y_{1:t}^i, \{t^k : z_k = j\}, \theta_{y,j}^{GP}\right) \quad (3.3)$$

where z_k indicates the motion pattern to which trajectory t^k is assigned, and $\theta_{x,j}^{GP}$ and $\theta_{y,j}^{GP}$ are the hyperparameters of the Gaussian process for motion pattern b_j. Equation 3.3 may be applied to trajectories with differing numbers of observations or even trajectories that are only partially complete, which is particularly important when we wish to determine a target's motion pattern given only a few observations.

Gaussian Process Motion Patterns

Observations from a target's trajectory represent a continuous path through space. The Gaussian process places a distribution over functions [Rasmussen and Williams, 2005], serving as a non-parametric form of interpolation. Gaussian process models are extremely robust to unaligned, noisy measurements and are well-suited for modeling the continuous paths underlying our non-uniformly sampled time-series samples of the target's locations.

The Gaussian process for a motion pattern that models a trajectory's derivative is specified by a set of mean and covariance functions. Specifically, given an input (x, y) location, the GP model for the motion pattern predicts the trajectory derivatives $(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$ at that location. We describe the mean trajectory-derivative functions as $E[\frac{\Delta x}{\Delta t}] = \mu_x(x, y)$ and $E[\frac{\Delta y}{\Delta t}] = \mu_y(x, y)$, and implicitly set both of them to initially be zero everywhere (for all x and y) by our choice of parameterization of the covariance function. This encodes the prior bias that, without any additional knowledge, we expect the target to stay in the same place. Zero-mean GP priors also simplify computations. The model assumes that trajectory derivatives in the x-direction and y-direction are independent; while a more sophisticated model could be used to model these trajectory derivatives jointly [Boyle and Frean, 2005], we found that our simple approach had good empirical performance and scaled well to larger datasets.

We denote the covariance function in the x-direction as $K_x(x, y, x', y')$, which describes the correlations between trajectory derivatives at two points (x, y) and (x', y'). Given locations $(x_1, .., x_k, y_1, .., y_k)$, the corresponding trajectory derivatives $(\frac{\Delta x_1}{\Delta t}, .., \frac{\Delta x_k}{\Delta t})$ are jointly distributed according to a Gaussian with mean $\{\mu_x(x_1, y_1), ..., \mu_x(x_k, y_k)\}$ and covariance Σ, where the $\Sigma_{ij} = K(x_i, y_i, x_j, y_j)$. We use the standard squared exponential covariance function

$$K_x(x, y, x', y') = \sigma_x^2 \exp\left(-\frac{(x - x')^2}{2{w_x}^2} - \frac{(y - y')^2}{2{w_y}^2}\right) + \sigma_n^2 \delta(x, y, x', y') \tag{3.4}$$

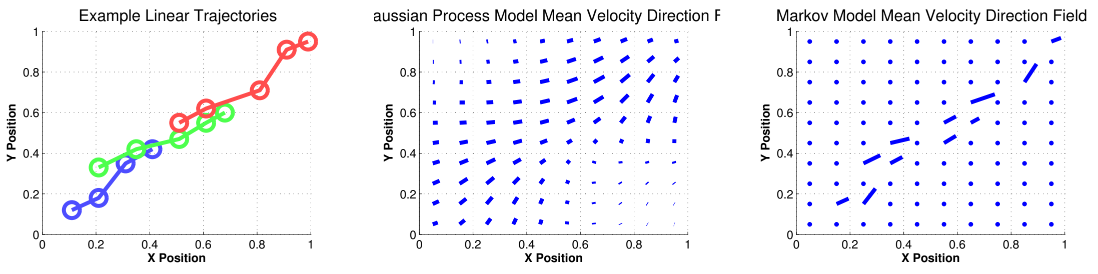
where $\delta(x, y, x', y') = 1$ if $x = x'$ and $y = y'$ and zero otherwise. The length-scale parameters w_x and w_y normalize for the scale of the data. The σ_n-term represents within-point variation (e.g., due to noisy measurements); the ratio of σ_n and σ_x weights the relative effects of noise and influences from nearby points. We use $\theta^{GP}_{x,j}$ to refer to the set of hyperparameters σ_x, σ_n, w_x, and w_y associated with motion pattern b_j (each motion pattern has a separate set of hyperparameters).[6] We chose a covariance function of the form of Equation 3.4 due to its straightforward and intuitive connection with driver trajectories and resulting high performance.

For a GP over trajectory derivatives trained with tuples $(x_k, y_k, \frac{\Delta x_k}{\Delta t})$, the predictive distribution over the trajectory derivative $\frac{\Delta x}{\Delta t}^*$ for a new point (x^*, y^*) is given by

$$\begin{aligned} \mu_{\frac{\Delta x}{\Delta t}^*} &= K_x(x^*,y^*,X,Y)K_x(X,Y,X,Y)^{-1}\frac{\Delta X}{\Delta t} \\ \sigma^2_{\frac{\Delta x}{\Delta t}^*} &= K_x(x^*,y^*,X,Y)K_x(X,Y,X,Y)^{-1}K_x(X,Y,x^*,y^*) \end{aligned} \tag{3.5}$$

where the expression $K_x(X, Y, X, Y)$ is shorthand for the covariance matrix Σ with terms $\Sigma_{ij} = K_x(x_i, y_i, x_j, y_j)$. The equations for $\frac{\Delta y}{\Delta t}^*$ are equivalent to those above, using the covariance K_y.

[6]We described the kernel for two dimensions, but it can be easily generalized to more.



(a) Example Trajectories (b) GP Mean Velocity Field (c) MM Mean Velocity Field

Figure 3-3: Velocity fields learned by a GP and a Markov model from three trajectories of an approximately linear motion pattern. The GP generalizes quickly from the irregularly observed trajectories, whereas the discretization in the Markov model slows down generalization.

Estimating Future Trajectories

As summarized in Equation 3.5, our Gaussian process motion model places a Gaussian distribution over trajectory derivatives $(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$ for every location (x, y). If the target's location is always known, we only need to predict the target's position one-step into the future to track it: even if it goes in an unexpected direction, we will know that a rare event has occurred and can plan accordingly. However, the target's position is not always known—for example, if it can only be observed within the agent's camera radius. In this case, the agent must be able to infer where the target might be multiple steps into the future to intercept it again from knowledge about where the target was located in the past.

In [Joseph et al., 2010] we used a simple approach to sample a target's possible trajectory multiple steps into the future: starting with the target's current location (x_1, y_1), we sampled a trajectory derivative $(\frac{\Delta x_1}{\Delta t_1}, \frac{\Delta y_1}{\Delta t_1})$ to get a next location (x_2, y_2). Then starting from (x_2, y_2), we sampled a trajectory derivative $(\frac{\Delta x_2}{\Delta t_2}, \frac{\Delta y_2}{\Delta t_2})$ to get a next location (x_3, y_3). We repeated this process until we had sampled a trajectory of length L. The entire sampling procedure was repeated from the current location (x_1, y_1) multiple times to get samples of the target's future trajectories.

While samples drawn from this procedure are an accurate representation of the posterior over trajectories, sampling N trajectories of where the target may be L steps in the future requires NL queries to the Gaussian process. It also does not take advantage of the unimodal, Gaussian distributions being used to model the trajectory derivatives. Key to efficiently predicting future trajectories is applying an approximation of Girard et al. [2003] and Deisenroth et al. [2009] that provides a fast, analytic approach of approximating the output of a Gaussian process given a distribution over the input distribution. In our case, our Gaussian process motion model over trajectory derivatives gives us a Gaussian distribution over possible target next-locations at each time step. The approximation of Girard et al. [2003] and Deisenroth et al. [2009] allows us to string these distributions together: we input a distribution of where the target may be at time t and a distribution of trajectory derivatives to get a distribution of where the target may be at time $t+1$. By being able to estimate the target's future trajectories analytically, we reduce the computations required—only L queries to the Gaussian process are needed to predict the target's location L steps into the future—and avoid the variance introduced by sampling future trajectories.

Comparison with a Markov chain model Instead of using a Gaussian process—which defines a distribution over velocities in a continuous state space—we could imagine a model that discretizes the state and velocity space into bins and learns a transition model between state-velocity bins. We call this alternative the "Markov model" because predictions about the target's next position depend only on the target's current position and velocity, not its past history.

A key question when trying to train such a Markov model is the appropriate level of discretization for the state space. In Figure 3-3, we consider modeling a motion pattern that consists of approximately linear trajectories observed at irregular intervals. By modeling the velocity field over the continuous space, the GP is able to quickly generalize the velocity field over region, whereas the Markov model has gaps induced by its discretization. These gaps could be filled by a coarser discretization; however, the modeling would also be coarser. The GP automatically adjusts the generalization as more data arrive.

Dirichlet Process Mixture Weights

Although a single Gaussian process can robustly model the variation within many closely related trajectories, it is not able to capture differences resulting from targets with different destinations or different preferred routes. To model qualitatively different motion patterns, we can represent the distribution over behaviors as a mixture of Gaussian processes. However, we do not know ahead of time how many behaviors are sufficient for the model. We use a Dirichlet process to allow for new behaviors to be added as they are observed.

We place a Dirichlet process (Section 2.2) prior over the number of motion patterns. If z_i indicates the motion pattern to which trajectory t^i is assigned, the prior probability that target trajectory t^i belongs to an existing motion pattern b_j is

$$p(z_i = j | z_{-i}, \alpha) = \frac{n_j}{N-1+\alpha}, \tag{3.6}$$

where z_{-i} refers to the motion pattern assignments for the remaining trajectories, α is the concentration parameter of the Dirichlet process, n_j is the number of trajectories assigned to motion pattern b_j, and N is the total number of observed trajectories. The probability that trajectory t^i exhibits a new motion pattern is

$$p(z_i = M+1 | z_{-i}, \alpha) = \frac{\alpha}{N-1+\alpha}. \tag{3.7}$$

where M is the number of observed motion patterns.

Equation 3.7 implies that the number of motion patterns can grow as more data is obtained. This property is key to realistically modeling targets: the more interception and tracking tasks we perform, the more varieties of target motion patterns we expect to encounter. Figure 3-4 shows how the number of motion patterns grows (under our model) as new trajectories are observed for the actual dataset of greater Boston taxi routes (described in Section 3.2). We show in Section 3.2 that we can efficiently plan even when the number of actively observed motion patterns is unknown; moreover, this flexibility yields significantly improved results in the performance of the planner.

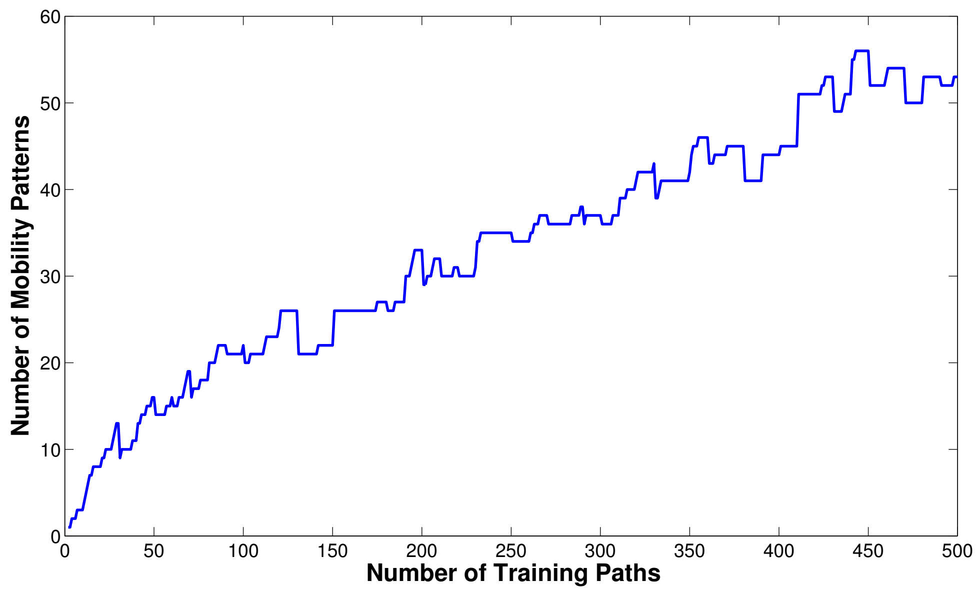


Figure 3-4: As expected, the number of motion patterns in the taxi dataset increases as more trajectories are added.

DP Trajectory Classifying Example Just as the Gaussian process in Section 3.1.1 allows us to model motion patterns without specifying a discretization, the Dirichlet process mixture model allows us to model mixtures of motion patterns without specifying the number of motion patterns. One could, of course, simply search over the number of motion patterns: we could train models with different numbers of patterns, examine how well each mixture model explains the data, and finally choose the best one. However, as we see below, this search requires much more computation time than using a Dirichlet process to automatically determine the number of patterns, with similar performance.

We compare the DPGP to a set of finite mixture models that also use Gaussian processes to model motion patterns (that is, the finite mixture model first described in Equation 3.2). We consider the helicopter-based tracking scenario for a data set of taxi trajectories. Each model was trained on a batch of 200 trajectories using five different initializations. We tested tracking performance on a set of 15 held-out test

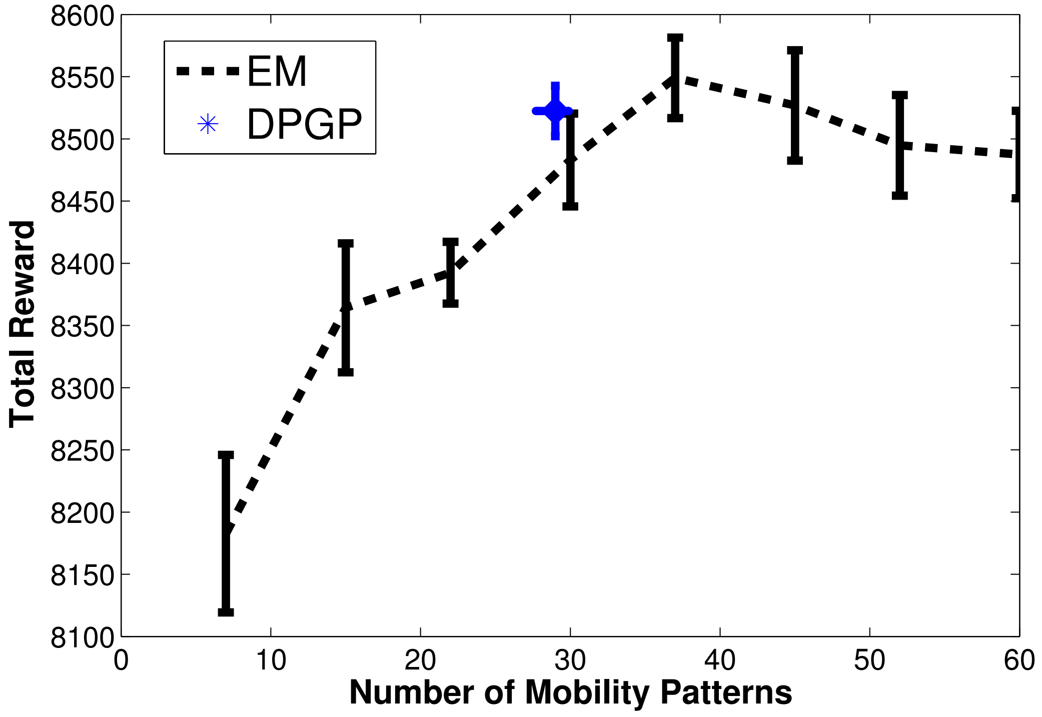


Figure 3-5: Performance on 15 held-out test trajectories vs. model size for a variety of finite models (black) and the DPGP (blue) trained on 200 trajectories. The error bars represent the standard deviation of the reward from five runs. Note the inferred DPGP model has model size error bars also due to variation in the estimated model size for each run.

trajectories. None of the models were updated during the testing phase.

The results in Figure 3-5 show that while the finite GP-based models perform well overall, our DPGP model has nearly the best performance *without having to perform a search over the finite model space*. This last point is important, not only because a search over finite models would require more computation but also because the search requires us to choose a regularization criterion to avoid over-fitting. Standard criteria, such as the Bayesian information criterion [Raftery, 1986] cannot be applied in this context because the GP contains an unbounded number of parameters; thus we must choose from various cross-validation or bootstrap procedures. The DPGP provides a principled, simple-to-use regularization criterion within its model.

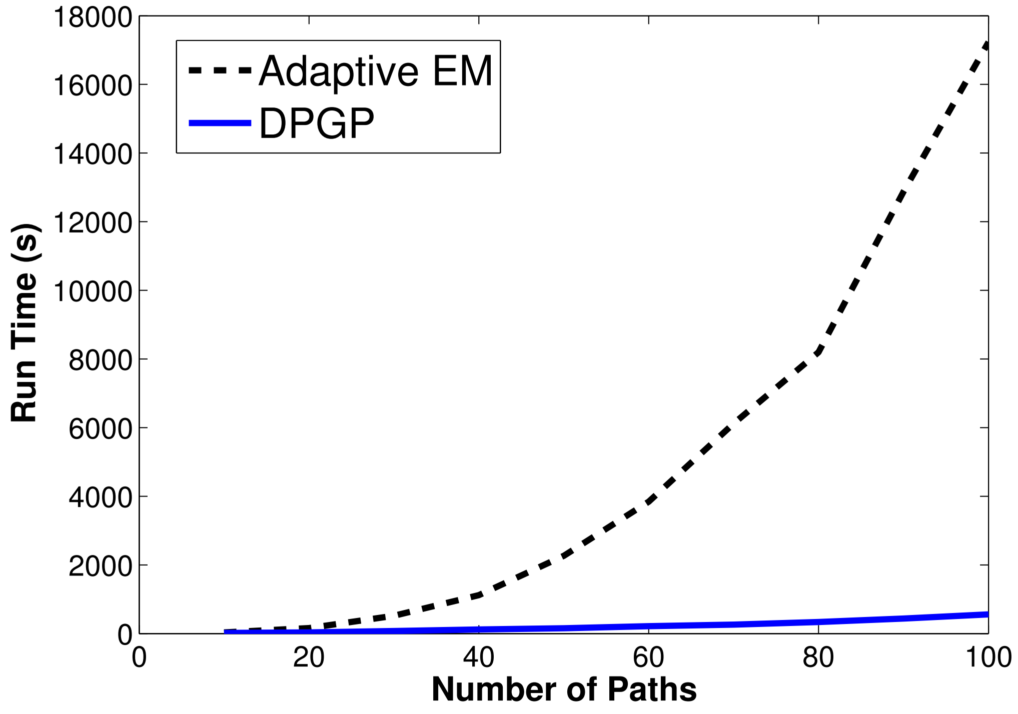


Figure 3-6: Run time vs. number of paths for adaptive EM and our DPGP model.

3.2 Interception and Tracking with Full Information

Searching in the space of finite models is especially computationally expensive when the data arrives online and the number of clusters are expected to grow with time. (The DP can update the number of clusters incrementally.) To gain insight into the extra computation cost of this search process we implemented EM where every 10 paths we search over models sizes that are within five clusters of the current model. Figure 3-6 shows run time as the number of training paths increase for our DPGP model and this adaptive EM technique. The running time grows exponentially longer for EM with model search compared to the DPGP.

We first consider the case in which our agent has access to the target's current position but needs to be able to predict its future position to track it effectively. We call this the "full information" case because this scenario implies that the agent has access to sensors covering the environment such that the target's current state is always known (up to time discretization). For example, we may be given location information from a dense sensor network. In this section, we formalize the tracking problem and describe the process of training a motion model for this full-information tracking task. We next provide results for our tracking problem applied to two targets with completely different motion models, one synthetic and one built from a real-world dataset. In Section 4, we will relax the assumption of a dense sensor network, and show how to extend our approach to target interception given information from a sparse sensor network.

3.2.1 Tracking Problem Formulation

Since the target's current position is known at every time step, we can formalize the scenario as a Markov decision process (MDP), a common tool for autonomous decision making. An MDP is defined by a set of states, a set of actions, a transition function, and a reward function. Here, the state is the joint position of our agent and the target $(x^a, y^a, x^{target}, y^{target})$. Given an action and our agent's current position (x^a_t, y^a_t), we assume that our agent's next position (x^a_{t+1}, y^a_{t+1}) is deterministic and known. In contrast, the target's transitions are stochastic over the continuous space; we can only place a distribution over the target's next position $(x^{target}_{t+1}, y^{target}_{t+1})$ based on our motion model. At each step, our agent incurs some small cost for moving, and receives a large positive reward each time it shares a grid cell with the target. For this type of interception and tracking scenario the policy is fairly insensitive to the reward values. Given an MDP, we can find the optimal policy using standard forward search techniques [Puterman, 1994].

3.2.2 Model Inference

Given a set of target trajectories, we can train the DPGP model from Section 3.1.1 and use it to make predictions about future trajectories. Since exact inference over the space of DPs and GPs is intractable, we describe a process for drawing samples from the posterior over motion models. These samples are then used by our agent for planning.

Training the Model

Our model contains two sets of parameters—the DP mixture weights $p(b_j)$, the motion pattern assignments z_i, and the DP hyperparameter α—the GP hyperparameters $\theta^{GP}_{x,j}, \theta^{GP}_{y,j}$ and the trajectories assigned to each motion pattern cluster. Following the work of Rasmussen and Ghahramani [2002] and Rasmussen [2000], learning the model involves Gibbs sampling the parameters (see Algorithm 1).

We first resample each z_i in turn, using the exchangeability properties of the DP and GP to model the target trajectory t^i as the most recently observed target. The probability that the trajectory t^i will be assigned to an instantiated motion pattern is

$$p(z_i = j | t^i, \alpha, \theta^{GP}_{x,j}, \theta^{GP}_{y,j}) \propto l(b_j; t^i) \left(\frac{n_j}{N - 1 + \alpha} \right) \tag{3.8}$$

where $l(b_j; t^i)$ is the likelihood of motion pattern b_j from Equation 3.3 and n_j is the number of trajectories currently assigned to motion pattern b_j. The probability that the trajectory t^i belongs to a new motion pattern is given by

$$p(z_i = M + 1 | t^i, \alpha) \propto \int l(b_{M+1}; t^i) d\theta^{GP}_{x,M+1} d\theta^{GP}_{y,M+1} \left(\frac{\alpha}{N - 1 + \alpha} \right), \tag{3.9}$$

and we use Monte Carlo integration [Bishop, 2006] to approximate the integral. The likelihood from Equation 3.8 also must be approximated for popular motion patterns, as the computations in Equation 3.5 are cubic in the cluster size n_j. Similar to Rasmussen and Williams [2005], we approximate the likelihood for these larger clusters using the N_{max} trajectories that are closest to the trajectory t^i.[7]

The DP concentration hyperparameter α is resampled using standard Gibbs sampling techniques [Rasmussen, 2000]. The GP length-scale and variance hyperparameters are more difficult to resample, so we leverage the fact that their posteriors are

[7]We tested the validity of this approximation by comparing approximations in which only the nearest points to the true likelihood were used and found no practical difference when discarding 75% of trajectories for large clusters.

Algorithm 1: Motion Model Inference

```
1: for sweep = 1 to # of sweeps do
2:    for each motion pattern $b_j$ do
3:       Draw the GP hyperparameters $\theta^{GP}_{x,j}, \theta^{GP}_{y,j}$
4:    end for
5:    Draw the DP hyperparameter $\alpha$
6:    for each trajectory $t^i$ do
7:       Draw $z_i$ using equations 3.8 and 3.9
8:    end for
9: end for
```

extremely peaked and instead always set them to their maximum likelihood values (using gradient ascent). In applications where the posteriors are less peaked, hybrid Monte Carlo techniques may be used [Duane et al., 1987].

Classification and Prediction with New Trajectories

The motion model from Algorithm 1 can now be used to predict a target's future locations, given a partial trajectory t^i. We first apply equations 3.8 and 3.9 to compute the relative probability of it belonging to each motion pattern b_j. Equation 3.3 is used to compute the likelihoods. Just as in Section 3.2.2 where we trained the model using complete target trajectories, the partial trajectory may contain any number of points. We can use the same equations 3.8 and 3.9 to determine the most likely motion patterns for the partial trajectory.

For each likely pattern b_j, we first compute the expected trajectory derivatives $(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})_j$ conditioned on GP parameters $(\theta^{GP}_{x,j}, \theta^{GP}_{y,j})$ (Equation 3.5). The expected trajectory derivative is a weighted average over all the conditional derivatives[8]

$$\sum_j p(b_j) \left(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t} \right)_j .$$

We apply this expected trajectory derivative to the target's most recent location to predict where it will be in the future.

[8]In practice, we found that the motion pattern likelihoods were highly peaked. In this situation, it was sufficient to only consider the maximum likelihood motion pattern when predicting the future locations in partial trajectories.

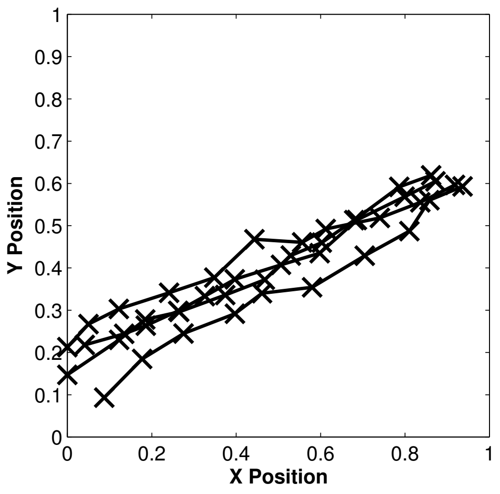


Figure 3-7: Several trajectory samples from the CORRIDOR scenario, where targets roughly following a straight line

3.2.3 Results

In this section we describe our results on two example scenarios. The first is a synthetic single-trajectory scenario where the agent must intercept and track 50 targets, one after the other. The second scenario is a (simulated) helicopter-based tracking scenario in which the targets are cars whose paths from the CarTel dataset. In both cases, we tested our models in an online fashion: initially our agent had no experience with the target; after each episode, the target's full trajectory was incorporated into the motion model.

We compare our DPGP motion model to a Markov model that projects positions and velocities to a discretized grid and uses the trajectory data to learn target transition probabilities between grid cells. The Markov model predicts a target's next grid cell using the transition probabilities stored at the grid cell closest to the target's current position and velocity. In contrast to the Markov model, which ignores trajectory history, the DPGP model considers the entire observed portion of the trajectory when predicting both the target's motion pattern and future trajectory.

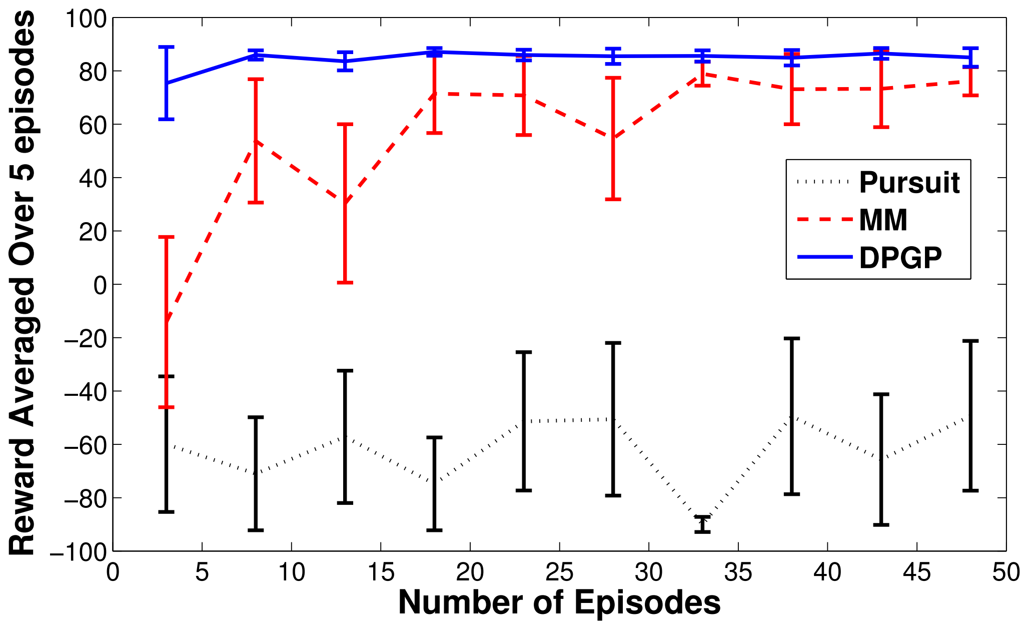
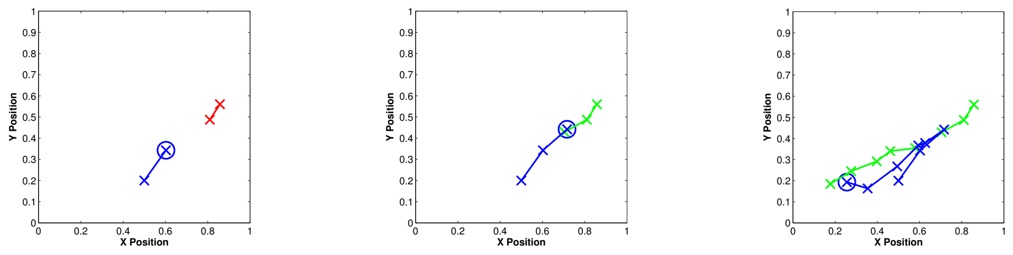


Figure 3-8: Sliding window average of per-episode rewards achieved by different models on the CORRIDOR scenario. Error bars show the 95% confidence interval of the mean from five repeated runs.

Results on a Simple Synthetic Example

We first apply our approach to a simple example involving a target following a straight line with occasional deviations (for example, walking along a puddle-covered road). The agent receives a reward of -10 for every time step until it intercepts the target, whereupon it receives a reward of +100. The agent's task involved intercepting and tracking 50 targets one after the other. We call this the CORRIDOR scenario. Figure 3-7 shows several trajectories from this example.

Figure 3-8 shows the results for five repetitions of this set of tasks. For comparison, we plot the results of both the Markov model and a naive pursuit approach that moves the agent to the target's most recent position. Overall, we see that while the agent planning with the Markov models with various initializations eventually reaches the same level of performance as the agent using the Gaussian process, the Gaussian process motion model learns faster from the data. Figure 3-9 shows an example planning sequence derived using the Gaussian process motion model in which the agent intercepts the target.



(a) $t = 2$ (b) $t = 3$ (c) $t = 9$

Figure 3-9: A planning episode for a single path in the CORRIDOR scenario. Agent positions are shown in blue and untagged target positions are shown in dashed red (before they are tagged) and dashed green (after they are tagged). The small blue circle around the agent signifies the tagging range.

While this is a simple and easy example, we note that the DPGP still outperforms the other models. The DPGP learns the model almost instantaneously, but the Markov model requires approximately 50 trials before matching the performance of the DPGP.

Results on a Helicopter-based Tracking Scenario

Next, we tested our approach on a helicopter-based target-tracking scenario. To model the helicopter and its rewards, we place a 20×20 grid over a city (an area of approximately 10 square miles) and represent the helicopter's state with the closest grid cell. At each time step, the helicopter can stay in place, move one cell, or move two cells. These actions result in rewards of 0, -1, and -2, respectively. The helicopter also receives a reward of 10 for each time step it shares a grid cell with the target car. While a real "chase" scenario would have many more complexities, this simplified tracking task allows us to show empirically that our model, initially trained on likelihood-based criteria, also performs well on a planning problem based on real data[9].

We tested both our DPGP and the Markov model on 500 trajectories taken from the CarTel dataset of time-stamped GPS coordinates of greater Boston area taxis.

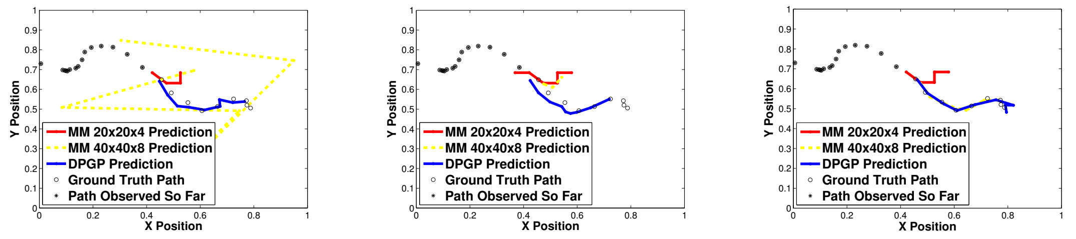
[9]Likelihood-based methods try to explain the data well, while the goal of the planning problem is to maximize rewards. A model that best explains the data is not guaranteed to be the best model for planning.

Training trajectories were randomly drawn from this set of 500 without replacement until all 500 trajectories were incorporated. The Markov model was initialized with a uniform prior, and its transition probabilities were updated as new trajectories arrived. To assess the effect of discretization granularity on the Markov model, we evaluated Markov models with different position and velocity resolutions. The x and y-positions were discretized on a 20×20, 40×40, or a 60×60 grid (the helicopter's discretization never changed). Velocity was either discretized into four or eight states. The models with finer discretizations were more expressive but require more data to train effectively.

After each trajectory was completed, our DPGP driver model was updated using Algorithm 1. Each update was initialized with the most recently sampled model. Since a full update required significant computation, new trajectories were initially clustered with their most likely motion pattern (which could have been a new pattern) using equations 3.8 and 3.9.

Every 10 new trajectories, a complete set of 5 Gibbs sweeps (Algorithm 1) were run to update the model parameters and trajectory assignments (we found that samples generally stopped changing after the first 2 sweeps). The noise parameter σ_n in Equation 3.4 was fit from the current trajectory set. While the DPGP model required more computation than the Markov model (about 10 times slower), it could still incorporate a new set of samples in minutes, an update rate fast enough for a real scenario where the model may be updated several times a day. The planning time was nearly instantaneous for both the DPGP and the Markov driver models.

We first carried out a series of experiments to evaluate the quality of our models. Example predictions of the DPGP and Markov models are seen in Figure 3-10. The solid circles show a partial trajectory; the open circles show the true continuation of the trajectory. The cyan, red, and blue curves show the continuations predicted by the DPGP model and two Markov models. With only 100 training trajectories, none of the models predict the full path, but the DPGP is close while the other models are completely lost. As more training data is added, the DPGP and the finer-grained Markov model match the true trajectory, while the simpler Markov model is not



(a) 100 paths (b) 300 paths (c) 500 paths

Figure 3-10: Predictions given a partial path for the DPGP and two Markov models for various amounts of training data. Trajectories were drawn randomly from the full dataset without replacement.

flexible enough to fit the data.

As the goal of our model is to predict the motion of mobile agents within a planner, we compared the performance of planners using the DPGP and Markov models, as well as a naive pursuit approach that simply assumed the vehicle's position at time $t+1$ would be the same as its location at time t. We also evaluated a simple k-nearest neighbor technique that, given an (x, y) point, simply searched the training set of trajectories for nearby (x, y) points and interpolated the trajectory derivatives $\frac{\Delta x}{\Delta t}$ and $\frac{\Delta y}{\Delta t}$ from the trajectory derivatives of nearby training points.[10] Finally, we evaluated a GP model that was fit to only the current trajectory and ignored all past training data. This single GP model ensured that the previous trajectories were important for making predictions about the current trajectory, that is, the current trajectory could not be well-predicted based on its own velocities.

Figure 3-11 shows the cumulative *difference* of total reward between all the approaches and naive pursuit method. The k-nearest neighbor and simple GP rarely out-perform pursuit. The Markov models initially do worse than pursuit because they have many parameters (making them vulnerable to over-fitting) and often make incorrect predictions when the agent observes a trajectory in a sparse region of their state space. In contrast, the DPGP starts out similar to pursuit, since the zero-mean prior on trajectory derivatives naturally encodes the bias that, in the absence of other

[10]For reasonably dense data, Gaussian process and nearest neighbor approximations are very close; thus, the k-nearest neighbor technique also served as a close approximation of a solution trained on a single GP for the entire dataset.

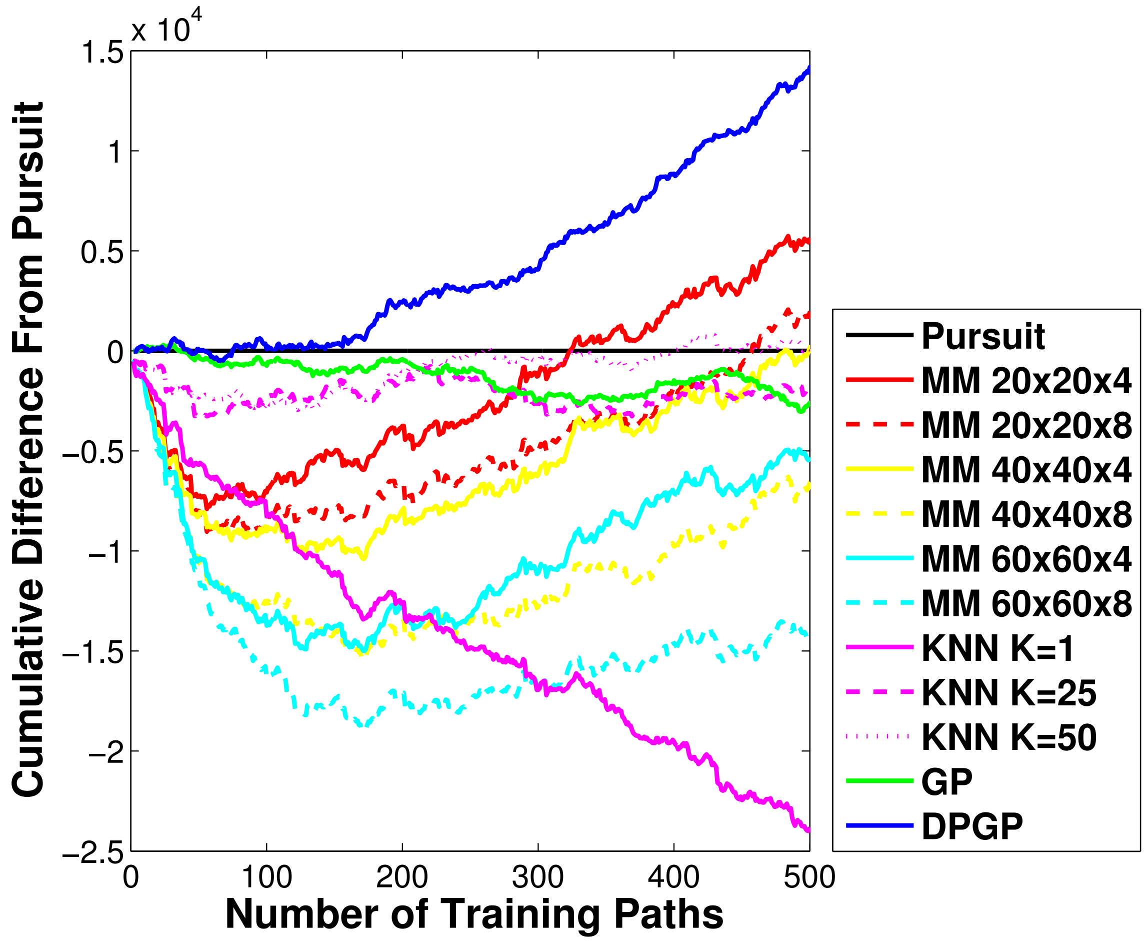


Figure 3-11: Cumulative difference in reward from the pursuit approach for the DPGP model, various Markov models (MM), k-nearest neighbors (KNN), and a GP fit to the current trajectory (GP) (higher values are better).

data, the car will likely stay still. The DPGP model quickly generalizes from a few trajectories and thus attains the highest cumulative rewards of the other methods. The Markov models eventually also exhibit similar performance, but they never make up for the initial lost reward.

3.3 Interception and Tracking with Partial Information

We now consider the case in which the agent does not always have access to the target's current location. Instead, we assume that the agent has a sensor that will provide a perfect measurement of the target's location if the target is within some observation radius of the agent, and no measurement otherwise. The agent's task is to first intercept the target — maneuver to within some small interception radius of the target for "inspection" — and then to keep the target within its larger observation radius.

In many senses, this problem formulation is a more realistic scenario in that we do not assume a sensor network will always provide the target's location. However, because the agent can only observe the target when it is near it, it no longer has full trajectories to cluster into motion patterns. Thus, a key additional step in the partially observable case is that the agent must now infer where the target when it was not being observed. This information is needed both to determine which motion pattern the target was exhibiting and to update the characteristics of a motion pattern cluster from partial trajectories.

We first formalize the model and detail the inference procedure; we next show how our motion model helps the agent intercept and track targets in a synthetic domain (Section 3.3.3) and a helicopter-based search and tracking scenario using the real-world taxi data (Section 3.3.3).

3.3.1 Interception and Tracking Problem Formulation

Since the target's current position is now potentially unknown at every time step, we formalize the interception and tracking scenario as a partially observable Markov decision process (POMDP). In addition to the states, actions, transition function, and reward function present in an MDP, a POMDP also includes a set of observations and an observation function.

As in the fully observable MDP case (Section 3.2), the state consists of the joint position of our agent and the target $(x^a, y^a, x^{target}, y^{target})$. Given an action and our agent's current position (x^a_t, y^a_t), we assume that our agent's next position (x^a_{t+1}, y^a_{t+1}) is deterministic and known. However, the target's position (x^{target}, y^{target}) may no longer be observed. Instead, our agent receives an (accurate) observation of the target's position if the target is within an observation radius r_{obs} of our agent. Otherwise

Algorithm 2: Partially Observable Motion Model Inference

```
1:  for sweep = 1 to # of sweeps do
2:    for each trajectory $t^i$ do
3:      for each time step $n$ do
4:        if $(x_t^{target}, y_t^{target})$ was not observed then
5:          Draw $(x_t^{target}, y_t^{target})$ using Equation 3.5
6:          if $(x_t^{target}, y_t^{target})$ was within $r_{obs}$ of $(x_t^a, y_t^a)$ then
7:            Reject sample, go to 5
8:          end if
9:        end if
10:     end for
11:   end for
12:   for each motion pattern $b_j$ do
13:     Draw the GP hyperparameters $\theta_{x,j}^{GP}, \theta_{x,j}^{GP}$
14:   end for
15:   Draw the DP hyperparameter $\alpha$
16:   for each trajectory $t^i$ do
17:     Draw $z_i$ using equations 3.8 and 3.9
18:   end for
19: end for
```

our agent receives no information about the target's position. Essentially, we are relaxing the assumption of the previous section that the target is tracked by a dense sensor network.

Our agent gets target information at irregular intervals from a sparse sensor network, and must model the target's behavior and plan trajectories to intercept the target given imperfect information about the current target's location. As before, the target's transitions are stochastic over the continuous space; we can place a distribution over the target's next position $(x_{t+1}^{target}, y_{t+1}^{target})$ based on our motion model. The agent receives a large one-time reward for being within a small interception radius of the target (which is significantly smaller than the observation radius and a small tracking reward for every target within its observation radius.

The inference procedure for learning the target motion models (Algorithm 2) is described next in Section 3.3.2; given this model and the remaining problem parameters, the agent chooses actions using a standard forward search [Ross et al., 2008].

3.3.2 Model Inference

Since our agent sees a target's location only when the target is within a given observation radius, the target trajectory that the agent observes will often be disjoint sections of the target's full trajectory. Fortunately, the Gaussian process does not require continuous trajectories to be trained, and the Dirichlet process mixture model can be used to classify partial paths that contain gaps during which the vehicle was not in sight. In this sense, the inference approach for the full information case (Section 3.2.2) also applies to the partial information case. However, using only the observed locations ignores a key piece of information: whenever the agent does not see the target, it knows that the target is not nearby. In this way, the lack of observations actually provides (negative) information about the target's location.

To leverage this information, we use Gibbs sampling to sample the unobserved target locations as well as the trajectory clusterings. Once the partially observed trajectories are completed, inference proceeds exactly as in the full information case. Specifically, we alternate resampling the cluster parameters (Section 3.2.2) with resampling the unobserved parts of each target's trajectory. Given all of the other trajectories in an incomplete trajectory's cluster, we can sample the missing sections using the prediction approach in Section 3.2.2; this approach also ensures that the filled in trajectories connect to observed segments smoothly. If the sampled trajectory crosses a region where the agent could have observed it—but did not—then that sample is rejected, and we sample a new trajectory completion. This rejection-sampling approach ensures that we draw motion patterns consistent with all of the available information (see Algorithm 2).

To predict future target positions, several of the sampled trajectory completions are retained and averaged to produce a final prediction. Each trajectory completion suggests a different Gaussian process motion model, and is weighted using Bayesian model-averaging. Using the final velocity prediction, computed as the weighted average of individual model predictions, we can then apply the prediction and classification approach in Section 3.2.2 for intercepting and tracking new targets.

3.3.3 Results

In this section, we apply our DPGP model to two partially observable interception and tracking problems. The first is a synthetic example designed to show the basic qualities of the DPGP in the partially observable case. In the second problem, we return to a more challenging, partially observable version of the taxi tracking scenario from Section 3.2. As in the fully observable case, we tested each model in an online fashion: initially the agent had no experience with the target; after each episode, any information it received about the target was incorporated into the motion model. Specifically, if the agent only observed the target at certain times, only those locations were used to update the motion model. The agent does not receive any additional information about the missed observations of a target's trajectory after an episode.

In all of the scenarios, we compared our DPGP algorithm to a pursuit forward search algorithm and a Markov model. The pursuit algorithm goes to the target's last observed location but uses forward search to plan about how best to intercept and track all three targets. The Markov models use a position discretization equal to the interception region with x and y velocity each discretized into two bins. The transition matrix is initialized with a small probability mass on self transitions to encode the bias that in the absence of data the target will tend to stay in the same location. Without this bias the model performs extremely poorly initially and would be an unfair comparison to our model which has a similar prior bias (Section 3.1.1). The Markov model also uses forward search to plan for the helicopter. While we could have used other Markov models with more bins, the results from Section 3.2.3 show us that these Markov models may perform better in the limit of infinite data but with the small data set here a Markov model with a small number of bins will perform the best.

Results on a Synthetic Multi-Target Scenario

We first illustrate our approach on a synthetic interception and tracking problem based on Roy and Earnest [2006]. In this problem, illustrated in Figure 3-12, the

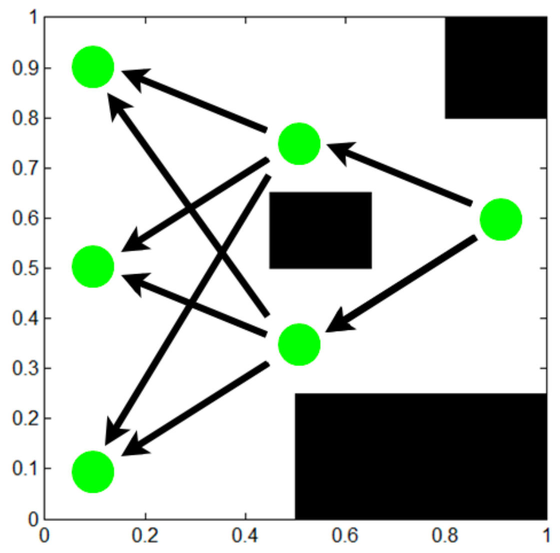


Figure 3-12: Search and tracking task in the synthetic BLOCKS scenario.

agent starts near the opening on the far right and must track three targets which start from the right side of the region and simultaneously move to three different target locations on the left wall. Targets have 0.75 probability of going above the central obstacle and 0.25 probability of going below it. The agent receives a reward of -10 for every time step until it intercepts the target, whereupon it receives a reward of +100. Additionally, it receives a reward of +1 for every target within its observation radius. We call this the BLOCKS scenario.

Figure 3-13 shows the performance of each approach over five runs, where each run consists of 100 episodes. The error bars show the 95% confidence interval of the mean from the five runs. In the figure, not only are the means of the DPGP approach higher than the other approaches, but in practice it scores significantly better on each individual run. The Markov models, despite requiring a fair amount of data to start making relatively good predictions, do outperform the simpler strategy. Figure 3-14 shows parts of a single planning episode, where the helicopter initially intercepts one target going below the obstacle before pursuing the last two above the obstacle.

Since this is a synthetic example, we can also compare the motion patterns found to the true underlying patterns in the model. The model has six patterns: the target can go either above or below the obstacle to reach one of the three final locations

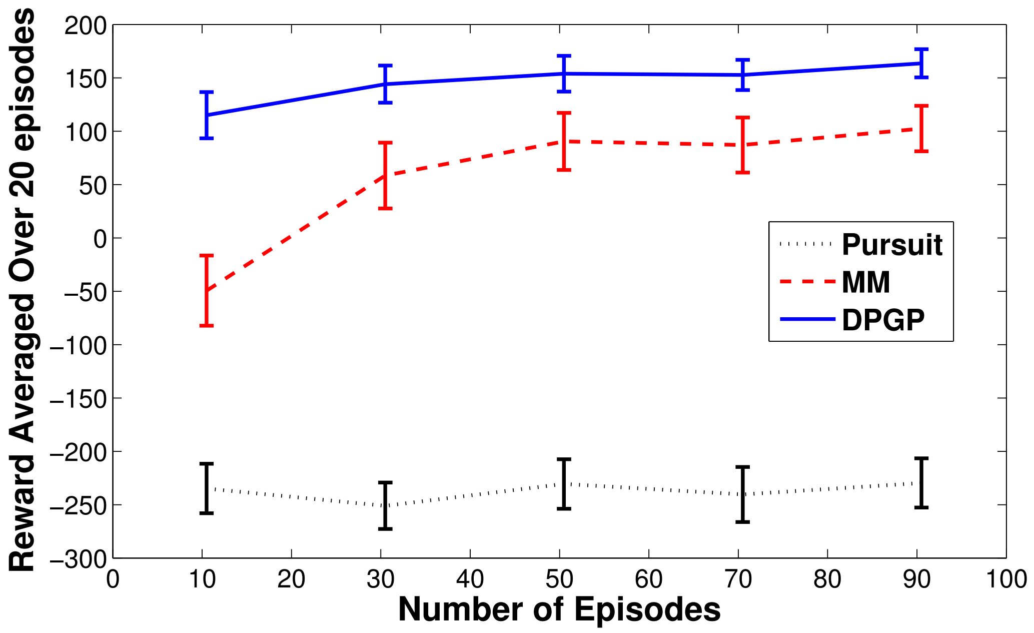
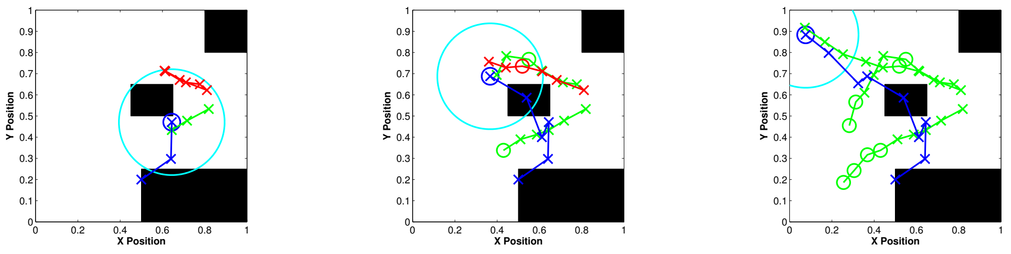


Figure 3-13: Sliding window average of per-episode rewards achieved by different models on the BLOCKS scenario. Error bars show the 95% confidence interval of the mean from five repeated runs.

on the left wall. The number of clusters found by our DPGP approach as a function of training paths is shown in Figure 3-15. In the beginning, when the agent has seen relatively little data, it maintains a smaller number of motion patterns. As the agent observes more trajectories, we see that the number of motion patterns settles around the true number (the error bars show 95% confidence intervals of the mean). By the end of the 100 trials, if two trajectories belonged to the same true cluster, then our DPGP model placed them in different clusters with probability 0.2625; if two trajectories actually belonged to separate clusters, then our DPGP model placed them in the same cluster with probability 0.1567. Some of this clustering error is due to our agent being out of range of the target resulting in some trajectories not containing the full location history. In fact, approximately 20% of the data points were not observed during the trails. These statistics, consistent over five runs of the 100 episodes, strongly suggest that our DPGP model was learning key clustering characteristics of the target motion patterns.



(a) $t = 3$ (b) $t = 6$ (c) $t = 9$

Figure 3-14: A planning episode in the Blocks scenario. Agent positions are shown in blue and target positions are shown in red before they are intercepted and green after. The small blue circle and the large cyan circle around the agent signify the interception region and observation radius, respectively. Target locations that were within the agent's sensor range are marked by $\times$ symbols, and target locations beyond the agent's sensor range are marked with $\circ$ symbols.

Results on a Helicopter-based Multi-Target Scenario

We next applied our approach to a helicopter-based search and tracking scenario that used the same taxi dataset described in Section 3.2.3. We assume that the agent was given the targets' true initial locations and velocities from a ground-based alert network. After being given this initial piece of information about the targets, the target states are no longer directly accessible, and the helicopter receives information about a target's location only if the target is within about 1.5 miles (a quarter the map area) of the helicopter. The interception radius is 0.25 miles (a twenty-fifth the map area). The reward function is identical to the one described in Section 3.3.3.

The results comparing our DPGP approach to the same control strategies from Section 3.3.3 are shown in Figure 3-16 and Figure 3-17, with the error bars showing the 95% confidence interval of the mean for the five runs of 150 tasks. Using our DPGP approach for modeling the targets results in much better interception and tracking performance from the start. Unlike the simpler Blocks scenario, the Markov models do no better than simple pursuit after 150 episodes. Figure 3-18 shows the number of clusters found by the DPGP approach as a function of training paths. As expected from a real-world dataset, the number of motion patterns grows with the number of episodes as new motion patterns observed in new trajectories. Finally, Figure 3-19 shows an example episode where the helicopter first intercepts each target and then

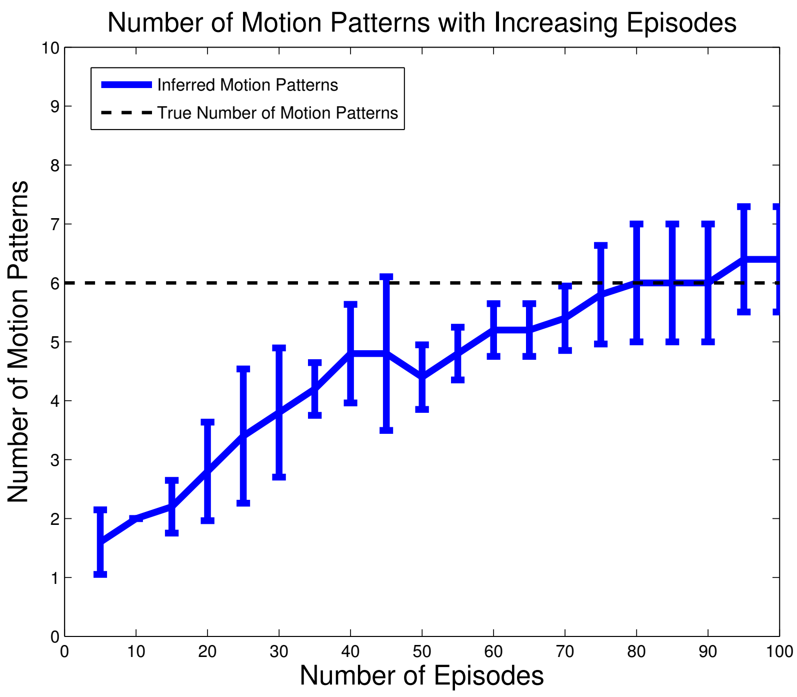


Figure 3-15: Number of discovered target motion patterns in the BLOCKS scenario.

finds a location where it can observe multiple targets to keep them localized.

3.4 Related Work

Much of the past work in modeling mobile agents has focused on two problems: expert systems (which require specialized data) and modeling a single agent (requiring data generated by the single agent). Letchner et al. [2006] built a model that predicted trajectories based on the optimal path between two locations (given factors such as the time of day) and the amount of "wasted time" a driver was willing to accept. Dia [2002] used a survey to classify drivers into different profiles to enable better prediction. Both of these works note that it is difficult to specify a model for human motion patterns based on logical reasoning. For example, Letchner et al. [2006] noted only 34.5% of drivers choose the fastest route between two locations.

Whether these statistics are a result of driver ignorance or another factor (*e.g.*, avoiding a stressful route) is highly debatable and difficult to incorporate into expert

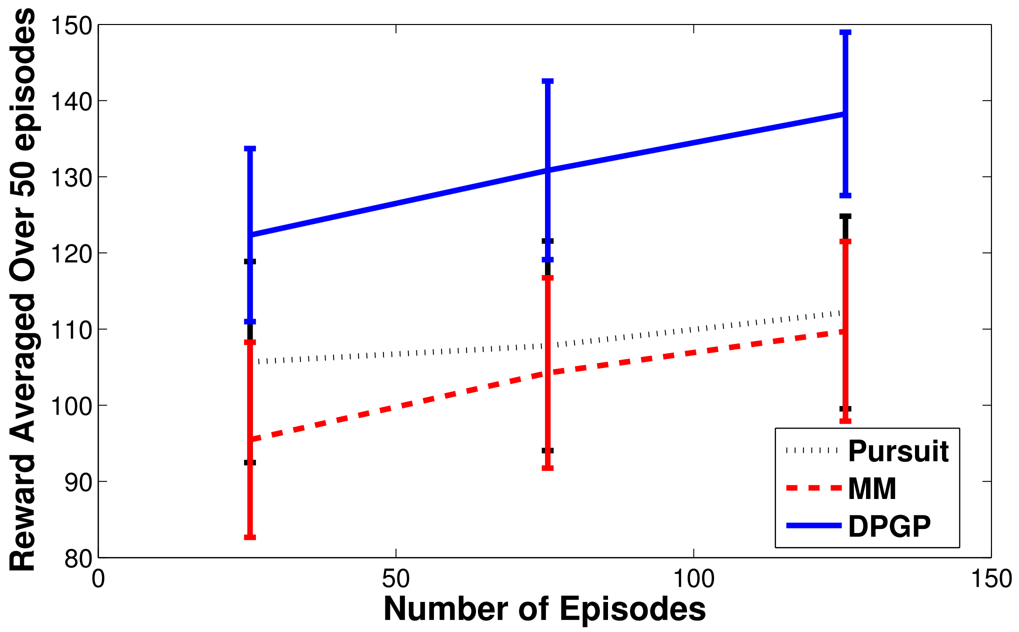


Figure 3-16: Sliding window average of per-episode rewards achieved by different models on the taxi multi-target interception and tracking task. Error bars show the 95% confidence interval of the mean from five repeated runs.

models of human motion patterns. Without access to similar data for the greater Boston area or having similar time-stamped GPS data for their models, we were unable to compare them to our approach; however, it would be interesting to see if incorporating expert features related to human psychology into the priors of our model could improve predictions.

Another body of literature has trained Markov models (generally using data from only one person) in which each road segment is a state and transition probabilities encode the probabilities of moving from one segment to another. For example, Patterson et al. [2003] treated the true driver state as hidden by GPS sensor noise and a hidden driver mode. Ashbrook and Starner [2003] modeled the end position and transition probabilities explicitly, but doing so prevented the method from updating the probabilities based on a partially observed trajectory. Using a hierarchy of Markov models, Liao et al. [2007] were able to make both local and destination predictions but still had difficulty in regions of sparse training data. Ziebart et al. [2008] take an inverse reinforcement learning approach to predicting the future locations of a target

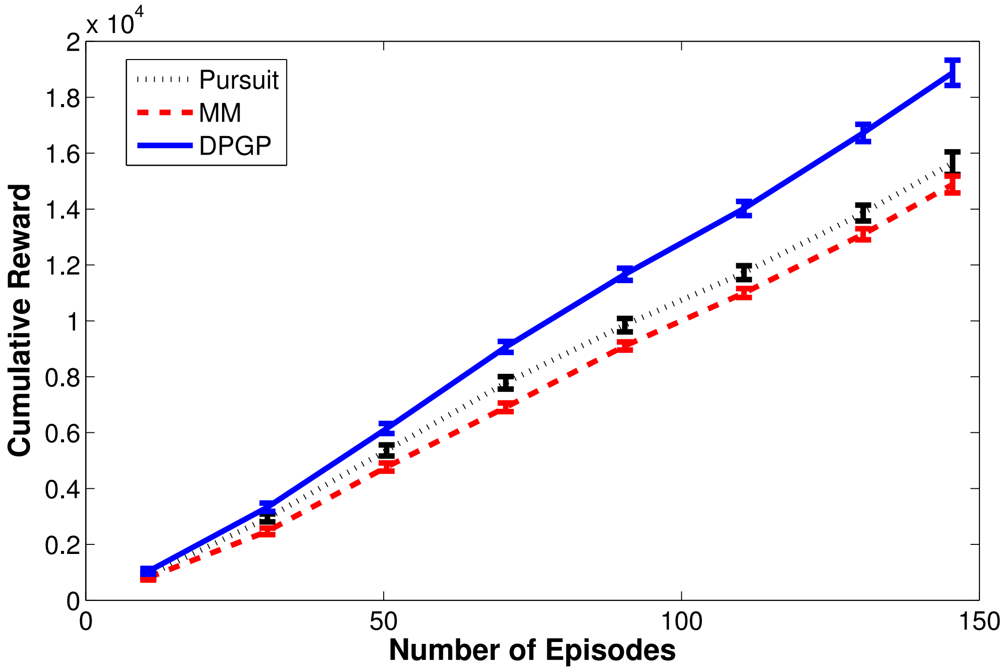


Figure 3-17: Results from the taxi multi-target interception and tracking task showing cumulative reward achieved by different models on the Blocks scenario. Error bars show the 95% confidence interval of the mean from five repeated runs.

of interest. In their work, they learn the reward function of drivers and use a known dynamics model (either designer provided or learned from a sufficient amount of data) to make their predictions. This is in contrast to the work presented in this chapter where we assume a known reward function and our focus is on learning a dynamics model from limited data.

Recently, Gaussian processes have been successfully applied to modeling and prediction in robotics tasks. Tay and Laugier [2007] used a finite mixture of Gaussian processes to model multiple moving targets in a small simulation environment. In the context of controlling a single vehicle, Ko and Fox [2009] demonstrated that Gaussian processes improved the model of a vehicle's dynamics.

Fox et al. [2007] took a related approach to ours and modeled the number of motion patterns with a Dirichlet process prior, with each motion pattern governed by a linear-Gaussian state space model. Unlike our approach, agents could switch between motion patterns using an underlying hidden Markov model. In our specific

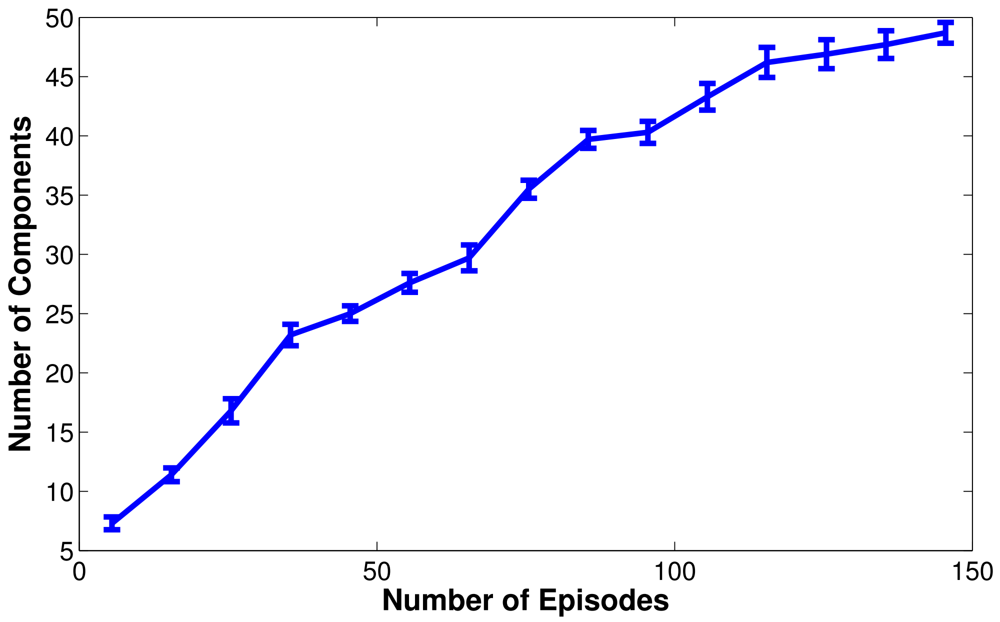


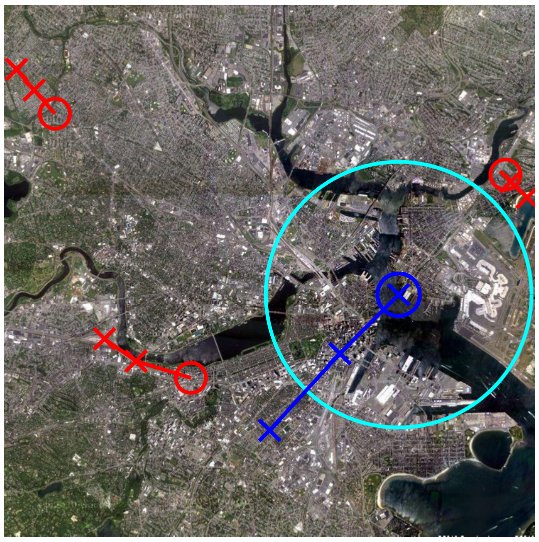
Figure 3-18: Number of discovered motion patterns for the taxi dataset search and tracking task.

dataset and application, the agents usually know their start and end destinations from the very beginning; not allowing motion pattern changes helped predict a car's path on roadways that were common to many motion patterns. However, our framework could certainly be extended to allow agents to change motion patterns.

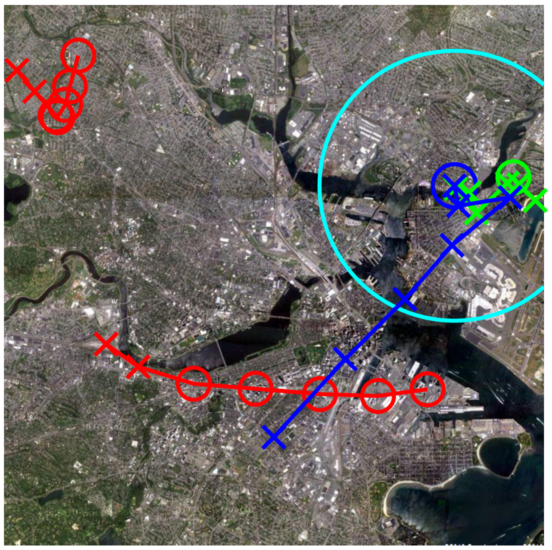
The target-tracking problem under partial observability conditions has a natural formulation as a POMDP, since the agent must make decisions with incomplete knowledge of the targets. Pineau et al. [2003] first applied the PBVI point-based solver to a small target-tracking problem, and more recent approximate point-based techniques, for example by Hsu et al. [2008] and Kurniawati et al. [2009], have expanded the applicability of general POMDP solvers to the target-tracking domain by rapidly exploring the reachable and high-value regions of the belief space.

Despite these advances, point-based POMDP methods still have limited utility in this domain. These methods typically discretize the agent and target state spaces to obtain a finite-dimensional belief space, and are unable to adapt to changing motion patterns due to substantial offline requirements.

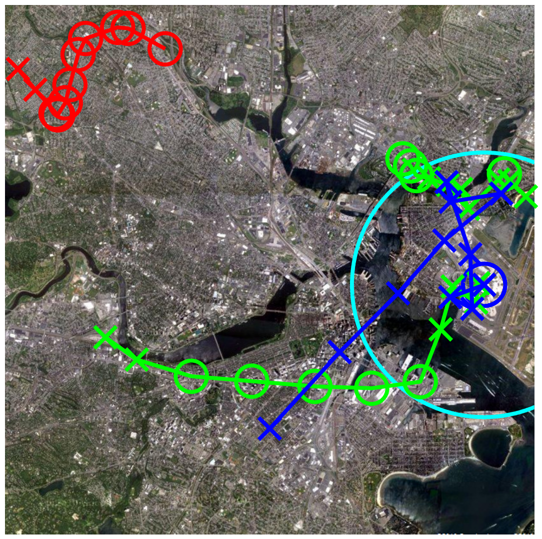
One approach to avoiding state space discretization is to represent beliefs using



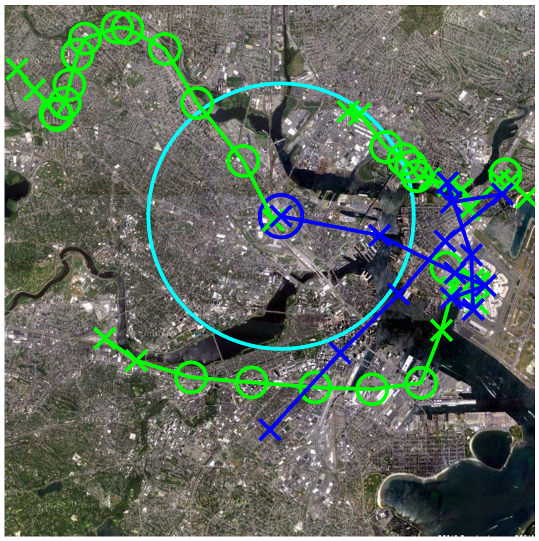
(a) $t = 3$



(b) $t = 7$



(c) $t = 11$



(d) $t = 13$

Figure 3-19: A planning episode from the taxi data set. Helicopter positions are shown in blue. Car positions are shown in red before interception and green after. The small blue circle and the large cyan circle around the helicopter signify the tagging and observation range, respectively. Car locations are marked with a $\times$ symbol when observed by the helicopter, and a $\circ$ symbol when beyond the helicopter's sensor range.

Gaussian distributions, as applied by Miller et al. [2009] to target tracking, or by He et al. [2010] with Gaussian mixture models. An advantage of these representations is the ability to analytically and exactly manipulate the belief state. However, these approaches focus on planning with accurate models, and do not address model learning or acquisition.

3.5 Conclusions and Discussion

Using our Bayesian nonparametric DPGP approach for modeling target motion patterns improved our agent's ability to predict a target's future locations from relatively few examples. A key advantage of the DPGP model is that it provides a way of scaling the sophistication of its predictions given the complexity of the observed target trajectories: we could model motion patterns directly over a continuous space without needing to specify discretization levels or expected curves. In contrast, the Markov models suffered because even at a "reasonable" discretization, these models needed to train the motion model for every grid cell—which required observing many more trajectories.

One way to think about the DPGP is as a type of hidden Markov model (HMM), where the future trajectories of the agent are Markov conditioned on some hidden (instead of observed) state. Indeed, introducing a hidden variable to describe a trajectory type or movement mode is a standard way to avoid issues such as the Markov model's confusion over crossing paths (Figure 3-2). However, standard HMM-based approaches would still typically need to define the number of trajectory types *a priori* and commit to a level of discretization. The DPGP can be viewed as an HMM model in a continuous space with an unknown number of trajectory types.

While we focused on the motion patterns of taxis in the Boston area, as seen in our synthetic example, the DPGP approach is not limited to modeling motion patterns of cars. It is meant as a far more general mobile agent model, which models a wide variety of trajectories over a continuous space as long as the targets motions obey local smoothness and continuity constraints—as seen in Section 3.3.3, paths and trajectory types can be inferred from even sparsely observed targets if the smoothness assumptions imposed by the GP model are true. We would expect the DPGP model to have difficulty modeling trajectories where smoothness assumptions about the trajectory derivatives could not be characterized by the single distance parameter in the GP covariance kernel: for example, if trajectories tended to have tight curves or kinks. Nonstationary GPs could be used in these situations [Meiring et al., 1997, Paciorek and Schervish, 2000]. In environments where movement in x and y directions are tightly coupled, GP models with multiple outputs may be more appropriate [Boyle and Frean, 2005].

The stationary, single-valued aspects of the GP motion model also make it inappropriate for modeling trajectories that loop onto themselves—that is, do different things at the same location based on some other context—and for adversarial situations. In these cases, additional information, such as the agent's location relative to the target, would need to be incorporated into the GP inputs. Thus, the DPGP model is best suited for situations where complex, non-overlapping dynamics and clusterings must be learned from relatively little data—as we saw in the results sections, the Markov models do catch up in performance once sufficient data is available; however, the DPGP makes significantly better predictions from only a few trajectories. In situations where the number of trajectory types is known and large batches of data exist, the DPGP will likely add little over a finite HMM-based model trained on the same large dataset. The Bayesian nature of our approach does allow available expert knowledge about target motion patterns to be given in the form of additional example trajectories without any need to adjust the rest of the inference process.

Finally, it is well-known that standard GPs require $O(N^3)$ computation to perform inference, where N is the number of data points. We were still able to process all of the data using the approximations described in Section 3.2.2; for larger datasets, there are fairly standard approximation algorithms with $O(N)$ running times [Csat and Opper, 2001, Snelson and Ghahramani, 2006].

Accurate agent modeling in large domains often breaks down from over-fitting or under-fitting the training data. We used a Bayesian nonparametric approach to motion-pattern modeling to circumvent these issues. This approach allows us to build flexible models that generalize sensibly with sparse data and add structure as more data is added. The reward models, the dynamics model of the agent, and the form of the agent's planner can all be adapted to the task at hand with few adjustments to the DPGP model or inference procedure.

We demonstrated our motion model on a set of helicopter-based interception and tracking tasks trained and tested on a real dataset of complex car trajectories. The results suggest that our approach will be useful in a variety of agent-modeling situations. Since the underlying structure of our model is based on a Gaussian process framework, our approach could easily be applied to beyond car domains to generic metric spaces. Finally, although we focused our approach on a set of interception and tracking tasks, we note that the DPGP motion model can be applied to any task where predictions about a target's future location are needed.

In the next chapter we apply a similar idea of growing the model to modeling the health of a battery. The same conceptual approach is taken where we use a Dirichlet process prior to learn the number of "behaviors" which, for a battery, are voltage and temperature traces. The intuition from this chapter of driver mobility patterns directly translates into battery behaviors.

Chapter 4

A Bayesian Nonparametric Approach to Modeling Battery Health

4.1 Battery Health Model

Batteries are often both the first point of failure and a significant fraction of a product's cost. Understanding battery failure is particularly important in robotics: at best, battery death at an inopportune time will require significant personnel intervention to rescue the robot; at worst, the robot may be lost. In robotic applications where battery death poses a large safety or financial hazard, batteries are usually replaced long before expected failure. However, this procedure introduces significant operational costs. Models to accurately predict the cycles left in a battery can help alleviate these costs.

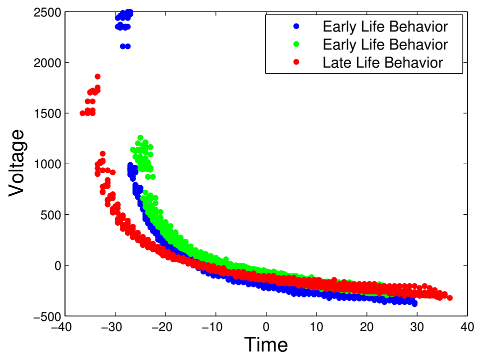
Unfortunately, modeling battery dynamics is far from simple. Previous efforts have generally relied on a significant amount of domain knowledge [Chen and Rincon-Mora, 2006, Gomadam, 2002, Dougal, 2002]; these methods cannot be used without a detailed understanding of the internal battery structure and chemical composition. In contrast, our model can be applied to any type of battery as long as we can collect empirical measurements from a representative set of similar batteries.

We take a data-driven, non-parametric approach to predicting time until a battery can no longer hold a charge, or what we call battery death. Specifically, we consider how the trajectories of voltage and temperature change as the battery cools after charging. As seen in Figure 4-1, batteries exhibit different voltage and temperature trajectory patterns while cooling. We see that these trajectory patterns vary both inside and across different points in their life-cycles. Our model first clusters together similar trajectories into *cooling behaviors*. Next, we learn a particular time-to-death for each cooling behavior. Given voltage and temperature trajectories from a new battery, we first map those trajectories to a cooling behavior learned from our training set of batteries that have already been cycled to death. We predict the new battery's remaining life based on the life that remained in the training batteries when they exhibited the same cooling behavior. Each cluster of our approach is associated with both a simple model of voltage trajectories and temperature trajectories. This is in contrast to data-driven, parametric methods, such as Saha et al. [2009], Wen et al. [2004], which learn a single complex model that can have more difficulty capturing the distinct trajectory patterns shown in Figure 4-1.

One of the key difficulties with predicting the time to battery death is that even "identical" batteries can have widely varying lifetimes. We focus on NiMH battery packs as commonly used with consumer electronics devices, including robots like the iRobot Roomba vacuum cleaning robot. For this project, iRobot Corporation[1] provided 13 Roomba batteries as research samples that did not meet iRobot's standards[2]. We used a custom-built charging station to cycle the batteries until death, a process that involves months of data collection per battery. The number of charge cycles in these batteries varied from 269 to 697 cycles, yet at the beginning of operations, all batteries behaved nearly identically and thus were difficult to differentiate. However, as each battery neared death, no matter how long it had been in operation, it tended to exhibit certain cooling behaviors after charging. A data-driven approach allows us

[1]Bedford, MA, USA

[2]We also tested production battery packs that ship with Roomba, but these packs were longer-lasting, and we were unable to kill them in the time frame of this study.



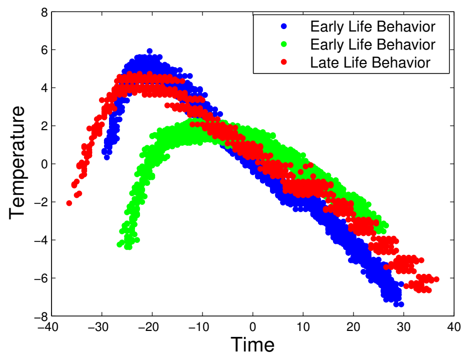


Figure 4-1: Sample voltage and temperature trajectories from different parts of a battery's life-cycle showing different patterns of cooling.

to learn the patterns that are strongly indicative of nearing battery death.

Finally, our nonparametric Bayesian approach allows us to make predictions given relatively little data, which is important since collecting data by cycling a fresh battery to death can take several months on a test stand. We can discover the number of cooling behaviors supported by the data automatically, without having to specify how many patterns may be present in advance.

4.1.1 Battery Data

We used data from 4683 charge cycles collected from 13 NiMH battery packs. Each pack was attached to a test stand which automatically cycled the battery through charging, cooling, and discharging. (Batteries heat up during charging; the charger automatically adds a cooling phase to prevent the batteries from overheating during the constant cycling.) The batteries started out new and were cycled until they could no longer hold charge, which we defined as battery death. Battery temperature and voltage were measured during each phase of each cycle.

To predict battery health, we focused on modelling the cooling phase of each cycle for two reasons. First, as seen in figure 4-1, batteries do exhibit a variety of distinct cooling behaviors over their lifetimes—thus, these data do contain information about battery health. Second, the cooling phase is a part of the cycle that can be most consistently measured in actual operation. Unlike on test stands, real-world discharge trajectories strongly depend on the robot's operations; charge trajectories, while somewhat more consistent, depend on the initial charge. Cooling trajectories are easily measured at the charging station and largely depend on the battery's final, fully charged state.

4.1.2 Battery Cooling Behavior Model

To model cooling trajectories, such as those in Figure 4-1, we assume that each voltage and temperature trajectory belongs to a particular cooling behavior. A cooling behavior is defined by a distribution over voltage and temperature trajectories. Let θ_v^j be the set of parameters that define the characteristic voltage trajectory for cooling behavior j, and let θ_e^j be the set of parameters that define the characteristic temperature trajectory for cooling behavior j.[3]

Suppose we are given a pair of voltage and temperature trajectories $v_0, v_1, ..., v_N$ and $e_0, e_1, ..., e_N$ measured at times $t_0, t_1, ..., t_N$. Our mixture model defines the probability of these data as

$$p(v_{0:N}, e_{0:N}|t_{0:N}) = \sum_{j=1}^{M} p(v_{0:N}|t_{0:N}, \theta_v^j) p(e_{0:N}|t_{0:N}, \theta_e^j) p(j) \tag{4.1}$$

where M is the total number of cooling behaviors. Factoring the probabilities of a voltage trajectory $p(v_{0:N}|t_{0:N}, \theta_v^j)$ and temperature trajectory $p(e_{0:N}|t_{0:N}, \theta_e^j)$ encodes our assumption that the voltage and temperature trajectories are independent given the mixture component j (which represents the battery's health state). Using a Bayesian approach, that is, placing a distribution over every parameter setting θ^j, helps avoid overfitting–a serious issue when the amount of data is limited.

Our goal is to learn the parameters θ^j for the characteristic curves and the probabilities $p(j)$ from the training data, and then learn a model of time-to-death for each behavior. (Note that only the temperature and voltage patterns for a cycle are used

[3]We also considered using the number of elapsed cycles as a feature, but we found this feature actually reduced prediction accuracy by splitting clusters with different numbers of elapsed cycles but similar remaining battery life — an acute issue given the bimodal nature of battery life.

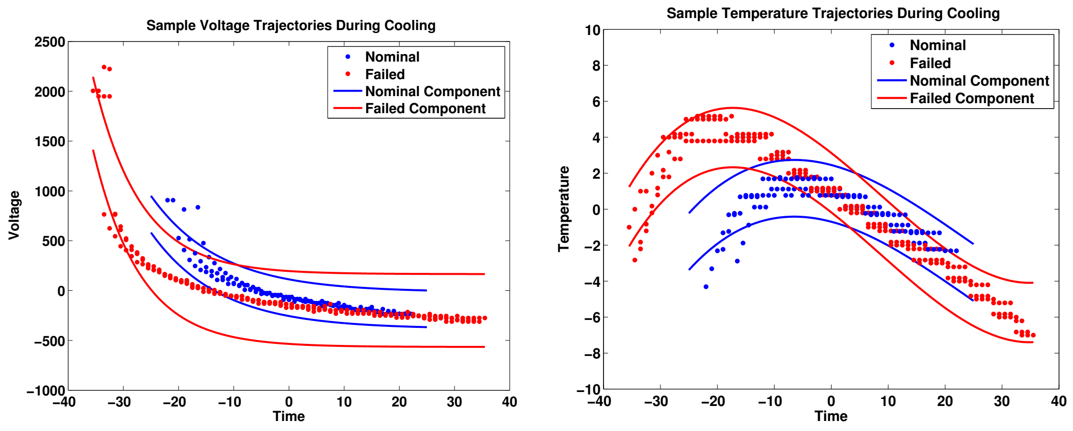


Figure 4-2: Cooling behaviors learned from the sample voltage and temperature trajectories in Figure 4-1.

as input for the initial clustering, since in testing, we will not know the number of cycles remaining.) We can then use this model to predict the time-to-death by inferring the type of cooling behavior exhibited by a new battery. We describe the model we use for the cooling behaviors below; in Section 4.1.2, we describe the clustering process used to define cooling behaviors out of measured trajectories.

Examples of cooling behaviors derived from the raw trajectory measurements in Figure 4-1 are shown in figure 4-2. The solid line shows the characteristic curve which defines the mean voltage and temperature trajectory for a particular cooling behavior, while the dashed line provides one standard-deviation range in which we expect most of the measured trajectories belonging to that behavior to lie. We model a measurement trajectory $p(v_{0:N}, e_{0:N}|t_{0:N}, \theta^j)$ as characteristic curves with additive Gaussian noise.

Voltage Trajectory Model The physics of electrical charge in batteries make an exponential curve a natural choice for the characteristic way in which we expect the voltage to change during the cooling cycle (see Figure 4-1 for sample trajectories). We posit that the measured voltages in the trajectory will vary around this characteristic exponential curve with independent, Gaussian noise. Thus, the voltage curve associated with particular cooling behavior is characterized by the parameters of the exponential curve and the amount of measurement noise.

Formally, we define the probability of a voltage trajectory $p(v_{0:N}|t_{0:N}, \theta_v^j)$ given a particular characteristic curve as

$$v_i \sim \mathcal{N}\left(b_0^j + b_1^j \exp(b_2^j + b_3^j t_i), (\sigma_v^j)^2\right) \tag{4.2}$$

and let $\theta_v^j = \{b_0^j, b_1^j, b_2^j, b_3^j, \sigma_v^j\}$. We place an independent Gaussian prior over the values of each element of θ_v^j.

We define the prior over the voltage trajectory model parameters to be

$$s_v^j|a_v, b_v \sim \mathcal{G}\left(a_v, b_v\right), \quad b_k^j|\mu_{v,k}, s_{v,k} \sim \mathcal{N}\left(\mu_{v,k}, s_{v,k}\right)$$

where $s_v = 1/\sigma_v^2$ is the precision of the voltage trajectory model and $k = 0, 1, 2, 3$. Based on the likelihood function defined in equation 4.2 the posterior distributions of parameters s_v and b_0 have closed form posteriors

$$\begin{aligned} s_v^j|v_{0:N}, t_{0:N}, b_{0:3}^j, a_v, b_v &\sim \\ & \mathcal{G}\left(a_v + \frac{N+1}{2}, \frac{1}{\frac{1}{2}\sum_{i=0}^N (v_i - b_2^j t_i^2 - b_1^j t_i - b_0^j)^2 + \frac{1}{b_v}}\right) \\ b_0^j|e_{0:N}, t_{0:N}, b_1^j, b_2^j, s_v^j, \mu_{v,0}, s_{v,0} &\sim \\ & \mathcal{N}\left(\frac{s_{v,0}\mu_{v,0} + s_v \sum_{i=0}^N v_i - b_2 t_i^2 - b_1 t_i}{(N+1)s_v + s_{v,0}}, \frac{1}{(N+1)s_v + s_{v,0}}\right) \end{aligned}$$

that are normally distributed. Unfortunately, the posterior distributions for parameters $b_{1:3}$ do not have a closed form and therefore we used importance sampling [Rubinstein, 1981] to sample from them.

Temperature Trajectory Model The temperature trajectory exhibits a more complex pattern because the battery initially continues to get warmer after the charging phase before cooling down. We evaluated first to fifth-order polynomial fits on various sample trajectories using a nested ANOVA, and a third-order polynomial was generally where the F-statistic reduced the most dramatically. Thus, we defined the probability of a temperature trajectory $p(e_{0:N}|t_{0:N}, \theta^j)$ given a particular characteristic mean curve as

$$e_i \sim \mathcal{N}\left(d_0^j + d_1^j t_i + d_2^j t_i^2 + d_3^j t_i^3, (\sigma_e^j)^2\right) \tag{4.3}$$

and let $\theta_e^j = \{d_0^j, d_1^j, d_2^j, d_3^j, \sigma_v^j\}$. We place an independent Gaussian prior over the values of each element of θ_e^j.

We define the prior over the temperature trajectory model parameters to be $s_e^j | a_e, b_e \sim \mathcal{G}(a_e, b_e)$ and $b_k^j | \mu_{e,k}, s_{e,k} \sim \mathcal{N}(\mu_{e,k}, s_{e,k})$ where $s_e = 1/\sigma_e^2$ is the precision of the temperature trajectory model and $k = 0, 1, 2, 3$. Based on the likelihood function defined in equation 4.3 the posterior distributions of the parameters have closed form posteriors

$$\begin{aligned}
s_e^j | e_{0:N}, t_{0:N}, d_{0:3}^j, a_e, b_e \;\sim&\\
&\mathcal{G}\left(a_e + \frac{N+1}{2}, \frac{1}{\frac{1}{2}\sum_{i=0}^N (e_i - d_2^j t_i^2 - d_1^j t_i - d_0^j)^2 + \frac{1}{b_e}}\right)\\
d_0^j | e_{0:N}, t_{0:N}, d_{1:3}^j, s_e^j, \mu_{e,0}, s_{e,0} \;\sim&\\
&\mathcal{N}\left(\frac{s_{e,0}\mu_{e,0} + s_e \sum_{i=0}^N e_i - d_3 t_i^3 - d_2 t_i^2 - d_1 t_i}{(N+1)s_e + s_{e,0}},\; \frac{1}{(N+1)s_e + s_{e,0}}\right)\\
d_1^j | e_{0:N}, t_{0:N}, d_0^j, d_2^j, d_3^j, s_e^j, \mu_{e,1}, s_{e,1} \;\sim&\\
&\mathcal{N}\left(\frac{s_{e,1}\mu_{e,1} + s_e \sum_{i=0}^N e_i t_i - d_3 t_i^4 - d_2 t_i^3 - d_0 t_i}{s_e \sum_{i=0}^N t_i^2 + s_{e,1}},\; \frac{1}{s_e \sum_{i=0}^N t_i^2 + s_{e,1}}\right)\\
d_2^j | e_{0:N}, t_{0:N}, d_0^j, d_1^j, d_3^j, s_e^j, \mu_{e,2}, s_{e,2} \;\sim&\\
&\mathcal{N}\left(\frac{s_{e,2}\mu_{e,2} + s_e \sum_{i=0}^N e_i t_i^2 - d_3 t_i^5 - d_1 t_i^3 - d_0 t_i^2}{s_e \sum_{i=0}^N t_i^4 + s_{e,2}},\; \frac{1}{s_e \sum_{i=0}^N t_i^4 + s_{e,2}}\right)\\
d_3^j | e_{0:N}, t_{0:N}, d_0^j, d_1^j, d_2^j, s_e^j, \mu_{e,3}, s_{e,3} \;\sim&\\
&\mathcal{N}\left(\frac{s_{e,2}\mu_{e,2} + s_e \sum_{i=0}^N e_i t_i^3 - d_2 t_i^5 - d_1 t_i^4 - d_0 t_i^3}{s_e \sum_{i=0}^N t_i^6 + s_{e,3}},\; \frac{1}{s_e \sum_{i=0}^N t_i^6 + s_{e,3}}\right)
\end{aligned}$$

that are normally distributed.

Mixtures of Measurement Trajectories

From Figure 4-1, we see that batteries exhibit a variety of cooling behaviors—that is, a variety of characteristic voltage and temperature curves during cooling—over the battery lifetime. We use a Dirichlet process 2.2 to guide the process of clustering the voltage and temperature trajectories that we measure into distinct cooling behaviors. The Dirichlet process (DP) posits that an infinite number of cooling behaviors may exist, but certain behaviors are likely to be very common and others quite rare. Of course, if only a finite number of voltage and temperature trajectories are observed, they can only belong to a finite number of cooling behaviors. We first provide background on the DP and then describe how we use it to infer the number of cooling behaviors in our battery model.

The Dirichlet Process Prior Let $n_1 ... n_M$ be the number of trajectories assigned to behaviors $1...M$, respectively, and let z_i be the labeled behavior of trajectory i (that is, if $z_i = 5$ then the i^{th} trajectory was generated according to the 5^{th} behavior). Then, the probability that a newly observed trajectory $N + 1$ belongs to behavior j, given the previous N data points, is

$$\begin{aligned} p(z_{N+1} = j, j \in 1, ..., M | n_{1:M}, \alpha) &= \frac{n_j}{N + \alpha} && (4.4) \\ p(z_{N+1} = j, j \notin 1, ..., M | n_{1:M}, \alpha) &= \frac{\alpha}{N + \alpha}. && (4.5) \end{aligned}$$

where α is a concentration parameter that governs how often we expect to see a completely new type of cooling behavior. Having a model that can suggest that an observed trajectory is unlike any previous trajectory gives us the flexibility to automatically infer the number of behaviors in a dataset.

We place a gamma prior on $\alpha \sim \mathcal{G}(a_\alpha, b_\alpha)$ which leads us to the posterior distribution

$$p(\alpha | n_1, ..., n_M, a_\alpha, b_\alpha) \propto p(n_1, .., n_M | \alpha) p(\alpha | a_\alpha, b_\alpha). \quad (4.6)$$

We draw samples from this distribution using importance sampling [Rubinstein, 1981].

Model Learning: Computing the Dirichlet Process Posterior We now describe how to use the DP prior to infer the number of cooling behaviors in a set of trajectory data and the corresponding parameters θ of the characteristic cooling curves in each behavior. Once the number and characteristic curves for the cooling behaviors have been inferred, we can use the cooling behaviors to predict the time remaining until battery death (Section 4.1.2).

Exact inference over an infinite number of possible cooling behaviors is intractable. Instead, we use an iterative process to sample the relevant parameters. First, suppose that we had a set of assignments $z_1....z_N$. Given the temperature and voltage trajectories assigned to a particular cooling behavior j, we can then sample the parameters θ^j associated with the characteristic trajectories of that behavior. We can also sample the concentration parameter α using Equation 4.6. Finally, given the parameters θ^j, we can resample trajectories into more probable assignments $z_1....z_N$. Repeating this process, known as Gibbs sampling, is guaranteed to converge to a set of samples that represent the posterior $p(\{z_i\}, \{\theta^j\}, \alpha | v_0, v_1, ..., v_N, e_0, e_1, ..., e_N)$.

Algorithm 3 summarizes our approach, which follows Rasmussen and Ghahramani [2001] and Rasmussen [1999]. The Gibbs sampling procedure begins by resampling each z_i given the remaining parameters $\{\theta^j\}, \alpha\}$. The probability that voltage and temperature trajectories i will be assigned to an instantiated cooling pattern j is

$$p(z_i = j | \theta^j, \alpha) \propto p(v_{0:N} | t_{0:N}, \theta_v^j) p(e_{0:N} | t_{0:N}, \theta_e^j) \left(\frac{n_j}{N - 1 + \alpha} \right) \tag{4.7}$$

where $p(v_{0:N} | t_{0:N}, \theta_v^j)$ and $p(e_{0:N} | t_{0:N}, \theta_e^j)$ are defined in equations 4.2 and 4.3, respectively. The probability that the trajectories belong to a new cooling behavior is given by

$$p(z_i = M + 1 | \alpha) \propto \left[\int p(v_{0:N} | t_{0:N}, \theta) p(e_{0:N} | t_{0:N}, \theta) d\theta \right] \left(\frac{\alpha}{N - 1 + \alpha} \right), \tag{4.8}$$

and we use Monte Carlo integration [Hammersley, 1960] to approximate the integral as suggested in Rasmussen and Ghahramani [2001].

Given a set of assignments, the trajectory parameters θ^j are resampled from their posterior distributions. In practice we found that simply choosing the maximum *a posteriori* θ^j worked well due to the peakiness of the posterior. Finally, the concentration parameter α is resampled according to Equation 4.6 from the previous section.

Algorithm 3: Cooling Behavior Inference

```
1: for sweep = 1 to # of sweeps do
2:     for each cooling behavior $j$ do
3:         Draw the parameters $\theta^j$ given $z_i$.
4:     end for
5:     Draw the DP hyperparameter $\alpha$ using Equation 4.6
6:     for each trajectory $i$ do
7:         Draw $z_i$ using equations 4.7 and 4.8
8:     end for
9: end for
```

Figure 4-3 shows an example of the cooling behaviors in a single battery that are discovered by the inference process. The fact that the blocks of color (denoting cooling cycles that are inferred to be from the same behavior) are close together in time suggests that clustering by trajectory type does indeed find groups of trajectories with similar times to battery death. We also see that at the beginning of the battery's lifetime (cycles 0 to 400), the distribution over cooling behaviors is quite spread and not strongly associated with a particular time to death. As the battery progresses through its life-cycle, each behavior marks a temporally-clustered block preceding battery death. In Section 4.2, we will see that the relative indistinguishably of cooling trajectories far from battery death means that predicting the lifetime of a fresh battery is prone to large errors; however, the more distinct characteristics of cooling trajectories closer to death allows for better predictions when the battery is nearing death.

Cycles-to-Death Prediction

Once the parameters $\{\theta^j\}_{j=1}^M$ have been learned, we compute a distribution $p(l|j, \theta^j)$ of the remaining cycles to death l for each cooling behavior j. Especially for cooling behaviors that occur early in a battery's lifetime, these distributions can be highly multimodal: a fresh long-life battery and a fresh short-life battery might have very similar voltage and temperature trajectories during their initial cooling phases, but they will have very different numbers of cycles remaining until death (that is, when the battery stops holding charge).

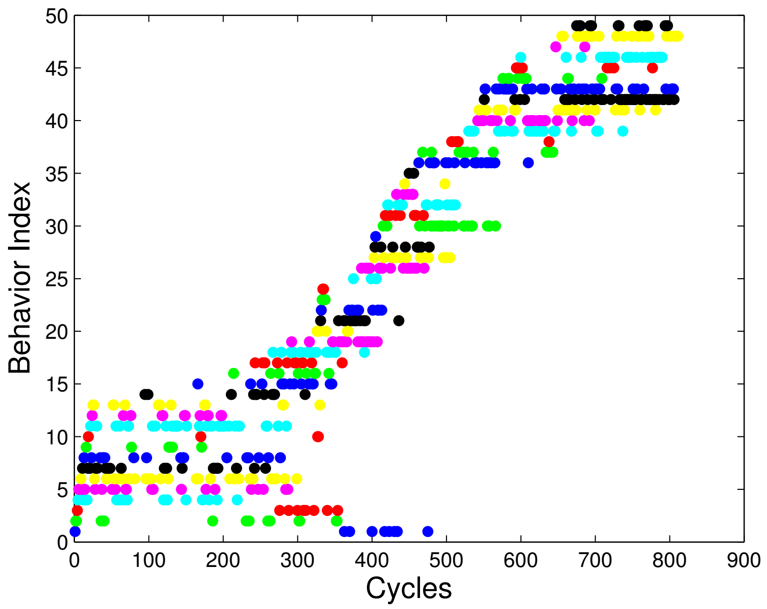


Figure 4-3: Cooling behaviors (indexed both on the y-axis and by color) for a single battery as it progresses through its life-cycle. (Cycles from the same behavior are plotted with the same color.) Initially, the cooling behaviors occur at a variety of points (see the behaviors for cycles 0 to 400), but as the battery progresses through its life-cycle, each stage more distinctly marks a block of time preceding battery death.

We use an empirical distribution to model the number of remaining cycles l. This distribution places a probability mass around each observed time-to-death l_i for each cooling behavior j in the training set.

$$p(l|j, \theta^j) \propto \sum_{i=1}^{N} \mathcal{N}\left(l; l_i, \sigma_l^2\right) \mathbf{1}\{z_i = j\} \tag{4.9}$$

where $\mathbf{1}\{z_i = j\}$ is the indicator function which equals one when trajectory i is assigned to behavior j and zero otherwise.

We can now make predictions about the remaining cycles left in a battery given voltage and temperature trajectories. The distribution over the cycles-to-death is given by taking the expectation over the cooling behavior j

$$p(l|t_{0:N}, v_{0:N}, e_{0:N}, \{\theta^j\}, \alpha) = \sum_{j=1}^{M+1} p(l|j, \theta^j) p(j|t_{0:N}, v_{0:N}, e_{0:N}, \theta^j, \alpha) \tag{4.10}$$

where $p(l|j, \theta^j)$ and $p(j|t_{0:N}, v_{0:N}, e_{0:N}, \theta^j, \alpha)$ are defined in sections 4.1.2 and 4.1.2.[4] Throughout a test run, we maintain a distribution over l, and after we observe a cycle of data we shift the distribution by -1 to simulate the amount of life degradation that occurred during the previous cycle.

4.2 Empirical Results

We tested our battery health model on a set of 13 non-standard Roomba batteries provided by iRobot for research purposes. Of these, 6 had relatively short lives (under 400 cycles), 5 had long lives (over 600 cycles), and 2 were in-between. The errors in the predicted cycles-to-death were computed using leave-one-out cross-validation, where the trajectories for one entire battery's run were held out each time. We chose to hold out an entire battery—rather than a random subset of cycles—because trajectories coming from the same battery tend to be more similar to each other than trajectories coming from different batteries; having trajectories from the same battery in both test and training sets would provide an unrealistic advantage to any algorithm. We normalized the time, voltage, and temperature values of all the cycles by subtracting the mean to avoid numerical problems during inference. Additionally, we set $\sigma_l = 25$ from Equation 4.9.

We compared our approach to four simple baselines. The naive approach estimated the remaining battery life by simply subtracting the number of elapsed cycles from the mean lifetime of the batteries in the training set. For the three remaining baselines, we first fit a curve to each temperature and each voltage trajectory using the parametric curves described in Section 4.1.2. This fit gave us a feature vector of trajectory parameters for each cycle that we could use as input to predict the remaining life l. We tested three baseline predictors: a linear regression of the form $l \sim \beta x$; k-nearest neighbor, which found the k nearest inputs x and returned their average time to death l; and k-means, which first formed k clusters based on the inputs x and then

[4]We also considered a Gaussian distribution for $p(l|j, \theta^j)$ but, due to the multimodal nature of $p(l|j, \theta^j)$, we found the empirical distribution a much better fit to the data.

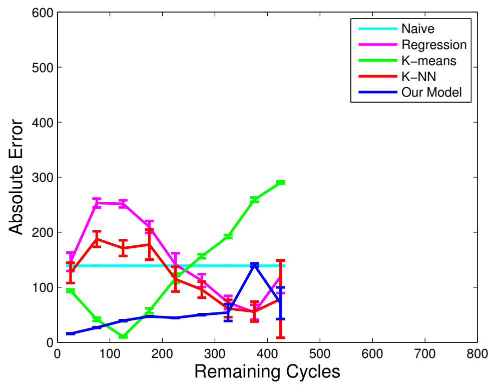
returned the mean l of the nearest cluster. For k-means and k-nearest neighbors, we tested values of k ranging from 1 to 300 (note that the entire dataset consisted of 4683 cooling trajectories). Each run of k-means had 10 restarts to avoid local optima.

4.2.1 Absolute Prediction Error

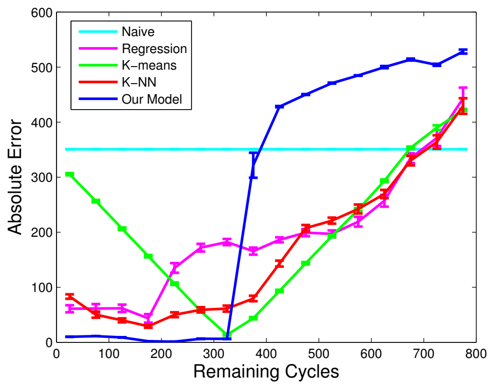
We first considered the absolute prediction error as a measure of prediction quality—after all, an ideal model would provide the user of the device an accurate picture of how many cycles are remaining in the battery's life. Figures 4-4(a) and 4-4(b) show the predictions of all the approaches on a typical short life and a typical long life battery, respectively. We chose the best-performing value of k for each plot, and the error bars show one standard error of the mean for blocks of 50 trajectories.

Our approach generally outperforms the baselines when predicting the remaining life in the short life battery. While we initially do poorly in predicting the remaining life in the long-life battery, we outperform the baselines as the long-life battery gets closer to failure. Thus, for both types of batteries, we predict the cycles to death well in regimes that matter most: when the battery is actually near death. From manual inspection, we note that the cooling behaviors near battery death tend to be more distinctive; we also have more data in this regime because every battery—long life or short life—has trajectories that are within 300 cycles of death. Only the long-life batteries have trajectories that are over 600 cycles out from death.

We see this trend when we consider all of the batteries from our complete cross-validation test. As before, the best performing values for k are plotted for the k-means and k-nearest neighbor. We see in Figure 4-5(a) that our approach outperforms the baselines when the actual remaining cycles is relatively small (roughly within the lifetime of a short-life battery). All approaches have difficulty with predictions when the battery is farther away from failure (Figure 4-5(b)).

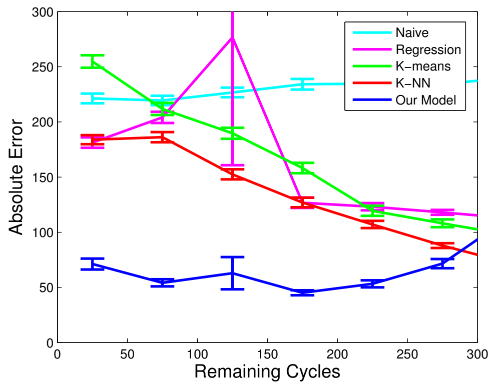


(a) Short-life batteries

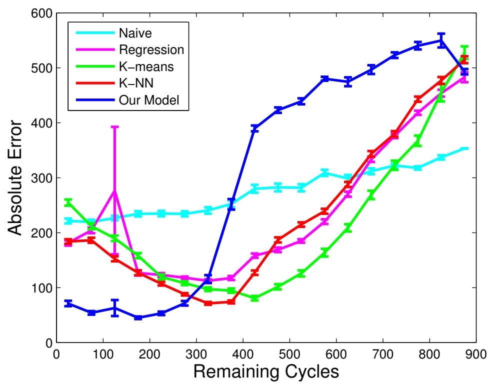


(b) Long-life batteries

Figure 4-4: Mean absolute prediction errors for the different types of batteries.



(a) Close to death

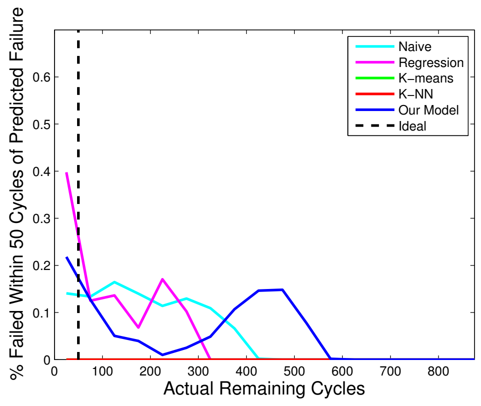


(b) Entire battery life

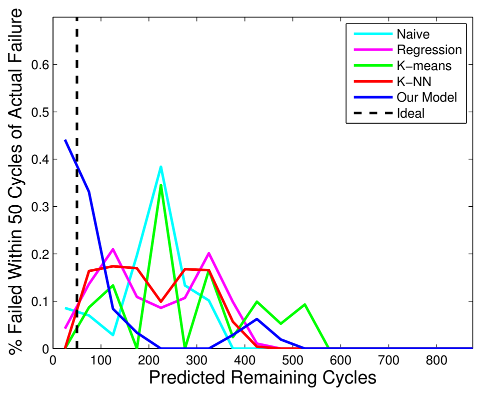
Figure 4-5: Mean absolute prediction errors across all batteries. Note that the model had many fewer training points in regimes more than 500-600 cycles to failure, since many of the batteries were short-life batteries.

4.2.2 Guiding Replacement Decisions

While knowing the number of cycles left in a battery is a useful figure, remaining life is usually an intermediate quantity needed to make the key decision of whether the battery should be replaced. When the lifetime predictions have error, the user of the device risks either waiting too long to replace the battery—and having it fail unexpectedly—or replacing the battery too early due to a false alarm. In this section, we analyze the predictions to determine how our model and the various baselines performed in this trade-off.



(a) Early battery replacement



(b) Late battery replacement

Figure 4-6: Risk of early and late battery replacement.

Figures 4-6(a) and 4-6(b) show how each of the models handle these trade-offs. Figure 4-6(a) plots how often the models predicted that a battery had less than 50 cycles remaining (an arbitrary threshold for replacement) against the actual remaining battery life; peaks far from 50 correspond to false alarms where the battery might have been replaced even when it had significant life left. Figure 4-6(b) plots how often the battery actually had less than 50 cycles remaining against the predicted battery life; peaks far from 50 correspond to situations when a battery might have failed despite a long predicted remaining life. Our model has peaks near 50 in both plots; our main source of error is confusing a short-life battery for a long-life battery or vica-versa (as seen in humps around 350-400). The other models seem to find less structure in the data; their errors are more spread out. (Note that k-nearest neighbor never predicts less than 50 cycles remaining.)

In the previous plots we created an arbitrary threshold of 50 cycles as when a battery ought to be replaced. We varied the threshold w to illustrate the trade-off more generally. Figure 4-7 plots false positive rates—how often did we predict that battery had more than w cycles when it had less than w cycles—against true positive rates—how often did we predict that the battery had less than w cycles when it actually had less than w cycles left. The ideal predictor would have points close to the top-left corner, corresponding to high true positive and low false positive rates; our approach achieves relatively high true positive rates without high false

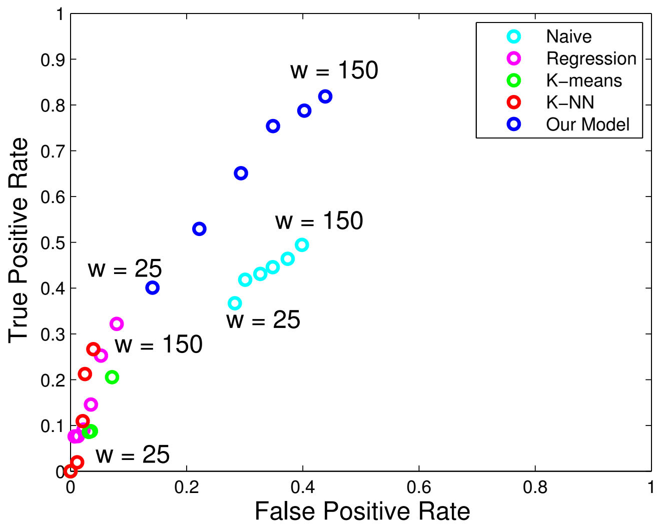


Figure 4-7: False positive rate (how often the battery was replaced too early) versus true positive rate (how often the battery was replaced in time) for various decision thresholds w.

positive rates for a variety of thresholds w. However, we also see that our model is somewhat conservative: if avoiding false positives (replacing too early) is more important than finding true positives (replacing before death), then the baselines might offer better options. Operationally, false negatives in predicting battery death are typically dangerous for robots but the desired performance is usually domain dependent.

4.3 Conclusions and Discussion

In Chapters 3 and 4 we answered the question: For complex environments where we have limited domain knowledge, how may we grow our model? We used Dirichlet processes as the technical centerpiece for allowing both our agent mobility model (Chapter 3) and our battery health model (Chapter 4) to grow. A lingering question is: What happens if these model classes (as flexible as they may be) are too limited to capture the true underlying dynamics? What we see in Chapter 5 is that, even with infinite data, these model classes may perform arbitrarily poorly when they cannot capture the true underlying dynamics. We learn that this is not necessarily due to the model classes themselves but the maximum a posterior approach used for model selection. The next chapter introduces Reward Based Model Search, an approach which remedies the problem of arbitrarily poor performance when using both limited parametric and nonparametric model classes.

4.3.1 Broken Exchangeability Assumption of DPs

As discussed in Section 2.2, a core assumption of the DP model is that the components are exchangeable. For DP applications on real-world data that will generally never be true. To see this, consider the taxi data described in Section 3.1. Say that during the process of learning we have found two mobility patterns around the greater Boston area. If the exchangeable assumption were to hold, these two previous mobility patterns should tell us nothing about the parametrization of the third mobility pattern – but they do. They contain a lot of information about the road network and we also know that the paths from the third mobility pattern will follow this same road network. Therefore the parametrization of the third component is not independent of the first two.

The technical reason why this happens is that the base distribution of the GP, H in Figure 2-2, is incorrect. For the exchangeability assumption to hold it needs to be a distribution on the flow on the greater Boston area's road network, not just over general flow fields. For the priors used in Chapter 3, we found that this error did not greatly impact performance but it came at the cost of requiring additional data to overcome the flat prior. We observed in Chapter 3 that the DPGP model did learn quickly so the use of the prior must not have come at too great an expense.

When the error introduced by using an incorrect base distribution begins to greatly impact performance there are two clear options we may use to overcome it. The first is to place a prior over H and infer H alongside the DP model from the data. By placing a prior over H we allow it to capture a larger range of distributions. A second approach is to increase the representation power of the Dirichlet Process by using, for example, the Dependent Dirichlet Process to model the correlation between the different components [Caron et al., 2007].

Chapter 5

Reward Based Model Search

In Chapters 3 and 4 we presented extremely flexible models that can grow as more data is seen. The hope of growing the model is that it will eventually capture the true world dynamics "close enough" that the resulting policy will perform well. Unfortunately, when the model class cannot capture the true dynamics the resulting model, from approaches like those in Chapters 3 and 4, may perform arbitrarily poorly. In this chapter we attempt to answer the question: How should we pick a model when the model class is too limited to capture the true world dynamics? To begin answering this question, we will discuss a real-world system which confronts us with the problem of having to plan with limited model classes.

Controlling a system with complex, unknown dynamics (*e.g.*, the interaction of a robot with a fluid) is a common and difficult problem in real-world domains. The hydrodynamic cart-pole, shown in Figure 5-1 is such an example, constructed to capture many of the fundamental challenges associated with fluid interaction (described in greater detail in Section 5.3). The system is composed of a thin flat plate (the pole) placed in a flowing channel of water with a linear actuator (the cart) attached to the pole's trailing edge via a pin joint. For this system, the objective is to learn a policy that stabilizes the pole pointing into the water current.

For real-world systems such as the hydrodynamic cart-pole, this trade-off means the chosen representation will often be *misspecified* (*i.e.*, the approximate representation cannot exactly capture the true dynamics). Such models can introduce *repre-*

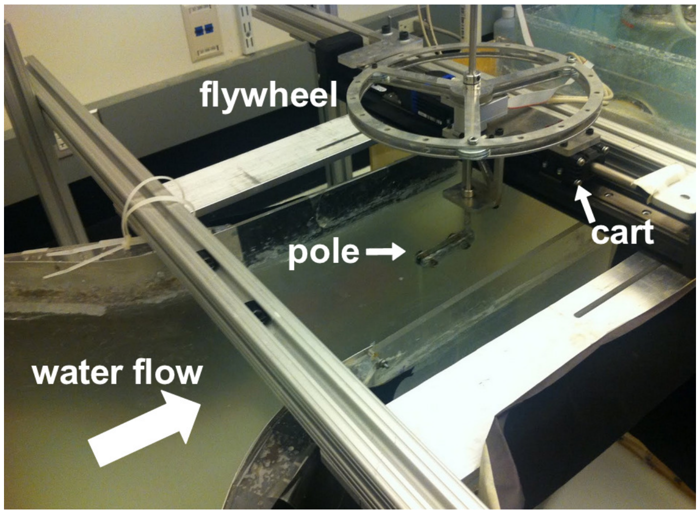


Figure 5-1: The hydrodynamic cart-pole system with the pole pointing upstream into the water current.

sentational bias, the difference between the performances of the true optimal policy and the best policy found based on the misspecified model class. Furthermore the learning algorithm using such model classes can introduce *learning bias*, which is the the difference between the performances of the best possible policy in the model class and the policy that is produced by the learner [Kalyanakrishnan and Stone, 2011].

The focus of this work is on identifying the cause of learning bias of MB RL, regardless of the model class, and presenting an algorithm to overcome it. For a given problem, typical MB RL algorithms choose a representation from the class of potential representations by minimizing a form of error measured on the training data (*e.g.*, maximum likelihood). Unfortunately, when no representation in the chosen class can capture the true representation, the fit by the standard minimum error metrics (Section 2.1.2) does not necessarily result in the highest possible performing policy. In other words, by optimizing a quantity other than the performance (*e.g.*, prediction error), the standard approaches introduce learning bias.

To address the issue of learning bias, Section 5.1 introduces Reward Based Model Search (RBMS), a batch MB RL algorithm that estimates the performance of models in the model class and explicitly searches for the model that achieves the highest performance. Given infinitely dense data and unlimited computation, Section 5.2

shows both that the asymptotic learning bias of RBMS for *any* representation class is zero and a probabilistic bound for limited data. Section 5.3 empirically demonstrates that RBMS often results in a large performance increase over minimum error on two common RL benchmark problems and on the real-world hydrodynamic cart-pole system.

5.1 Algorithm

Here we briefly reintroduce a portion of the content from Section 2.1 on finite time Markov Decision Processes (MDPs) with slightly different notation in order to clarify the analysis presented later in this chapter. An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{W}, m, \rho, s^{start}, T)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{W}$ is the disturbance space[1], $m : \mathcal{S} \times \mathcal{A} \times \mathcal{W} \rightarrow \mathcal{S}$ is the deterministic dynamics model, $\rho : \mathcal{S} \rightarrow \mathbb{R}$ is the reward function, $s^{start} \in \mathcal{S}$ is the starting state, and T is the maximum length of an episode[2]. For a policy[3], $\pi : \mathcal{S} \rightarrow \mathcal{A}$, we define its return[4] as

$$V(\pi) = \int_{w_{0:T-1}} \left[\rho(s^{start}) + \sum_{t=0}^{T-1} \rho(m(s_t, \pi(s_t), w_t)) \right] \; p(w_{0:T-1}) \; dw_{0:T-1} \tag{5.1}$$

$$= E_{w_{0:T-1}} \left[\sum_{t=0}^{T-1} \rho(s_t) \middle| \, s_0 = s^{start}, \; s_{t+1} = m(s_t, \pi(s_t), w_t) \right]. \tag{5.2}$$

We call the sequence $e_\pi = \{s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}\}$ an episode of data where $s_{t+1} = m(s_t, a_t, w_t)$ and $a_t = \pi(s_t)$.

Throughout this chapter we assume that the dynamics model, reward function,

[1]The disturbance space is introduced so we may assume the dynamics model is deterministic and add noise through w. This is primarily done to facilitate theoretical analysis and is equivalent to the standard RL notation which uses a stochastic dynamics model that does not include w.

[2]For simplicity we assume ρ and $s_0 = s^{start}$ are known and deterministic and ρ is only a function of the current state. Without loss of generality this allows us to write V as a scalar everywhere. This work can be straightforwardly extended an unknown and stochastic ρ and s^{start} and the more general $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$.

[3]For computational reasons, in this chapter we focus on learning time-invariant policies for finite horizon problems. Although we use a single policy across all time steps, the algorithm and bound presented in this chapter are equally applicable to time-varying policies. See Kakade [2003], Bagnell et al. [2003] for a discussion on the use of time-invariant policies in finite horizon problems.

[4]$V(\pi)$ is shorthand for $V^\pi(s^{start})$, the expected sum of rewards from following policy π from state s^{start}.

and policy are Lipschitz continuous with constants L_m, L_ρ, and L_π, respectively Fonteneau et al. [2012], where

$$||m(s,a,w) - m(s',a',w)||_{\mathcal{S}} \leq L_m(||s-s'||_{\mathcal{S}} + ||a-a'||_{\mathcal{A}}),$$

$$|\rho(s,a,w) - \rho(s',a',w)| \leq L_\rho(||s-s'||_{\mathcal{S}} + ||a-a'||_{\mathcal{A}}),$$

$$||\pi(t,s) - \pi(t,s')||_{\mathcal{A}} \leq L_\pi ||s-s'||_{\mathcal{S}},$$

$$\forall (s,s',a,a',w) \in \mathcal{S}^2 \times \mathcal{A}^2 \times \mathcal{W},\ \forall t \in \{0,..,T-1\}.$$

The objective of an MDP is to find the policy

$$\pi^* = \arg\max_{\pi \in \Pi} V(\pi) \tag{5.3}$$

from a given policy class, Π. Typically in RL the dynamics model, m, is unavailable to us and therefore we are unable use Equation 5.2 to solve Equation 5.3.

Section 2.1.2 discussed that minimum error techniques in RL are vulnerable to learning poor policies when the representations for the dynamics model is not expressive enough to capture the true dynamics. This section presents a novel batch MB RL approach that learns a representation which explicitly maximizes Equation 2.1. We treat the dynamics model class as a parametrization of the policy and search for $\hat{\theta} \in \Theta$ that achieves the highest return, rather than optimizing a different metric (*e.g.*, maximum likelihood). We can think of the policy as being indirectly parametrized by θ which defines the model and is updated using a policy gradient approach, such that

$$\hat{\theta} = \hat{\theta} + c\frac{\partial V^{\pi^\theta}(s_0)}{\partial \hat{\theta}}, \tag{5.4}$$

where $c > 0$.

Section 5.1.1 outlines a method for estimating the return of a policy in continuous state spaces called Model-Free Monte Carlo (MFMC) [Fonteneau et al., 2010]. MFMC estimates a policy's return directly from data using Equation 2.3 rather than from a dynamics model learned from the data. Section 5.1.2 describes a method of gradient

Algorithm 4: MFMC Policy Return Estimation

```
  Input: $\pi,\ \mathcal{D},\ s_0,\ \Delta,\ T,\ p$
  Output: $V^{\pi}_{MFMC}(s_0)$
1 for $n = 1$ to $p$ do
2     $\tilde{s} \leftarrow s_0,\ e_n \leftarrow [\ ]$
3     for $t = 0$ to $T$ do
4         $\tilde{a} \leftarrow \pi(\tilde{s})$
5         $(s^n_t, a^n_t, s^n_{t+1}) \leftarrow \arg\min_{(s,a,s')\in\mathcal{D}} \Delta((\tilde{s},\tilde{a}),(s,a))$
6         $\mathcal{D} \leftarrow \mathcal{D} \setminus (s^n_t, a^n_t, s^n_{t+1})$
7         Append $(s^n_t, a^n_t, s^n_{t+1})$ to $e_n$
8         $\tilde{s} \leftarrow s^n_{t+1}$
9 return $V^{\pi}_{MFMC}(s_0)$ (computed using Equation 2.3)
```

ascent in order to maximize Equation 2.1.

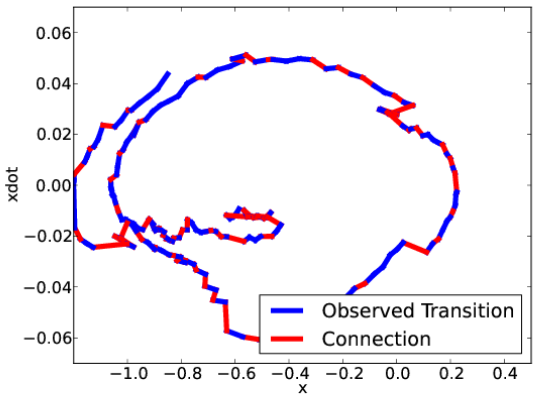
5.1.1 Off-Policy Policy Evaluation

Ideally, we would choose policies using Equations 2.3 and 2.1 with on-policy data. This results in a sample complexity at least linear in the number of evaluated policies, an impractical amount for most real-world problems. We can reduce this sample complexity by sharing data across iterations, using *off-policy* data – data generated under a policy different from the one we are evaluating. Therefore, we need an approach that allows us to perform MC-like policy evaluation from off-policy data.

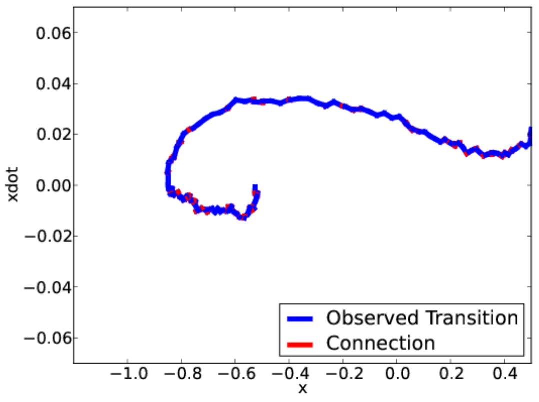
In continuous state spaces, creating on-policy episodes from off-policy data is challenging because a single state is rarely visited more than once. We use Model-free Monte Carlo (MFMC) [Fonteneau et al., 2010] for policy evaluation from off-policy data in continuous state spaces. For a policy, π^θ, MFMC creates pseudo on-policy episodes from off-policy training data to compute statistics of the policy with bounded estimation error.

To construct an episode of on-policy data, MFMC uses a set of training data and a distance function. We define the data set as $\mathcal{D} = \{(s_i, a_i, s'_i)\}_{i=0}^{|\mathcal{D}|}$, where $s'_i \sim m(s_i, a_i, \cdot)$. The designer-provided distance function takes the form

$$\Delta\left((s_i, a_i), (s_j, a_j)\right) = ||s_i - s_j||_S + ||a_i - a_j||_A, \qquad (5.5)$$



(a) 5 episodes of data



(b) 50 episodes of data

Figure 5-2: Phase space plots for pseudo on-policy episodes constructed for a well performing policy using the MFMC algorithm with 5 and 50 episodes of batch data.

where i and j are one-step transitions from $\mathcal{D}$. MFMC constructs episodes starting at s_0 and sequentially adds transitions such that at time t, when the agent is in state $\tilde{s}$, the next transition $(s_t, a_t, s_{t+1}) = \arg\min_{(s,a,s')\in\mathcal{D}} \Delta((\tilde{s}, \pi^\theta(\tilde{s})), (s, a))$. Each transition in $\mathcal{D}$ can only be used once and episodes are terminated after T transitions.

MFMC also requires p, the number of episodes used for policy evaluation. The decision of p, relative to $|\mathcal{D}|$, trades off bias and variance (see [Fonteneau et al., 2010] for analysis, bounds on bias and variance, and discussion on the trade-off). Algorithm 4 is a reproduction of MFMC from Fonteneau et al. [2010] for the reader's convenience.

A visualization the of episode construction procedure is shown in Figure 5-2 for the mountain car domain (described in Section 5.3) where the agent begins near the center of the diagram and attempts to reach $x \geq 0.5$. The figure shows two phase-space plots of constructed episodes, where transitions observed in the data set are shown in blue, and are connected by red lines for illustration purposes. Episodes were generated using a well performing policy that reaches the goal. Figures 5-2(a) and 5-2(b) show two episodes constructed based on 5 and 50 episodes, respectively. Actual observed interactions are shown in blue while discontinuities in tailoring the episodes are highlighted as red segments. Notice that more data leads to an overall smoother prediction of the system evolution and the episode more accurately approximates an on-policy sequence of interactions.

5.1.2 Policy Improvement

The second step to solving Equation 2.1 is the maximization over Θ. As discussed in Section 2.1, enumerating and evaluating all possible parameter values is impractical for real-world problems. Policy gradient approaches overcome this hurdle by updating the parameter value in the direction of increasing performance (Equation 5.4).

We take the policy gradient type approach as described in Section 2.1, only we are updating the parameters of the dynamics model rather than a parametrization of the policy. Unfortunately, gradient ascent is known to be susceptible to local maxima. To reduce the likelihood of becoming stuck in a poor local maxima, we use random restarts in addition to including the maximum likelihood solution in the set of potential starting points (see Section 5.2). For this work, we chose the basic form of gradient ascent for policy improvement described in Section 5.1.[5]

Our overall approach is presented in Algorithm 5. The inputs to the algorithm are the data, $\mathcal{D}$, our representation, Θ, the initial gradient ascent step size, δ^{init}, and the minimum gradient ascent step size, δ^{min}.

RBMS begins gradient ascent by selecting a dimension of the model space to search and computes the policies for models θ, $\theta + \delta$, and $\theta - \delta$ (line 5), where the parameter δ is the step size in the model space. Once the policies are computed, we use MFMC to estimate the return of each of these policies (line 6) and estimate the difference in return between the current best model θ and the models $\theta + \delta$ and $\theta - \delta$. If neither of these models yield an increase in the estimated return, we continue to the next dimension. If $\theta + \delta$ or $\theta - \delta$ are positive steps in terms of estimated return, we take the step in the steepest direction. When all dimensions have been stepped through, δ is decreased (line 13) and the process is continues until δ reaches δ^{min}. The algorithm is then repeated for many random restarts, including the ML parameters.

[5]While more intelligent optimization methods [Bertsekas, 1999] were considered, we found that this basic form of gradient ascent was sufficient for our optimization needs and rarely encountered difficulties with local maxima.

Algorithm 5: RBMS

```
   Input: $\mathcal{D}$, $\tilde{\Theta}$, $\delta^{init}$, $\delta^{min}$
   Output: $\pi^{\theta_{RBMS}}$
1  $V^{\pi^{\theta_{start}}}(s_0) \leftarrow -\infty$, $\theta \leftarrow \theta_{start}$, $\delta \leftarrow \delta^{init}$
2  while $\delta > \delta^{min}$ do
3      for each dimension of $\Theta$ do
4          while True do
5              $\{\pi^{\theta-}, \pi^{\theta}, \pi^{\theta+}\} \leftarrow$ Compute the policies for $\{\theta - \delta, \theta, \theta + \delta\}$
6              $\{V^{\pi^{\theta-}}(s_0), V^{\pi^{\theta}}(s_0), V^{\pi^{\theta+}}(s_0)\} \leftarrow$ Estimate the return of
                   $\{\pi^{\theta-}, \pi^{\theta}, \pi^{\theta+}\}$
7              if $\max(V^{\pi^{\theta+}}(s_0), V^{\pi^{\theta-}}(s_0)) \leq V^{\pi^{\theta}}$ then
8                  Break
9              if $V^{\pi^{\theta+}}(s_0) > V^{\pi^{\theta-}}(s_0)$ then
10                 $\theta \leftarrow \theta + \delta$
11             else
12                 $\theta \leftarrow \theta - \delta$
13     $\delta \leftarrow \delta/2$
14 return $\pi^{\theta}$
```

5.2 Theoretical Analysis

As described in Section 5.1.1, MFMC constructs a set of on-policy episodes from off-policy data and uses Equation 2.3 to estimate the return of π^{θ}. This section provides the formal analysis which shows that with infinitely dense data and unlimited computation, RBMS is guaranteed to find the highest performing model from the model class (Section 5.2.1). We relax these assumptions in Section 5.2.1 and guarantee that using RBMS after finding the ML solution will never result in worse performance in expectation. Section 5.2.2 presents a probabilistic bound on estimation error.

5.2.1 Bound on Expected Performance

Naively, using Equation 5.9 to approximate and solve Equation 5.3, would require N data for each $\pi \in \Pi$, an infinite amount of data for infinite policy classes. Off-policy policy evaluation aims to alleviate this issue by estimating $V(\pi)$ using episodes $e^1_{\pi_1}, e^2_{\pi_2}, \ldots, e^N_{\pi_N}$ where $\pi_1, \ldots, \pi_N$ may be different than π. To perform off-policy evaluation, we use an approach called Model-Free Monte Carlo-like policy evaluation (MFMC) [Fonteneau et al., 2012], which attempts to approximate the estimator from Equation 5.9 by piecing together artificial episodes of on-policy data from off-policy, batch data.

Consider a set of data $\{e^1_{\pi_1}, e^2_{\pi_2}, \ldots, e^N_{\pi_N}\}$, which we re-index as one-step transitions $\mathcal{D} = \{(s_0, a_0, s_1), \ldots, (s_{NT-1}, a_{NT-1}, s_{NT})\}$. To evaluate a policy, π, MFMC uses a distance function

$$\Delta((s,a),(s',a')) = ||s - s'||_{\mathcal{S}} + ||a - a'||_{\mathcal{A}}$$

and pieces together $\tilde{N}$ artificial episodes from $\mathcal{D}$ such that $\tilde{e}^n_\pi = \{\tilde{s}^n_0, \tilde{a}^n_0, \ldots, \tilde{s}^n_T, \tilde{a}^n_T\}$ is an artificial on-policy episode approximating an episode π for $n = 1, \ldots, \tilde{N}$. To construct $\tilde{e}^n_\pi$, MFMC starts with $\tilde{s}^n_0 = s^{start}$ and for $t = 1, \ldots, T$ we find

$$\tilde{s}^n_{t+1} = \arg \min_{s':(s,a,s') \in \mathcal{D}} \Delta((s,a),(\tilde{s}^n_t, \tilde{a}^n_t)) \tag{5.6}$$

where $\tilde{a}^n_t = \pi(\tilde{s}^n_t)$ and once a transition, (s, a, s'), is chosen using Equation 5.6 it is removed from $\mathcal{D}$. Following the construction of episodes $\tilde{e}^1_\pi, \ldots, \tilde{e}^{\tilde{N}}_\pi$, MFMC estimates the return of π using

$$V_{MFMC}(\pi; \mathcal{D}) = \frac{1}{\tilde{N}} \sum_{n=1}^{\tilde{N}} \sum_{t=0}^{T-1} \rho(\tilde{s}^n_t). \tag{5.7}$$

We can bound the return using Theorem 4.1, Lemma A.1, and Lemma A.2 of Fonteneau et al. [2010] and say that

$$V(\pi) \geq E_{\mathcal{D}}\left[V_{MFMC}(\pi)\right] - d(\pi; \mathcal{D}) \tag{5.8}$$

where

$$d(\pi; \mathcal{D}) = \max_{n \leq \tilde{N}} \sum_{t=0}^{T-1} L_{T-t} \delta^n_t, \quad \delta^n_t = \min_{s':(s,a,s') \in \mathcal{D}} \Delta((s,a),(\tilde{s}^n_t, \tilde{a}^n_t))$$

for each $\tilde{s}^n_{t+1}$ chosen using Equation 5.6, and

$$L_{T-t} = L_\rho \sum_{i=0}^{T-t-1} [L_m(1 + L_\pi)]^i.$$

The term $d(\pi; \mathcal{D})$ is the maximum deviation between the true return of any policy and the expected MFMC estimate of return of that policy. See Fonteneau et al. [2010, 2012] for a discussion regarding the choice of Δ and $\tilde{N}$.

Infinitely Dense Data and Unlimited Computation

Solving Equation 2.1 is difficult because of two complex calculations: accurately estimating $V^{\pi^\theta}(s_0)$ and performing the optimization over policies. Equation 5.8 tells us that as the density of the data increases, our estimate $\hat{V}^{\pi^\theta}(s_0) \to V^{\pi^\theta}(s_0)$.

With unlimited computation, we have the ability to search over the entire representation space Θ to find $\theta^* = \arg\max_{\theta \in \Theta} V^{\pi^\theta}(s_0)$.[6] This implies that with infinitely dense data and unlimited computation, RBMS is guaranteed to find θ^*, the highest performing model from the model class.

Limited Data and Computation

While the guarantee of finding θ^* is a nice theoretical property (Section 5.2.1), for real-world domains we can never expect to have infinitely dense data nor unlimited computation. Here we relax both assumptions and guarantee that using RBMS after finding the ML solution will never result in worse performance in expectation.

With limited computation, we no longer have the ability to perform the maximization over Θ and instead resort to gradient ascent. Due to the non-convexity of $V^{\pi^\theta}(s_0)$ as a function of θ, we can no longer guarantee that θ^* is found. By initializing gradient ascent with θ_{ML}, the ML model, we guarantee that RBMS takes steps in the direction of improving on the ML solution. In addition to θ_{ML}, we also initialize gradient ascent with many random starting locations drawn from Θ to further increase our chances of improving on the ML solution.[7]

The consequence of limited data is that $\hat{V}^{\pi^\theta}(s_0)$ can be an inaccurate estimate of $V^{\pi^\theta}(s_0)$ and cause gradient steps to decrease $V^{\pi^\theta}(s_0)$. Hence, one could imagine using

[6]This is trivially accomplished for discrete spaces and can be done to an arbitrary precision in continuous spaces.

[7]In the analysis, θ_{ML} can be trivially replaced by any prediction error metric (e.g., minimum squared error).

the bound from Equation 5.8 and only step from θ_i to θ_{i+1} if $\hat{V}^{\pi^{\theta_{i+1}}}(s_0) \geq \hat{V}^{\pi_i^\theta}(s_0) + d(\pi;\mathcal{D})$. This would ensure that the step from θ_i to θ_{i+1} result in $V^{\pi^{\theta_{i+1}}}(s_0) \geq V^{\pi^{\theta_i}}(s_0)$ and consequently that $V^{\pi^{\theta_{RBMS}}}(s_0) \geq V^{\pi^{\theta_{ML}}}(s_0)$, where θ_{RBMS} is the model selected by RBMS. Unfortunately, this bound only holds in expectation over the data. For future work, we plan develop a probabilistic bound based on Hoeffding's inequality for MFMC to ensure that gradient steps increase $V^{\pi^\theta}(s_0)$ with high probability.

5.2.2 Probabilistic Bound on Performance for MFMC

Given a set of N episodes of data $\{e_\pi^1, e_\pi^2, \ldots, e_\pi^N\}$ collected using π, we can estimate $V^\pi(s_0)$ using empirical return (analogous to Equation 2.12) where

$$V_\pi^{emp}(s_0;\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{T-1}\rho(s_t^n), \qquad (5.9)$$

$$V^\pi(s_0) \geq V^{emp}(\pi;\mathcal{D}) - (B-A)\sqrt{\frac{-\ln\delta}{2N}}, \qquad (5.10)$$

$e_\pi^n = \{s_0^n, a_0^n, \ldots, s_{T-1}^n, a_{T-1}^n\}$, $s_{t+1}^n = m(s_t^n, a_t^n, w_t^n)$, $a_t^n = \pi(s_t^n)$, and Equation 5.10 holds with probability $1-\delta$ by Hoeffding's inequality [Hoeffding, 1963]. This approach, where episodes are generated on-policy and then the return is estimated using Equation 5.9, is called Monte Carlo policy evaluation [Sutton and Barto, 1998]. Since we do not make any assumptions about the policies that will be evaluated nor the policy under which the data was generated, we cannot directly use Equations 5.9 and 5.10 but we will build upon them in the following sections.

Unfortunately, the bound provided in Equation 5.8 only allows us to bound the return using the *expectation* of the MFMC estimate, not the *realized* estimate based on the data. We present such a bound, beginning with Hoeffding's inequality [Hoeffding, 1963],

$$e^{2N(\epsilon - d(\pi;\mathcal{D}))^2/(B-A)^2} > \Pr\left[V_{MFMC}(\pi;\mathcal{D}) - E[V_{MFMC}(\pi)] + d(\pi;\mathcal{D}) > \epsilon\right] \qquad (5.11)$$

$$\geq \Pr\left[V_{MFMC}(\pi;\mathcal{D}) - V(\pi) > \epsilon\right] \qquad (5.12)$$

where we move from Equation 5.11 to Equation 5.12 using Equation 5.8. Setting $\delta = e^{2N(\epsilon - d(\pi;\mathcal{D}))^2/(B-A)^2}$, solving for ϵ, and substituting the quantity into Equation 5.12 we see that with at least probability $1-\delta$

$$V(\pi) \geq V_{MFMC}(\pi;\mathcal{D}) - d(\pi;\mathcal{D}) - (B-A)\sqrt{\frac{-\ln\delta}{2N}}. \tag{5.13}$$

While Equation 5.13 is useful for bounding the return estimate using MFMC, in Section 6 we will require a bound between $V_{MFMC}(\pi;\mathcal{D})$ and $V^{emp}(\pi)$. By combining Equations 5.10 and 5.13, we have with probability $1-2\delta$

$$V^{emp}(\pi;\mathcal{D}) \geq V_{MFMC}(\pi;\mathcal{D}) - d(\pi;\mathcal{D}) - 2(B-A)\sqrt{\frac{-\ln\delta}{2N}}. \tag{5.14}$$

5.3 Empirical Results

The experiments described in this section highlight the performance improvement of Reward Based Model Search (RBMS) over maximum likelihood (ML) learners on the RL benchmark problems of mountain car and cart-pole and the real-world hydrodynamic cart-pole system. The two benchmark domains allow for easier understanding and more extensive evaluation to study how quickly the performance of ML and RBMS degrade as the model classes become increasingly misspecified. To study the different types of misspecification, Section 5.3.1 investigates both sharp misspecification (unmodeled discontinuities in the true dynamics) and irrelevant data (unhelpful data from a region of our state-space that well performing policies will not visit). Section 5.3.2 investigates unmodeled noise and smooth misspecification (*i.e.*, a gradual change in misspecification as the agent moves through the state-space).

We compare RBMS and ML with misspecified model classes to a large tabular Markov model fit using ML.We show the resulting performance versus the amount of training data to highlight the sample complexity advantage of RBMS using small model classes over tabular representations, despite being misspecified. For all model classes, policies were found by first finely discretizing the continuous model and performing value iteration using the same discretization for the value function [Sutton and Barto, 1998]. RBMS was run with $p = 10$ and, unless otherwise specified, used the distance function

$$\Delta\left((s,a)),(s',a')\right) = \begin{cases} \sum_{d=1}^{D} \frac{|s^d - s'^d|}{s^d_{max} - s^d_{min}} & \text{if } a = a' \\ \infty & \text{otherwise} \end{cases},$$

where s^d is the d-th dimension.

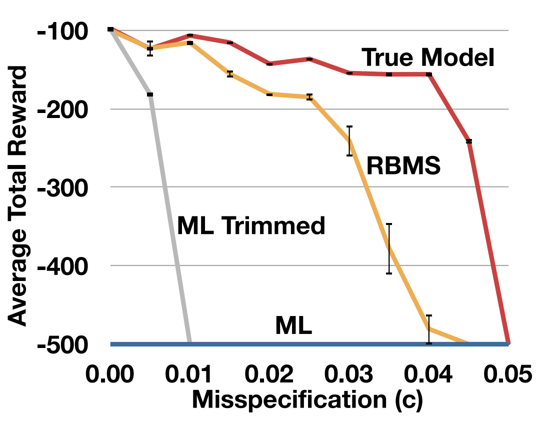
5.3.1 Mountain Car

Mountain car [Singh et al., 1996] is a standard RL benchmark simulation where the agent starts in a valley between two hills and takes actions to reach the top of the right hill and receives -1 reward for every time step the agent is not at the goal. Episodes end when the agent reaches the goal or after 500 steps. The standard mountain car dynamics are

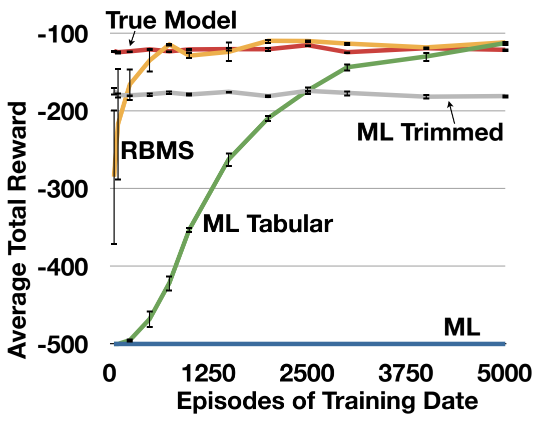
$$x_{t+1} = x_t + \dot{x}_t, \quad \dot{x}_{t+1} = \dot{x}_t + a + \theta_1 \cos(\theta_2 x)$$

with action $a \in \{-0.001, 0.001\}$. A uniform discretization of x and $\dot{x}$ with a 500×250 grid was used to represent value functions. Note the differing numbers of cells in each dimension were chosen to keep the representation small, because it is also used as the dynamics representation for the large tabular model approach. This ensured the large tabular model was made as competitive as possible in terms of data requirements.

To study how the two approaches' performances change as their model class becomes increasingly misspecified, we modified the standard dynamics in three ways. First, we added stochasticity to the car's starting location by uniformly sampling from the interval $[{-\pi}/6 - 0.1, {-\pi}/6 + 0.1]$. Second, we simulated a rock at $x = 0.25$, such that when the car hits the rock its velocity ($\dot{x}$) decreased by c. Therefore, increasing c corresponds to the standard car dynamics model class becoming increasingly misspecified. Third, we included a second valley to the left of the standard valley with significantly different dynamics (the goal is still on the right hill of the right valley). For the right valley we used $\theta_1^{right} = -0.0025$, $\theta_2^{right} = 3$ for the dynamics, as is com-



(a) Performance vs Misspecification



(b) Performance vs Data Size

Figure 5-3: Performance versus misspecification (a) and performance versus data size for $c = 0.005$ (b) for RBMS with the standard mountain car model (yellow), ML with the standard mountain car model (blue), ML with the standard mountain car model with irrelevant data removed (gray), ML with the large tabular Markov model (green), and the true model (red). The error bars show one standard error.

monly done in the literature [Sutton and Barto, 1998], and $\theta_1^{left} = -0.01$, $\theta_2^{left} = 3$ for the left valley. For this experiment ML and RBMS were given the standard mountain car dynamics model, parametrized by θ_1 and θ_2, with both hills sharing the same parameters. The training data was generated from uniformly random policies.

Figure 5-3(a) shows the results of ML (blue, gray) and RBMS (yellow) as their model class becomes increasingly misspecified (with increasing c) for 2500 episodes of training data. For reference, the return of the planner with the true model[8] is shown in red. The blue line shows that the standard ML approach never performs well due to the data from the left hill biasing its model estimate. While it may seem straightforward to remove irrelevant data before fitting a model (*e.g.*, data from the left valley), in general for difficult problems (*e.g.*, the hydrodynamic cart-pole), identifying regions of irrelevant data can be extremely difficult. Shown in gray is a demonstration that even if we were able to trim irrelevant data before using ML, other unmodeled effects (a small influence of the rock) still cause ML to perform poorly.

In contrast, RBMS is able to learn models which allow it to reach the goal for

[8]Note that the planner first discretizes the continuous model to find a policy, so the policy plotted for the "true model" is actual a finely discretized representation of the true continuous model.

a large range of misspecification both from the increasing effect of the rock and irrelevant data. Additionally, we can see that when the rock had little effect ($c \leq .005$), RBMS performed as well as the true model. Eventually, the influence of the rock is too significant for any policy to escape the valley, as shown by the red line eventually dropping to -500.

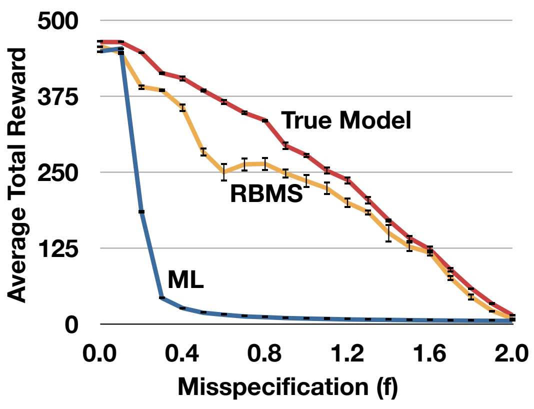
Figure 5-3(b) shows the results of ML (blue, gray), RBMS (yellow), the large tabular Markov model (green) as the amount of training data increases for fixed $c = 0.005$. For reference, the return of the planner with the true model is shown in red. This figure illustrates that less data is required to use a small model, as opposed to a large tabular Markov model with many parameters. Although the large tabular model does eventually achieve the performance of RBMS and the true model, such large data requirements are often prohibitive for real-world problems. Our approach, on the other hand, performs well using only 500 episodes, an order of magnitude reduction in the amount of training data compared to ML Tabular.

5.3.2 Cart-Pole

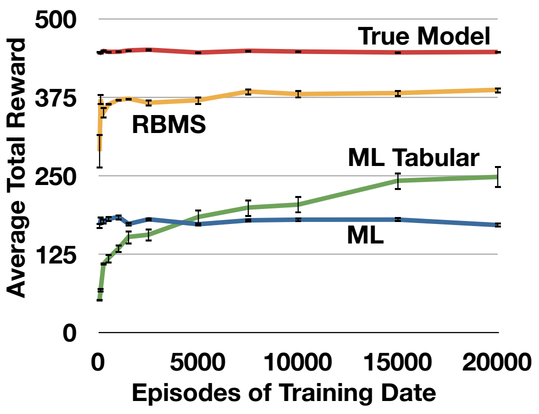
The cart-pole system [Spong, 1998] is composed of a cart that moves along a single axis with a pendulum (the pole) attached by a pin joint. We focus on the task of stabilizing the pole in an upright position against gravity.

The cart-pole deterministically starts at $x = \psi = \dot{x} = \dot{\psi} = 0$, where x is the position of the cart, ψ is the angle of the pole, and the action space $A = \{-5, 5\}$. An episode ends when the pole falls over (defined as $|\psi| > \pi/15$) or after 500 steps. The reward function is $1 - |\theta|/(\pi/15)$. The training data was generated from a uniformly random policy.

The dynamics were parametrized by three quantities: mass of the cart (m_c), mass of the pole (m_p), and the length of the pole (l) [Barto et al., 1983]. For the true dynamics, we chose $m_c = m_p = l = 1$ and introduced downward wind on the system which applied a force, f, in the direction of gravity (*i.e.*, increasing force as θ moves away from zero) causing θ to be displaced. We also added to zero mean noise on ψ with standard deviation of 0.01.



(a) Performance vs Misspecification



(b) Performance vs Data Size

Figure 5-4: Performance versus misspecification (a) and performance versus data size for $f = 0.2$ (b) for RBMS with the standard cart-pole model (yellow), ML with the standard cart-pole model (blue), ML with the large tabular Markov model (green), and the true model (red). The error bars show one standard error.

Similarly to the experiment described in 5.3.1, the purpose of this experiment is to understand how the approaches' performance changes as the model class becomes increasingly misspecified. For ML and RBMS, we used the standard cart-pole model class, parametrized by m_c, m_p, and l and represented the value function using of a $50 \times 30 \times 30$ grid over ψ, $\dot{x}$, and $\dot{\psi}$.

Figure 5-4(a) shows the results of ML (blue) and RBMS (yellow) as their standard cart-pole dynamics model class becomes increasingly misspecified (with increasing f) for 10,000 episodes of training data. For reference, the return of the true model is shown in red. The figure shows that the ML approach has difficulty coping with the model misspecification and for $f \geq 0.3$ the approach never learns a stabilizing policy. In contrast, RBMS is able to learn models resulting in policies that stabilized the pole for a much larger range of f. Eventually, the influence of the wind is too significant for any policy to stabilize the pole, as shown by the red line eventually dropping to near zero.

Figure 5-4(b) shows the results of ML (blue), RBMS (yellow), the large tabular Markov model (green) as the amount of training data increases for a fixed $f = 0.2$. Again, for reference, the return of the true model is shown in red. While it is not shown in the figure, the large tabular Markov model does eventually achieve a level of performance equal to that of the true model after approximately 50,000 episodes. This further illustrates that less data is required to use a small model, as opposed to a large tabular Markov model with many parameters. Our approach, on the other hand, achieves a high level of return after 500 episodes, two orders of magnitude reduction in the amount of training data compared to the ML Tabular, but may never achieve performance equal to that of the true model due to the limited representational power of the misspecified model.

5.3.3 Hydrodynamic Cart-Pole

The hydrodynamic cart-pole, shown in Figure 5-1, is a real-world, experimental, fluid analog of the cart-pole system. It is composed of a thin flat plate (the pole) attached at its trailing edge via a pin joint to a linear actuator (the cart) that is operated at 50 Hz. The flat plate (6.5 cm wide by 20 cm tall) is submerged in a channel (22 cm wide) of flowing water, with the flat plate placed at the "upright position." In this position, the water attempts to turn the wing downstream via a "weather vane" effect, making the upright position passively unstable and the downright position passively stable.[9] The system was operated at a Reynolds number of 15,000, considered to be an intermediate flow. As opposed to large and small Reynolds number flows, intermediate flows are particularly challenging due to the difficulty of modeling and simulating them.

Learning a policy which stabilizes the pole at the upright is difficult because the coupling between the motion of the cart and the motion of the pole is dictated by the complex dynamics of the fluid, which itself has many states and is highly nonlinear. Additionally, the flywheel mounted to the pole adds extra inertia to the system and there is a free surface which can add secondary effects when the pole moves violently through the water. Success on this system will demonstrate the robustness of the technique to the many uncertainties inevitably present in actual hardware, and

[9] Due to fluid effects, the downright position is not precisely stable as vortex shedding from the wing will cause it to oscillate slightly, but these are small deviations compared to those considered in the experiments in this chapter.

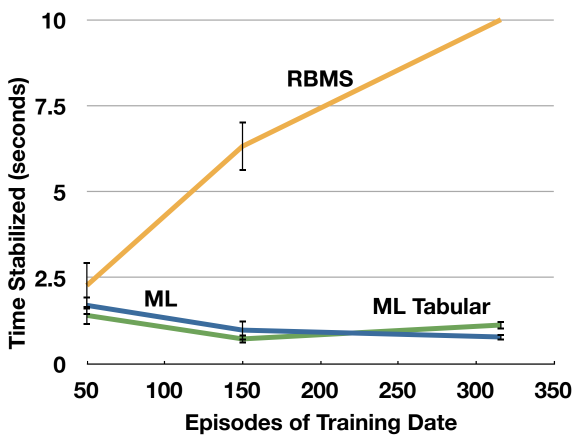


Figure 5-5: Time stabilized versus number of episodes for RBMS with the standard cart-pole model (yellow), ML with the standard cart-pole model (blue), and ML with a large tabular Markov model (green) on hydrodynamic cart-pole. The error bars represent one standard error.

particularly present in fluid systems.

Our goal is to learn a controller from training data that stabilizes the pole similar to the experiment performed in Section 5.3.2. We collected a data set of 316 episodes (approximately 45 minutes of data) from the hydrodynamic cart-pole system from a variety of hand-designed, poor controllers, none of which stabilized the pole for more than a few seconds and many of them quickly fell over. For the system, an episode was ended when the pole fell over (defined as $|\psi| > \pi/15$) or a maximum length of 500 steps (10 seconds) was reached. For our model class we used the standard cart-pole system, parametrized by m_c, m_p, l as a model class of this far more complex system. To represent the value function, we discretized the state space $[x, \psi, \dot{x}, \dot{\psi}]$ into $7 \times 11 \times 11 \times 11$ bins with an action space $A = \{-1, -0.9, -0.8, ..., 1\}$. We used a quadratic cost function and used the distance function

$$\begin{aligned}\Delta\left((s,a)),(s',a')\right) = \frac{|x-x'|}{x_{max}-x_{min}} + \frac{|\psi-\psi'|}{\psi_{max}-\psi_{min}} \\ + \frac{|\dot{x}-\dot{x}'|}{\dot{x}_{max}-\dot{x}_{min}} + \frac{|\dot{\psi}-\dot{\psi}'|}{\dot{\psi}_{max}-\dot{\psi}_{min}} + \frac{|a-a'|}{a_{max}-a_{min}}.\end{aligned}$$

Figure 5-5 shows the performance on the real system of the learned controllers for the standard cart-pole model fit using ML (blue), the standard cart-pole model fit using RBMS (yellow), and a large tabular Markov model fit using ML (green), where each controller was run 10 times and one standard error is shown on the figure. The results show that the 45 minutes of training data was sufficient to learn a controller which stabilized the system for the maximum amount of time in all 10 trials from the small representation. Neither ML with the misspecified model nor ML with the large Markov model were able to achieve any statistically significant improvement in performance. This experiment demonstrates that despite the misspecified representation, RBMS was still able to use limited off-policy data to perform well on an extremely complex system, because it focused on finding the model that achieves the highest performance rather than minimizing the prediction error.

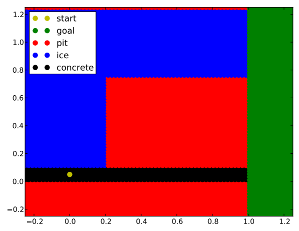
5.4 Bayesian Nonparametric Reward Based Model Search

In this section we expand on the work of Section 5.1 to apply RBMS to Bayesian nonparametric model (BNM) classes.

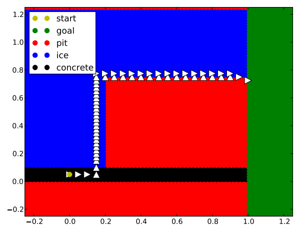
5.4.1 Algorithm

The approach described in Section 5.1.1 for policy evaluation will work, unchanged, for nonparametric model classes. We will adapt the model improvement step (Section 5.1.2) performed by gradient ascent in the model classes' parameter space to fit nonparametric models. In BNMs, the model is not only a function of some hyperparameters, but also of the data, so it is not clear what taking the gradient in the model classes' parameter space would mean for BNM classes.

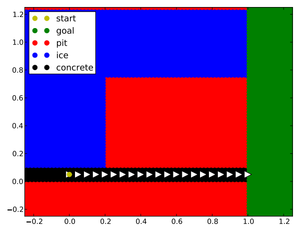
The purpose of the model improvement step is to search through the model class in the direction of increasing return. Therefore, following this purpose, we propose three approaches for searching through the model class: removing data from the model, sampling new data from the model, and treating the data as parameters themselves to be adjusted. Note that these approaches may be used together or



(a)



(b)



(c)

Figure 5-6: The domain (a) and the episode of data produced by the policy from the MAP model (b) and the RBMS model (c).

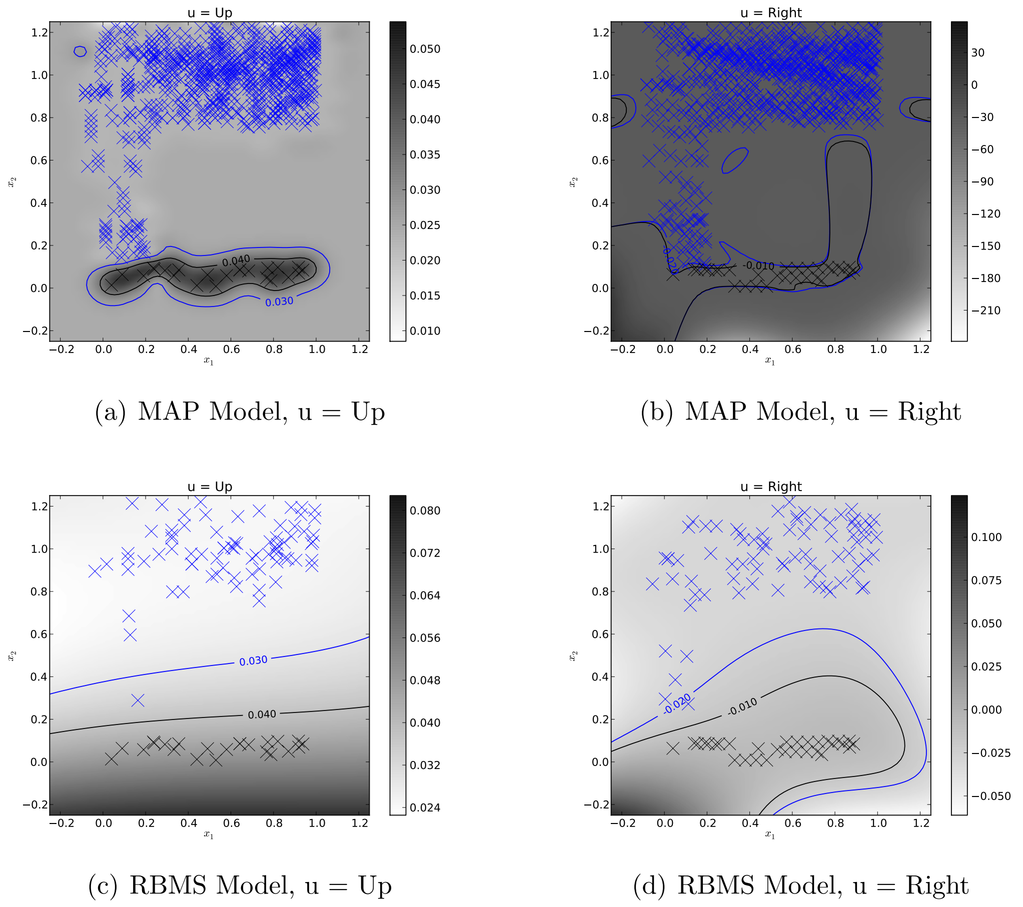
separately. These are in addition to using the standard gradient technique with the BNM classes' hyperparameters, which can also be used to search through the space.

In Section 5.4.2 we experimentally test RBMS with the approach of removing data from the model for model improvement with BNM classes. We leave it to future work to further explore the other two approaches and the trade-offs between all the approaches and their combinations

5.4.2 Empirical Results

We empirically validated RBMS for Bayesian nonparametric models on the domain shown in Figure 5-6(a). In the domain the agent starts at the yellow point and uses available actions $\{up, right\}$ to try and reach the goal (green) while avoiding falling in any of the pits (red). The agent experiences two different dynamics across the world, concrete (black) and ice (blue). On the concrete the agent's actions achieve their desired outcome and on the ice the agent will move in the chosen direction but will also "slide" south. The dynamics are deterministic and the reward function is -1 for any action, -100 for falling in a pit, and the episode ends when the agent either falls in a pit or reaches the goal. One hundred episodes of training data were generated by randomly starting an agent in a state and randomly choosing actions until the episode ended.

To model the dynamics, two separate Gaussian processes (GPs) were used to model the transitions for the *up* action and the *right* action. We used the standard



(a) MAP Model, u = Up (b) MAP Model, u = Right

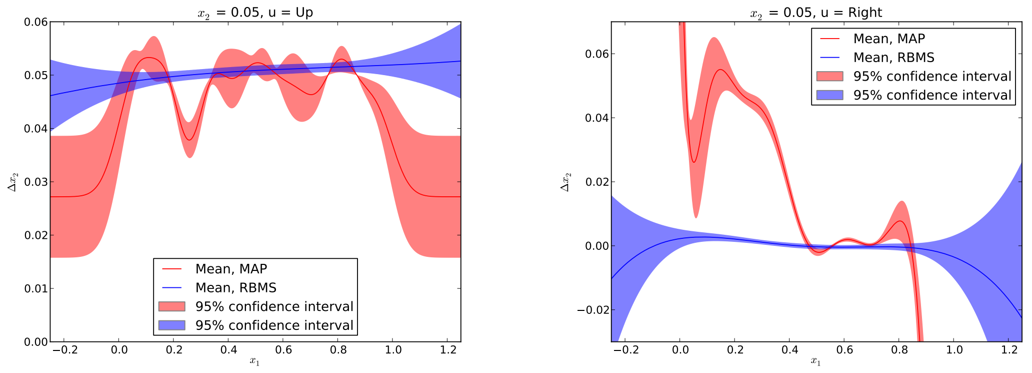
(c) RBMS Model, u = Up (d) RBMS Model, u = Right

Figure 5-7: The GPs' mean of the MAP model (a,b) and the MBRS model (c,d) for both actions.

squared exponential form of the covariance function for both GPs. Figures 5-6(b) and 5-6(c) show the episode of data generated by the policy for the maximum a posteriori (MAP) model and the RBMS model, respectively. For the RBMS model improvement step we used the strategy of stochastically removing data, which proved sufficient.

Figures 5-7 is a visualization of the mean for both the learned MAP model and RBMS model. Data points included in each model are shown as Xs and colored blue if they were on the ice and black if on the concrete. The sharp contour of the MAP GP around $x_2 = 0.1$ shows each GP attempting to cope with the discontinuity in the dynamics, in contrast to the smooth dynamics found by RBMS.

Figures 5-8 shows the GPs' means and confidence intervals, plotted over x_1 with $x_2 = 0.05$ to show the learned variances of the models in the center of the concrete.



(a) MAP and RBMS Models, u = Up, $x_2 = 0.05$ (b) MAP and RBMS Models, u = Right, $x_2 = 0.05$

Figure 5-8: The GPs' means and confidence intervals with $x_2 = 0.05$.

These figures demonstrate how the GPs, which assume the dynamics are smooth, cannot cope with the discontinuity at $x_2 = 0.1$. In other words, the discontinuity of the dynamics violates the implicit assumption of the GPs' covariance function and leads to their misspecification. RBMS mitigated this problem by searching for the model which achieved the highest return, not by attempting to fit the data well. Note that RBMS has essentially chosen a model which says, "everywhere is concrete," and that works well for this domain. If the domain were modified in a way that it would be necessary to model both the ice and concrete well (*e.g.*, because the optimal route involved both ice and concrete) we may have to turn to a more powerful model class, such as a mixture of GPs (Chapter 3).

5.5 Related Work

The approaches most closely related to our work are Policy search (PS) techniques, which search the policy space for high performing policies using gradient techniques [Baird and Moore, 1999, Guestrin et al., 2002]. In order to estimate the gradient, some methods obtain new samples for the new policy in order to avoid the learning bias, which can translate into high sample complexity. Existing off-policy PS methods [Meuleau et al., 2000] use importance sampling to reuse the data while attempting to avoid learning bias. In general, such methods are limited to discrete domains and stochastic policies. Additionally, sample efficient PS methods generally require data from high performing policies [Kober et al., 2010]. We suspect that applying RBMS in a PS setting, where the policy is directly parametrized by θ, would be successful in cases where we, as designers, struggle to construct a model class that contains high performing policies learnable from limited data. In these cases we may prefer to directly parameterize the policy instead of using a dynamics model. We plan to investigate this approach and compare it to the model-based approach described here in future work.

Model Based (MB) methods aim to capture the unknown dynamics using a representation of the dynamics model. While expressive representations [Deisenroth and Rasmussen, 2011, Joseph et al., 2011] can capture nearly any type of world dynamics, they are still vulnerable to choosing models which perform arbitrarily poorly, especially from limited data due to the bias introduced by the learner. Compact representations may eliminate the sample complexity problem, yet once combined with classical learning methods (*e.g.*, maximum likelihood), they incur substantial learning bias, as shown in our empirical results. We hypothesize that we can remedy this shortcoming by applying RBMS to Bayesian nonparametric models and plan to explore that direction in future work.

Model-free Value Based (VB) techniques sidestep learning an explicit world model and directly compute a value function, although Parr et al. [2008] showed that MB and VB RL methods using linear representations are equivalent. In the online setting, VB methods *e.g.*, Rummery and Niranjan [1994] eliminate the learning bias by using on-policy data, yet are sample inefficient. For real-world problem, batch VB techniques [Lagoudakis and Parr, 2003b] have been shown to be sample efficient, yet they are sensitive to the distribution of the training data[10]. Manually filtering the training data by a domain expert has been used to correct for the sampling distribution; for example, in a bicycle domain, episodes were trimmed after a certain length to expose

[10]For comparison note that the policies learned by the large tabular Markov model are equivalent to the policies LSPI [Lagoudakis and Parr, 2003b] would learn if given the tabular representation for its value function.

the agent to a higher proportion of the data from the bicycle balancing, which is more likely to be seen under good policies [Petrik et al., 2010]. When the representation containing the true value function is unknown, batch VB methods cannot guarantee that the highest performing value function is chosen from the representation, and generally only show convergence [Sutton and Barto, 1998]. We believe we can overcome this limitation by using the parameterization $V(s;\theta)$ for RBMS to find the highest performing value function in the VB setting.

While there has been relatively little work on overcoming learning bias, there has been a great deal of work in reducing representational bias by growing the representation using nonparameteric dynamics models [Vasquez Govea et al., 2009, Ure et al., 2012], Bayesian nonparametric dynamics models [Joseph et al., 2011], nonparametric value functions [Whiteson et al., 2007b, Geramifard et al., 2011], and kernel-based methods [Ormoneit and Glynn, 2000, Barreto et al., 2011]. The work specifically focused on reducing misspecification error generally has relied on strong assumptions about the true model [White, 1982, Sugiyama, 2006]. Kalyanakrishnan and Stone [2011] states that we must "*pick and tailor learning methods to work with imperfect representations*," and specifically highlights meta-learning [Vilalta and Drissi, 2002], and combining value-based RL and policy search [Baird and Moore, 1999, Guestrin et al., 2002] as work headed in this direction. Despite these previous algorithms being a significant step toward the goal of reducing learning bias, for a given problem and representation, it is extremely difficult to know which method will provide the desired reduction in learning bias without implementing a variety of methods.

5.5.1 Conclusions

In this chapter we have shown how choosing models by maximizing estimated return in both parametric and nonparametric settings can outperform minimum prediction error techniques (*e.g.* maximum likelihood, maximum a posteriori) with misspecified model classes. RBMS's guarantees us the ability to find the best model from a misspecified model class with infinite data, a claim that minimum prediction error techniques cannot make. Additionally, as Figures 5-3(b) and 5-4(b) provides some intuition that using smaller models (which may be misspecified) may let us learn faster. In Chapter 6 we attempt to answer the question: Why would we want to choose a misspecified model class? We see in the chapter that the intuition from the plots is correct, and the amount of data available to us directly relates how large a model class we should prefer.

Chapter 6

Structural Return Maximization

The goal of this chapter is to answer the question: Why would we choose a misspecified model class? We find that the answer is the model class we choose should strongly depend on the amount of data available to us. With limited data, approaches that explicitly maximize estimated return are vulnerable to learning policies which perform poorly due to the return being difficult to estimate. We aim to overcome this problem by applying the principle of Structural Risk Minimization (SRM) [Vapnik, 1998], which, in terms of RL, states that instead of choosing the policy which maximizes the estimated return we should instead maximize the bound on return. In SRM the policy class size is treated as a controlling variable in the optimization of the bound, allowing us to naturally trade-off between estimated performance and estimation confidence. By controlling policy class size in this principled way we can overcome the poor performance of approaches which explicitly maximize estimated return with small amounts of data.

The main contribution of this chapter is to apply the principle of SRM to RL for general policy classes under mild assumptions. We first map RL to classification, allowing us to transfer generalization bounds developed for classification based on Rademacher complexity [Bartlett and Mendelson, 2003] which results in a bound on the return of any policy from a policy class. Given a structure of policy classes, we then apply the principle of SRM to find the highest performing policy from the family of policy classes.

6.1 Bounding Return

In this section we discuss a variety of bounds on the return of policies. It is important for the reader to note the distinction between three types of bounds:

1. A bound on the return for a policy

2. A bound on the return for a policy chosen from a policy class

3. A bound on the return for a policy chosen from a policy class which was chosen from a set of policy classes

For example, Section 5.2.2 presented a bound on the return of a policy (bullet (1)) when its return is estimated using MFMC but this is not a bound on a policy produced from RBMS (which is presented in Section 6.1.1). Another way to understand bullets (1), (2), and (3) is to think of them as bounds on policy evaluation, policy learning, and policy class selection, respectively. Note that one cannot simply apply a method for doing (2) to many policy classes and pick the tightest bound as a solution for (3) Vapnik [1998].

In this chapter we will first present a bound on RBMS (a policy chosen from a policy class) based on bounds used in the classification literature. By using this type of bound, combined with a specific structure of a set of policy classes, we then present a bound on a policy chosen from a policy class which was chosen from a set of policy classes.

6.1.1 Bound on the Return of a Policy Chosen from a Policy Class

Our goal is to develop a bound on the return for a policy chosen from a policy class from a set of policy classes that can be used for real-world, complex systems from which we only have batch data. Despite significant work having been done to bound the return of a policy chosen from a policy class Ng and Jordan [2000], Kearns et al. [2002], Bagnell et al. [2003], Kakade [2003] there is no clear way to use these bounds to choose between policy classes. Additionally, the previous work focused on producing bounds for specify types of policy classes, where as the bounds contained in this chapter require only mild assumptions on the policy class. Throughout this chapter we use RBMS as the learning algorithm due to its minimal assumptions on the true world dynamics with batch data but the bound presented in Section 6.1.2 could be applied to any learning algorithm which has a bound on the return for a policy chosen from the policy class and a policy class of which we can compute the Rademacher complexity (Section 2.3.3).

Mapping of Classification Bounds to Reinforcement Learning

To show the mapping we begin by defining the *return function*,

$$G(s_0, w_{0:T-1}, a_{0:T-1}) \triangleq \rho(s_0) + \sum_{t=0}^{T-1} \rho(m(s_t, a_t, w_t)), \tag{6.1}$$

where s_t, a_t, and w_t are the state, action, and disturbance at time t, ρ is the reward function, and m is the true (unknown) dynamics model. Using the return function we can then show that the classification objective of minimizing risk is equivalent to the RL objective of maximizing return. Using Equation 2.11 and a loss function that does not depend on y[1], we set $\mathcal{L}(y, f(x)) = \tilde{\mathcal{L}}(x, f)$ and see that

$$\begin{aligned}
\mathcal{R}(f) &= \int \tilde{\mathcal{L}}(x, f)\ p(x)\ dx & (6.2)\\
&= \int -\tilde{G}(s_0, w_{0:T-1}, f)\ p(s_0, w_{0:T-1})\ dw_{0:T-1} & (6.3)\\
&= -V(f) & (6.4)
\end{aligned}$$

where we go from Equation 6.2 to Equation 6.3 by setting $x = [s_0, w_{0:T-1}]$ and noting that $\tilde{\mathcal{L}}(x, f) = -\tilde{G}(s_0, w_{0:T-1}, f) = -G(s_0, w_{0:T-1}, a_{0:T-1})$ for $a_t = f(s_t)$, and we move from Equation 6.3 to Equation 6.4 using Equation 5.2. Therefore, minimizing $\mathcal{R}(f)$ for $f \in \mathcal{F}$ is identical to maximizing $V(f)$ for $f \in \mathcal{F}$. We see that G is the term in brackets

[1]While it may seem that writing $\mathcal{L}$ without y is an abuse of notation, if instead we view $\mathcal{L}$ as a measure of performance the relationship between RL and classification becomes more clear.

in Equation 5.1; G encodes both the reward function and dynamics model and is analogous to classification's $\mathcal{L}$. This is a crucial relationship since transferring the bounds from Section 2.3.3 depends on our being able to calculate the RL equivalent of $\mathcal{L} \circ \mathcal{F}$, which we denote $G \circ \Pi$.

Using this mapping we can rewrite Equations 2.12 and 2.15 for RL as

$$V^{emp}(\pi; \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} G(s_0, w_{0:T-1}^{\pi,n}, a_{0:T-1}^{\pi,n}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \rho(s_t^{\pi,n}) \tag{6.5}$$

$$V(\pi) \geq V^{emp}(\pi; \mathcal{D}) - \Omega(G \circ \Pi, \mathcal{D}, \delta). \tag{6.6}$$

where $s_{t+1}^{\pi,n} = m((s_t^{\pi,n}, \pi(s_t^{\pi,n}), w_{0:T-1}^{n})$. Note that in Equation 6.5, $w_{0:T-1}^{n}$ is needed to compute the empirical return which typically is not observed in practice. Since we assume the state is fully observable, we can use Equation 6.5 to calculate empirical return. Combining Equations 5.14 and 6.6,

$$V(\pi) \geq V_{MFMC}(\pi; \mathcal{D}) - d(\pi; \mathcal{D}) - 2(B - A)\sqrt{\frac{-\ln \delta}{2N}} - \Omega(G \circ \Pi, \mathcal{D}, \delta) \tag{6.7}$$

with probability $1-3\delta$ and is a bound on return for all policies in Π. The remainder of this section describes the use of VC dimension and Rademacher complexity (Section 2.3.3), respectively, to compute Ω in Equation 6.7. We also assume that the return is bounded by $A \leq \sum_{t=0}^{T-1} \rho(s_t) \leq B$. An alternative to the approach described in this section for computing Ω may be similar to the one taken in Kearns et al. [2002].

Bound Based on VC Dimension for RL To use the bound described in Section 2.3.3 we need to know the VC dimension of $G \circ \Pi$. Unfortunately, the VC dimension is only known for specific function classes (*e.g.*, linear indicator functions [Vapnik, 1998]), and, since the only assumptions we made about functional form of ρ, m, and π is that they are Lipschitz continuous, $G \circ \Pi$ will not in general have known VC dimension.

There also exist known bounds on the VC dimension for *e.g.*, neural networks [Anthony and Bartlett, 1999], decision trees [Asian et al., 2009], support vector machines

[Vapnik, 1998], smoothly parametrized function classes [Lee et al., 1995], but, again, due to our relatively few assumptions on ρ, m, and π, $G \circ \Pi$ is not a function class with a known bound on the VC dimension. Cherkassky and Mulier [1998] notes that for some classification problems, the VC dimension of $\mathcal{F}$ can be used as a reasonable approximation for the VC dimension $\mathcal{L} \circ \mathcal{F}$, but we have no evidence to support this being an accurate approximation when used in RL.

Other work has been done to estimate the VC dimension from data [Shao et al., 1969, Vapnik et al., 1994] and bound the VC dimension estimate [McDonald et al., 2011]. While, in principle, we are able to estimate the VC dimension using one of these techniques, the approach described in the following section is a more straightforward method for computing Ω in Equation 2.15 based on data.

Bound Based on Rademacher Complexity for RL Using the Rademacher complexity bound (Section 2.3.3), allows us to calculate Ω (Equation 6.6) based on data. The only remaining piece is how to calculate the summation inside the absolute value sign of Equation 2.18 for RL. Mapping the Rademacher complexity estimator (Equation 2.18) into RL yields

$$\hat{R}^N(G \circ \Pi; \mathcal{D}) = E_{\sigma_{1:\tilde{N}}}\left[\sup_{\pi \in \Pi} \frac{2}{\tilde{N}} \left| \sum_{n=1}^{\tilde{N}} \sigma_n \sum_{t=0}^{T-1} \rho(s_t^n) \right| \middle| \mathcal{D} \right]. \tag{6.8}$$

Therefore, with probability $1 - \delta$

$$\hat{R}^N(G \circ \Pi; \mathcal{D}) \geq -2 + E_{\sigma_{1:\tilde{N}}}\left[\sup_{\pi \in \Pi} \frac{2}{\tilde{N}} \left| \sum_{n=1}^{\tilde{N}} \sigma_n \left(V_{MFMC}(\pi; \mathcal{D}) - d(\pi; \mathcal{D}) - 2\sqrt{\frac{-\ln \delta}{2N}} \right) \right| \middle| \mathcal{D} \right] \tag{6.9}$$

where we move from Equation 6.8 to Equation 6.9 using

$$\sigma_n \sum_{t=0}^{T-1} \rho(s_t^n) \geq -\tilde{N} + \sigma_n \left(V_{MFMC}(\pi; \mathcal{D}) - d(\pi; \mathcal{D}) - 2\sqrt{\frac{-\ln \delta}{2N}} \right) \tag{6.10}$$

from Equation 5.14 and assumed $-1 \leq \sum_{t=0}^{T-1} \rho(s_t) \leq 0$ for simplicity. Note that Equation 6.8 is an estimate of the true (unknown) Rademacher complexity and we use Equation 6.9, a bound on Rademacher complexity, in the computation of Ω.

6.1.2 Bound on the Return of Policy Classes Selection

Policy Class Structures

In Section 2.3.4 we defined a structure of function classes, from which a function class and function from that class are chosen using Equation 2.20. To use structural risk minimization for RL, we must similarly define a structure,

$$\Pi_1 \subseteq \Pi_2 \subseteq \cdots \subset \Pi_k \subseteq \cdots , \tag{6.11}$$

of policy classes[2]. For RL we add the additional constraint that the policies must be Lipschitz continuous (Section 2.1) in order to use the bound provided in Section 5.2.2. Note that the structure (*e.g.*, the indexing order $1, \ldots, k, \ldots$) must be specified *before* any data is seen.

Fortunately, some function classes have a "natural" ordering which may be taken advantage of, for example support vector machines [Vapnik, 1998] use decreasing margin size as an ordering of the structure. In RL, many common policy representations contain natural structure and are also Lipschitz continuous. Consider a policy class consisting of the linear combinations of radial basis functions [Menache et al., 2005]. This class is Lipschitz continuous and using this representation, we may impose a structure by progressively increasing a limit on the magnitude of all basis functions, therefore high k allows for a greater range of actions a policy may choose. This policy class consists of policies of the form

$$\pi(s) = \sum_{i=1}^{M} \phi_i e^{-c(s-\bar{s}_i)^2} \tag{6.12}$$

where $c \geq 0$, $\bar{s}_i \in \mathcal{S}$ are fixed beforehand and the progressively increasing limit $l_k \geq |\phi_i|$, $i \in \{1, \ldots, M\}$ and k is the index of the policy class structure. A second

[2]Note that the structure is technically over $G \circ \Pi_k$ but $\Pi_i \subseteq \Pi_j \implies G \circ \Pi_i \subseteq G \circ \Pi_j$.

policy representation which meets our requirements consists of policies of the form

$$\pi(s) = \sum_{i=1}^{M} \frac{\phi_i}{||s - l_i||_{\mathcal{S}}} \tag{6.13}$$

where $l_i \in \mathcal{S}$ are fixed beforehand and a policy of this representation is described by set $\phi_i \in \mathcal{A},\ i \in \{1, \dots, M\}$. Using this representation, we then present two possible structures, the second of which we use for the experiments in Section 4.2. The first places a cap on ϕ_i, where $-a_k \leq \phi_i \leq a_k,\ \forall i \in \{1, \dots, M\}$ and $a_k \leq a_{k+1}$. The second structure "ties" together $\phi_i,\ i \in \{1, \dots, M\}$ such that for $k = 1$, $\phi_1 = \phi_2 = \cdots = \phi_M$. For $k = 2$, we untie ϕ_1 but still maintain that $\phi_2 = \cdots = \phi_M$ and continue for the remaining k such that Equation 6.11 is maintained. While we believe the representation described in this section is a natural structure that can be used for many RL policy classes, we leave further investigation into automatically constructing structures or making the structure data-dependent (*e.g.*, Shawe-Taylor et al. [1996]) to future work.

Overall Bound

Using a structure of policy classes as described in the previous section, we may now reformulate the objective of SRM into Structural Return Maximization for RL as

$$\hat{k} = \arg\min_{k} V_{MFMC}(\hat{\pi}_k; \mathcal{D}) - d(\hat{\pi}_k; \mathcal{D}) - 2(B - A)\sqrt{\frac{-\ln\delta}{2N}} - \Omega(G \circ \Pi_k, \mathcal{D}, \delta) \tag{6.14}$$

$$\hat{\pi}_k = \arg\min_{\pi \in \Pi_k} V_{MFMC}(\pi; \mathcal{D}) - d(\pi; \mathcal{D}), \tag{6.15}$$

where Ω is computed using Equations 2.17, 2.19, and 6.9. Therefore, with a small batch of data, Equations 6.14 and 6.15 will choose a small k and as we acquire more data k naturally grows. The only term in Equations 6.14 and 6.15 that we must assume is L_m, the Lipschitz continuity constant of the world (Section 5.1). To solve Equation 6.15 we follow Joseph et al. [2013] and use standard gradient decent with random restarts.

6.2 Empirical Results

The experiments described in this section are all instances of batch, off-policy learning used to compare Structural Return Maximization (SRM), described in Section 6.1.2, to a maximum return (MR) learner [Joseph et al., 2013], which maximizes the estimated return. For each experiment, the chosen approaches[3] choose policies from a designer-provided policy class. To use SRM we imposed a structure on the policy class (where the original policy class was the largest, most expressive class in the structure) and the methodology from Section 6.1.2 allowed SRM to select an appropriately sized policy class and policy from that class. The MR learner maximizes the empirical return (Equation 5.9) using the single, largest policy class.

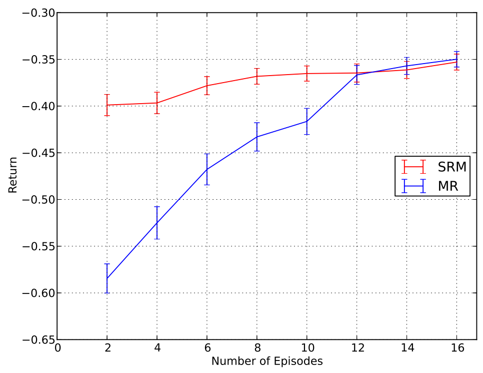
We compare these approaches on three simulated problems: a 1D toy domain, the inverted pendulum domain, and an intruder monitoring domain. The domains demonstrate how SRM naturally grows the policy class as more data is seen and is far less vulnerable to over-fitting with small amounts of data than a MR learner. Policy classes were comprised of linear combinations of radial basis functions (Section 6.1.2) and training data was collected from a random policy. To piece together artificial trajectories for MFMC we used $\tilde{N} = (0.1)N$ where N is the number of data episodes.

6.2.1 1D Toy Domain

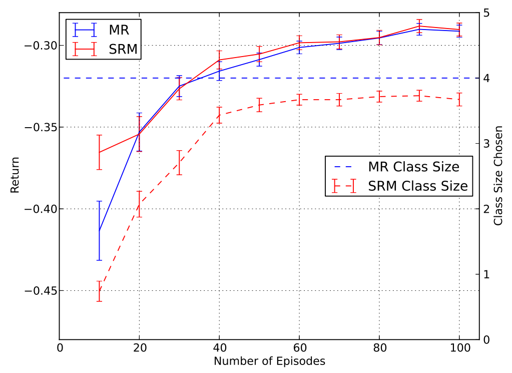
The purpose of the 1D toy domain is to enable understanding for the reader. The domain is a single dimensional world consisting of an agent who begins at $s_0 = 0$ and attempts to "stabilize" at $s = 0$ in the presence of noise. The dynamics are $s_{t+1} = s_t + a + e$, $s_t \in [-1, 1]$ and e is a uniform random variable over $[-1/4, 1/4]$. The agent takes actions $a \in [-0.5, 0.5]$ and has reward function $\rho(s_t) = 5|s_t|$. For the policy representation we used four evenly spaced radial basis functions and for SRM we imposed five limits $l_k \in \{0, 0.125, 0.25, 0.375, 0.5\}$ on $|\phi_i|$ (see Equation 6.12).

Figures 6-1(a) and 6-1(b) show the performance of SRM (red solid line) and MR

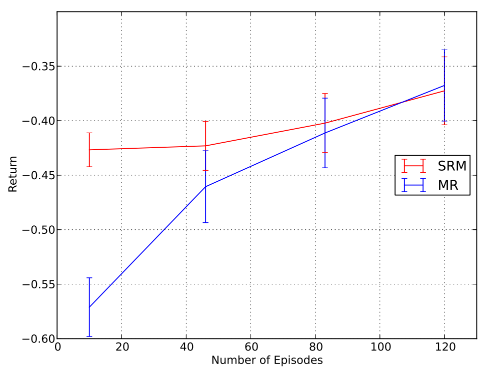
[3]See Section 2.1.2 for our discussion on the pitfalls of using model-based approaches in this setting.



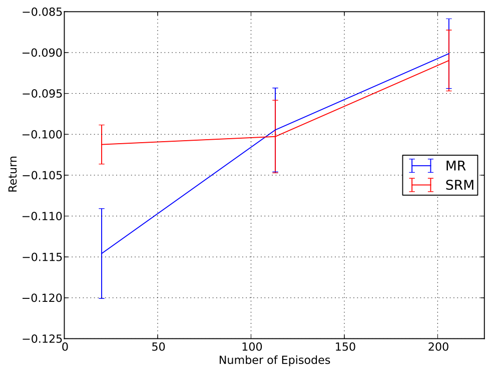
(a) 1D Toy Domain



(b) 1D Toy Domain



(c) Inverted Pendulum



(d) Intruder Monitoring

Figure 6-1: Performance versus the amount of training data and class size versus the amount of training data (a, b) on the 1D toy domain. Performance versus the amount of training data on the inverted pendulum domain (c) and the intruder monitoring domain (d). Error bars represent the 95% confidence interval of the mean.

(blue solid line) on the 1D toy domain, where figure 6-1(a) is zoomed in to highlight the SRM's performance with small amounts of data. The plot shows that MR over-fits the small amounts of data, resulting in poor performance. On the other hand, SRM overcomes the problem of over-fitting by selecting a policy class which is appropriately sized for the amount of data. Figure 6-1(b) illustrates how SRM (red dashed line) selects a larger policy class as more data is seen, in contrast to MR, which uses a fixed policy class (blue dashed line). The figures show that as SRM is given more data, it selects increasing larger classes, allowing it learn higher performing policies without over-fitting.

6.2.2 Inverted Pendulum

The inverted pendulum is a standard RL benchmark problem (see Lagoudakis and Parr [2003b] for a detailed explanation and parametrization of the system). In our experiments we started the pendulum upright with the objective of learning policies which stabilize the pendulum in the presence of noise. For the policy representation we placed 16 evenly spaced radial basis functions and for SRM we imposed two limits on $|\phi_i|$ (Equation 6.12), $l_k \in \{0, 50\}$.

Figure 6-1(c) shows the performance of SRM (red) and MR (blue) on inverted pendulum. Similar to the results on the 1D toy domain, we see that MR over-fits the training data early on, resulting in poor performance. In contrast, SRM achieves higher performance early on by using a small policy class with small amounts of data and growing the policy class as more data is seen.

6.2.3 Intruder Monitoring

The intruder monitoring domain models the scenario of an intruder traversing a two dimensional world where a camera must monitor the intruder around a sensitive location. The camera observes a circle of radius $r_{\text{cam}} = 0.5$ centered at s_{cam}, and the intruder, located at s_I, wanders toward the sensitive location with additive uniform noise. The camera dynamics follow $s_{\text{cam}} = s_{\text{cam}} + a$, where the agent takes action $a \in [-0.1, 0.1]$ and has reward

$$\rho(s_{\text{cam}}, s_I) = \sum_i \frac{min(||s_{\text{cam}} - X_i||, 0.5)}{max(||s_{\text{sensitive}} - X_i||, 0.05)}$$

where $s_{\text{sensitive}} = [0, 0]$. For our policy representation, we placed 16 radial basis points on a grid inside $[[-1, 1], [-1, 1]]$ and for SRM we imposed two limits on $|\phi_i|$, $l_k \in \{0, 0.1\}$.

Figure 6-1(d) shows the performance of SRM (red) and MR (blue) on inverted pendulum. Similar to the results on the previous domains, we see that SRM outperforms MR due to using a small policy class with small amounts of data and growing the policy class as more data is seen.

6.3 Related Work

While there has been a significant amount of prior work relating Reinforcement Learning (RL) and classification [Langford and Zadrozny, 2003, Lagoudakis and Parr, 2003a, Barto and Dietterich, 2004, Langford, 2005], to the best of our knowledge, our work is the first to sufficiently develop the mapping to allow the analysis presented in this Chapter on general policy classes under mild assumptions to be transferred to RL. Other work has been done to bound the return of a policy chosen from a policy class Ng and Jordan [2000], Kearns et al. [2002], Bagnell et al. [2003], Kakade [2003] there is no clear way to use these bounds to choose between policy classes and they focus on producing bounds for specify types of policy classes, where as the bounds contained in this chapter require only mild assumptions on the policy class. There has also been work to use classifiers to represent policies in RL [Rexakis and Lagoudakis, 2008, Dimitrakakis and Lagoudakis, 2008, Blatt and Hero, 2006], which is tangential to our work; our focus is on using bounds on return to choose the size of policy classes.

The RL literature also has a great deal of work growing representations as more data is seen. Past work in the model-based [Doshi-Velez, 2009, Joseph et al., 2011] and value-based [Ratitch and Precup, 2004, Whiteson et al., 2007a, Geramifard et al., 2011] settings have proven successful but generally require either prior distributions or a large amount of training data collected under a specific policy. Additionally, our approach applies in model-based, value-based, and policy search settings by treating either the model class or value function class as indirect policy representations.

6.4 Conclusions and Discussion

In this chapter we answered the question: Why would we choose a misspecified model class? We learned that the class size should strongly depend on the amount of data available. By using the Principle of Structural Risk Minimization (and transferring the theory over from the statistical learning theory literature) we were able to show how to choose an appropriately sized policy class for a given amount of data.

Chapter 7

Conclusions

In this thesis we looked at three questions:

1. How big should our model be?
2. How do we pick a policy when the model class is too limited to capture the true dynamics?
3. Why would we pick a limited model class?

In Chapters 3 and 4 we answered the first question using Bayesian nonparametric model (BNM) classes. The BNM classes were extremely flexible and allowed the models to grow with the size of the data. For the first application (Chapter 3), mobile agent modeling, we represented a mobility patterns as a Gaussian process. Not knowing how many mobility patterns to expect, we used a Dirichlet process (DP) prior over the number of mobility patterns. The DP prior allowed us to grow the number of mobility patterns based on the given data. We applied to the model to simulated interception and tracking domains and a real-world dataset of taxis driving around the greater Boston area. We continued using BNM classes in Chapter 4 to model battery health where we applied a similar model of a DP prior over battery behaviors to iRobot Roomba battery data.

We addressed the second question in Chapter 5 by stating that we should pick the model based on the performance of the resulting policy. The chapter introduced

Reward Based Model Search (RBMS) for both parametric model classes and BNM classes. RBMS leveraged Model-free Monte Carlo [Fonteneau et al., 2010] to perform policy evaluation and gradient ascent for policy improvement. We applied RBMS to a variety of parametric and nonparametric simulated environments and the real-world hydrodynamic cart-pole system.

In Chapter 6 we answered the third question by following the Principle of Structural Risk Minimization [Vapnik, 1995] and let both the estimated performance and the performance estimation accuracy dictate the size of the model class. We developed an algorithm called Structural Return Maximization (SRM) which utilized a bound on the true return to decide the appropriate size of the policy class. We evaluated SRM's performance versus naively maximizing performance on a variety of simulated domains.

7.1 Relationship Between Bayesian Nonparametric Modeling and SRM

A natural question to ask is what the relationship between Structural Return Maximization (SRM, Chapter 6) and Bayesian Nonparametric Models (BNM, Chapters 3 and 4) is since they both seem to answer the question: How large should our model class be?

An important part of understanding the relationship between SRM and BNMs is to first understand the assumptions of each learning technique discussed in this thesis. For model inference, both maximum likelihood (ML) and maximum a posteriori (MAP) learning both assume the true world dynamics are contained within the model class they are inferring over. Reward Based Model Search (RBMS) assumes a designer provided distance function (for MFMC, Section 5.1.1) and a known Lipschitz continuity constant of the world dynamics (Section 5.1). Structural Return Maximization (SRM) inherits both assumptions of RBMS (since RBMS is inside of the SRM algorithm) and additionally that the model classes it is searching over are nested subsets (Section 6.1.2).

To understand the relationship between SRM and BNMs we must first note that SRM is a technique for choosing between model classes and a BNM is a model class. So the relationship we are technically interested in is between SRM and BNMs with MAP inference (BNM+MAP). This relationship, between SRM and BNM+MAP, is best understood by first considering how RBMS is related to ML learning. To begin, ML learning is far more well known in the literature and is, for many classes of models, computationally simpler. With this in mind, the decision to use RBMS depends on when the assumptions of ML cannot be relied on and in these situations we would then turn to RBMS. For example, in Section 5.3.3, we learned a cart-pole model of the hydrodynamic cart-pole system. This clearly broke the ML assumption that the mode class (the class of standard cart-pole models) contains the true system (the hydrodynamic cart-pole).

Figures 5-3(a) and 5-4(a) illustrate how the broken assumptions of ML manifest in poor performance. In these situations where we can not rely on the ML assumptions we have two choices: design a larger model class that we can rely on the ML assumptions for or choose a model from the original model class based on a different set of assumptions. RBMS squarely addresses the latter approach.

Unfortunately, the move from ML to RBMS does not come without a cost. First, RBMS requires a designer-provided distance function and a known Lipschitz continuity constant of the world dynamics. Additionally, RBMS, in general, has a far higher computational cost and it may be intractable to use in some situations either because of the difficulty running the optimization or because many, long episodes must be pieced together by MFMC.

The relationship between RBMS and BNM+MAP follows a similar story to RBMS and ML. BNM+MAP is far better understood and, in general, is computationally far easier than SRM. In Section 5.4.2 we illustrated how the GP kernel function assumed smooth dynamics but the true world contained a sharp discontinuity between the concrete and ice. The result of this were poorly inferred models by BNM+MAP and better models inferred by RBMS (see Figure 5-8). To more clearly understand this, we will discuss the two BNM classes used in this thesis, Dirichlet Process (DP) and Gaussian Process (GP), separately.

When using DPs with MAP inference (DP+MAP) the DP places a likelihood on the number of components and MAP chooses the number of the components with the highest likelihood. Choosing the size of the representation (for either the model or policy) in this way is common in the literature and generally straightforward to implement. SRM then comes in when we cannot rely on the assumptions of DP+MAP or DP+RBMS. For example, in the case of DP+MAP, this could be due to the broken exchangeability assumption as described in Section 4.3.1. In the case of DP+RBMS, this could be due to, for example, needing the DP's components to have different models (not just different parameter values). We are then faced with a decision similar to the one faced with ML, make the model class more powerful (*e.g.*, using dependent DPs, as suggested in Section 4.3.1) or change DP+MAP to DP+RBMS (Section 5.4). If we cannot rely on the assumptions of DP+MAP or if the model class is not powerful enough for DP+RBMS (*e.g.*, needing different models across components), SRM provides us with a third choice. SRM also is the approach to use third because it is 1) more poorly understood than MAP and RBMS inference and 2) is more computationally difficult than DP+MAP and DP+RBMS.

An alternate way of understanding the choice of DP+MAP/DP+RBMS or SRM is to examine their assumptions. DP+MAP/DP+RBMS assumes all components have the same representation and the true world both contains an infinite number of these components and can be perfectly captured by the model class. SRM requires the representation classes to be nested subsets (which is more general than the DP assumption) and only requires mild Lipschitz continuity assumptions (Section 5.2). So DP+MAP has harsher assumptions and benefits from these assumptions by easier computational complexity and implementation.

The other BNM used in this thesis, Gaussian processes using MAP inference (GP+MAP), does not increase in "size" in a way similar to DP+MAP or SRM. As GP+MAP sees more data it can increasingly place higher probability mass on more complex functions (measured by, *e.g.*, Lipschitz continuity) if the data warrants it but there is no clear way in which the GP "grows in size". Hence, there is no meaningful relationship between a GP+MAP, a fixed size model class, and SRM, a method for choosing between model classes; although GP+RBMS is a perfectly valid choice when the MAP assumptions do not hold for GPs (Section 5.4). And when GP+RBMS is a poor choice because the GP model class is too limited, for example, because a mixture of GPs is needed, then a designer can make the model class larger (*e.g.*, with DPGP+MAP, see Chapter 3) or SRM can be used to appropriately grow the size of the model class.

The cost of moving from BNM+MAP to SRM also follows a similar story of the move from ML to RBMS. Since RBMS is inside SRM, to use SRM we are required to have a designer-provided distance function and a known Lipschitz continuity of the world dynamics. SRM also, in general, has a far higher computational cost than BNM+MAP and therefore may be intractable to use in some situations. On additional requirement is knowledge of L_m, the Lipschitz continuity constant of the world (Section 5.1), which will generally be a weaker assumption than the BNM+MAP assumptions, but is still needed by the algorithm.

In this section so far we have only addressed the relationship between BNM+MAP and SRM in computational and implementation difficulty and in terms of previous use in the literature. A final relationship that the reader may be interested in is in regards to data complexity. We can again look to the comparison between ML and RBMS. As we learned from Chapter 5, ML can learn extremely quickly (*e.g.*, ML Trimmed on Figure 5-3(b)) if the model class can capture the truth (or close to it). On the other hand, it may also never learn a high performing policy (*e.g.*, ML on Figure 5-3(b)). Similarly, BNM+MAP can learn very quickly (Chapters 3 and 4) but can also never learn a high performing policy (Section 5.4) even when one exists and given unlimited data and computation. Whereas, if a high performing policy exists, SRM is guaranteed to find it with unlimited data and computation.

In conclusion, the question of when we should use BNM+MAP and SRM is answered by the following decision tree:

- Can we rely on the BNM+MAP assumptions?

- If no, can we either make the BNM more powerful or exchange MAP for RBMS and rely on the resulting assumptions?
- If no, design a structure of policy or model classes and use SRM.

7.2 Future Work

We believe there are two main promising directions for future research based on the work presented in this thesis. The first is to extend SRM to an active learning setting using *e.g.*, Koltchinskii [2010]. The type of methodology introduced in Koltchinskii [2010] should directly fit into the overall approach from Chapter 6. The second direction for future work would be to perform a much deeper investigation into structures of policy classes to be used for SRM. Additionally, structures of model class or value function classes may be more appropriate for specific applications. This may also lead into interesting work automatically generating structures of policy classes that can be used in SRM, as opposed to the designer provided structures we described in this thesis.

Appendix A

Probability Distributions

A.1 Gamma

A gamma distribution is a continuous distribution defined by shape parameter, a, and scale parameter, b. A random variable $x \sim \mathcal{G}(a, b)$ implies

$$\mathrm{P}(x) = f_{\mathcal{G}}(x; a, b) = \frac{1}{\Gamma(a) b^a} x^{a-1} e^{-x/b}. \tag{A.1}$$

In (A.1), $\Gamma(a) = \int_0^\infty z^{a-1} e^{-z} dz$ is the gamma function, the extension of the factorial function to complex numbers. There are other common parameterizations of the gamma distribution that the reader may be familiar with but this is the only one used in this thesis.

A.2 Dirichlet

A Dirichlet distribution is a continuous distribution with parameters $\alpha_1, ..., \alpha_k$ that defines a distribution over $\pi_1, ..., \pi_k$ where $\sum_i \pi_i = 1$. It can be thought of as a distribution on k dimensional discrete distributions, explicitly written

$$\begin{aligned} \mathrm{P}(\pi_1, ..., \pi_k | \alpha_1, ..., \alpha_k) &= f_{Dir}(\pi_1, ..., \pi_k; \alpha_1, ..., \alpha_k) && \text{(A.2)} \\ &= \frac{\prod_i \Gamma(\alpha_i)}{\Gamma(\sum_i \alpha_i)} \prod_i \pi_i^{\alpha_i - 1}. && \text{(A.3)} \end{aligned}$$

Bibliography

D. Aldous. Exchangeability and Related Topics. In *Ecole d'Ete de Probabilities de Saint-Flour XIII 1983*, pages 1–198. Springer, 1985.

Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

C. E. Antoniak. Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems. *The Annals of Statistics*, 2:1152–1174, November 1974.

D. Ashbrook and T. Starner. Using GPS to Learn Significant Locations and Predict Movement Across Multiple Users. *Personal Ubiquitous Computing*, 7(5):275–286, 2003.

Ozlem Asian, Olcay Taner Yildiz, and Ethem Alpaydin. Calculating the vc-dimension of decision trees. In *ISCIS 2009, 14-16 September 2009, North Cyprus*, pages 193–198. IEEE, 2009.

Bernardo Avila Pires, Mohammad Ghavamzadeh, and Csaba Szepesvari. Cost-sensitive Multiclass Classification Risk Bounds. In *International Conference on Machine Learning*, 2013.

J. Andrew Bagnell, Sham Kakade, Andrew Ng, and Jeff Schneider. Policy search by dynamic programming. In *Neural Information Processing Systems*, volume 16. MIT Press, 2003.

Leemon Baird and Andrew Moore. Gradient descent for general reinforcement learning. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*. MIT Press, 1999.

Andre S. Barreto, Doina Precup, and Joelle Pineau. Reinforcement learning using kernel-based stochastic factorization. In J. Shawe-Taylor, R.S. Zemel, P. Bartlett, F.C.N. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 720–728. 2011.

Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: risk bounds and structural results. *J. Mach. Learn. Res.*, 3:463–482, March 2003. ISSN 1532-4435.

Peter L. Bartlett, Stéphane Boucheron, and Gábor Lugosi. Model selection and error estimation. *Machine Learning*, 48(1-3):85–113, 2002a.

Peter L. Bartlett, Olivier Bousquet, and Shahar Mendelson. Local rademacher complexities. In *Annals of Statistics*, pages 44–58, 2002b.

Peter L. Bartlett, Michael I. Jordan, and Jon D. Mcauliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, March 2006.

A.G. Barto and T.G. Dietterich. Reinforcement learning and its relationship to supervised learning. *Handbook of learning and approximate dynamic programming. John Wiley and Sons, Inc*, 2004.

Andrew Barto and Michael Duff. Monte carlo matrix inversion and reinforcement learning. In *In Advances in Neural Information Processing Systems 6*, pages 687–694. Morgan Kaufmann, 1994.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):835–846, 1983.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 2001.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

Shai Ben-David, David Loker, Nathan Srebro, and Karthik Sridharan. Minimizing the misclassification error rate using a surrogate convex loss. In *ICML*, 2012.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, August 2006. ISBN 0387310738.

D. Blackwell and J. B. Macqueen. Ferguson distributions via Pólya urn schemes. *The Annals of Statistics*, 1:353–355, 1973.

Doron Blatt and Alfred Hero. From weighted classification to policy search. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *NIPS*. 2006.

Phillip Boyle and Marcus Frean. Dependent gaussian processes. In *In Advances in Neural Information Processing Systems 17*, pages 217–224. MIT Press, 2005.

Francois Caron, Manuel Davy, and Arnaud Doucet. Generalized polya urn for time-varying dirichlet process mixtures. In *Proceedings of the Twenty-Third Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-07)*, pages 33–40, Corvallis, Oregon, 2007. AUAI Press.

M Chen and G A Rincon-Mora. Accurate Electrical Battery Model Capable of Predicting Runtime and I-V Performance. *IEEE Transactions on Energy Conversion*, 21(2):504–511, 2006.

Vladimir S. Cherkassky and Filip Mulier. *Learning from Data: Concepts, Theory, and Methods*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998. ISBN 0471154938.

Lehel Csat and Manfred Opper. Sparse representation for gaussian process models. In *Advances in Neural Information Processing Systems*, pages 444–450. MIT Press, 2001.

Marc P. Deisenroth and Carl E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, June 2011.

Marc P. Deisenroth, Marco F. Huber, and Uwe D. Hanebeck. Analytic Moment-based Gaussian Process Filtering. In L. Bouttou and M. Littman, editors, *Proceedings of the 26th International Conference on Machine Learning*, pages 225–232, Montreal, Canada, June 2009. Omnipress.

Hussein Dia. An Agent-Based Approach to Modeling Driver Route Choice Behaviour Under the Influence of Real-Time Information. *The American Statistician*, 10, 2002.

Christos Dimitrakakis and Michail G. Lagoudakis. Rollout Sampling Approximate Policy Iteration. *Machine Learning*, 72(3):157–171, September 2008.

Finale Doshi-Velez. The infinite partially observable markov decision process. In *NIPS*, 2009.

R. A. Dougal. Dynamic Lithium-ion Battery Model for System Simulation. *IEEE Transactions on Components and Packaging Technologies*, 25(3):495–505, 2002.

Simon Duane, A. D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2), September 1987.

Raphael Fonteneau, Susan A. Murphy, Louis Wehenkel, and Damien Ernst. Model-free monte carlo-like policy evaluation. *Journal of Machine Learning Research - Proceedings Track*, 2010.

Raphael Fonteneau, SusanA Murphy, Louis Wehenkel, and Damien Ernst. Batch mode reinforcement learning based on the synthesis of artificial trajectories. pages 1–34, 2012.

Emily Fox, Erik Sudderth, and Alan S. Willsky. Hierarchical Dirichlet Processes for Tracking Maneuvering Targets. In *Proc. Inter. Conf. Information Fusion*, January 2007.

Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In *ICML*, 2011.

Agathe Girard, Carl Edward Rasmussen, Joaquin Quintero-Candela, and Roderick Murray-smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems*, pages 529–536. MIT Press, 2003.

P Gomadam. Mathematical Modeling of Lithium-ion and Nickel Battery Systems. *Journal of Power Sources*, 110(2):267–284, 2002.

Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 227–234, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-873-7.

J. M. Hammersley. Monte-Carlo Methods for Solving Multivariable Problems. *Ann. New York Acad. Sci.*, 86:844–874, 1960.

Ruijie He, Abraham Bachrach, and Nicholas Roy. Efficient planning under uncertainty for a target-tracking micro-air vehicle. In *Proc. IEEE International Conference on Robotics and Automation*, May 2010.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

R. A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, 1960.

David Hsu, Wee Sun Lee, and Nan Rong. A point-based POMDP planner for target tracking. In *Proc. IEEE International Conference on Robotics and Automation*, Pasadena, CA, May 2008.

Joshua Joseph. Nonparametric bayesian behavior modeling. 2008.

Joshua Joseph, Finale Doshi-Velez, Albert S. Huang, and Nicholas Roy. A Bayesian Nonparametric Approach to Modeling Motion Patterns. *Autonomous Robots*, 31 (4):383–400, 2011.

Joshua Joseph, Alborz Geramifard, John W. Roberts, Jonathan P. How, and Nicholas Roy. Reinforcement learning with misspecified model classes. In *ICRA*, 2013.

Joshua Mason Joseph, Finale Doshi-Velez, and Nicholas Roy. A bayesian nonparametric approach to modeling mobility patterns. In *AAAI*, 2010.

Sham Machandranath Kakade. On the sample complexity of reinforcement learning. 2003.

Shivaram Kalyanakrishnan and Peter Stone. On learning with imperfect representations. In *Proceedings of the 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 17–24. IEEE, April 2011. ISBN 978-1-4244-9886-4.

Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.

Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, July 2009.

J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.

Vladimir Koltchinskii. Rademacher complexities and bounding the excess risk in active learning. *Journal of Machine Learning Research*, 11:2457–2485, 2010.

H. Kurniawati, Y. Du, D. Hsu, and W.S. Lee. Motion Planning under Uncertainty for Robotic Tasks with Long Time Horizons. In *Proc. International Symposium of Robotics Research*, 2009.

Michail G. Lagoudakis and Ronald Parr. Reinforcement learning as classification: Leveraging modern classifiers. In *ICML*, 2003a.

Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *JMLR*, 4: 1107–1149, 2003b.

John Langford. Relating reinforcement learning performance to classification performance. In *ICML*, 2005.

John Langford and Bianca Zadrozny. Reducing t-step reinforcement learning to classification, 2003.

Wee Sun Lee, Peter L. Bartlett, and Robert C. Williamson. Lower bounds on the vc-dimension of smoothly parametrized function classes. *Neural Computaion*, 7: 7–1040, 1995.

Julia Letchner, John Krumm, and Eric Horvitz. Trip Router with Individualized Preferences (TRIP): Incorporating Personalization into Route Planning. In *AAAI*, 2006.

Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. Learning and Inferring Transportation Routines. *Artif. Intell.*, 171(5-6):311–331, 2007. ISSN 0004-3702.

Daniel McDonald, Cosma Shalizi, and Mark Schervish. Estimated vc dimension for risk bounds. 2011.

Edward Meeds and Simon Osindero. An alternative infinite mixture of Gaussian process experts. In *NIPS 18*, 2006.

W. Meiring, P. Monestiez, P.D. Sampson, and P. Guttorp. Developments in the modelling of nonstationary spatial covariance structure from space-time monitoring data. In E.Y. Baafi and N. Schofield, editors, *Geostatistics Wallongong 1996*, pages 162–173, Dordrecht, 1997. Kluwer.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Basis function adaptation in temporal difference reinforcement learning. *Annals OR*, 134(1):215–238, 2005.

Nicolas Meuleau, Leonid Peshkin, Leslie Kaelbling, and Kee Kim. Off-policy policy search. Technical report, MIT Articical Intelligence Laboratory, 2000.

Scott A. Miller, Zachary A. Harris, and Edwin K. P. Chong. A POMDP framework for coordinated guidance of autonomous UAVs for multitarget tracking. *EURASIP Journal on Advances in Signal Processing*, 2009.

Andrew Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

Dirk Ormoneit and Peter Glynn. Kernel-based reinforcement learning in average-cost problems: An application to optimal portfolio choice. In *Advances in Neural Information Processing Systems*, 2000.

Christopher J. Paciorek and Mark J. Schervish. Nonstationary covariance functions for gaussian process regression. In *Neural Information Processing Systems*, 2000.

Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, 2008.

D. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring High-Level Behavior From Low-Level Sensors. In *Proc. UBICOMP*, 2003.

Leonid Peshkin. Reinforcement learning by policy search, 2000.

Marek Petrik, Gavin Taylor, Ronald Parr, and Shlomo Zilberstein. Feature selection using regularization in approximate linear programs for Markov decision processes. In *ICML*, 2010.

Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for pomdps. In *International Joint Conference Artificial Intelligence*, pages 1025 – 1032, August 2003.

Jim Pitman. Poisson–Dirichlet and GEM Invariant Distributions for Split-and-Merge Transformations of an Interval Partition. *Comb. Probab. Comput.*, 11(5):501–514, 2002. ISSN 0963-5483. doi: http://dx.doi.org/10.1017/S0963548302005163.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

Adrian Raftery. Choosing models for cross-classifications. *Americal Sociological Review*, 51, 1986.

C. E. Rasmussen. The Infinite Gaussian Mixture Model. In *NIPS 12*, 2000.

C. E. Rasmussen and Z. Ghahramani. Infinite mixtures of Gaussian process experts. In *NIPS 14*, 2002.

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. 2005. ISBN 026218253X.

Carl Edward Rasmussen. The Infinite Gaussian Mixture Model. In *NIPS*, pages 554–560, 1999.

Carl Edward Rasmussen and Zoubin Ghahramani. Infinite Mixtures of Gaussian Process Experts. In *NIPS*, pages 881–888, 2001.

Bohdana Ratitch and Doina Precup. Sparse distributed memories for on-line value-based reinforcement learning. In *Machine Learning: ECML*, Lecture Notes in Computer Science, 2004.

Ioannis Rexakis and Michail G. Lagoudakis. Classifier-based policy representation. In *Seventh International Conference on Machine Learning and Applications, ICMLA*, 2008.

Stephane Ross, Joelle Pineau, Sebastien Paquet, and Brahim Chaib-Draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32: 663–704, July 2008.

Nicholas Roy and Caleb Earnest. Dynamic action spaces for information gain maximization in search and exploration, 2006.

R.Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., 1981.

G. A. Rummery and M. Niranjan. Online Q-learning using connectionist systems (tech. rep. no. cued/f-infeng/tr 166). *Cambridge University Engineering Department*, 1994.

B. Saha, K. Goebel, and J. Christophersen. Comparison of Prognostic Algorithms for Estimating Remaining Useful Life of Batteries. *Transactions of the Institute of Measurement and Control*, 31(3-4):293–308, 2009.

J. Sethuraman. A Constructive Definition of Dirichlet Priors. *Statistica Sinica*, 4: 639–650, 1994.

Xuhui Shao, Vladimir Cherkassky, and William Li. Measuring the vc-dimension using optimized experimental design. *Neural Computation*, 12:2000, 1969.

John Shawe-Taylor, Royal Holloway, Peter L. Bartlett, Robert C. Williamson, and Martin Anthony. Structural risk minimization over data-dependent hierarchies, 1996.

Satinder Singh, Richard S. Sutton, and P. Kaelbling. Reinforcement learning with replacing eligibility traces. In *Machine Learning*, pages 123–158, 1996.

Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT press, 2006.

Mark W. Spong. Underactuated mechanical systems. In *Control Problems in Robotics and Automation*. Springer-Verlag, 1998.

Masashi Sugiyama. Active learning for misspecified models. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1305–1312. MIT Press, Cambridge, MA, 2006.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, May 1998.

Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

Christopher Tay and Christian Laugier. Modelling smooth paths using gaussian processes. In *International Conference on Field and Service Robotics*, 2007.

Y. W. Teh. Dirichlet Processes. In *Encyclopedia of Machine Learning*. Springer, 2010.

K. Ure, A. Geramifard, G. Chowdhary, and J. P. How. Adaptive Planning for Markov Decision Processes with Uncertain Transition Models via Incremental Feature Dependency Discovery. In *European Conference on Machine Learning (ECML)*, 2012.

Vladimir Vapnik. *Statistical learning theory*. Wiley, 1998. ISBN 978-0-471-03003-4.

Vladimir Vapnik, Esther Levin, and Yann LeCun. Measuring the vc-dimension of a learning machine. *Neural Computation*, 6(5):851–876, 1994.

Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc, 1995.

Dizan Alejandro Vasquez Govea, Thierry Fraichard, and Christian Laugier. Growing Hidden Markov Models: An Incremental Tool for Learning and Predicting Human and Vehicle Motion. *International Journal of Robotics Research*, 28(11-12):1486–1506, November 2009.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of Meta-Learning. *Artificial Intelligence Review*, 18:77–95, 2002.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3): 279–292, May 1992. doi: 10.1007/BF00992698.

Ye Wen, Rich Wolski, and Chandra Krintz. Online Prediction of Battery Lifetime for Embedded and Mobile Devices. In *Special Issue on Embedded Systems*. Springer-Verlag Heidelberg Lecture Notes in Computer Science, 2004.

Halbert White. Maximum likelihood estimation of misspecified models. *Econometrica*, 50(1):1–25, January 1982. ISSN 00129682. doi: 10.2307/1912526.

Shimon Whiteson, Matthew Taylor, and Peter Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007a.

Shimon Whiteson, Matthew Taylor, and Peter Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007b.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

Brian D. Ziebart, Andrew Maas, James Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, July 2008.