Bayesian Nonparametric Reward Learning from Demonstration

by
Bernard J. Michini
S.M., Aeronautical/Astronautical Engineering
Massachusetts Institute of Technology (2009)
S.B., Aeronautical/Astronautical Engineering
Massachusetts Institute of Technology (2007)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 2013

[SEPTEMBER 2013]

Author .....
Department of Aeronautics and Astronautics
August, 2013

Certified by .....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Nicholas Roy
Associate Professor of Aeronautics and Astronautics

Certified by .....
Mary Cummings
Associate Professor of Aeronautics and Astronautics

Accepted by ... .
Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Bayesian Nonparametric Reward Learning from Demonstration

by

Bernard J. Michini

Submitted to the Department of Aeronautics and Astronautics
on August, 2013, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Learning from demonstration provides an attractive solution to the problem of teaching autonomous systems how to perform complex tasks. Demonstration opens autonomy development to non-experts and is an intuitive means of communication for humans, who naturally use demonstration to teach others. This thesis focuses on a specific form of learning from demonstration, namely inverse reinforcement learning, whereby the *reward* of the demonstrator is inferred. Formally, inverse reinforcement learning (IRL) is the task of learning the reward function of a Markov Decision Process (MDP) given knowledge of the transition function and a set of observed demonstrations. While reward learning is a promising method of inferring a rich and transferable representation of the demonstrator's intents, current algorithms suffer from intractability and inefficiency in large, real-world domains. This thesis presents a reward learning framework that infers multiple reward functions from a single, unsegmented demonstration, provides several key approximations which enable scalability to large real-world domains, and generalizes to fully continuous demonstration domains without the need for discretization of the state space, all of which are not handled by previous methods.

In the thesis, modifications are proposed to an existing Bayesian IRL algorithm to improve its efficiency and tractability in situations where the state space is large and the demonstrations span only a small portion of it. A modified algorithm is presented and simulation results show substantially faster convergence while maintaining the solution quality of the original method. Even with the proposed efficiency improvements, a key limitation of Bayesian IRL (and most current IRL methods) is the assumption that the demonstrator is maximizing a *single* reward function. This presents problems when dealing with unsegmented demonstrations containing multiple distinct tasks, common in robot learning from demonstration (e.g. in large tasks that may require multiple subtasks to complete). A key contribution of this thesis is the development of a method that learns *multiple* reward functions from a single demonstration. The proposed method, termed Bayesian nonparametric inverse reinforcement learning (BNIRL), uses a Bayesian nonparametric mixture model

to automatically partition the data and find a set of simple reward functions corresponding to each partition. The simple rewards are interpreted intuitively as subgoals, which can be used to predict actions or analyze which states are important to the demonstrator. Simulation results demonstrate the ability of BNIRL to handle cyclic tasks that break existing algorithms due to the existence of multiple subgoal rewards in the demonstration. The BNIRL algorithm is easily parallelized, and several approximations to the demonstrator likelihood function are offered to further improve computational tractability in large domains.

Since BNIRL is only applicable to discrete domains, the Bayesian nonparametric reward learning framework is extended to general continuous demonstration domains using Gaussian process reward representations. The resulting algorithm, termed Gaussian process subgoal reward learning (GPSRL), is the only learning from demonstration method that is able to learn multiple reward functions from unsegmented demonstration in general continuous domains. GPSRL does not require discretization of the continuous state space and focuses computation efficiently around the demonstration itself. Learned subgoal rewards are cast as Markov decision process options to enable execution of the learned behaviors by the robotic system and provide a principled basis for future learning and skill refinement. Experiments conducted in the MIT RAVEN indoor test facility demonstrate the ability of both BNIRL and GPSRL to learn challenging maneuvers from demonstration on a quadrotor helicopter and a remote-controlled car.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

Foremost, I'd like to thank my family and friends. None of my accomplishments would be possible without their constant support. To Mom, Dad, Gipper, Melissa, and the rest of my family: there seems to be very little in life that can be counted on, yet you have proven time and again that no matter what happens you will always be there for me. It has made more of a difference than you know. To the many many wonderful friends I've met over the last 10 years at the Institute (of whom there are too many to name), and to the men of Phi Kappa Theta fraternity who I've come to consider my brothers: despite the psets, projects, and deadlines, you've made life at MIT the most enjoyable existence I could ask for. I constantly think on the time we shared, both fondly and jealously, and I truly hope that we continue to be a part of each others lives.

I'd also like to thank the academic colleagues who have given me so much assistance during my time as a graduate student. To my advisor, Professor Jon How: it's been my pleasure to learn your unique brand of research, problem solving, technical writing, and public speaking, and I'm extremely grateful for your mentorship over the years (though I sometimes hear *"what's the status"* in my sleep). Thanks also to my thesis committee, Professors Missy Cummings and Nicholas Roy, for their guidance and support. To the many past and present members of the Aerospace Controls Lab, including Dan Levine, Mark Cutler, Luke Johnson, Sam Ponda, Josh Redding, Frant Sobolic, Brett Bethke, and Brandon Luders: I couldn't have asked for a better group of colleagues to spend the many work days and late nights with. I wish you all the best of luck in your professional careers.

Finally, I'd like to thank the Office of Naval Research Science of Autonomy program for funding this work under contract #N000140910625.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

As humans, we perform a wide variety of tasks every day: determining when to leave home to get to work on time, choosing appropriate clothing given a typically-inaccurate weather forecast, braking for a stoplight with adequate margin for error (and other drivers), deciding how much cash to withdraw for the week's expenses, taking an exam, or locating someone's office in the Stata Center. While these tasks may seem mundane, most are deceivingly-complex and involve a myriad of pre-requisites like motor skills, sensory perception, language processing, social reasoning, and the ability to make decisions in the face of uncertainty.

Yet we are born with only a tiny fraction of these skills, and a key method of filling the gap is our incredible ability to *learn from others.* So critical is the act of learning that we spend our entire lifetime doing it. While we often take for granted its complexities and nuances, it is certain that our survival and success in the world are directly linked to our ability to learn.

A direct analogy can be drawn to robotic (and more generally *autonomous*) systems. As these systems grow in both complexity and role, it seems unrealistic that they will be programmed *a priori* with all of the skills and behaviors necessary to perform complex tasks. An autonomous system with the ability to learn from others has the potential to achieve far beyond its original design. While the notion of a robot with a human capacity for learning has long been a coveted goal of the artificial intelligence community, a multitude of technical hurdles have made realization of such

of a goal extremely difficult. Still, much progress has been and continues to be made using the tools available, highlighting the potential for an exciting future of capable autonomy.

1.1 Motivation: Learning from Demonstration in Autonomy

As technology continues to play a larger role in society, humans interact with autonomous systems on a daily basis. Accordingly, the study of human-robot interaction has seen rapid growth [29, 34, 39, 47, 64, 85, 92]. It is reasonable to assume that non-experts will increasingly interact with robotic systems and will have an idea of how the system *should* act. For the most part, however, autonomy algorithms are currently developed and implemented by technical experts such as roboticists and computer programmers. Modifying the behavior of these algorithms is mostly beyond the capabilities of the end-user. Learning from demonstration provides an attractive solution to this problem for several reasons [6]. The demonstrator is typically not required to have expert knowledge of the domain dynamics. This opens autonomy development to non-robotics-experts and also reduces performance brittleness from model simplifications. Also, demonstration is already an intuitive means of communication for humans, as we use demonstration to teach others in everyday life. Finally, demonstrations can be used to focus the automated learning process on useful areas of the state space [53], as well as provably expand the class of learnable functions [97].

There have been a wide variety of successful applications that highlight the utility and potential of learning from demonstration. Many of these applications focus on teaching basic motor skills to robotic systems, such as object grasping [83, 89, 96], walking [58], and quadraped locomotion [48, 73]. More advanced motor tasks have also been learned, such as pole balancing [7], robotic arm assembly [19], drumming [43], and egg flipping [67]. Demonstration has proved successful in teaching robotic

systems to engage in recreational activities such as soccer [4, 40, 41], air hockey [11], rock-paper-scissors [18], basketball [17], and even music creation [32]. While the previous examples are focused mainly on robotics, there are several instances of learning from demonstration in more complex, high-level tasks. These include autonomous driving [2, 21, 66], obstacle avoidance and navigation [44, 82], and unmanned acrobatic helicopter flight [23, 63].

1.2 Problem Formulation and Solution Approach

This thesis focuses on learning from demonstration, a demonstration being defined as a set of state-action pairs:

$$\boldsymbol{O} = \{(s_1, a_1), (s_2, a_2), ..., (s_N, a_N)\} \tag{1.1}$$

where $\boldsymbol{O}$ is the demonstration set, s_i is the state of the system, and a_i is the action that was taken from state s_i. In the thesis, it is assumed that states and actions are fully-observable, and problems associated with partial state/action observability are not considered. The demonstration may not necessarily be given in temporal order, and furthermore could contain redundant states, inconsistent actions, and noise resulting from imperfect measurements of physical robotic systems.

Learning from demonstration methods can be distinguished by *what is learned* from the demonstration. Broadly, there are two classes: those which attempt to learn a *policy* from the demonstration, and those which attempt to learn a *task description* from the demonstration. In policy learning methods, the objective is to learn a mapping from states to actions that is consistent with the state-action pairs observed in the demonstration. In that way, the learned policy can be executed on the autonomous system to generate behavior similar to that of the demonstrator.

Policy methods are not concerned with *what* is being done in the demonstration, but rather *how* it is being done. In contrast, task learning methods use the demonstration to infer the objective that the demonstrator is trying to achieve. A common way

of specifying such an objective is to define an associated *reward function*, a mapping from states to a scalar reward value. The task can then be more concretely defined as reaching states that maximize accumulated reward. This thesis focuses primarily on the problem of *reward learning from demonstration.*

Reward learning is challenging for several fundamental reasons:

- Learning rewards from demonstration necessitates a *model* of the demonstrator that predicts what actions would be taken given some reward function (or objective). The actions predicted by the model are compared to the demonstration as a means of inferring the reward function of the demonstrator. The demonstrator model is typically difficult to obtain in that it requires solving for a policy which maximizes a candidate reward function.

- The demonstration typically admits many possible corresponding reward functions, i.e. there is no unique reward function that explains a given set of observed state-action pairs.

- The demonstration itself can be inconsistent and the demonstrator imperfect. Thus, it cannot be assumed that the state-action pairs in the demonstration optimize reward, only that they *attempt* to do so.

Despite these difficulties, reward learning has several perceived advantages over policy learning. A policy, due to its nature as a direct mapping from states to actions, becomes invalid if the state transition model changes (actions may have different consequences than they did in the original demonstration). Also, a policy mapping must be defined for every necessary state, relying on a large demonstration set or additional generalization methods. A learned reward function, however, can be used to *solve for a policy* given knowledge of the state transition model, making it invariant to changes in domain dynamics and generalizable to new states. Thus, a reward function is a succinct and transferable description of the task being demonstrated and still provides a policy which generates behavior similar to that of the demonstrator.

This thesis focuses primarily on developing reward learning from demonstration techniques that are scalable to large, real-world, continuous demonstration domains

while retaining computational tractability. While previous reward learning methods assume that a single reward function is responsible for the demosntration, the framework developed in this thesis is based on the notion that the demonstration itself can be partitioned and explained using a class of simple reward functions. Two new reward learning methods are presented that utilize Bayesian nonparametric mixture models to simultaneously partition the demonstration and learn associated reward functions. Several key approximation methods are also developed with the aim of improving efficiency and tractability in large continuous domains. Simulation results are given which highlight key properties and advantages, and experimental results validate the new algorithms applied to challenging robotic systems.

The next section highlights relevant previous work in the field of learning from demonstration, and is followed by a more detailed summary of the thesis contributions.

1.3 Literature Review

The many methods for learning from demonstration can be broadly categorized into two main groups based on what is being learned [6]: a policy mapping function from states to actions, or a task description.

In the policy mapping approach, a function is learned which maps states to actions in either a discrete (classification) or continuous (regression) manner. Classification architectures used to learn low-level tasks include Gaussian Mixture Models for car driving [20], decision trees for aircraft control [77], and Bayesian networks [44] and k-Nearest Neighbors [78] for navigation and obstacle avoidance. Several classifiers have also been used to learn high-level tasks including Hidden Markov Models for box sorting [76] and Support Vector Machines for ball sorting [21]. Continuous regression methods are typically used to learn low-level motion-related behaviors, and a few of the many examples include Neural Networks for learning to drive on various road types [66], Locally-Weighted Regression [22] for drumming and walking [43, 58], and Sparse Online Gaussian Processes for basic soccer skills [41]. Actions are often defined

along with a set of necessary pre-conditions and resulting post-conditions [33]. Some examples include learning object manipulation [50], ball collection [94], navigation from natural language dialog [51], and single-demonstration learning [36].

Of the learning from demonstration methods which learn a task description, most do so by learning a reward function. In [7], the transition model is learned from repeated attempts to perform an inverted pendulum task, and the reward function (the task itself) is learned from human demonstrations. The demonstrations double as a starting point for the policy search to focus the computation on a smaller volume of the state space. Similar approaches approaches are taken in [28, 36, 93]. When the transition function is assumed to be known (at least approximately), a reward function can be found that rationalizes the observed demonstrations. In the context of control theory this problem is known as Inverse Optimal Control, originally posed by Kalman and solved in [16]. Ng and Russell cast the problem in the reinforcement learning framework in [62] and called it Inverse Reinforcement Learning (IRL), highlighting the fact that the reward function in many RL applications is often not known *a priori* and must instead be learned. IRL seeks to learn the reward function which is argued in [62] to be the "most succinct, robust, and transferable definition of the task".

There have since been a number of IRL methods developed, many of which use a weighted-features representation for the unknown reward function. Abbeel and Ng solve a quadratic program iteratively to find feature weights that attempt to match the expected feature counts of the resulting policy with those of the expert demonstrations [2]. A game-theoretic approach is taken in [90], whereby a minimax search is used to minimize the difference in weighted feature expectations between the demonstrations and learned policy. In this formulation, the correct signs of the feature weights are assumed to be known and thus the learned policy can perform better than the expert. Ratliff et al. [73, 74] take a max-margin approach, finding a weight vector that explains the expert demonstrations by essentially optimizing the margin between competing explanations. Ziebart et al. [99, 100] match feature counts using the principle of maximum entropy to resolve ambiguities in the resulting reward function. In [61], the parameters of a generic family of parametrized rewards

are found using a more direct gradient method which focuses on policy matching with the expert. Finally, Ramachandran and Amir [71] take a general Bayesian approach, termed Bayesian Inverse Reinforcement Learning (BIRL).

All of the aforementioned IRL algorithms are similar in that they attempt to find a single reward function that explains the entirety of the observed demonstration. This reward function must then be necessarily complex in order to explain the data sufficiently, especially when the task being demonstrated is itself complicated. Searching for a complex reward function is fundamentally difficult for two reasons. First, as the complexity of the reward model increases, so too does the number of free parameters needed to describe the model. Thus the search is over a larger space of candidate functions. Second, the process of testing candidate reward functions requires solving for the MDP value function, the computational cost of which typically scales poorly with the size of the MDP state space, even for approximate solutions [13]. Thus finding a single, complex reward function to explain the observed demonstrations requires searching over a large space of possible solutions and substantial computational effort to test each candidate.

The algorithms presented in this thesis avoid the search for a single reward function by instead partitioning the demonstration and inferring a reward function for each partition. This enables the discovery of multiple reward functions from a single, unsegmented demonstration. Several methods have been developed that also address the issue of multimodal learning from unsegmented demonstration. Grollman *et al.* characterize the demonstration as a mixture of Gaussian process experts [41] and find multiple policies to describe the demonstration. Also using a Bayesian nonparametric framework, Fox *et al.* cast the demonstration as a switched linear dynamic system, and infer a hidden Markov model to indicate switching between systems [35]. In Constructing Skill Trees (CST) the overall task is represented as a hierarchy of subtasks, and Markov decision process options (skills) are learned for each subtask. [49]. Of these methods, none attempt to learn multiple *reward* functions from unsegmented demonstration.

Throughout the thesis, subgoals are used as simple reward representations to

explain partitioned demonstration data. The notion of defining tasks using a corresponding subgoal was proposed by Sutton *et al.* along with the options MDP framework [88]. Many other methods exist which learn options from a given set of trajectories. In [55], diverse density across multiple solution paths is used to discover such subgoals. Several algorithms use graph-theoretic measures to partition densely-connected regions of the state space and learn subgoals at bottleneck states [56, 81]. Bottleneck states are also identified using state frequencies [84] or using a local measure of relative novelty [80]. Of these methods, most require large amounts of trajectory data and furthermore none have the ability to learn *reward* functions from demonstration.

1.4 Summary of Contributions

This thesis focuses broadly on improving existing reward learning from demonstration methods and developing new methods that enable scalable reward learning for real-world robotic systems. A reward learning framework is developed that infers multiple reward functions from a single, unsegmented demonstration, provides several key approximations which enable scalability to large real-world domains, and generalizes to fully continuous demonstration domains without the need for discretization of the state space, none of which are handled by previous methods.

The first contribution of the thesis is the proposal of several modifications to the Bayesian IRL algorithm to improve its efficiency and tractability in situations where the state space is large and the demonstrations span only a small portion of it. The key insight is that the inference task should be focused on states that are similar to those encountered by the expert, as opposed to making the naive assumption that the expert demonstrations contain enough information to accurately infer the reward function over the entire state space. With regard to the improvement of Bayesian IRL, the thesis makes the following contributions:

- Two key limitations of the Bayesian IRL algorithm are identified. Foremost, it is shown that the set of demonstrations given to the algorithm often contains

a limited amount of information relative to the entire state space. Even so, standard BIRL will attempt to infer the reward of *every* state. Second, the MCMC sampling in BIRL must search over a reward function space whose dimension is the number of MDP states. Even for toy problems, the number of MCMC iterations needed to approximate the mean of the posterior will become intractably large.

- A fundamental improvement is proposed which introduces a kernel function quantifying similarity between states. The BIRL inference task is then scaled down to include only those states which are similar to the ones encountered by the expert (the degree of "similarity" being a parameter of the algorithm). The resulting algorithm is shown to have much improved computational efficiency while maintaining the quality of the resulting reward function estimate. If the kernel function provided is simply a constant, the original BIRL algorithm is obtained.

- A new acceptance probability is proposed similar to a cooling schedule in Simulated Annealing to improve speed of convergence to the BIRL prior mode. Use of the cooling schedule in the modified BIRL algorithm allows the MCMC process to first find areas of high posterior probability and focus the samples towards them, speeding up convergence.

Even with the proposed efficiency improvements, a key limitation of Bayesian IRL (and most current IRL methods) is the assumption that the demonstrator is maximizing a *single* reward function. This presents problems when dealing with unsegmented demonstrations containing multiple distinct tasks, common in robot learning from demonstration (e.g. in large tasks that may require multiple subtasks to complete). The second contribution of this thesis is the development of a method that learns *multiple* reward functions from a single demonstration. With respect to learning multiple reward functions, the thesis makes the following contributions:

- A new reward learning framework is proposed, termed Bayesian nonparametric inverse reinforcement learning (BNIRL), which uses a Bayesian nonparamet-

ric mixture model to automatically partition the data and find a set of simple reward functions corresponding to each partition. The simple rewards are interpreted intuitively as subgoals, which can be used to predict actions or analyze which states are important to the demonstrator.

- Convergence of the BNIRL algorithm in 0-1 loss is proven. Several computational advantages of the method over existing IRL frameworks are shown, namely the search over a finite (as opposed to infinite) space of possible rewards and the ability to easily parallelize the majority of the method's computational requirements.

- Simulation results are given for simple examples showing comparable performance to other IRL algorithms in nominal situations. Moreover, the proposed method handles cyclic tasks (where the agent begins and ends in the same state) that would break existing algorithms without modification due to the existence of multiple subgoal rewards in a single demonstration.

- Two approximations to the demonstrator likelihood function are developed to further improve computational tractability in large domains. In the first method, the Real-time Dynamic Programming (RTDP) framework is incorporated to approximate the optimal action-value function. RTDP effectively limits computation of the value function only to necessary areas of the state space. This allows the complexity of the BNIRL reward learning method to scale with the size of the demonstration set, *not* the size of the full state space. Simulation results for a Grid World domain show order of magnitude speedups over exact solvers for large grid sizes. In the second method, an existing closed-loop controller takes the place of the optimal value function. This avoids having to specify a discretization of the state or action spaces, extending the applicability of BNIRL to continuous demonstration domains when a closed-loop controller is available. Simulation results are given for a pedestrian data set, demonstrating the ability to learn meaningful subgoals using a very simple closed-loop control law.

While BNIRL has the ability to learn multiple reward functions from a single demonstration, it is only generally applicable in discrete domains when a closed-loop controller is not available. A main focus area of the thesis is achieving scalable reward learning from demonstration in real-world robotic systems, necessitating the extension of the Bayesian nonparametric reward learning framework to general, continuous demonstration domains. With respect to reward learning in continuous domains, this thesis makes the following contributions:

- The Bayesian nonparametric reward learning framework is extended to general continuous demonstration domains using Gaussian process reward representations. The resulting algorithm, termed Gaussian process subgoal reward learning (GPSRL), is the only learning from demonstration method able to learn multiple reward functions from unsegmented demonstration in general continuous domains. GPSRL does not require discretization of the continuous state space and focuses computation efficiently around the demonstration itself.

- Learned subgoal rewards are cast as Markov decision process options to enable execution of the learned behaviors by the robotic system and provide a principled basis for future learning and skill refinement. Definitions of the option initiation set, terminating criteria, and policy follow directly from data already inferred during the GPSRL reward learning process. This enables execution of learned subgoals without the requirement for further learning.

- A method is developed for choosing the key confidence parameter in the GPSRL likelihood function. The method works by instructing the demonstrator to execute a single maneuver several times, and doing a sweep of the parameter to identify regions of under- and over-fitting. Furthermore, this method can be used to quantify the relative skill level of the demonstrator, enabling comparison between multiple demonstrators.

Since the broad focus of this work is to enable scalable reward learning from demonstration, the final contribution of the thesis is to provide experimental results

demonstrating the ability of the proposed methods to learn reward from demonstrations in real-world robotic domains. With respect to experimental validation of the methods presented herein, the thesis makes the following contributions:

- Quadrotor flight maneuvers are learned from a human demonstrator using only hand motions. The demonstration is recorded using a motion capture system and then analyzed by the BNIRL algorithm with action comparison. Learned subgoal rewards (in the form of waypoints) are passed as commands to an autonomous quadrotor which executes the learned behavior in actual flight. The entire process from demonstration to reward learning to robotic execution takes on the order of 10 seconds to complete using a single computer. Thus, the results highlight the ability of BNIRL to use data from a safe (and not necessarily dynamically feasible) demonstration environment and quickly learn subgoal rewards that can be used in the actual robotic system.

- GPSRL is experimentally applied to a robotic car domain. In the experiments, multiple difficult maneuvering skills such as drifting turns are identified from a single unsegmented demonstration. The learned subgoal rewards are then executed autonomously using MDP options and shown to closely match the original demonstration. Finally, the relative skill level of the demonstrator is quantified through *a posteriori* analysis of the confidence likelihood parameter.

1.5 Thesis Outline

The thesis proceeds as follows. Chapter 2 provides background material on the mathematical concepts that the thesis builds on. Chapter 3 presents several fundamental modifications to the existing Bayesian IRL method to improve efficiency and tractability in large domains. In Chapter 4, a new Bayesian nonparametric reward learning framework is developed enabling the discovery of multiple reward functions from a single demonstration. Chapter 5 offers several approximations to the BNIRL likelihood function that further enables scalability to large domains. In Chapter 6, the GPSRL

algorithm is developed as a generalized, continuous extension of BNIRL. Chapter 7 provides experimental results demonstrating the application of BNIRL and GPSRL to quadrotor helicopter and remote-controlled car domains. Finally, Chapter 8 offers concluding remarks and highlights areas for future research.

Chapter 2

Background

This chapter provides a background in the mathematical concepts that this thesis builds upon. Throughout the thesis, boldface is used to denote vectors and subscripts are used to denote the elements of vectors (i.e. z_i is the ith element of vector $\boldsymbol{z}$).

2.1 Markov Decision Processes and Options

A finite-state Markov Decision Process (MDP) [69] is a tuple (S, A, T, R, γ) where S is a set of *states*, A is a set of *actions*, $T : S \times A \times S \mapsto [0, 1]$ is the function of *transition probabilities* such that $T(s, a, s')$ is the probability of being in state s' after taking action a from state s, $R : S \mapsto \mathbb{R}$ is the *reward function*, and $\gamma \in [0, 1)$ is the *discount factor*.

A *stationary policy* is a function $\pi : S \mapsto A$. From [87] we have the following set of definitions and results:

1. The infinite-horizon expected reward for starting in state s and following policy π thereafter is given by the *value function* $V^\pi(s, R)$:

$$V^\pi(s, R) = E_\pi \left[\sum_{i=0}^{\infty} \gamma^i R(s_i) \middle| s_0 = s \right] \tag{2.1}$$

where s_i is the state at time i. Assuming state-based reward (i.e. rewards that do not depend on actions), the value function satisfies the following Bellman

equation for all $s \in S$:

$$V^{\pi}(s, R) = R(s) + \sum_{s'} \gamma \, T(s, \pi(s), s') V^{\pi}(s') \tag{2.2}$$

The so-called Q-function (or action-value function) $Q^{\pi}(s, a, R)$ is defined as the infinite-horizon expected reward for starting in state s, taking action a, and following policy π thereafter:

$$Q^{\pi}(s, a, R) = R(s) + \sum_{s'} \gamma \, T(s, a, s') V^{\pi}(s') \tag{2.3}$$

2. A policy π is optimal iff, for all $s \in S$:

$$\pi(s) = \underset{a \in A}{\operatorname{argmax}} \, Q^{\pi}(s, a, R) \tag{2.4}$$

An optimal policy is denoted as π^* with corresponding value function V^* and action-value function Q^*.

There are many methods available for computing or approximating V^* (and thus Q^*) when the transition function T is either known or unknown [13, 69, 87]. Throughout the thesis, T is assumed known (either exactly or approximately). A principal method for iteratively calculating the optimal value function V^* when T is known is called *value iteration*, an algorithm based on *dynamic programming* [10]. In value iteration, the Bellman equation (2.2) is used as an update rule which provides a successive approximation to the optimal value function V^*. Let V_k be the estimated value function at iteration k, then:

$$V_{k+1}(s, R) = R(s) + \max_{a} \sum_{s'} \gamma \, T(s, a, s') V_k(s') \qquad \forall s \in S \tag{2.5}$$

The sequence $\{V_k\}$ can be shown to converge to V^* under the same mild conditions that guarantee the existence of V^* [13]. In general, value iteration requires an infinite number of iterations to converge. In practice, the algorithm terminates when the maximum change in value from one iteration to the next is less than some threshold,

i.e. when:

$$\max_s |V_k(s) - V_{k-1}(s)| < \epsilon \tag{2.6}$$

Value iteration is used throughout the thesis as a simple and reliable method for calculating optimal value functions in relatively small domains. However, since value iteration requires a sweep of the entire state space at each update, it is often impractical to use for larger domains. Many approximate methods exists that are based on value iteration but avoid sweeping the entire state space. Two such approximate methods (presented later in the thesis) are real-time dynamic programming [9] and Gaussian process dynamic programming [27].

Many hierarchical methods have been developed which employ temporally-extended macro actions, often referred to as *options*, to achieve complex tasks in large and challenging domains. An option, o, is defined by the tuple (I_o, π_o, β_o) [88]. $I_o : S \mapsto \{0, 1\}$ is the *initiation set*, defined to be 1 where the option can be executed and 0 elsewhere. $\pi_o : S \mapsto A$ is the *option policy* for each state where the option is defined according to I_o. Finally, $\beta_o : S \mapsto [0, 1]$ is the *terminating condition*, defining the probability that the option will terminate in any state for which the option is defined. Any method which creates new skills (in the form of options) must define at least I_o and β_o. The option policy π_o can be learned using standard RL methods.

2.2 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL)[62] is the problem of inferring the reward function responsible for generating observed optimal behavior. Formally, IRL assumes a given MDP/R, defined as a MDP for which everything is specified except the state reward function $R(s)$. Observations (demonstrations) are provided as a set of state-action pairs:

$$\boldsymbol{O} = \{(s_1, a_1), (s_2, a_2), ..., (s_N, a_N)\} \tag{2.7}$$

where each pair $O_i = (s_i, a_i)$ indicates that the demonstrator took action a_i while in state s_i. Inverse reinforcement learning algorithms attempt to find a reward function

that rationalizes the observed demonstrations, i.e. find a reward function $\widehat{R}(s)$ whose corresponding optimal policy π^* matches the observations $\boldsymbol{O}$.

It is clear that the IRL problem stated in this manner is ill-posed. Indeed, $\widehat{R}(s) = c \ \forall s \in S$, where c is any constant, will make any set of state-action pairs O trivially optimal. Also, O may contain inconsistent or conflicting state-action pairs, i.e. (s_i, a_1) and (s_i, a_2) where $a_1 \neq a_2$. Furthermore, the "rationality" of the demonstrator is not well-defined (e.g., is the demonstrator perfectly optimal, and if not, to what extent sub-optimal).

Most existing IRL algorithms attempt to resolve the ill-posedness by making some assumptions about the form of the demonstrator's reward function. For example, in [2] it is assumed that the reward is a sum of weighted state features, and a reward function is found that matches the demonstrator's feature expectations. In [74] a linear-in-features reward is also assumed, and a maximum margin optimization is used to find a reward function that minimizes a loss function between observed and predicted actions. In [71] it is posited that the demonstrator samples from a prior distribution over possible reward functions, and thus Bayesian inference is used to find a posterior over rewards given the observed data. An implicit assumption in these algorithms is that the demonstrator is using a single, fixed reward function.

The three IRL methods mentioned above (and other existing methods such as [53, 61, 90]) share a generic algorithmic form, which is given by Algorithm 1, where the various algorithms use differing definitions of "similar" in Step 6. We note that each iteration of the algorithm requires solving for the optimal MDP value function V^* in Step 4, and the required number of iterations (and thus MDP solutions) is potentially unbounded.

2.3 Chinese Restaurant Process Mixtures

The algorithms developed throughout the thesis combine IRL with a Bayesian non-parametric model for learning multiple reward functions, namely the Chinese restaurant process mixture model. The *Chinese restaurant process* (CRP) is a sequential

Algorithm 1 Generic inverse reinforcement learning algorithm

function GENERICIRL(MDP/R, Obs. $O_{1:N}$, Reward representation $\widehat{R}(s|w)$)
  $w^{(0)} \leftarrow$ Initial reward function parameters
  while iteration $t < t_{\max}$ **do**
    $V^* \leftarrow$ Optimal MDP value function for reward function $\widehat{R}(s|w^{(t-1)})$
    $\widehat{\pi} \leftarrow$ Optimal policy according to V^*
    $w^{(t)} \leftarrow$ Parameters to make $\widehat{\pi}$ more similar to demonstrations $O_{1:N}$
  end while
  return Reward function given by $\widehat{R}(s|w^{(t_{\max})})$
end function

construction of random partitions used to define a probability distribution over the space of all possible partitions, and is often used in machine learning applications which involve partitioning observed data[65]. The process by which partitions are constructed follows a metaphor whereby customers enter a Chinese restaurant and must choose a table. In the analogy, tables are used to represent partitions, and the Chinese restaurant has a potentially infinite number of tables available. The construction proceeds as follows:

1. The first customer sits at the first table.

2. Customer i arrives and chooses the first unoccupied table with probability $\frac{\eta}{i-1+\eta}$, and an occupied table with probability $\frac{c}{i-1+\eta}$, where c is the number of customers already sitting at that table.

The concentration hyperparameter η controls the probability that a customer starts a new table. Using $z_i = j$ to denote that customer i has chosen table j, C_j to denote the number of customers sitting at table j, and J_{i-1} to denote the number of tables currently occupied by the first $i-1$ customers, the assignment probability can be formally defined by:

$$P(z_i = j|z_{1\ldots i-1}) = \begin{cases} \frac{C_J}{i-1+\eta} & j \leq J_{i-1} \\ \frac{\eta}{i-1+\eta} & j = J_{i-1}+1 \end{cases} \tag{2.8}$$

This process induces a distribution over table partitions that is *exchangeable* [37], meaning that the order in which the customers arrive can be permuted and any partition with the same proportions will have the same probability. A Chinese restaurant process mixture is defined using the same construct, but each table is endowed with parameters θ of a probability distribution which generates data points x_i:

1. Each table j is endowed with parameter θ_j drawn i.i.d. from a prior $P(\theta)$.
2. For each customer i that arrives:
   (a) The customer sits at table j according to (2.8) (the assignment variable $z_i = j$).
   (b) A datapoint x_i is drawn i.i.d. from $P(x|\theta_j)$.

Thus each datapoint x_i has an associated table (partition) assignment $z_i = j$ and is drawn from the distribution $P(x|\theta_j)$ [1]. The CRP mixture is in the class of *Bayesian nonparametric models*, meaning that the number of resulting partitions is potentially infinite. This property arises from the fact that, as new a customer arrives, there is always a non-zero probability that a new table will be started. The ability of the CRP mixture to model data which are generated from a random and potentially infinite number of partitions is critical to the algorithms presented throughout the thesis.

2.4 CRP Inference via Gibbs Sampling

The CRP mixture from Section 2.3 describes a *generative model* for the data $\boldsymbol{x}$, i.e. the process by which each datapoint x_i was generated. For the algorithms presented in the thesis, the task will be to invert this process: given a set of observed data $\boldsymbol{x}$, infer each partition assignment $z_i = j$, and the associated mixture parameters θ_j.

Formally, this means inferring the *posterior* distribution over assignments and mixture parameters given observed data, $P(\boldsymbol{z}, \boldsymbol{\theta}|\boldsymbol{x})$. Bayes rule can be used to decompose this posterior:

[1] It is noted that the CRP mixture is directly analogous to the Dirichlet process mixture, whereby datapoints are generated directly from posterior draws of a Dirichlet process. CRPs are used throughout the thesis for consistency.

$$P(\boldsymbol{z}, \boldsymbol{\theta}|\boldsymbol{x}) \propto \underbrace{P(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta})}_{\text{likelihood}} \underbrace{P(\boldsymbol{z}, \boldsymbol{\theta})}_{\text{prior}} \tag{2.9}$$

where $P(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{\theta})$ is the *likelihood* of the data given the parameters and $P(\boldsymbol{z}, \boldsymbol{\theta})$ is the *prior* over parameters. Calculating the posterior distribution (2.9) analytically is not possible in the general case where the likelihood and prior are non-conjugate [37]. Even when the mixture parameters $\boldsymbol{\theta}$ come from a finite discrete distribution (which is the case throughout the thesis), exhaustive search for the maximum likelihood value of each z_i and θ_j is intractable due to the combinatorial number of possible partition assignments $\boldsymbol{z}$. A tractable alternative is to draw samples from the posterior (2.9), and approximate the desired statistics (e.g. the mode of the distribution) from a finite number of samples.

Gibbs sampling [38] is in the family of Markov chain Monte Carlo (MCMC) sampling algorithms and is commonly used for approximate inference of Bayesian non-parametric mixture models [30, 60, 86]. The Gibbs sampler works under the assumption that each target random variable can be tractably sampled *conditioned* on all of the others (it samples one variable at a time while holding the others constant).

Algorithm 2 outlines the generic Gibbs sampling procedure for the CRP mixture posterior (2.9). Note that the posterior variables to be inferred are sampled separately while others are held constant, i.e. each z_i is sampled in Step 17 and each θ_j is sampled in Step 10. The assignment sampling in Step 17 of the algorithm relies on the exchangeability of the CRP mixture model by assuming that each x_i is the last datapoint to arrive. The posterior assignment probability $p(z_i = j|z_{-i}, \theta_j)$ is then the direct product of the CRP prior (2.8) and the likelihood given the associated partition parameters:

$$p(z_i = j|z_{-i}, \theta_j) \propto \underbrace{p(z_i|z_{-i})}_{\text{CRP}} \underbrace{p(x_i|\theta_j)}_{\text{likelihood}} \tag{2.10}$$

It is assumed that $P(\theta|x)$, the conditional of θ given x, can be sampled.

Given that each sampling update of z_i and θ_j occurs infinitely often and some mild conditions on the update probabilities are met [60], the resulting samples $\boldsymbol{z}^{(t)}$

Algorithm 2 Generic CRP mixture Gibbs sampler

1: **while** iteration $t < T$ **do**
2: **for** each observation $x_i \in \boldsymbol{x}$ **do**
3: **for** each current partition $j^{(t)}$ **do**
4: $p(z_i = j|z_{-i}, \theta_j) \leftarrow$ Probability of partition j from (2.10)
5: **end for**
6: $p(z_i = k|z_{-i}, \theta_k) \leftarrow$ Probability of new partition with parameter $\theta_k \sim P(\theta|x_i)$
7: $z_i^{(t)} \leftarrow$ Sample partition assignment from normalized probabilities in lines 13–16
8: **end for**
9: **for** each current partition $j^{(t)}$ **do**
10: $\theta_j^{(t)} \leftarrow$ Resample from $P(\theta|\{x_i : z_i = j\})$
11: **end for**
12: **end while**
13: **return** samples $\boldsymbol{z}^{(1:T)}$ and $\boldsymbol{\theta}^{(1:T)}$, discarding samples for burn-in and lag if desired

and $\boldsymbol{\theta}^{(t)}$ can be shown to form a Markov chain whose stationary distribution is the target posterior (2.9). In other words, the samples $(\boldsymbol{z}^{(t)}, \boldsymbol{\theta}^{(t)})$ will converge to a sample from (2.9) as $t \to \infty$.

In practice, for a finite number of iterations T, the chain will be dependent on the initial state of the posterior variables and consecutive samples from the chain will be correlated (not i.i.d.). To mitigate the effect of arbitrary initial conditions, the first N *burn-in* samples are discarded. To mitigate correlation between samples, only every n^{th} *lagged* sample is kept, and the rest discarded. There is considerable debate as to whether these ad-hoc strategies are theoretically or practically justified, and in general it has proven difficult to characterize convergence of the Gibbs sampler to the stationary Markov chain [75].

Figure 2-1 shows an illustrative example of Algorithm 2 applied to a Chinese restaurant process mixture model where data are generated from 2-dimensional Gaussian clusters. In the example, the data $x \in \mathbb{R}^2$ are drawn randomly from five clusters, and the parameters to be estimated $\theta = \{\mu, \Sigma\}$ are the means and covariance matrices of each inferred cluster. The likelihood function from (2.10) is simply the unnormal-

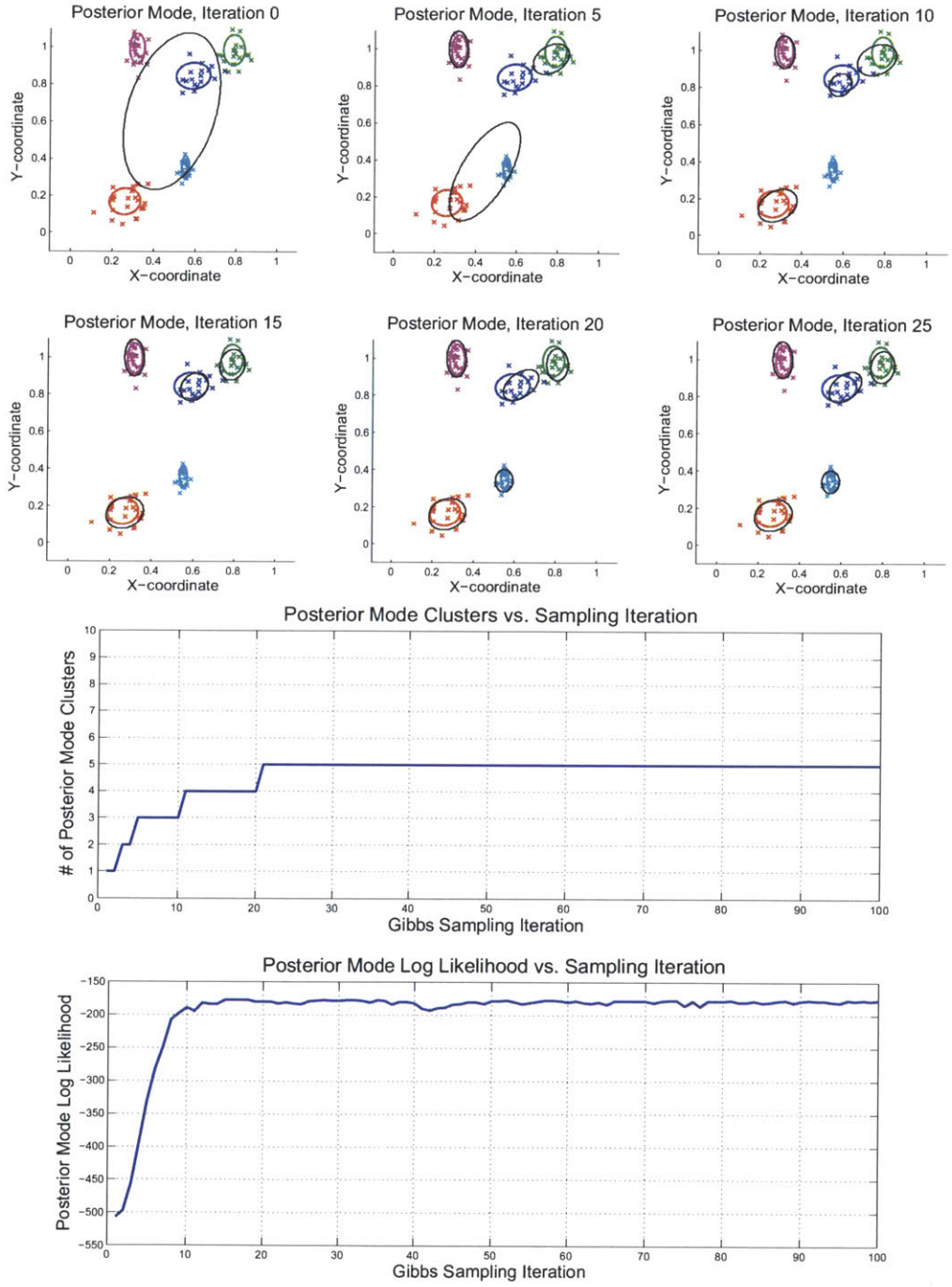


Figure 2-1: Example of Gibbs sampling applied to a Chinese restaurant process mixture model, where data are generated from 2-dimensional Gaussian clusters. Observed data from each of five clusters are shown in color along with cluster covariance ellipses (top). Gibbs sampler posterior mode is overlaid as black covariance ellipses after 0,5,10,15,20, and 25 sampling iterations (top). Number of posterior mode clusters versus sampling iteration (middle). Posterior mode log likelihood versus sampling iteration (bottom).

ized multivariate Gaussian probability density function (PDF), and $P(\theta|\boldsymbol{x})$ is taken to be the maximum likelihood estimate of the mean and covariance. Observed data from the five true clusters are shown in color along with the associated covariance ellipses (Figure 2-1 top). The Gibbs sampler posterior mode is overlaid as black covariance ellipses representing each cluster after 0,5,10,15,20, and 25 sampling iterations (Figure 2-1 top). The number of posterior mode clusters (Figure 2-1 middle) shows that the sampling algorithm, although it is not given the number of true clusters *a priori*, converges to the correct model within 20 iterations. The posterior mode log likelihood (Figure 2-1 bottom) shows convergence in model likelihood in just 10 iterations. The CRP concentration parameter in (2.8) used for inference is $\eta = 1$. Nearly identical posterior clustering results are attained for η ranging from 1 to 10000, demonstrating robustness to the selection of this parameter.

2.5 Gaussian Processes

A *Gaussian process* (GP) is a distribution over functions, widely used in machine learning as a nonparametric regression method for estimating continuous functions from sparse and noisy data [72]. In this thesis, Gaussian processes will be used as a subgoal reward representation which can be trained with a single data point but has support over the entire state space.

A training set consists of input vectors $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]$ and corresponding observations $\boldsymbol{y} = [y_1, ..., y_n]^\top$. The observations are assumed to be noisy measurements from the unknown target function f:

$$y_i = f(\boldsymbol{x}_i) + \epsilon \tag{2.11}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is Gaussian noise. A zero-mean Gaussian process is completely specified by a covariance function $k(\cdot, \cdot)$, called a *kernel*. Given the training data $\{\boldsymbol{X}, \boldsymbol{y}\}$ and covariance function $k(\cdot, \cdot)$, the Gaussian process induces a predictive marginal distribution for test point $\boldsymbol{x}_*$ which is Gaussian distributed so that $f(\boldsymbol{x}_*) \sim$

$\mathcal{N}(\mu_{f_*}, \sigma^2_{f_*})$ with mean and variance given by:

$$\mu_{f_*} = k(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y} \tag{2.12}$$

$$\sigma^2_{f_*} = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, \boldsymbol{X}) \left(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} k(\boldsymbol{X}, \boldsymbol{x}_*) \tag{2.13}$$

where $\boldsymbol{K} \in \mathbb{R}^{n \times n}$ is the Gram matrix with $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Selecting a kernel is typically application-specific, since the function $k(\boldsymbol{x}, \boldsymbol{x}')$ is used as a measure of correlation (or distance) between states $\boldsymbol{x}$ and $\boldsymbol{x}'$. A common choice (used widely throughout the thesis) is the squared exponential (SE) kernel:

$$k_{\text{SE}}(\boldsymbol{x}, \boldsymbol{x}') = \nu^2 \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^\top \Lambda^{-1} (\boldsymbol{x} - \boldsymbol{x}')\right) \tag{2.14}$$

where $\Lambda = \text{diag}(\ [\lambda_1^2, ..., \lambda_{n_x}^2]\ )$ are the characteristic length scales of each dimension of $\boldsymbol{x}$ and ν^2 describes the variability of f. Thus $\theta_{\text{SE}} = \{\nu, \lambda_1, ..., \lambda_{n_x}\}$ is the vector of hyperparameters which must be chosen for the squared exponential kernel. Choosing hyperparameters is typically achieved through maximization of the log evidence:

$$\begin{aligned} \log p(\boldsymbol{y}|\boldsymbol{X}, \theta) &= \log \int p(\boldsymbol{y}|f(\boldsymbol{X}), \boldsymbol{X}, \boldsymbol{\theta})\ p(f(\boldsymbol{X})|\boldsymbol{X}, \boldsymbol{\theta})\ df \\ &= -\frac{1}{2}\boldsymbol{y}^\top \left(\boldsymbol{K}_\theta + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}\ - \frac{1}{2}\log\left|\boldsymbol{K}_\theta + \sigma_\epsilon^2 \boldsymbol{I}\right| + c \end{aligned} \tag{2.15}$$

where c is a constant. Maximization of (2.15) w.r.t the hyperparameters involves unconstrained non-linear optimization which can be difficult in many cases. In practice, the optimization need only be carried out once for a representative set of training data, and local optimization methods such as gradient descent often converge to satisfactory hyperparameter settings [72].

The computational complexity of GP prediction is dominated by the inversion of the kernel matrix in (2.12), and is thus $\mathcal{O}(n^3)$ where n is the number of training points. This is in contrast to parametric regressors (such as least squares) where the complexity scales instead with the number of representational parameters. In practice, the system $\left(\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}$ from (2.12) need only be solved once and cached for a

given set of training data, reducing the complexity of new predictions to $\mathcal{O}(n^2)$. Also, many approximate GP methods exist for reducing the computational requirements for large training sets [24, 70, 72].

As an example of the ability of GPs to represent complex functions with a small amount of training data, Figure 2-2 shows Gaussian process approximations of an arbitrary Grid World value function. The full tabular value function for a 40×40 Grid World MDP requires storage of 1600 values (Figure 2-2, upper left). A Guassian process with a squared exponential kernel and 64 training points yields an average error of 1.7% over the original grid cells (Figure 2-2, upper right). A GP with just 16 training points yields an average error of 3.5% (Figure 2-2, lower left). A GP with 3 training points manually placed at each of the three value function peaks yields an average error of 5.9% (Figure 2-2, lower right). All kernel hyperparameters are learned using 10 iterations of standard gradient descent of the evidence (2.15). These examples demonstrate the ability of Gaussian processes (with appropriately selected kernel functions and training points) to represent a complex function using orders of magnitude fewer stored training points.

2.6 Summary

This chapter provided background in the mathematical concepts that this thesis builds upon. In the next chapter, several fundamental modifications are made to the Bayesian inverse reinforcement learning algorithm to improve its efficiency and tractability in situations where the state space is large and the demonstrations span only a small portion of it.

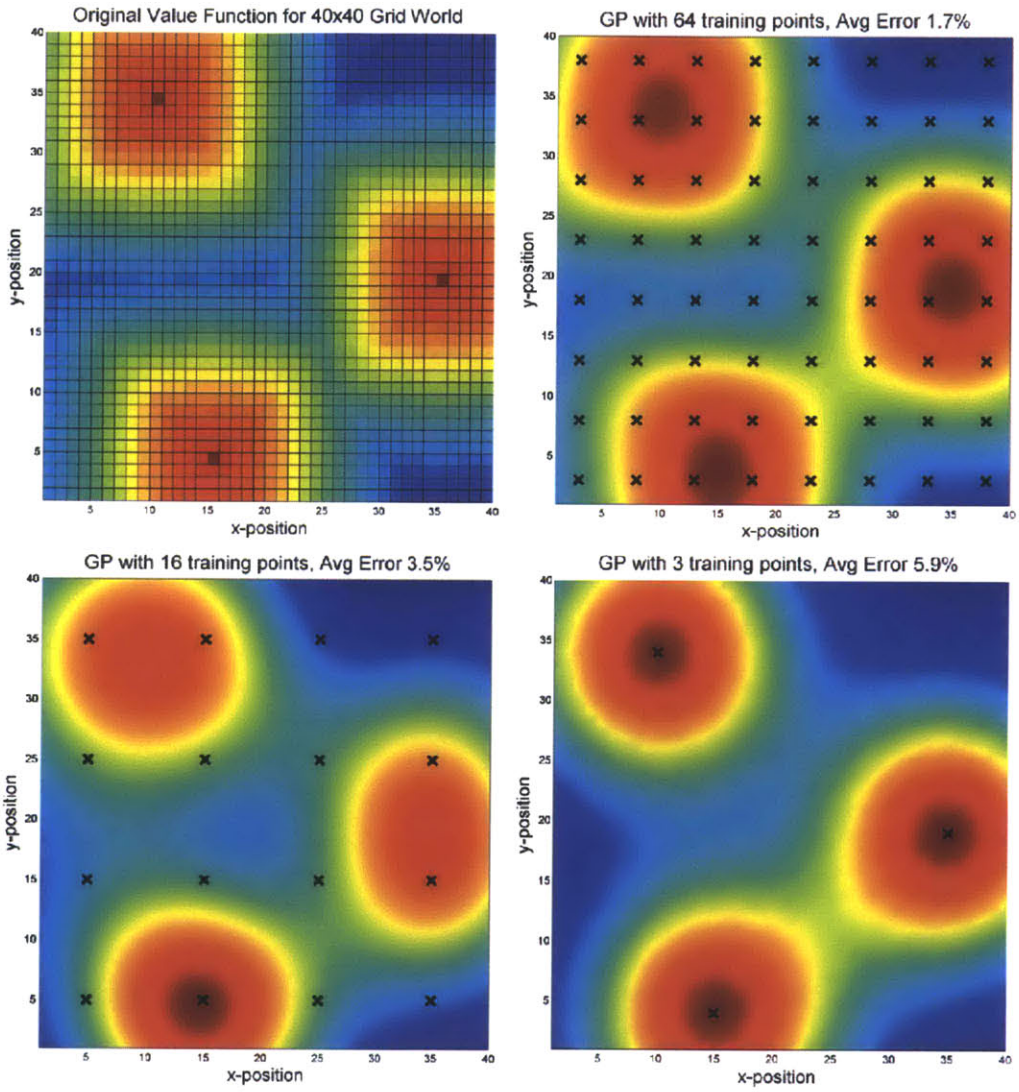


Figure 2-2: Example of Gaussian process (GP) approximation of a Grid World value function using the squared exponential kernel. The original 40×40 tabular value function for an arbitrary reward function requires 1600 values to be stored (upper left). A GP approximation with 160 training points (shown as black x's) yields an average error of 1.7% (upper right). A GP with 16 training points yields 3.5% average error (lower left), and a GP with just 3 training points yields 5.9% average error (lower right).

Chapter 3

Improving the Efficiency of Bayesian IRL

Inverse reinforcement learning (IRL) is the subset of learning from demonstration methods in which the reward function, or equivalently the task description, is learned from a set of expert demonstrations. The IRL problem is formalized using the Markov Decision Process (MDP) framework in the seminal work [62]: Given expert demonstrations in the form of state-action pairs, determine the reward function that the expert is optimizing assuming that the model dynamics (i.e. transition probabilities) are known.

In the larger context of learning from demonstration, many algorithms attempt to directly learn the policy (sometimes in addition to the model dynamics) using the given demonstrations [6]. IRL separates itself from these methods in that it is the *reward function* that is learned, not the policy. The reward function can be viewed as a high-level description of the task, and can thus "explain" the expert's behavior in a richer sense than the policy alone. No information is lost in learning the reward function instead of the policy. Indeed, given the reward function and model dynamics an optimal policy can be recovered (though many such policies may exist). Thus the reward function is also transferable, in that changing the model dynamics *would not* affect the reward function but *would* render a given policy invalid. For these reasons, IRL may be more advantageous than direct policy learning methods

in many situations.

It is evident that the IRL problem itself is ill-posed. In general, there is no single reward function that will make the expert's behavior optimal [53, 61, 74, 99, 100]. This is true even if the expert's policy is *fully specified* to the IRL algorithm, i.e. many reward functions may map to the same optimal policy. Another challenge in IRL is that in real-world situations the demonstrator may act sub-optimally or inconsistently. Finally, in problems with a large state space there may be a relatively limited amount of demonstration data.

Several algorithms address these limitations successfully and have shown IRL to be an effective method of learning from demonstration [1, 61, 73, 74, 90, 99, 100]. A general Bayesian approach is taken in Bayesian inverse reinforcement learning (BIRL) [71]. In BIRL, the reward learning task is cast as a standard Bayesian inference problem. A prior over reward functions is combined with a likelihood function for expert demonstrations (the evidence) to form a posterior over reward functions which is then sampled using Markov chain Monte Carlo (MCMC) techniques. BIRL has several advantages. It does not assume that the expert behaves optimally since a *distribution* over reward functions is recovered. Thus, the ambiguity of an inconsistent or uncertain expert is addressed explicitly. External *a priori* information and constraints on the reward function can be encoded naturally through the choice of prior distribution. Perhaps most importantly, the principled Bayesian manner in which the IRL problem is framed allows for the algorithm designer to leverage a wide range of inference techniques from the statistics and machine learning literature. Thus BIRL forms a general and powerful foundation for the problem of reward learning.

As discussed below, the Bayesian IRL algorithm as presented in [71] suffers from several practical limitations. The reward function to be inferred is a *vector* whose length is equal to the number of MDP states. Given the nature of the MCMC method used, a large number of iterations is required for acceptable convergence to the mean of the posterior. The problem stems mainly from the fact that each of these iterations requires re-solving the MDP for the optimal policy, which can be computationally expensive as the size of the state space increases (the so-called "curse

of dimensionality").

In this chapter, a modified Bayesian IRL algorithm is proposed based on the simple observation that the information contained in the expert demonstrations may very well *not* apply to the entire state space. As an abstract example, if the IRL agent is given a small set of expert trajectories that reside entirely in one "corner" of the state space, those demonstrations may provide little, if any, information about the reward function in some opposite "corner", making it naive to perform reward function inference over the entire state space. The proposed method takes as input a kernel function that quantifies similarity between states. The BIRL inference task is then scaled down to include only those states which are similar to the ones encountered by the expert (the degree of "similarity" being a parameter of the algorithm). The resulting algorithm is shown to have much improved computational efficiency while maintaining the quality of the resulting reward function estimate. If the kernel function provided is simply a constant, the original BIRL algorithm from [71] is obtained.

3.1 Bayesian IRL

The following summarizes the Bayesian inverse reinforcement learning framework [71]. The basic premise of BIRL is to infer a posterior distribution for the reward vector $\mathbf{R}$ from a prior distribution and a likelihood function for the evidence (the expert's actions). The evidence O takes the form of observed state-action pairs, so that $O = \{(s_1, a_1), (s_2, a_2), ..., (s_k, a_k)\}$. Applying Bayes Theorem, the posterior can be written as:

$$Pr(\mathbf{R}|O) = \frac{Pr(O|\mathbf{R})Pr(\mathbf{R})}{Pr(O)} \tag{3.1}$$

where each term is explained below:

- $Pr(\mathbf{R}|O)$: The posterior distribution of the reward vector given the observed actions of the expert. This is the target distribution whose mean will be estimated.

- $Pr(O|\mathbf{R})$: The likelihood of the evidence (observed expert state-action pairs) given a particular reward vector $\mathbf{R}$. A perfect expert would always choose optimal actions, and thus state-action pairs with large $Q^*(s_i, a_i, \mathbf{R})$ would be more likely. However, the expert is assumed to be imperfect, so the likelihood of each state-action pair is given by an exponential distribution:

$$Pr(a_i|s_i, \mathbf{R}) = \frac{e^{\alpha Q^*(s_i,a_i,\mathbf{R})}}{\sum_{b \in A} e^{\alpha Q^*(s_i,b,\mathbf{R})}} \tag{3.2}$$

  where α is a parameter representing our confidence that the expert chooses actions with high value (the lower the value of α the more "imperfect" the expert is expected to be). The likelihood of the entire evidence is thus:

$$Pr(O|\mathbf{R}) = \frac{e^{\alpha \sum_i Q^*(s_i,a_i,\mathbf{R})}}{\sum_{b \in A} e^{\alpha \sum_i Q^*(s_i,b,\mathbf{R})}} \tag{3.3}$$

- $Pr(\mathbf{R})$: Prior distribution representing how likely a given reward vector is based *only* on prior knowledge. This is where constraints and *a priori* knowledge of the rewards can be injected.

- $Pr(O)$: The probability of O over the entire space of reward vectors $\mathbf{R}$. This is very difficult to calculate but is not be needed for the MCMC methods used throughout the thesis.

For the reward learning task, we wish to estimate the expert's reward vector $\mathbf{R}$. One common way to determine the accuracy of an estimate is the *squared loss function*:

$$L_{SE}(\mathbf{R}, \hat{\mathbf{R}}) = ||\mathbf{R} - \hat{\mathbf{R}}||_2 \tag{3.4}$$

where $\mathbf{R}$ and $\hat{\mathbf{R}}$ are the actual and estimated expert reward vectors, respectively. It is shown in [71] that the *mean* of the posterior distribution (3.1) minimizes (3.4).

The posterior distribution of $\mathbf{R}$ must also be used to find a policy that is close to the expert's. Given some reward vector $\mathbf{R}$, a sensible measure of the closeness of

policy π to the optimal policy obtained using $\mathbf{R}$ is a *policy loss function*:

$$L^p_{\text{policy}}(\mathbf{R}, \pi) = ||V^*(\mathbf{R}) - V^\pi(\mathbf{R})||_p \tag{3.5}$$

where p is a norm. It is shown in [71] that the policy which minimizes (3.5) is the optimal policy obtained using the *mean* of the posterior (3.1).

Thus, for both the reward estimation and policy learning tasks, inference of the mean of the posterior (3.1) is required. Markov chain Monte Carlo (MCMC) techniques are appropriate for this task [5]. The method proposed in [71], termed `PolicyWalk`, iterates as follows. Given a current reward vector $\mathbf{R}$, sample a new proposal $\tilde{\mathbf{R}}$ randomly from the neighbors of $\mathbf{R}$ on a grid of length δ, i.e. $\mathbf{R} = \tilde{\mathbf{R}}$ except for one randomly chosen $s \in S$:

$$\tilde{\mathbf{R}}(s) = \mathbf{R}(s) \pm \delta \tag{3.6}$$

The proposal is accepted ($\mathbf{R} := \tilde{\mathbf{R}}$) with probability $\min\left\{1, \frac{Pr(\tilde{\mathbf{R}}|O)}{Pr(\mathbf{R}|O)}\right\}$, where the posteriors are given by (3.1) so that:

$$\frac{Pr(\tilde{\mathbf{R}}|O)}{Pr(\mathbf{R}|O)} = \frac{Pr(O|\tilde{\mathbf{R}})Pr(\tilde{\mathbf{R}})}{Pr(O)} \cdot \frac{Pr(O)}{Pr(O|\mathbf{R})Pr(\mathbf{R})} = \frac{Pr(O|\tilde{\mathbf{R}})Pr(\tilde{\mathbf{R}})}{Pr(O|\mathbf{R})Pr(\mathbf{R})} \tag{3.7}$$

The mean of the posterior is thus approximated by the empirical mean of $\mathbf{R}$ over all of the iterations. Note that none of the normalizing constants are needed and thus the likelihood and prior only need to be known to a constant. Here it is also noted that finding Q^* for the likelihood calculation requires the MDP to be solved using $\tilde{\mathbf{R}}$, and this must be done at *every* MCMC iteration. Solving the MDP each iteration is typical among IRL algorithms and is computationally challenging [13], highlighting the need to reduce the number of iterations to the extent possible.

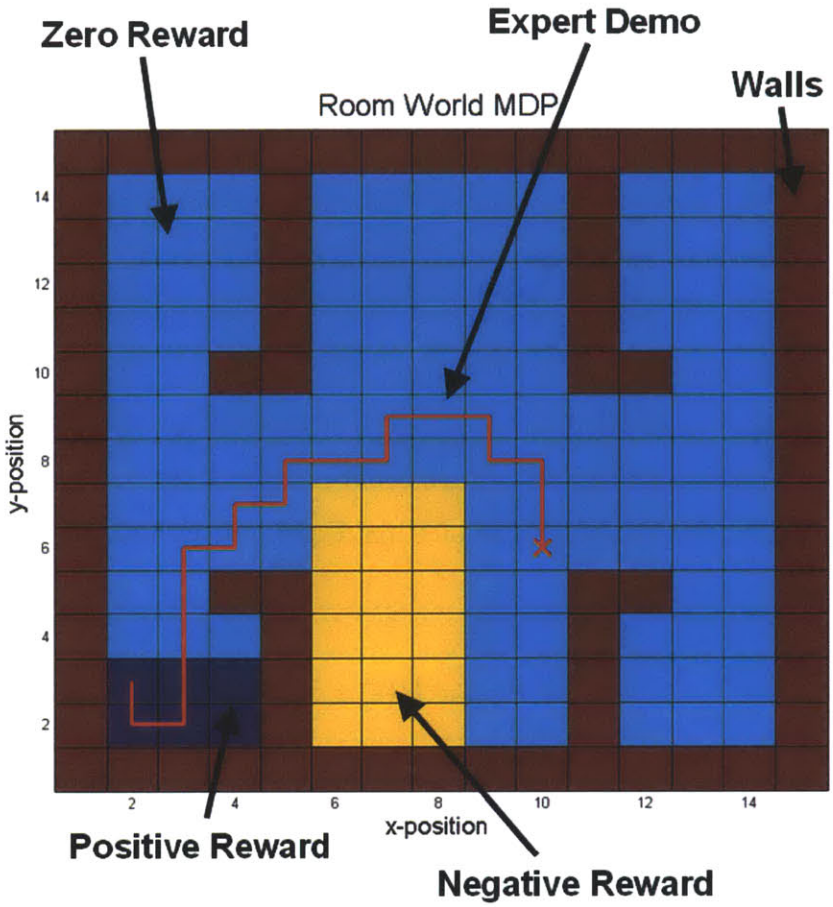


Figure 3-1: Room World MDP, showing the walls (maroon), zero reward (cyan), positive reward (dark blue), negative reward (yellow), and example expert demonstration (red).

3.2 Limitations of Standard Bayesian IRL

In this section, a simple example is presented that illustrates some practical limitations of the original Bayesian IRL algorithm from [71].

3.2.1 Room World MDP

"Room World", shown in Figure 3-1, is a 15×15 grid with walls that form five rooms. The true reward function consists of a block of negative reward in the center room, positive reward in the lower-left room, and zero reward elsewhere. The agent can

choose from four actions (up, down, left, right). If "up" is chosen, the agent moves up with probability 0.75, left or right each with probability 0.1, and stays in the same cell with probability 0.05 (and similarly for the other actions). The agent is not allowed to enter wall states. The discount factor is 0.93 and the magnitude of rewards is 0.01. The "expert" executes the optimal policy found using the true reward function. To simulate an imperfect expert, the optimal action is chosen with probability 0.95, and a random action is chosen otherwise. The expert always starts in the cell $(x, y) = (10, 6)$. An example expert demonstration is shown in red in Figure 3-1.

3.2.2 Applying Bayesian IRL

The Bayesian IRL algorithm presented in [71] is applied to attempt to learn the reward function for the Room World MDP given a set of 100 expert demonstrations shown in Figure 3-3. The reward vector is assumed to be composed of independently identically distributed (i.i.d.) components, each with prior distribution:

$$Pr(R) = 0.4e^{-0.001(R-R_{\max}).^2} + 0.4e^{-0.001(R-R_{\min})^2} + e^{-0.001(R)^2} \tag{3.8}$$

as shown in Figure 3-2 (recall that the prior only needs to be known to a constant). This prior reflects external knowledge that for any given state the reward is most likely zero, and if not than will likely take the minimum or maximum reward value. [1]

For the Room World MDP with 225 states, the policy loss (defined in Section 3.4) converges after roughly 600 iterations as seen in Figure 3-4. Recall that each iteration requires solving the MDP to find the optimal policy according to the proposed reward function. While time can sometimes be saved by bootstrapping from the previous solution, the number of MCMC iterations required for a more realistically-large state space will quickly become prohibitive. There are two main reasons for this inefficiency, each discussed below.

[1] Note that the ability of Bayesian IRL to impose a prior such as this effectively reduces the ambiguity and ill-posedness of the IRL reward estimation problem by intuitively limiting the space of possible reward functions.

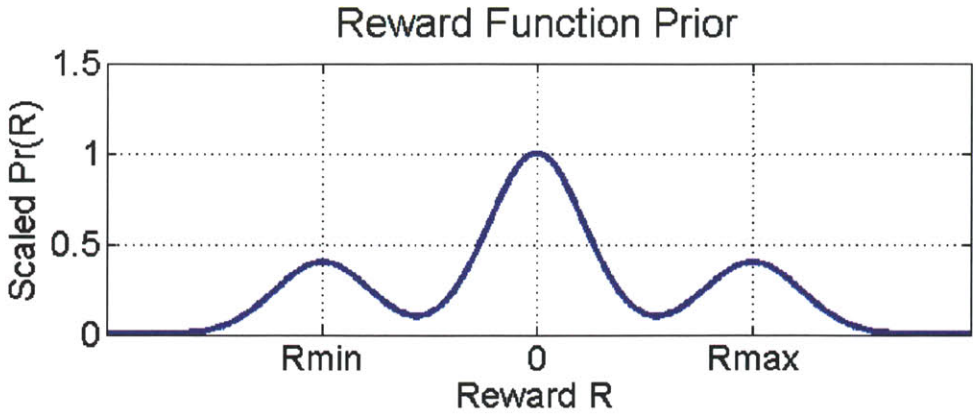


Figure 3-2: Reward function prior distribution (scaled).

Limited expert information: Foremost, in large state spaces, the set of expert demonstrations given to the BIRL algorithm contains a limited amount of information relative to the entire state space. Intuitively, it would be difficult to infer much about the reward function in the upper-right room since there are no observed state-action pairs near that area. Even so, standard BIRL will attempt to infer the reward of *every* state. Empirically, the estimates in states far from any expert demonstrations tend to "wander", slowing convergence of the posterior reward distribution as a whole. More concretely, when a new proposal is drawn in which a far-away state is perturbed, the effect on the likelihood of the evidence and the prior as a whole is very small. Thus $Pr(\tilde{\mathbf{R}}|O) \approx Pr(\mathbf{R}|O)$ and the acceptance probability $\frac{Pr(\tilde{\mathbf{R}}|O)}{Pr(\mathbf{R}|O)} \approx 1$, meaning that the new proposal will most likely be accepted. As a result, the reward estimate at far-away states will change frequently and in a way that is not affected by the evidence. The efficiency of the algorithm suffers since it spends much of its time attempting to infer the reward in states for which it has little expert evidence.

Exploration vs. exploitation: The MCMC algorithm must search over a reward function space whose dimension is the number of MDP states. Even for toy problems this can grow quickly, and as mentioned before the number of MCMC iterations needed to approximate the mean of the posterior will become extremely large. Simulated annealing is a method used to focus the sampled distribution around it's maximum by using a modified acceptance probability of $\left(\frac{Pr(\tilde{\mathbf{R}}|O)}{Pr(\mathbf{R}|O)}\right)^{1/T_i}$ where T_i is a

decreasing "cooling schedule" [5]. While this method is typically used to estimate the maximum of the posterior (MAP), it provides a "knob" to focus the samples on areas of higher posterior probability essentially trading exploration of the full distribution for exploitation of it's peaks. For high-dimensional reward spaces (i.e. MDPs with a large number of states), this is necessary to reduce the number of samples needed to converge to a high-probability area of the posterior.

3.3 Modifications to the BIRL Algorithm

This section describes two modifications to the original Bayesian IRL algorithm to address the aforementioned limitations. The modified BIRL method is shown in Algorithm 3 and explained below.

3.3.1 Kernel-based Relevance Function

It is unlikely that the observed expert demonstrations will span every state of the MDP, or even provide a sparse covering of the entire state space for large problems. Thus it is naive to assume that the reward function over the entire state space can be accurately inferred. Instead, it would make intuitive sense to learn the rewards in states "similar to" those encountered by the expert. The notion of similarity must be rigorously defined, and for this a kernel function is used. Kernel functions are commonly used in machine learning for exactly this purpose [79], and are defined as the dot product of two *feature vectors*. A feature is a mapping from states to feature space $\mathbf{\Phi} : S \mapsto \mathbb{R}^{k \times 1}$, so that the corresponding kernel function is given by:

$$k(s, s') = \mathbf{\Phi}^T(s) \cdot \mathbf{\Phi}(s') \tag{3.9}$$

While $k(s, s')$ corresponds to a dot product of feature vectors, this dot product need not be explicated. For instance, the popular radial basis kernel $k(s, s') = e^{-||s-s'||_2/2\sigma^2}$ represents the dot product of an infinitely long feature vector [79]. The kernel function is passed in as a parameter to the modified algorithm, and is used to define the *state*

relevance function $\rho : S \mapsto [0, 1]$:

$$\rho(s) = \frac{\sum_{s' \in O} k(s, s')}{Z} \tag{3.10}$$

where O is the set of expert state-action pairs and $Z = \max_{s \in S} \rho(s)$ is a normalizing constant. Intuitively $\rho(s)$ is a normalized measure of how similar state s is to the set of states encountered by the expert.

The state relevance $\rho(s)$ is used in the modified BIRL algorithm shown in Algorithm 3 as follows. To propose a new reward vector $\tilde{\mathbf{R}}$, a state $\tilde{s} \in S$ is sampled at random. The state is accepted with probability $\rho(\tilde{s})$, and the new reward vector proposal is chosen such that $\tilde{\mathbf{R}} := \mathbf{R}$, except for $\tilde{\mathbf{R}}(\tilde{s}) = \mathbf{R}(\tilde{s}) \pm \delta$. If $\tilde{s}$ is rejected, the process repeats until a state is accepted. This process models the original BIRL algorithm closely, except that now the reward search is focused more heavily on states that are more similar to those encountered by the expert. In the trivial case of a constant kernel $k(s, s') = C$ (i.e. each state s is equally similar to all other states), the original BIRL algorithm `PolicyWalk` from [71] is obtained.

The modified BIRL algorithm initializes the reward vector $\mathbf{R}$ to the maximum of the prior. This is because the state relevance modification causes the algorithm to effectively *not* infer the reward vector for states with a low relevance score, and thus the reward in these states needs to be initialized to a reasonable value. The relevance function can thus be thought of as a state-by-state measure of how much the expert demonstrations will affect the reward estimate at that state.

3.3.2 Cooling Schedule

As discussed in Section 3.2, the original BIRL algorithm lacks the ability to trade off exploration for exploitation in order to speed up convergence of the posterior. To address this, a small modification to the acceptance probability is made. As in

Algorithm 3 Modified BIRL Algorithm

function MODIFIEDBIRL(Posterior $Pr(\mathbf{R}|O)$, MDP/R M, Kernel k, Cooling Sched. T_i, Step Size δ)

  Initialize reward vector $\mathbf{R}$ to the max of the prior

  $(\pi^*, Q^*) \leftarrow$ `ValueIteration`$(M, \mathbf{R})$

  while iteration $i < I_{\max}$ **do**

    Draw $\tilde{s} \in S$ at random, accept $\tilde{s}$ with prob. $\rho(\tilde{s})$ from (3.10) or repeat

    $\tilde{\mathbf{R}} \leftarrow \mathbf{R}$, except for $\tilde{\mathbf{R}}(\tilde{s}) = \mathbf{R}(\tilde{s}) \pm \delta$

    $(\tilde{\pi}^*, \tilde{Q}^*) \leftarrow$ `ValueIteration`$(M, \tilde{\mathbf{R}}, \pi^*)$

    $\mathbf{R} \leftarrow \tilde{\mathbf{R}}$ and $\pi^* \leftarrow \tilde{\pi}^*$ with prob. $\min\left\{1, \left(\frac{Pr(\tilde{\mathbf{R}}|O)}{Pr(\mathbf{R}|O)}\right)^{1/T_i}\right\}$

  end while

  return R

end function

Simulated Annealing, the new acceptance probability is:

$$p_{\text{accept}} = \min\left\{1, \left(\frac{Pr(\tilde{\mathbf{R}}|O)}{Pr(\mathbf{R}|O)}\right)^{1/T_i}\right\} \tag{3.11}$$

where T_i is a cooling schedule (which is a function of iteration number i) passed into the algorithm. As T_i decreases, the proposals will focus more heavily on areas of large posterior probability (favoring exploitation). Selection of the cooling schedule is left as a parameter, though there are many popular methods in the literature [5]. As will be shown in Section 3.4, the use of a simple decreasing cooling schedule in the modified BIRL algorithm allows the MCMC process to first find areas of high posterior probability then focus the samples towards them.

3.4 Simulation Results

To compare the performance of the original Bayesian IRL algorithm to the modified BIRL method proposed in Section 3.3, the Room World MDP presented in Section 3.2.1 is used with the imperfect expert as described in Section 3.1 providing 100 demonstrations each of length 50 (shown in Figure 3-3). Four variations are compared:

1. `PolicyWalk` **BIRL:** The algorithm exactly as presented in [71] with likelihood given by (3.3), prior given by (3.8), $\alpha = 0.95$, and $\delta = 1/3$.

2. `PolicyWalk` **BIRL with Cooling:** Same as above, but with the a cooling schedule added as described in Section 3.3.2. The cooling parameter was set to $1/T_i = 25 + i/50$ where i is the MCMC iteration number.

3. **Modified BIRL with narrow state relevance kernel:** BIRL with cooling as above, but also using the state relevance function from Section 3.3.1. The relevance kernel is a simple radial basis kernel that uses Euclidean distance as the measure of similarity:

$$k(s, s') = e^{-||s-s'||_2 \ / \ 2\sigma^2} \tag{3.12}$$

   with $\sigma = 0.1$. Figure 3-3 (left) shows the corresponding state relevance given the expert demonstrations (overlaid).

4. **Modified BIRL with wide state relevance kernel:** Same as above but with a "wider" state relevance kernel defined using $\sigma = 1$. This is shown in Figure 3-3 (right), and compared to the narrower kernel above it has high value over a wider set of states around the expert demonstrations.

Figure 3-4 compares the 0-1 policy loss for each of the four algorithms as a function of the MCMC iteration number, averaged over ten episodes. At each iteration, the current MCMC reward vector $\mathbf{R}$ is used to find the optimal policy π^*, and the 0-1 policy loss is simply the number of expert state-action pairs that do not agree with π^* (i.e. the number of times the expert made the wrong decision according to the current reward function estimate). Policy loss is chosen as the measure of algorithm performance over reward loss given the ill-posedness of the IRL problem to recover the exact rewards.

Bayesian IRL with a cooling schedule (green triangles) is shown to converge roughly three times faster than standard Bayesian IRL (blue line). Both losses reach the same final value of roughly 500. Intuitively this is explained by the fact that the

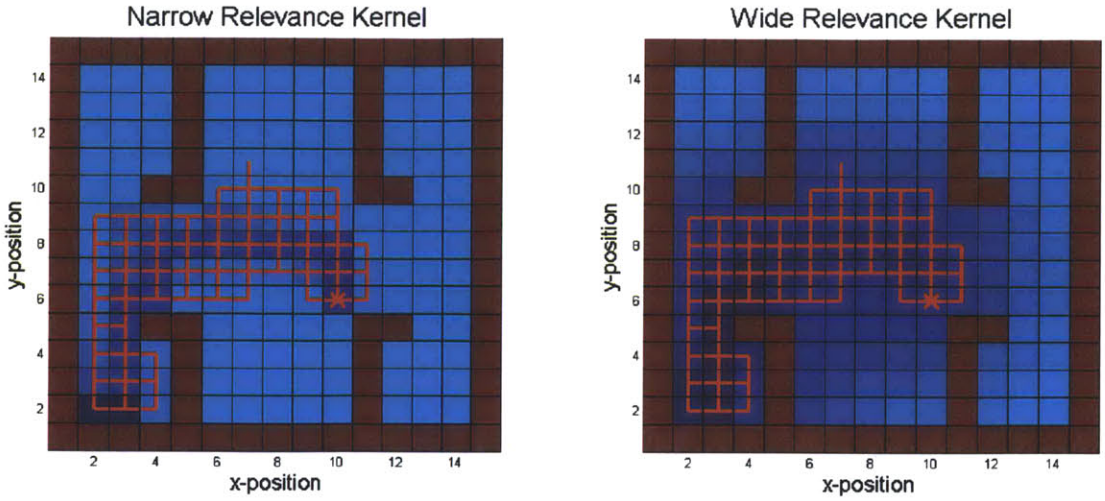


Figure 3-3: State relevance scores for a narrow RBF relevance kernel (left) and a wide RBF relevance kernel (right). Cyan corresponds to zero and dark blue corresponds to one. The set of 100 expert demonstrations are overlaid in red.

cooling schedule allows the algorithm to more quickly focus the samples on peaks in the posterior reward distribution.

The modified BIRL algorithms, which make use of the relevance kernel (cyan crosses and red dashed line), converge much more quickly (roughly ten times faster than standard BIRL). This is a result of the inference being directed towards states where there is more expert information instead of wasting time in irrelevant states. In addition, the modified BIRL algorithms converge to about half the loss of original BIRL, implying that the solutions not only converge faster but are also more accurate.

It is interesting to note the difference in performance between the two modified IRL algorithms (one using the narrow kernel and one using the wide kernel). The algorithm using the narrow kernel converges faster but to a larger steady-state loss; i.e. inferring the rewards over less states yields faster convergence but restricts the algorithm's ability to accurately explain the evidence. Intuitively this gives the algorithm designer the ability to tradeoff accuracy for lowered computation time by varying the width of the relevance kernel.

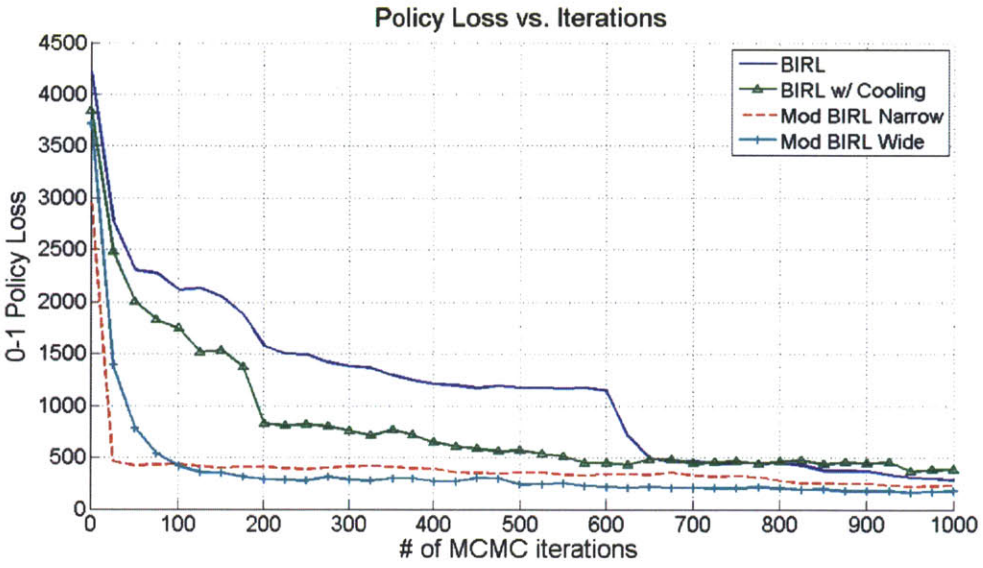


Figure 3-4: The 0-1 policy loss versus number of MCMC iterations for the RoomWorld example comparing original BIRL, BIRL with a cooling schedule, modified Bayesian IRL with a narrow relevance kernel, and modified Bayesian IRL with a wide relevance kernel.

3.5 Summary

This chapter presents two key modifications to the original Bayesian IRL framework based on the observation that the information contained in the expert demonstrations may *not* apply to the entire state space. The modifications are shown to reduce convergence time substantially while maintaining solution quality. The proposed methods allow the user to tradeoff computation time for solution accuracy by defining a kernel function that focuses the inference task on states similar to those encountered by the expert. If the kernel function provided is simply a constant, the original BIRL algorithm from [71] is obtained.

Active IRL [53] is a related algorithm which also attempts to improve the efficiency of BIRL by asking the expert for additional demonstrations in states where the policy is most uncertain. While Active IRL is shown to improve performance, there are two main drawbacks. First, Active IRL relies on the fact that the expert can be asked for more information whereas in many situations this is not possible. Second, Active IRL does nothing to improve the tractability of the initial solution (*before* the expert

is asked for more demonstrations). Thus, like BIRL, Active IRL remains intractable for large state spaces.

Chapter 4

Bayesian Nonparametric Inverse Reinforcement Learning

Of the many IRL algorithms developed [2, 53, 61, 71, 74, 90, 100] (including the two BIRL methods from Chapter 3), all attempt to find one single reward function that explains the entirety of the observed demonstration set. This reward function must be necessarily complex in order to explain the data sufficiently, especially when the task being demonstrated is itself complicated. Searching for a complex reward function is fundamentally difficult for two reasons. First, as the complexity of the reward model increases, so too does the number of free parameters needed to describe the model. Thus the search is over a larger space of candidate functions. Second, the process of testing candidate reward functions requires solving for the MDP value function, the computational cost of which typically scales poorly with the size of the MDP state space, even for approximate solutions [13]. Finding a single complex reward function to explain the observed demonstrations consequently requires searching over a large space of possible solutions and substantial computational effort to test each candidate.

One potential solution to these problems would be to partition the observations into sets of smaller sub-demonstrations. Then, each sub-demonstration could be attributed to a smaller and less-complex class of reward functions. However, such a method would require manual partitioning of the data into an unknown number of

groups, and inferring the reward function corresponding to each group.

The primary contribution of this chapter is to present an IRL algorithm that automates this partitioning process using Bayesian nonparametric methods. Instead of finding a single complex reward function the demonstrations are partitioned and each partition is explained with a simple reward function. A generative model is assumed in which these simple reward functions can be interpreted as *subgoals* of the demonstrator. The generative model utilizes a Chinese Restaurant Process (CRP) prior over partitions so that the number of partitions (and thus subgoals) need not be specified *a priori* and can be potentially infinite.

A key advantage of this method is that the reward functions representing each subgoal can be extremely simple. For instance, one can assume that a subgoal is a single coordinate of the state space (or feature space). The reward function could then consist of a single positive reward at that coordinate, and zero elsewhere. This greatly constrains the space of possible reward functions, yet complex demonstrations can still be explained using a sequence of these simple subgoals. Also, the algorithm has no dependence on the sequential (i.e. temporal) properties of the demonstrations, instead focusing on partitioning the observed data by associated subgoal. Thus the resulting solution does not depend on the initial conditions of each demonstration, and moreover naturally handles cyclic tasks (where the agent begins and ends in the same state).

4.1 Subgoal Reward and Likelihood Functions

This section describes the Bayesian nonparametric subgoal IRL algorithm. The following are two definitions necessary to the algorithm.

Definition 1. *A state subgoal g is simply a single coordinate $g \in S$ of the MDP state space. The associated state subgoal reward function $R_g(s)$ is:*

$$R_g(s) = \begin{cases} c & \text{at state } g \\ 0 & \text{at all other states} \end{cases} \tag{4.1}$$

where c is a positive constant.

While the notion of a state subgoal and its associated reward function may seem trivial, a more general *feature* subgoal will be defined in the following sections to extend the algorithm to a feature representation of the state space.

Definition 2. *An agent in state s_i moving towards some state subgoal g chooses an action a_i with the following probability:*

$$P(a_i|s_i, g) = \pi(a_i|s_i, g) = \frac{e^{\alpha Q^*(s_i, a_i, R_g)}}{\sum_a e^{\alpha Q^*(s_i, a, R_g)}} \tag{4.2}$$

Thus π defines a stochastic policy as in [87], and is essentially our model of rationality for the demonstrating agent (this is the same rationality model as in [71] and [8]). In Bayesian terms, it defines the likelihood of observed action a_i when the agent is in state s_i. The hyperparameter α represents our degree of confidence in the demonstrator's ability to maximize reward.

4.2 Generative Model

The set of observed state-action pairs $\boldsymbol{O}$ defined by (2.7) are assumed to be generated by the following model. The model is based on the likelihood function above, but adds a CRP partitioning component. This addition reflects our basic assumption that the demonstrations can be explained by partitioning the data and finding a simple reward function for each partition.

The model assumes that an agent finds itself in state s_i (because of the Markov property, the agent need not consider how he got to s_i in order to decide which action a_i to take). In analogy to the CRP mixture described in Section 2.3, the agent chooses which partition a_i should be added to, where each existing partition j has its own associated subgoal g_j. The agent can also choose to assign a_i to a new partition whose subgoal will be drawn from the base distribution $P(g)$ of possible subgoals. The assignment variable z_i is set to denote that the agent has chosen partition z_i,

and thus subgoal g_{z_i}. As in equation (2.8), $P(z_i|z_{1:i-1}) = CRP(\eta, z_{1:i-1})$. Now that a partition (and thus subgoal) has been selected for a_i, the agent generates the action according to the stochastic policy $a_i \sim \pi(a_i|s_i, g_{z_i})$ from equation (4.2).

The joint probability over $\boldsymbol{O}$, $\boldsymbol{z}$, and $\boldsymbol{g}$ is given below, since it will be needed to derive the conditional distributions necessary for sampling:

$$
\begin{aligned}
P(\boldsymbol{O}, \boldsymbol{z}, \boldsymbol{g}) &= P(\boldsymbol{O}|\boldsymbol{z}, \boldsymbol{g})\ P(\boldsymbol{z}, \boldsymbol{g}) && (4.3)\\
&= P(\boldsymbol{O}|\boldsymbol{z}, \boldsymbol{g})\ P(\boldsymbol{z})\ P(\boldsymbol{g}) && (4.4)\\
&= \prod_{i=1}^{N} \underbrace{P(o_i|g_{z_i})}_{\text{likelihood}} \underbrace{P(z_i|z_{-i})}_{\text{CRP}} \prod_{j=1}^{J_N} \underbrace{P(g_j)}_{\text{prior}} && (4.5)
\end{aligned}
$$

where (4.4) follows since subgoal parameters g_j for each new partition are drawn independently from prior $P(g)$ as described above. As shown in (4.5), there are three key elements to the joint probability. The likelihood term is the probability of taking each action a_i from state s_i given the associated subgoal g_{z_i}, and is defined in (4.2). The CRP term is the probability of each partition assignment z_i given by (2.8). The prior term is the probability of each partition's subgoal (J_N is used to indicate the number of partitions after observing N datapoints). The subgoals are drawn i.i.d. from discrete base distribution $P(g)$ each time a new partition is started, and thus have non-zero probability given by $P(g_j)$.

The model assumes that o_i is conditionally independent of o_j for $i \neq j$ given g_{z_i}. Also, it can be verified that the CRP partition probabilities $P(z_i|z_{-i})$ are exchangeable. Thus, the model implies that the data $\boldsymbol{O}$ are exchangeable [37]. Note that this is weaker than implying that the data are independent and identically distributed (i.i.d.). The generative model instead assumes that there is an underlying grouping structure that can be exploited in order to decouple the data and make posterior inference feasible.

The CRP partitioning allows for an unknown and potentially infinite number of subgoals. By construction, the CRP has "built-in" complexity control, i.e. its concentration hyperparameter η can be used to make a smaller number of partitions

more likely.

4.3 Inference

The generative model (4.5) has two sets of hidden parameters, namely the partition assignments z_i for each observation o_i, and the subgoals g_j for each partition j. Thus the job of the IRL algorithm will be to infer the posterior over these hidden variables, $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$. While both $\boldsymbol{z}$ and $\boldsymbol{g}$ are discrete, the support of $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$ is combinatorially large (since $\boldsymbol{z}$ ranges over the set of all possible partitions of N integers), so exact inference of the posterior is not feasible. Instead, approximate inference techniques must be used. As summarized in Section 2.4, Gibbs sampling [38] is in the family of Markov chain Monte Carlo (MCMC) sampling algorithms and is commonly used for approximate inference of Bayesian nonparametric mixture models [30, 60, 86]. Since the posteriors of both the assignments and subgoals are of interest, uncollapsed Gibbs sampling is used where both the $\boldsymbol{z}$ and $\boldsymbol{g}$ are sampled in each sweep.

Each Gibbs iteration involves sampling from the conditional distributions of each hidden variable given all of the other variables (i.e. sample one unknown at a time with all of the others fixed). Thus the conditionals for each partition assignment z_i and subgoal g_j must be derived.

The conditional probability for partition assignment z_i can be derived as follows:

$$
\begin{aligned}
P(z_i|z_{-i}, \boldsymbol{g}, \boldsymbol{O}) &\propto P(z_i, o_i \mid z_{-i}, \boldsymbol{O}_{-i}) && (4.6)\\
&= P(z_i|z_{-i}, \boldsymbol{g}, \boldsymbol{O}_{-i}) P(o_i|z_i, z_{-i}, \boldsymbol{g}, \boldsymbol{O}_{-i}) && (4.7)\\
&= P(z_i|z_{-i}) \;\; P(o_i|z_i, z_{-i}, \boldsymbol{g}, \boldsymbol{O}_{-i}) && (4.8)\\
&= \underbrace{P(z_i|z_{-i})}_{\text{CRP}} \;\; \underbrace{P(o_i|g_{z_i})}_{\text{likelihood}} && (4.9)
\end{aligned}
$$

where (4.6) is the definition of conditional probability, (4.7) applies the chain rule, (4.8) follows from the fact that assignment z_i depends only on the other assignments z_{-i}, and (4.9) follows from the fact that each o_i depends only on its assigned subgoal g_{z_i}.

When sampling from (4.9), the exchangeability of the data is utilized to treat z_i as if it was the last point to be added. Probabilities (4.9) are calculated with z_i being assigned to each existing partition, and for the case when z_i starts a new partition with subgoal drawn from the prior $P(g)$. While the number of partitions is potentially infinite, there will always be a finite number of groups when the length of the data N is finite, so this sampling step is always feasible.

The conditional probabilities for each partition's subgoal g_j is derived as follows:

$$\begin{aligned} P(g_j|\boldsymbol{z},\boldsymbol{O}) &\propto P(\boldsymbol{O}_{I_j}|g_j,\boldsymbol{z},\boldsymbol{O}_{-I_j})P(g_j|\boldsymbol{z},\boldsymbol{O}_{-I_j}) && (4.10)\\ &= \sum_{i\in I_j} P(o_i|g_{z_i})\ P(g_j|\boldsymbol{z},\boldsymbol{O}_{-I_j}) && (4.11)\\ &= \sum_{i\in I_j} \underbrace{P(o_i|g_{z_i})}_{\text{likelihood}}\ \underbrace{P(g_j)}_{\text{prior}} && (4.12) \end{aligned}$$

where (4.10) applies Bayes' rule, (4.11) follows from the fact that each o_i depends only on its assigned subgoal g_{z_i}, and (4.12) follows from the fact that the subgoal g_j of each partition is drawn i.i.d. from the prior over subgoals. The index set $I_j = \{i : z_i = j\}$.

Sampling from (4.12) depends on the form of the prior over subgoals $P(g)$. When the subgoals are assumed to take the form of *state subgoals* (Definition 1), then $P(g)$ is a discrete distribution whose support is the set S of all states of the MDP. In this chapter, the following simplifying assumption is proposed to increase the efficiency of the sampling process.

Proposition 1. *The prior $P(g)$ is assumed to have support only on the set S_O of MDP states, where $S_O = \{s \in S : s = s_i$ for some observation $o_i = (s_i, a_i)\}$.*

This proposition assumes that the set of all possible subgoals is limited to only those states encountered by the demonstrator. Intuitively it implies that during the demonstration, the demonstrator achieves each of his subgoals. This is not the same as assuming a perfect demonstrator (the expert is not assumed to get to each subgoal optimally, just eventually). Sampling of (4.12) now scales with the number of unique states in the observation set $\boldsymbol{O}$. While this proposition may seem limiting, the

simulation results in Section 4.7 indicate that it does not affect performance compared to other IRL algorithms and greatly reduces the required amount of computation.

Algorithm 4 Bayesian nonparametric IRL

1: **function** BNIRL(MDP/R, Observations $\boldsymbol{O}$, Confidence α, Concentration η)
2: **for** each unique $s_i \in \boldsymbol{O}$ **do**
3: Solve for and store $V^*(R_g)$, where $g = s_i$ and R_g is defined by (4.1)
4: Sample initial subgoal $g_1^{(0)}$ from prior $P(g)$ and set all assignments $z_i^{(0)} = 1$
5: **end for**
6: **while** iteration $t < t_{\max}$ **do**
7: **for** each current subgoal $g_j^{(t-1)}$ **do**
8: Sample subgoal $g_j^{(t)}$ from (4.12)
9: **end for**
10: **for** each observation $o_i \in \boldsymbol{O}$ **do**
11: **for** each current subgoal $j^{(t)}$ **do**
12: $p(z_i = j|\boldsymbol{z}, \boldsymbol{O}, R_j) \leftarrow$ Probability of subgoal j from (4.9)
13: **end for**
14: $p(z_i = k|\boldsymbol{z}, \boldsymbol{O}, R_k) \leftarrow$ Probability of new subgoal R_k drawn from $P(g)$
15: $z_i^{(t)} \leftarrow$ Sample assignment from normalized probabilities in lines 12–14
16: **end for**
17: **end while**
18: **return** samples $z^{(1:t_{\max})}$ and $g^{(1:t_{\max})}$, discarding samples for burn-in and lag
19: **end function**

Algorithm 4 defines the Bayesian nonparametric inverse reinforcement learning method. The algorithm outputs samples which form a Markov chain whose stationary distribution is the posterior, so that sampled assignments $\boldsymbol{z}^{(T)}$ and subgoals $\boldsymbol{g}^{(T)}$ converge to a sample from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$ as $T \to \infty$ [5, 38]. Note that instead of solving for the MDP value function in each iteration (as is typical with IRL algorithms, see Algorithm 1), Algorithm 4 pre-computes all of the necessary value functions. The number of required value functions is upper bounded by the number of elements in the support of the prior $P(g)$. When Proposition 1 is assumed, then the support of $P(g)$ is limited to the set of unique states in the observations $\boldsymbol{O}$. Thus

the required number of MDP solutions scales with the size of the observed data set $\boldsymbol{O}$, *not* with the number of required iterations. This is a perceived advantage in a learning scenario when the size of the MDP is potentially large but the amount of demonstrated data is small.

4.4 Convergence in Expected 0-1 Loss

To demonstrate convergence, it is common in IRL to define a loss function which in some way measures the difference between the demonstrator and the predictive output of the algorithm [61, 71, 74]. In Bayesian nonparametric IRL, the assignments $\boldsymbol{z}$ and subgoals $\boldsymbol{g}$ represent the hidden variables of the demonstration that must be learned. Since these variables are discrete, a 0-1 loss function is suitable:

$$\mathcal{L}\left[(\boldsymbol{z}, \boldsymbol{g}), (\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}})\right] = \begin{cases} 0 & \text{if } (\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}}) = (\boldsymbol{z}, \boldsymbol{g}) \\ 1 & \text{otherwise} \end{cases} \tag{4.13}$$

The loss function evaluates to 0 if the estimated parameters $(\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}})$ are exactly equal to the true parameters $(\boldsymbol{z}, \boldsymbol{g})$, and 1 otherwise. It must be shown that, for the Bayesian nonparametric IRL algorithm (Algorithm 4), the expected value of the loss function (4.13) given a set of observations $\boldsymbol{O}$ is minimized as the number of iterations T increases. Theorem 1 establishes this.

Theorem 1. *Assuming observations $\boldsymbol{O}$ are generated according to the generative model defined by (4.5), the expected 0–1 loss defined by (4.13) is minimized by the empirical mode of the samples $(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})$ output by Algorithm 4 as the number of iterations $T \to \infty$.*

Proof. It can be verified that the maximum *a posteriori* (MAP) parameter values, defined here by

$$(\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}}) = \underset{(\boldsymbol{z}, \boldsymbol{g})}{\operatorname{argmax}}\, P(\boldsymbol{z}, \boldsymbol{g} | \boldsymbol{O})$$

minimize the expected 0–1 loss defined in (4.13) given the observations $\boldsymbol{O}$ (see [12]). By construction, Algorithm 4 defines a Gibbs sampler whose samples $(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})$

converge to samples from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$ so long as the Markov chain producing the samples is ergodic [38]. A sufficient condition for ergodicity of the Markov chain in Gibbs sampling requires only that the conditional probabilities used to generate samples are non-zero [59]. For Algorithm 4, these conditionals are defined by (4.9) and (4.12). Since clearly the likelihood (4.2) and CRP prior (2.8) are always non-zero, then the conditional (4.9) is always non-zero. Furthermore, the prior over subgoals $P(g)$ is non-zero for all possible g by assumption, so that (4.12) is non-zero as well.

Thus the Markov chain is ergodic and the samples $(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})$ converge to samples from the true posterior $P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$ as $T \to \infty$. By the strong law of large numbers, the empirical mode of the samples, defined by

$$(\widetilde{\boldsymbol{z}}, \widetilde{\boldsymbol{g}}) = \underset{(\boldsymbol{z}^{(1:T)}, \boldsymbol{g}^{(1:T)})}{\operatorname{argmax}} P(\boldsymbol{z}, \boldsymbol{g}|\boldsymbol{O})$$

converges to the true mode $(\widehat{\boldsymbol{z}}, \widehat{\boldsymbol{g}})$ as $T \to \infty$, and this is exactly the MAP estimate of the parameters which was shown to minimize the 0–1 loss. □

It is noted that, given the nature of the CRP prior, the posterior will be multimodal (switching partition indices does not affect the partition probability even though the numerical assignments $\boldsymbol{z}$ will be different). As such, the argmax above is used to define the *set* of parameter values which maximize the posterior. In practice, the sampler need only converge on one of these modes to find a satisfactory solution.

The rate at which the loss function decreases relies on the rate the empirical sample mode(s) converges to the true mode(s) of the posterior. This is a property of the approximate inference algorithm and, as such, is beyond the scope of this chapter (convergence properties of the Gibbs sampler have been studied, for instance in [75]). As will be seen empirically in Section 4.7, the number of iterations required for convergence is typically similar to (or less than) that required for other IRL methods.

4.5 Action Prediction

IRL algorithms find reward models with the eventual goal of learning to predict what action the agent will take from a given state. As in Algorithm 1, the typical output of the IRL algorithm is a single reward function that can be used to define a policy which predicts what action the demonstrator would take from a given state.

In Bayesian nonparametric IRL (Algorithm 4), in order to predict action a_k from state s_k, a subgoal must first be chosen from the mode of the samples $\widehat{\boldsymbol{g}} = \text{mode}(\boldsymbol{g}^{(1:T)})$. This is done by finding the most likely partition assignment z_k after marginalizing over actions using Equation (4.6):

$$z_k = \underset{z_i}{\text{argmax}} \sum_a P\left(z_i \mid \widehat{z}_{-i}, \widehat{\boldsymbol{g}}, O_k = (s_k, a)\right) \tag{4.14}$$

where $\widehat{\boldsymbol{z}}$ is the mode of samples $\boldsymbol{z}^{(1:T)}$. Then, an action is selected using the policy defined by (4.2) with $\widehat{g}_{z_k}$ as the subgoal.

Alternatively, the subgoals can simply be used as waypoints which are followed in the same order as observed in the demonstrations. In addition to predicting actions, the subgoals in $\widehat{\boldsymbol{g}}$ can be used to analyze which states in the demonstrations are important, and which are just transitory.

4.6 Extension to General Linear Reward Functions

Linear combinations of state features are commonly used in reinforcement learning to approximately represent the value function in a lower-dimensional space [13, 87]. Formally, a k-dimensional feature vector is a mapping $\Phi : S \mapsto \mathbb{R}^k$. Likewise, a discrete k-dimensional feature vector is a mapping $\Phi : S \mapsto \mathbb{Z}^k$, where $\mathbb{Z}$ is the set of integers.

Many of the IRL algorithms listed in Section 2.2 assume that the reward function can be represented as a linear combination of features. Algorithm 4 is extended to accommodate discrete feature vectors by defining a *feature subgoal* in analogy to the state subgoal from Definition 1.

Definition 3. *Given a k-dimensional discrete feature vector Φ, a feature subgoal $g(\boldsymbol{f})$ is the set of states in S which map to the coordinate $\boldsymbol{f}$ in the feature space. Formally, $g(\boldsymbol{f}) = \{s \in S : \Phi(s) = \boldsymbol{f}\}$ where $\boldsymbol{f} \in \mathbb{Z}^k$. The associated feature subgoal reward function $R_{g(\boldsymbol{f})}(s)$ is defined as follows:*

$$R_{g(\boldsymbol{f})}(s) = \begin{cases} c, & s \in g(\boldsymbol{f}) \\ 0, & s \notin g(\boldsymbol{f}) \end{cases} \tag{4.15}$$

where c is a positive constant.

From this definition it can be seen that a state subgoal is simply a specific instance of a feature subgoal, where the features are binary indicators for each state in S. Algorithm 4 runs exactly as before, with the only difference being that the support of the prior over reward functions $P(g)$ is now defined as the set of unique feature coordinates induced by mapping S through ϕ. Proposition 1 is also still valid should the set of possible subgoals be limited to only those feature coordinates in the observed demonstrations, $\Phi(s_{1:N})$. Finally, feature subgoals do not modify any of the assumptions of Theorem 1, thus convergence is still attained in 0-1 loss.

4.7 Simulation Results

Simulation results are given for three test cases. All three use a 20×20 Grid World MDP (total of 400 states) with walls. Note that while this is a relatively simple MDP, it is similar in size and nature to experiments done in the seminal papers of each of the compared algorithms. Also, the intent of the experiments is to compare basic properties of the algorithms in nominal situations (as opposed to finding the limits of each).

The agent can move in all eight directions or choose to stay. Transitions are noisy, with probability 0.7 of moving in the chosen direction and probability 0.3 of moving in an adjacent direction. The discount factor $\gamma = 0.99$, and value iteration is used to find the optimal value function for all of the IRL algorithms tested. The demonstrator in each case makes optimal decisions based on the true reward function. While this is

not required for Bayesian nonparametric IRL, it is an assumption of one of the other algorithms tested [2]. In all cases, the 0-1 policy loss function is used to measure performance. The 0-1 policy loss simply counts the number of times that the learned policy (i.e. the optimal actions given the learned reward function) does not match the demonstrator over the set of observed state-action pairs.

4.7.1 Grid World Example

The first example uses the state-subgoal Bayesian nonparametric IRL algorithm. The prior over subgoal locations is chosen to be uniform over states visited by the demonstrator (as in Proposition 1). The demonstrator chooses optimal actions towards each of three subgoals $(x, y) = \{(10, 12), (2, 17), (2, 2)\}$, where the next subgoal is chosen only after arrival at the current one. Figure 4-1 shows the state-action pairs of the demonstrator (top left), the 0-1 policy loss averaged over 25 runs (top right), and the posterior mode of subgoals and partition assignments (colored arrows denote assignments to the corresponding colored boxed subgoals) after 100 iterations (bottom). The algorithm reaches a minimum in loss after roughly 40 iterations, and the mode of the posterior subgoal locations converges to the correct coordinates. It is noted that while the subgoal locations have correctly converged after 100 iterations, the partition assignments for each state-action pair have not yet converged for actions whose subgoal is somewhat ambiguous.

4.7.2 Grid World with Features Comparison

In the next test case, Bayesian nonparametric IRL (for both state- and feature-subgoals) is compared to three other IRL algorithms, using the same Grid World setup as in Section 4.7.1: "Abbeel" IRL using the quadratic program variant [2], Maximum Margin Planning using a loss function that is non-zero at states not visited by the demonstrator [74], and Bayesian IRL [71]. A set of six features $\phi_{1:6}(s)$ are used, where feature k has an associated state s_{ϕ_k}. The value of feature k at state s is

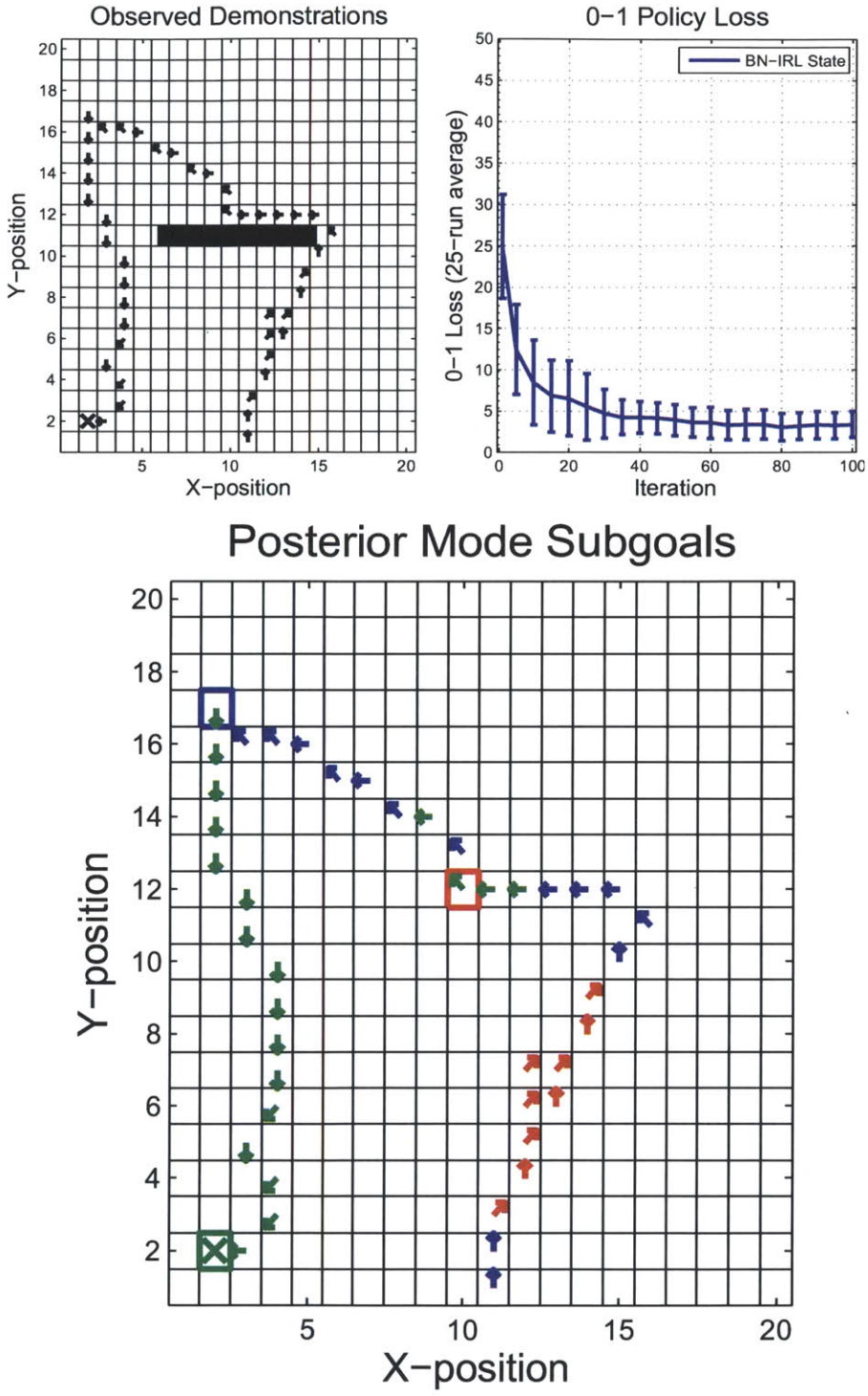


Figure 4-1: Observed state-action pairs for simple grid world example (top left), where arrows indicate direction of the chosen action and X's indicate choosing the "stay" action. 0-1 policy loss for Bayesian nonparametric IRL (top right). Posterior mode of subgoals and partition assignments (bottom). Colored arrows denote assignments to the corresponding colored boxed subgoals.

simply the Manhattan distance (1-norm) from s to s_{ϕ_k}:

$$\phi_k(s) = ||s - s_{\phi_k}||_1 \tag{4.16}$$

The true reward function is defined as $R(s) = \boldsymbol{w}^T \boldsymbol{\phi}(s)$ where $\boldsymbol{w}$ is a vector of randomly-chosen weights. The observations consist of five demonstrations starting at state $(x, y) = (15, 1)$, each having 15 actions which follow the optimal policy corresponding to the true reward function. Note that this dataset satisfies the assumptions of the three compared algorithms, though it does not strictly follow the generative process of Bayesian nonparametric IRL. Figure 4-2 shows the state-action pairs of the demonstrator (top) and the 0-1 policy loss, averaged over 25 runs versus iteration for each algorithm (bottom). All but Bayesian IRL achieve convergence to the same minimum in policy loss by 20 iterations, and Bayesian IRL converges at roughly 100 iterations (not shown). Even though the assumptions of Bayesian nonparametric IRL were not strictly satisfied (the assumed model (4.5) did not generate the data), both the state- and feature-subgoal variants of the algorithm achieved performance comparable to the other IRL methods.

4.7.3 Grid World with Loop Comparison

In the final experiment, five demonstrations are generated using subgoals as in Section 4.7.1. The demonstrator starts in state $(x, y) = (10, 1)$, and proceeds to subgoals $(x, y) = \{(19, 9), (10, 17), (2, 9), (10, 1)\}$. Distance features (as in Section 4.7.2) are placed at each of the four subgoal locations. Figure 4-3 (left) shows the observed state-action pairs. This dataset clearly violates the assumptions of all three of the compared algorithms, since more than one reward function is used to generate the state-action pairs. However, the assumptions are violated in a reasonable way. The data resemble a common robotics scenario in which an agent leaves an initial state, performsome tasks, and then returns to the same initial state.

Figure 4-3 (center) shows that the three compared algorithms, as expected, do not converge in policy loss. Both Bayesian nonparametric algorithms, however, perform

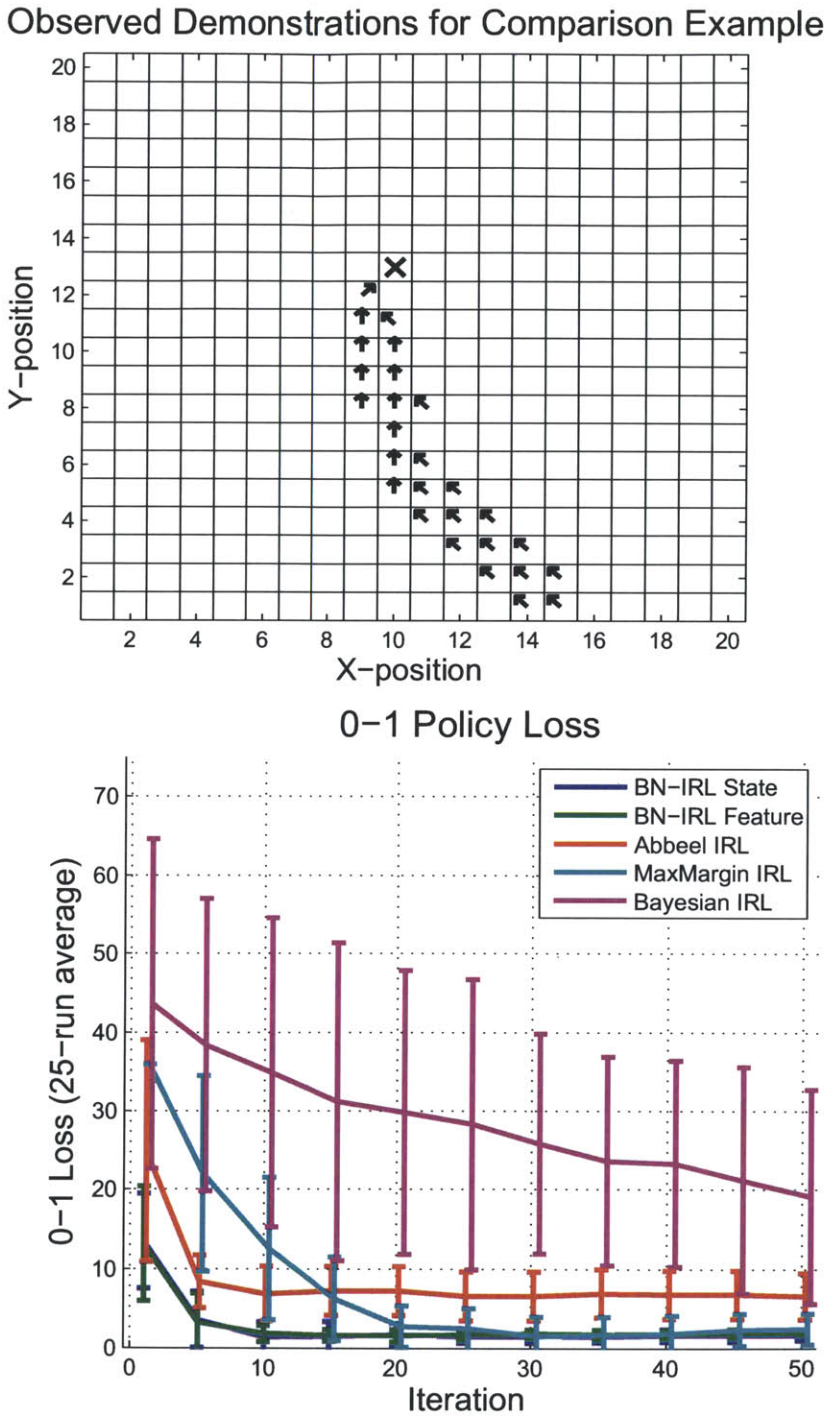


Figure 4-2: Observed state-action pairs for grid world comparison example (top). Comparison of 0-1 Policy loss for various IRL algorithms (bottom).

almost exactly as before and the mode posterior subgoal locations converge to the four true subgoals (Figure 4-3 right). Again, the three compared algorithms would have worked properly if the data had been generated by a single reward function, but such a reward function would have to be significantly more complex (i.e. by including temporal elements). Bayesian nonparametric IRL is able to explain the demonstrations without modification or added complexity.

4.7.4 Comparison of Computational Complexities

BNIRL has two stages of computation:

- In the initialization stage, optimal action-value functions are computed for each candidate reward state, i.e. for each unique demonstration state by Proposition 1. Since many methods exist for finding optimal action-value functions [13, 87], the computational complexity of the operation will be referred to as $O(\text{MDP})$.

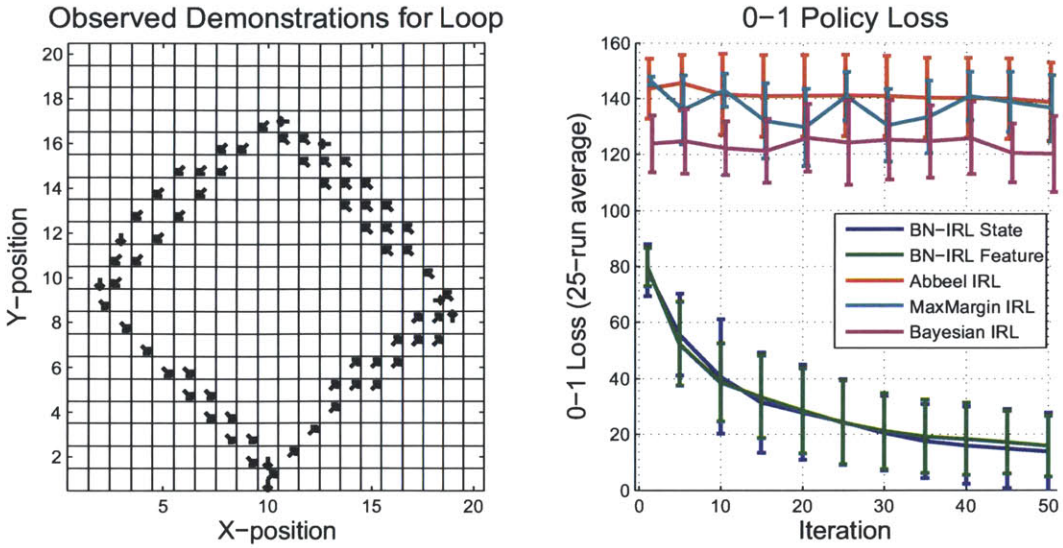
- In the sampling stage, each iteration requires assigning observations to a subgoal reward. The complexity of each sampling iteration is

$$O(N_{\text{obs}}\, \eta \log N_{\text{obs}}),$$

  where N_{obs} is the number of observed state-action pairs in the demonstration, η is the CRP concentration parameter, and $\eta \log N_{\text{obs}}$ is the expected number of active subgoals in the CRP.

The overall complexities of each stage are not directly comparable since results on the number of iterations required for Gibbs sampling convergence are not well established [75]. However, in practice the first stage (calculating optimal action-value functions) dominates the complexity of the overall algorithm.

As outlined below, other IRL algorithms calculate optimal action value functions once per iteration. Since BNIRL calculates the optimal action value function at a maximum of once per demonstration state, a rough complexity comparison can be



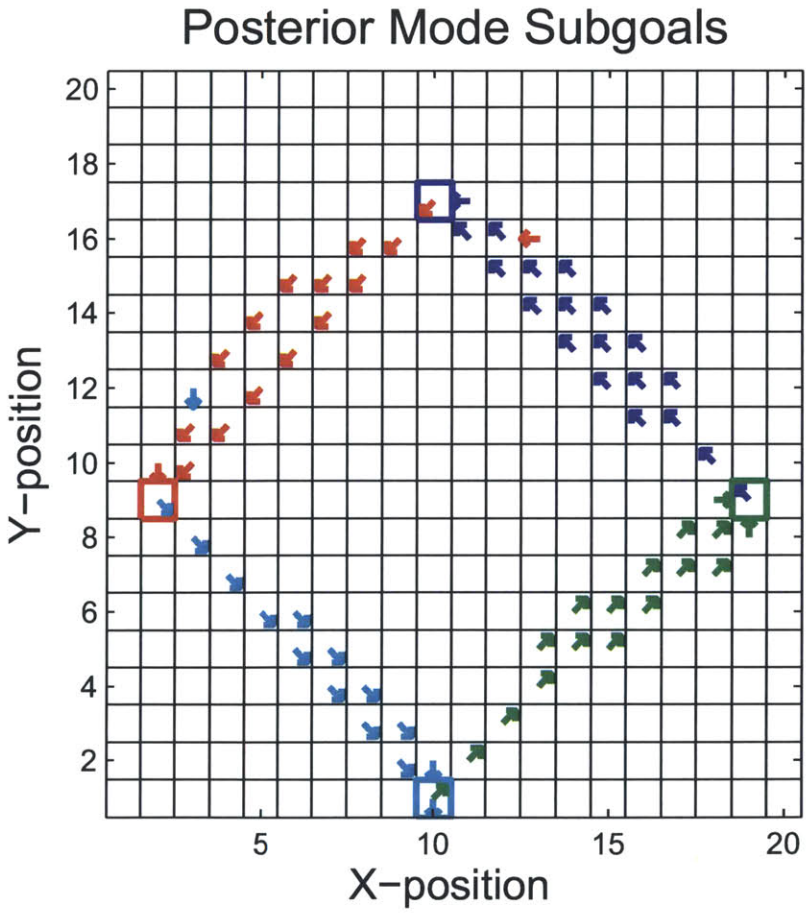


Figure 4-3: Observed state-action pairs for grid world loop example (top left). Comparison of 0-1 Policy loss for various IRL algorithms (top right). Posterior mode of subgoals and partition assignments (bottom).

made by comparing the number of times the MDP must be solved for each algorithm. The following summarizes a complexity analysis given in each respective original work:

1. **Abbeel IRL [1]:** An upper bound on the number of iterations required to guarantee feature count matching error $\frac{\epsilon}{4}$ is given as $\frac{48k}{(1-\gamma)^2\epsilon^2}$, where k is the dimension of the state space and γ is the MDP discount factor. Each iteration requires computation of an optimal action-value function.

2. **Bayesian IRL [71]:** The number of iterations required is related to the mixing time of the MCMC method used. The chain is said to "rapidly mix" in $O(n^2d^2\alpha^2e^{2\beta}\log\frac{1}{\epsilon})$ sampling iterations, where n is the dimension of the state space, d is a bound on the reward magnitude, and α, β and ϵ are parameters. Each sampling iteration requires computation of an optimal action-value function.

3. **MaxMargin IRL [74]:** While no analytical expression is given, convergence is said to be sub-linear for a diminishing step-size rule which achieves a minimum in error under a strong convexity assumption. As in the above two algorithms, each iteration requires computation of an optimal value function.

Thus, the effective complexity of the other algorithms (the number of optimal value functions that must be computed) scales with the number of iterations needed for convergence. As shown above, the number of required iterations can depend on many parameters of each algorithm. In BNIRL, the effective complexity is upper-bounded by the number of unique states in the demonstration. This highlights a fundamental computational difference of BNIRL versus previous methods.

To give an empirical sense of computation times for the example in Section 4.7.2, Table 4.1 compares average initialization and per-iteration run-times for each of the algorithms. These are given only to show general trends, as the Matlab implementations of the algorithms were not optimized for efficiency. The initialization of Bayesian nonparametric IRL takes much longer than the others, since during this period the algorithm is pre-computing optimal value functions for each of the possible subgoal

locations (i.e. each of the states encountered by the demonstrator). However, the Bayesian nonparametric IRL per-iteration time is roughly an order of magnitude less than the other algorithms, since the others must re-compute an optimal value function each iteration.

Table 4.1: Run-time comparison for various IRL algorithms.

	Initialization (sec)	Per-iteration (sec)	Iter. to Converge	Total (sec)
BNIRL	15.3	0.21	10	17.4
Abbeel-IRL	0.42	1.65	10	16.9
MaxMargin-IRL	0.41	1.16	20	23.6
Bayesian-IRL	0.56	3.27	105	344

The example which generated the data in Table 4.1 was selected for BNIRL to perform comparably in overall runtime to Abbeel IRL such that a fair comparison of initialization versus per-iteration runtimes can be made. This selection highlights the fundamental performance tradeoff between BNIRL and the other IRL methods compared. By Proposition 1, BNIRL limits the candidate subgoals to the states observed in the demonstration. This proposition limits the potential complexity of the reward representation, but it also places an upper-bound on the number of value functions that must be calculated. In the compared IRL methods, the reward function is parametrized and the algorithms iteratively search over a continuous parameter space, computing a new value function at each iteration. In this case, no assumption is made about the number of candidate reward functions (other than the reward parameterization itself) at the cost of an asymptotic number of value functions to be computed.

As a result of this fundamental difference in algorithmic structure, there are scenarios when BNIRL will perform computationally faster than the other methods, and vice versa. In cases where the demonstration set is small and there are a large number of demonstrator subgoals, BNIRL will generally execute faster since its computation scales with the number of unique demonstration states and it has the ability to learn multiple subgoal reward functions. The other IRL methods will generally execute slower in this case, since they must search for a more complex representation. In

cases where there is a large amount of demonstration data and there are not multiple subgoals, BNIRL will generally execute slower since it must find a value function for each unique demonstration state. The other IRL will generally execute faster in this case, since their computation does not scale with demonstration size and a less-complex reward representation is required.

4.8 Summary

This chapter proposed a new inverse reinforcement learning method which is able to learn *multiple reward functions* from a single demonstration. The method uses a Bayesian nonparametric mixture model to automatically partition the data and find a set of simple reward functions corresponding to each partition. The simple rewards are interpreted intuitively as subgoals, which can be used to predict actions or analyze which states are important to the demonstrator.

The example in Section 4.7.2 shows that, for a simple problem, Bayesian nonparametric IRL performs comparably to existing algorithms in cases where the data are generated using a single reward function. Approximate computational run-times indicate that overall required computation is similar to existing algorithms. As noted in Section 4.3, however, Bayesian nonparametric IRL solves for the MDP value function once for each unique state in the demonstrations. The other algorithms solve for the MDP value function once per iteration.

The loop example in Section 4.7.3 highlights several fundamental differences between Bayesian nonparametric IRL and existing algorithms. The example breaks the fundamental assumption of existing IRL methods, i.e. that the demonstrator is optimizing a *single* reward function. These algorithms could be made to properly handle the loop case, but not without added complexity or manual partitioning of the demonstrations. Bayesian nonparametric IRL, on the other hand, is able to explain the loop example without any modifications. The ability of the new algorithm to automatically partition the data and explain each group with a simple subgoal reward eliminates the need to find a single, complex temporal reward function. Furthermore,

the Chinese restaurant process prior naturally limits the number of partitions in the resulting solution, rendering a parsimonious explanation of the data.

Chapter 5

Approximations to the Demonstrator Likelihood

Chapter 4 presented a Bayesian nonparametric inverse reinforcement learning (BNIRL) algorithm to address the scalability of IRL methods to larger problems. The BNIRL algorithm automatically partitions the observed demonstrations and finds a simple reward function to explain each partition using a Bayesian nonparametric mixture model. Using simple reward functions (which can be interpreted as *subgoals*) for each partition eliminates the need to search over a large candidate space. Also, the number of partitions is assumed to be unconstrained and unknown a priori, allowing the algorithm to explain complex behavior.

Results from Section 4.7 show that BNIRL performs similarly to a variety of conventional IRL methods for small problems, and furthermore can handle cyclic tasks which break the assumptions of the other methods. However, BNIRL (like other IRL methods) still relies on computing the optimal MDP value function in order to test reward function candidates. Calculating the optimal value function becomes infeasible for large state spaces [87], thus limiting the applicability of BNIRL to small problems.

The key contribution of this chapter is to offer several effective methods to avoid computing the optimal MDP value function, enabling BNIRL to scale to much larger problem domains. In the first method, we modify BNIRL to use a framework known

as Real-time Dynamic Programming (RTDP) [9]. RTDP effectively limits computation of the value function to necessary areas of the state space only. This allows the complexity of the BNIRL reward learning method to scale with the size of the demonstration set, *not* the size of the full state space as in Chapter 4. Experimental results are given for a Grid World domain and show order of magnitude speedups over exact solvers for large grid sizes.

In the second method, we modify BNIRL to utilize an existing closed-loop controller in place of the optimal value function. This avoids having to specify a discretization of the state or action spaces, extending the applicability of BNIRL to continuous domains. Simulation results are given for a pedestrian data set, demonstrating the ability to learn meaningful subgoals using a very simple closed-loop control law. In Chapter 7, the approximation is applied experimentally for a quadrotor flight example which, if discretized, would require over 10^{10} states. In the experiment, quadrotor flight maneuvers are learned from a human demonstrator using only hand motions. The demonstration is recorded using a motion capture system and then analyzed by the BNIRL algorithm. Learned subgoal rewards (in the form of waypoints) are passed as commands to an autonomous quadrotor which executes the learned behavior in actual flight. The entire process from demonstration to reward learning to robotic execution takes on the order of 10 seconds to complete using a single computer. Thus, the results highlight the ability of BNIRL to use data from a safe (and not necessarily dynamically feasible) demonstration environment and quickly learn subgoal rewards that can be used in the actual robotic system.

5.1 Action Likelihood Approximation

The action likelihood (3.3) requires evaluation of the optimal action-value function $Q^*(s_i, a_i, R_{z_i})$ which, in turn, involves computing the optimal value function $V^*(s, R_{z_i})$. This computation must be performed for each candidate reward function $\widehat{R}$ (most other IRL methods [2, 61, 71, 74, 90, 100] have the same requirement). Standard methods for finding the optimal value function, including value iteration and

linear programming, scale polynomially with the number of states [13]. As a result, an approximation for the action likelihood (3.3) is required in order to scale reward learning to large problems. The following section describes two such approximations, and the experimental results in Section 3.4 show that both scale well to large domains.

5.1.1 Real-time Dynamic Programming

One method of approximating the action likelihood (3.3) is to approximate the optimal action-value function Q^* itself. Approximating Q^* given a fully-specified MDP is a popular area of research known as approximate dynamic programming [13].

Real-time dynamic programming (RTDP) [9] is one such method particularly well-suited to IRL. The basic premise of RTDP is to start with a set of sample states $\tilde{S}$, which is a small subset of the full state-space, i.e. $\tilde{S} \subset S$. Value iteration [87] is then performed on the sample states to generate a value function which is defined only over $\tilde{S}$. A greedy simulation is performed using $V^*(\tilde{S})$ starting from some state $s \in \tilde{S}$. Each state encountered in the simulation (as well as any neighboring states) is then added to $\tilde{S}$, and the process repeats.

As an example of using RTDP in BNIRL, consider the Grid World domain shown in Figure 5-1. The agent can move in all eight directions or choose to stay. Transitions are noisy with probability 0.7 of moving in the chosen direction, and the discount factor $\gamma = 0.99$. The demonstration set $\boldsymbol{O}$ is denoted by arrows, indicating actions chosen from each state.

The initial set of RTDP sample states $\tilde{S}$ is chosen to be the set of states encountered in the demonstration $\boldsymbol{O}$, as well as any other states reachable in one transition. Value iteration is performed on these states for an example candidate reward function, and the resulting value function is shown in Figure 5-1a. A random state $s \in \boldsymbol{O}_i$ is then chosen, and a greedy simulation is performed. All states encountered during the simulation (as well as any other states reachable in one transition) are added to $\tilde{S}$. The cycle repeats, and Figures 5-1b and 5-1c shows the progression of sample states and corresponding value functions. The process terminates when the greedy simulation fails to add any new states to $\tilde{S}$.

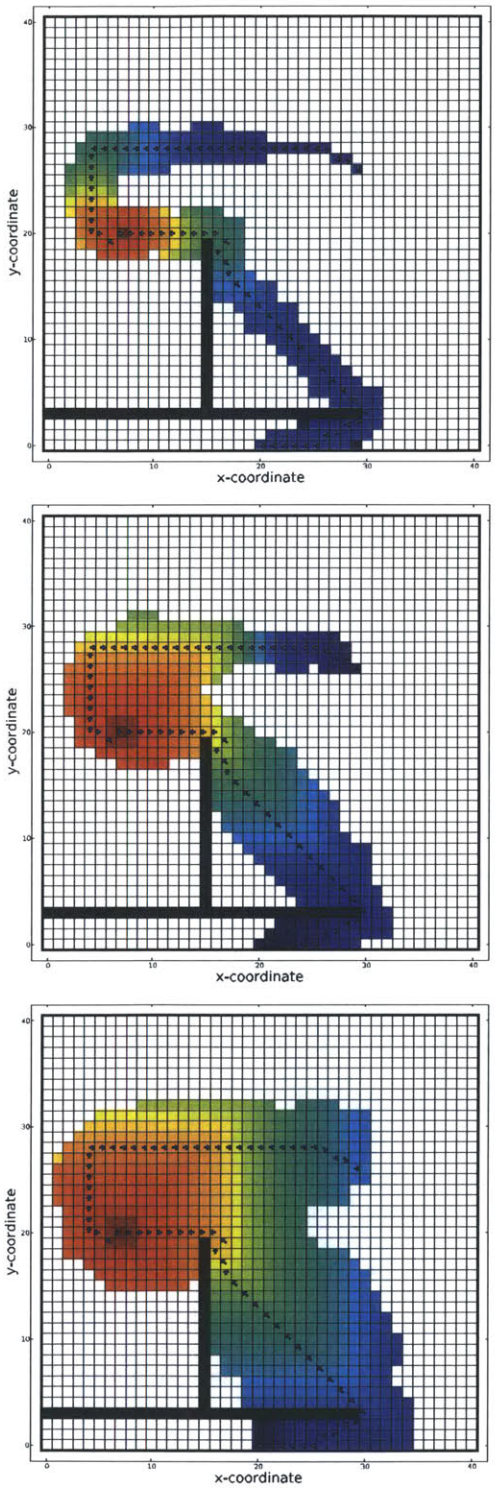


Figure 5-1: Progression of real-time dynamic programming [9] sample states for the Grid World example. The algorithm starts with the initial set (top) based on the demonstration set (denoted by arrows), and uses greedy simulations to progressively expand the set of sample states (middle and bottom) over which value iteration is performed.

As the figures illustrate, RTDP only adds states necessary to improve the quality of the value function around $\boldsymbol{O}_i$, thus avoiding unnecessary computation in remote states. The net result is a reward learning algorithm that scales with the size of the demonstration set $\boldsymbol{O}$, and is agnostic to how large the surrounding state space may be. Experimental results in Section 3.4 show that BNIRL combined with RTDP results in order of magnitude computation time decreases as compared to methods which use exact value function solvers.

5.1.2 Action Comparison

Many learning scenarios involve demonstrations in a continuous domain. Before Q^* can be calculated with conventional techniques, the domain must be discretized. Even for relatively simple domains, however, the discretization process can result in an extremely large state space.

Take, for instance, the 2-dimensional quadrotor model shown in Figure 5-2. The state-space is six-dimensional (two positions, one angle, and their time-derivatives). Even using modest discretization intervals (1cm, 1cm/s, $\pi/16$ rad, $\pi/16$ rad/sec) would require over 10^{10} states to cover a 1-meter by 1-meter grid. This is unwieldy even for approximate dynamic programming/model-based RL methods. Thus, trying to approximate Q^* for such domains quickly becomes infeasible.

An alternative to approximating Q^* is to instead approximate the entire action likelihood (3.3) itself. In words, (3.3) represents the likelihood that the demonstrator took action a_i from state s_i in order to maximize the subgoal reward R_{z_i}. As defined in (4.1), BNIRL subgoal rewards comprise a single positive reward for reaching some coordinate in the state (or feature) space. Thus, approximating (3.3) would simply require a measure of how likely action a_i would be if the demonstrator wanted to go from s_i to subgoal g_{z_i}.

One method for approximating this likelihood would be to compare action a_i with the action a_{CL} given by some closed-loop controller tasked with getting from s_i to g_{z_i}.

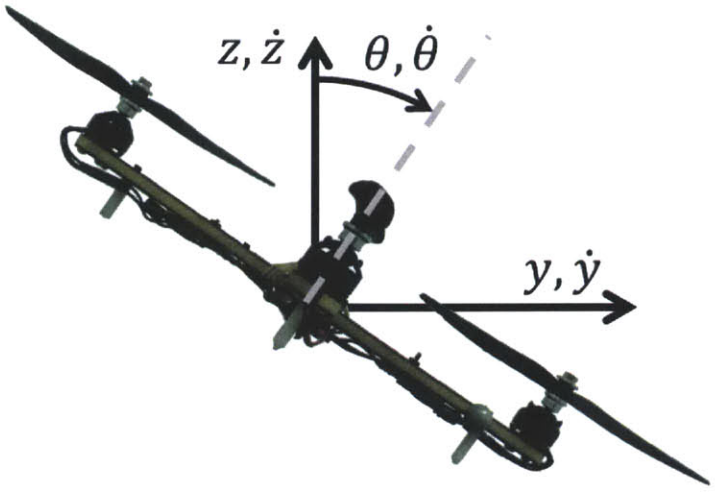


Figure 5-2: Two-dimensional quadrotor model, showing y, z, and θ pose states along with $\dot{y}$, $\dot{z}$, and $\dot{\theta}$ velocity states.

An approximate action likelihood in place of (3.3) would thus be:

$$P(O_i|R_{z_i}) = P(a_i|s_i, z_i) \propto e^{-\alpha||a_i - a_{\text{CL}}||_2} \tag{5.1}$$

where a_{CL} is the action given by some closed-loop controller attempting to go from s_i to subgoal g_{z_i}. It is noted that the scaling of the norm $||a_i - a_{\text{CL}}||_2$ is inconsequential, since probabilities are normalized in Step 15 of the BNIRL sampling procedure (Algorithm 4).

The form of the closed-loop controller is problem-dependent, but in many cases a simple controller can be easily synthesized (or already exists). Take, for example, the 2-dimensional quadrotor in Figure 5-2. Let the states of the demonstration be a set of observed poses $s_i = (x_i, z_i, \theta_i)$ and the "actions" of the demonstration be the corresponding set of observed velocities $a_i = (\dot{x}_i, \dot{z}_i, \dot{\theta}_i)$. A simple controller simply generates an action a_{CL} that commands a velocity in the direction of the pose error between s_i and g_{z_i}, i.e.:

$$a_{\text{CL}} \propto g_{z_i} - s_i \tag{5.2}$$

While this may seem like an overly-simple control law, it is used to generate pedestrian

subgoal learning simulation results in Section 5.2.2 as well as experimental results in Section 7.2 which demonstrate the successful learning of autonomous quadrotor flight maneuvers from hand-held recorded demonstrations.

We see action comparison as a powerful method for approximating the likelihood (3.3) for several reasons. First, the method requires no discretization of the state or action spaces, as would be the case for methods which attempt to approximate Q^*. This makes the method well-suited for continuous domains. Second, calculation of the control action a_{CL} is typically extremely fast compared to calculating (or approximating) an entire action-value function Q^*. This allows for real-time reward learning in many situations, as is shown in Section 7.2. Finally, the form of the closed-loop controller can be refined based on the degree of knowledge of the expert, enabling a trade-off between computational complexity and accuracy of the action likelihood approximation.

5.2 Simulation Results

The following section presents experimental results which apply the two action likelihood approximations described in Section 5.1 to relatively large problem domains.

5.2.1 Grid World using RTDP

Consider the Grid World example presented in Section 5.1.1 (shown in Figure 5-1). In order to test the scalability of the RTDP Q^* approximation, the CPU run-times of five different methods are compared: BNIRL using full value iteration, Abbeel IRL (from [2], a representative conventional IRL method) using full value iteration, BNIRL using RTDP, Abbeel IRL using RTDP, and BNIRL using parallelized RTDP. In the parallel BNIRL case, the pre-computation of the required approximate value functions is done on a cluster of 25 computing cores. The ability to compute value functions in parallel is an algorithmic feature of the BNIRL algorithm (since the number of reward function candidates is finite, see [57]). To the author's knowledge, Abbeel IRL (as well as other conventional IRL methods [2, 61, 71, 74, 90, 100]) does

not have a corresponding parallel implementation. Computation is performed on a Pentium i7 3.4GHz processor with 8GB RAM. Implementations of each algorithm have not been optimized, and results are only meant to demonstrate trends.

Figure 5-3a shows average CPU run-times of each method (lower is better) for Grid World domains ranging from 100 to 1,000,000 states. For each domain size, demonstrations are generated with a greedy controller starting and ending at randomly-chosen states. As can be seen, both BNIRL and Abbeel IRL using full value iteration become extremely slow for problems larger than 10^3 states (data points for 10^6 states are not included, as they would take weeks to computing time). Methods using RTDP are slower for small problem sizes (this is due to the extra time needed for simulations to expand the set of sample states). However, beyond problems with 10^3 states, the RTDP methods are roughly an order of magnitude faster than full value iteration. Finally, the parallelized BNIRL method using RTDP shows significantly faster performance than the non-parallelized version and Abbeel IRL with RTPD. This is due to the fact that 25 computing cores can be used in parallel to calculate the necessary value functions for the BNIRL sampling procedure.

To ensure that the RTDP Q^* approximation does not affect the quality of the learned reward function, Figure 5-3b shows the average 0-1 policy loss of each algorithm (lower is better) for each grid size. The 0-1 policy loss simply counts the number of times that the learned policy (i.e. the optimal actions given the learned reward function) does not match the demonstrator over the set of observed state-action pairs. As can be seen, using RTDP to approximate Q^* does not have an adverse effect on reward learning performance, as the loss for the RTDP methods is only slightly higher than the full value iteration methods.

5.2.2 Pedestrian Subgoal Learning using Action Comparison

To demonstrate the ability of the action likelihood approximation to learn meaningful subgoals using comparison to a simple closed-loop control law, pedestrian data from the Edinburgh Informatics Forum Pedestrian Database [54] is analyzed. Figure 5-4 (top) shows a subset of pedestrian trajectories through a busy forum. In this example,

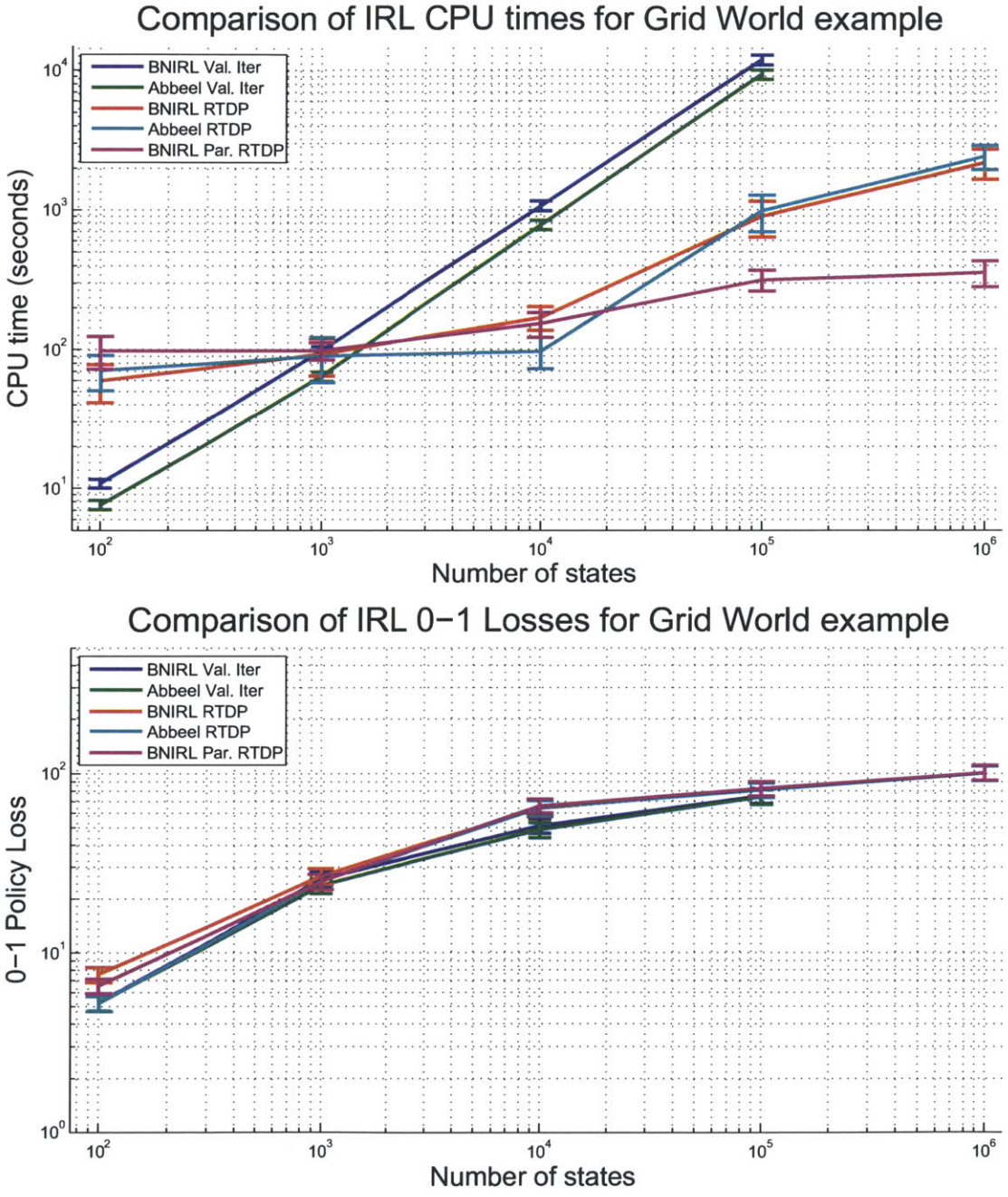


Figure 5-3: Comparison of average CPU runtimes for various IRL algorithms for the Grid World example (top), along with the average corresponding 0-1 policy loss (bottom). Both plots use a log-log scale, and averages were take over 30 samples.

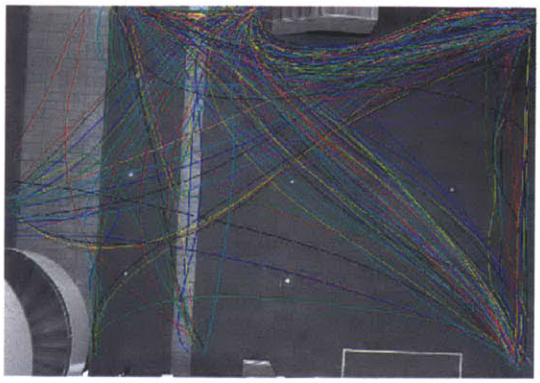
the states $s_i = (x_i, y_i)$ are the measured and sampled positions of the pedestrians, and the actions $a_i = (\dot{x}_i, \dot{y}_i)$ are the corresponding post-processed velocities. The closed-loop control law is taken to be the simple velocity controller in (5.2). BNIRL is applied to the subset of 1,000 trajectory state-action pairs shown in Figure 5-4 (middle), where the trajectories start in the upper-right and typically go to one of four main locations in the atrium. The approximate likelihood function (5.1) is used, and the confidence parameter $\alpha = 10$.

Figure 5-4 (middle) shows the learned subgoals in red. Despite the use of the very simple control law in (5.2), the algorithm finds subgoals at each of four key locations in the forum. An extra subgoal is found in the lower right, which is attributed to noisy post-processed velocity data. Figure 5-4 (bottom) shows the number of learned subgoals versus sampling iteration, indicating convergence in posterior mode after roughly 400 iterations. The results demonstrate the ability of BNIRL with action comparison to learn subgoals using noisy real-world data without the need to specify an overly-complicated likelihood model or discretize the demonstration domain.

5.3 Summary

This chapter presented several effective methods to avoid computing the optimal MDP value function, enabling BNIRL to scale to much larger problem domains. In the first method, BNIRL is modified to use a framework known as Real-time Dynamic Programming (RTDP) [9]. RTDP effectively limits computation of the value function to necessary areas of the state space only. This allows the complexity of the BNIRL reward learning method to scale with the size of the demonstration set, *not* the size of the full state space as in Chapter 4. In the second method, an existing closed-loop controller is used in place of the optimal value function. This avoids having to specify a discretization of the state or action spaces, extending the algorithm's applicability to continuous demonstration domains for which a closed-loop controller is available.

The simulation results presented demonstrate several fundamental improvements over conventional IRL reward learning methods. BNIRL limits the size of the candi-



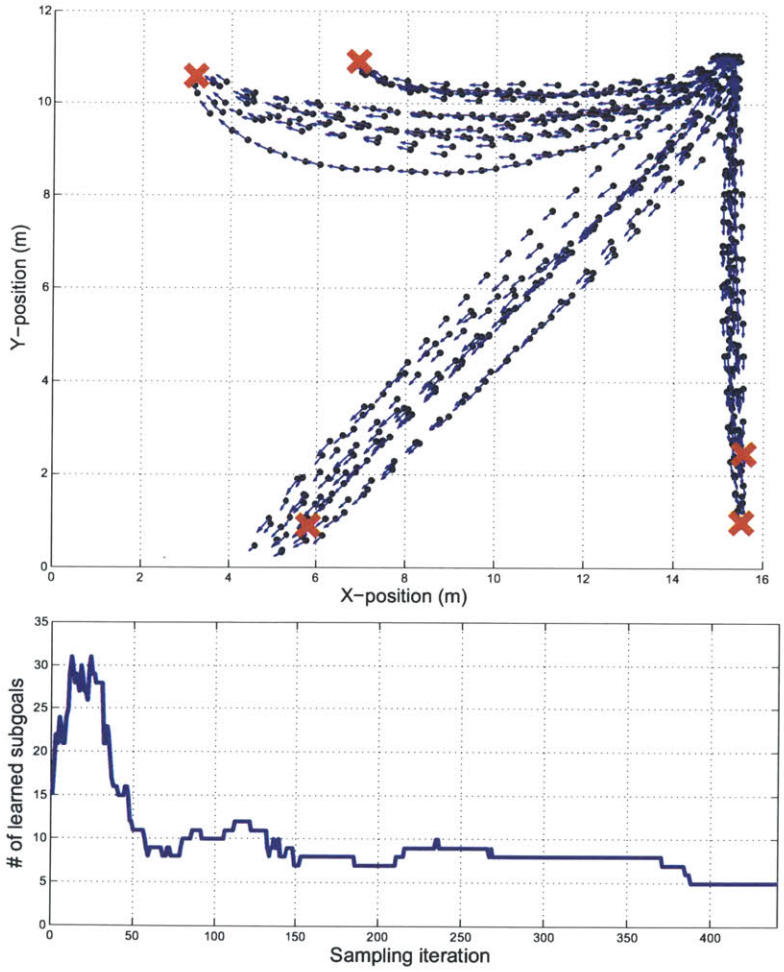


Figure 5-4: Example pedestrian trajectories from the Edinburgh Informatics Forum Pedestrian Database [54] (top). Subset of pedestrian data with states (position) in black and actions (velocities) in blue, along with learned subgoals in red (middle). Number of learned subgoals versus sampling iteration for a representative trial (bottom).

date reward space to a finite set, allowing for parallelized pre-computation of (approximate) action value functions. This is shown to lead to order-of-magnitude computational speedups over previous methods. Also, the BNIRL likelihood function can be approximated using action comparison to an existing closed-loop controller, avoiding the need to discretize the state space and allowing for learning in continuous demonstration domains. Results from a noisy pedestrian dataset show promising subgoal learning using a very simple closed-loop control law.

Chapter 6

Gaussian Process Subgoal Reward Learning

In this chapter, the Bayesian nonparametric inverse reinforcement learning (BNIRL) method is generalized to handle fully continuous demonstration domains by using Gaussian process reward representation and Gaussian process dynamic programming [27] as a method of finding approximate action-value functions. Further, the option MDP (skills) framework [88] enables execution of the learned behaviors by the robotic system and provides a principled basis for future learning and skill refinement. In Chapter 7, experimental results are given for a robotic car domain, identifying maneuvering skills such as drifting turns, executing the learned skills autonomously, and providing a method for quantifying the relative skill level of the original demonstrator.

6.1 Gaussian Process Subgoal Reward Learning Algorithm

While the BNIRL algorithm (Algorithm 4) learns multiple reward functions, it is only applicable in discrete domains. The method is extended to a continuous domain in Section 5.1.2, but this extension relies on access to an existing closed-loop controller which may not be available in general. This section presents the Gaussian process

subgoal reward learning (GPSRL) algorithm, which learns Gaussian process subgoal reward representations from unsegmented, continuous demonstration. The GPSRL algorithm assumes two key inputs:

1. The demonstration set of state-action pairs, $\boldsymbol{O} = \{(s_1, a_1), (s_2, a_2), \ldots, (s_N, a_N)\}$. The continuous demonstration must be measured and downsampled to the desired time interval. Note that this is not the same as discretization of the continuous state space; it is instead sampling a continuous trajectory at finite time intervals. As an example, the motion capture data used for the experiments in Section 7.3 is sampled at 100Hz.

2. A state transition function, $s' = f(s, a)$, which models the dynamics that generated the demonstration. Note that this is not a model of the demonstrator, just a model of the state transition *given* an action. In general, reward learning necessitates a system model. If no model is available, many system identification techniques exist which can learn a transition model.

The full GPSRL method is shown in Algorithm 5, comprising three main stages that are explained in the subsections below. First, the set of candidate subgoal rewards is constructed (lines 2-7). Next, Gaussian process dynamic programming is used to approximate the optimal action-value function Q^* for each candidate subgoal in parallel (lines 8-10). Finally, approximate posterior inference is done using Gibbs sampling (lines 11-19).

6.2 Subgoal Reward Representation

Since the demonstration space is assumed to be continuous, a subgoal reward at a single coordinate of the state space (as in BNIRL [57]) is ill-defined. A subgoal reward representation with broader support is achieved using Gaussian processes (GPs). Each subgoal reward is simply a GP with one training point, $GP(s_g, r, k_g)$, where s_g is the subgoal state, r is a positive scalar reward (the magnitude of which

Algorithm 5 Gaussian Process Subgoal Reward Learning

1: $\boldsymbol{R}_{sg}, \boldsymbol{S}_{\text{supp}} \leftarrow \{\}$

2: **for** each demonstration state $s_i \in \boldsymbol{O}$ **do**
3: **if** $||s_i - s|| > \epsilon \quad \forall s \in \boldsymbol{S}_{\text{supp}}$ **then**
4: $\boldsymbol{S}_{\text{supp}} \leftarrow \{\boldsymbol{S}_{\text{supp}}, \ s_i\}$ ▷ Build set of support states for GPDP
5: $\boldsymbol{R}_{sg} \leftarrow \{\boldsymbol{R}_{sg}, \ GP(s_i, r, k_g)\}$ ▷ Build set of candidate subgoal rewards
6: **end if**
7: **end for**

8: **for** each candidate subgoal $R_j \in \boldsymbol{R}_{sg}$ (in parallel) **do**
9: $\widehat{Q}^*(R_j) \leftarrow \text{GPDP}(S_{\text{supp}}, R_j, f(s'|s,a))$ ▷ Gaussian process DP [27]
10: **end for**

11: **while** iteration $t < N_{\text{iter}}$ **do** ▷ Gibbs sampling of subgoal posterior assignments
12: **for** each observation $o_i \in \boldsymbol{O}$ **do**
13: **for** each current partition $j^{(t)}$ **do**
14: $p(z_i = j|\boldsymbol{z}, \boldsymbol{O}, R_j) \leftarrow$ Probability of partition j from (6.3)
15: **end for**
16: $p(z_i = k|\boldsymbol{z}, \boldsymbol{O}, R_k) \leftarrow$ Probability of new partition with R_k drawn from $\boldsymbol{R}_{sg}$
17: $z_i^{(t)} \leftarrow$ Sample assignment from normalized probabilities in lines 13–16
18: **end for**

19: **end while**

20: **return** mode of samples $\boldsymbol{z}^{(1:N_{\text{iter}})}$ and associated subgoal rewards R_j

is not critical to the algorithm), and $k_g(\cdot,\cdot)$ is a kernel function. The GP spreads the reward to the neighborhood around s_g, according to the kernel function k_g.

As in BNIRL, a key assumption of GPSRL is that the set of possible subgoals comes from the demonstration itself, avoiding *a priori* discretization of the state space to generate a candidate subgoal reward set. The set of possible rewards $\boldsymbol{R}_{sg}$ is thus the set of subgoal rewards corresponding to the sampled demonstration:

$$\boldsymbol{R}_{sg} = \{ \ GP(s, r, k_g), \quad \forall s \in \boldsymbol{O} \ \} \tag{6.1}$$

To avoid redundant subgoals, the set is built incrementally such that a new subgoal is not added if it is ϵ-close to a subgoal already in $\boldsymbol{R}_{sg}$ (lines 2-7 of Algorithm 5). The parameter ϵ is thus chosen to scale the size of the candidate set of subgoal rewards, and correspondingly the computational requirements of the GPSRL algorithm.

6.3 Action Likelihood

Inferring reward requires a model for the likelihood of a given observation O_i, i.e. the relative probability that the demonstrator takes action a_i from state s_i, given some subgoal reward R_j. A softmax likelihood based on the optimal Q^* function is used similar to (4.2), but requires a maximum in the denominator due to the continuous action space:

$$p(O_i|R_j) = p(a_i|s_i, R_j) \propto \frac{\exp\left(\ \alpha Q^*(s_i, a_i|R_j)\ \right)}{\max\limits_a \exp\left(\ \alpha Q^*(s_i, a|R_j)\ \right)} \tag{6.2}$$

Since the demonstration is not assumed to be optimal, α is a parameter that represents the expected degree to which the demonstrator is able to maximize reward. In the limit as $\alpha \to \infty$, the demonstrator is assumed perfectly optimal. In the limit as $\alpha \to 0$, the demonstrator is assumed to choose arbitrary actions that do not attempt to maximize reward. Thus α is a key parameter of the algorithm, and further considerations for its selection are given in Section 7.3.1.

6.4 Gaussian Process Dynamic Programming

In order to compute the likelihood value in (6.2), the optimal Q^* function is necessary. In general, calculating the optimal value function for discrete systems is computationally difficult, and even more so for continuous systems. This necessitates the use of an approximate method, and Gaussian process dynamic programming (GPDP) [27] is chosen for this purpose. GPDP generalizes dynamic programming to continuous domains by representing the value and action-value functions with Gaussian processes. Thus, the algorithm requires only a set of *support states* to train the value function GPs, instead of requiring discretization or feature mapping of the state space.

GPDP is particularly well-suited for the task of approximating the Q^* function in (6.2) for several reasons. Foremost, the support points used to learn the value function in GPDP come directly from the demonstration, i.e. $\boldsymbol{S}_{\text{support}} = \{s : s \in \boldsymbol{O}\}$. This effectively focuses computational effort only on areas of the state space that

are relevant to the reward learning task. Also, the Gaussian process subgoal reward representation from Section 6.2 is naturally compatible with the GPDP framework. Finally, the output of the GPDP algorithm enables evaluation of the approximated optimal action-value function $\widehat{Q}^*$ at any arbitrary state s via evaluation of a GP mean, allowing for efficient calculation of the likelihood (6.2). Note that the max in the denominator of (6.2) must be found numerically by evaluating $\widehat{Q}^*$ at several test actions, which in practice are distributed uniformly between action bounds.

The computational complexity of Algorithm 5 is dominated by the GPDP calculation of $\widehat{Q}^*(R_j)$ for each candidate subgoal reward $R_j \in \boldsymbol{R}_{sg}$ (lines 8-10). However, this is easily parallelized on a computing cluster allowing for substantial savings in computation time.

6.5 Bayesian Nonparametric Mixture Model and Subgoal Posterior Inference

The GPSRL algorithm learns multiple subgoals reward functions from the demonstration set. To avoid the need to prespecify or constrain the number of learned subgoals, a Bayesian nonparametric model is used similar to BNIRL. In the model, each state-action pair in the demonstration $\boldsymbol{O}$ is assigned to a partition. The vector $\boldsymbol{z} \in \mathbb{R}^{|\boldsymbol{O}|}$ stores partition assignments, so that $z_i = j$ implies that observation $O_i = (s_i, a_i)$ is assigned to partition j. Each partition has an associated subgoal from the set of candidate GP subgoal reward functions $\boldsymbol{R}_{sg}$. The posterior probability of assignment z_i to partition j is defined as follows (see [57] for a more detailed derivation):

$$p(z_i = j|\boldsymbol{z}, \boldsymbol{O}, R_j) \propto \underbrace{p(z_i = j|\boldsymbol{z}_{-i})}_{\text{CRP prior (2.8)}} \quad \underbrace{p(O_i|R_j)}_{\text{action likelihood (6.2)}} \tag{6.3}$$

where R_j is the GP subgoal reward corresponding to partition j. The CRP prior, which encourages clustering into large partitions, is defined by (2.8). The action likelihood term, which encourages similarity within partitions, is defined by (6.2).

As summarized in Section 2.4, Gibbs sampling is a common choice for approximate inference of Bayesian nonparametric mixture models [60]. Gibbs sampling exploits the exchangeability of the model by sampling one conditional z_i assignment at a time as if it is the last datapoint to arrive, holding the rest constant. To sample z_i, the relative posterior probability (6.3) is calculated for each non-empty partition j, in addition to a new partition randomly sampled from $\boldsymbol{R}_{sg}$. The probabilities are then normalized, and a new assignment is sampled. After the desired number of sampling iterations, the mode of the assignment vector $\boldsymbol{z}$ and the corresponding subgoal rewards R_j are returned. The Gibbs sampling procedure comprises lines 11-19 of Algorithm 5.

6.6 Converting Learned Subgoals to MDP Options

Once subgoal rewards are learned using GPSRL they can be easily cast in the options MDP framework [88]. As summarized in Section 2.1, an option is defined by the initiation set I_o, the option policy π_o, and the terminating condition β_o. For a learned subgoal reward R_j centered at subgoal state s_g, the initiation set is defined as those states which are ϵ-close (where ϵ is the parameter from Section 6.2) to a demonstration state which is assigned to subgoal R_j by the GPSRL sampling step. Since the approximate optimal action value function $\widehat{Q}^*(s, a|R_j)$ for subgoal reward R_j is already calculated in the GPDP step of GPSRL, the option policy π_o is simply the corresponding optimal policy. Finally, the set of terminating states is simply the set of states which are ϵ-close to s_g:

$$I_o(R_j) \triangleq \{s \in S : ||s - s_i|| < \epsilon, \text{where } s_i \in \boldsymbol{O} \text{ and } z_i = j\} \tag{6.4}$$

$$\pi_o(s|R_j) \triangleq \underset{a}{\text{argmax}} \ \widehat{Q}^*(s, a|R_j) \tag{6.5}$$

$$\beta_o(R_j) \triangleq \{s \in S : ||s - s_g|| < \epsilon\} \tag{6.6}$$

The conversion of learned GPSRL subgoals into MDP options is validated experimentally in Chapter 7, in which RC car driving maneuvers are learned from demonstration and executed by the autonomous system.

6.7 Summary

This chapter extended the Bayesian nonparametric inverse reinforcement learning (BNIRL) method developed in Chapter 4 to handle general, continuous demonstration domains. Experimental results applying GPSRL are given in Chapter 7. Unlike previous IRL methods ([2, 61, 62, 74, 100]), GPSRL learns *multiple reward functions* and explicitly incorporates GPDP [27] as a method for operating in large, continuous domains while retaining computational tractability.

The Bayesian nonparametric framework of GPSRL, specifically the use of Dirichlet process mixtures to partition observations, is similar to that of Grollman et al. [41] and Fox et al. [35]. However, neither of those other methods learns *reward* representations of behavior and instead classify the demonstration by policy or dynamics.

Finally, GPSRL is similar to constructing skill trees (CST) [49] in that multiple options (skills) are learned by segmenting a single demonstration. While CST also learns a hierarchical structure of the overall task, it requires access to a reward signal. Thus, GPSRL is complementary to CST in that it assumes unknown rewards (which are learned) and instead requires a transition model.

Chapter 7

Experimental Results

The broad focus of this thesis is to enable scalable reward learning from demonstration for real-world robotic systems. This section presents experimental results that validate the use of BNIRL and GPSRL on experimental robotic hardware systems.

7.1 Experimental Test Facility

Experiments are performed in the MIT RAVEN facility, shown in Figure 7-1. The Realtime indoor Autonomous Vehicle test ENvironment (RAVEN) enables rapid prototyping and testing of a variety of unmanned vehicle technologies, such as adaptive flight control, automated UAV recharging, autonomous UAV air combat, and coordinated multivehicle search and track missions, in a controlled, indoor flight test volume [42]. RAVEN utilizes an camera-based motion capture system to simultaneously track multiple air and ground-based vehicles, and provide accurate position, attitude, and velocity information at rates up to 200 Hz. Measured vehicle information is distributed to a group of ground computers on which learning and control algorithms are executed. Control commands are then sent to the autonomous vehicles via radio transmitter.

Figure 7-1: RAVEN indoor test facility with quadrotor flight vehicles, ground vehicles, autonomous battery swap and recharge station, and motion capture system.

7.2 Learning Quadrotor Flight Maneuvers from Hand-Held Demonstration with Action Comparison

To test the action comparison likelihood approximation described in Section 5.1.2, BNIRL is used to learn quadrotor flight maneuvers from a hand-held demonstration. First, the maneuver is demonstrated by motioning with a disabled quadrotor helicopter (Figure 7-2a) while the pose and velocities of the quadrotor are tracked and recorded by the motion capture system down-sampled to 20Hz (Figure 7-2b). In this case, states are positions and actions are velocities. Using the 2-D quadrotor model described in Section 5.1.2 and the closed-loop controller action comparison likelihood defined by (5.1) and (5.2), the BNIRL algorithm is used to generate an approximate posterior distribution over the demonstrator's subgoals. Figure 7-2c shows the mode of the sampled posterior, which converges to four subgoals, one at at each corner of the demonstrated trajectory. The subgoals are then sent as waypoints to an autonomous quadrotor which executes them in actual flight, thus recreating the demonstrated

trajectory. Flight tests are conducted in the RAVEN indoor testbed [42] using the flight control law described in [25]. Figure 7-2d plots the hand-held trajectory against the autonomous flight, showing a close matchup between the demonstration and the resulting learned behavior.

To demonstrate the ability of BNIRL to handle cyclic, repetitive demonstrations, Figure 7-3 shows a cluttered trajectory where the demonstrator moves randomly between the four corners of a square. Overlayed are the four subgoals of the converged posterior, which correctly identify the four key subgoals inherent in the demonstration.

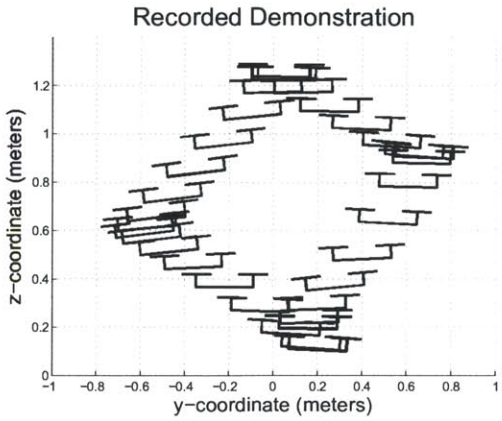
Figure 7-4a shows another example, this time where the demonstrated trajectory is a flip. As shown in Figure 7-4b, the BNIRL algorithm using action comparison likelihood converges to posterior subgoals at the bottom and the top of the trajectory, with the quadrotor being inverted at the top. The subgoal waypoints are executed by the autonomous flight controller and the actual flight path is overlaid on Figure 7-4a, again showing the matchup between demonstrated and learned behavior.

Finally, it is noted that the BNIRL sampling process for the three examples above takes roughly three seconds to converge to the posterior mode. This is due to the fact that evaluation of the closed-loop control action in (5.1) is fast, making BNIRL suitable for online reward learning.

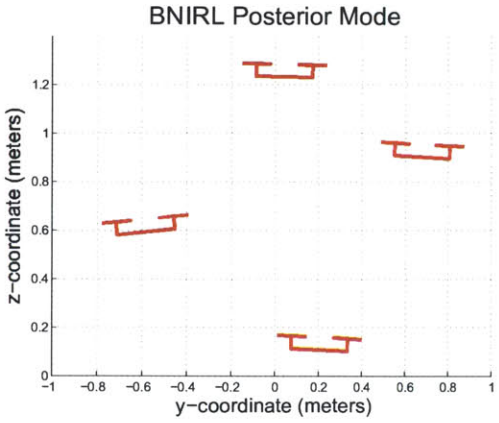
7.3 Learning Driving Maneuvers from Demonstration with GPSRL

This section presents experimental results demonstrating the ability of GPSRL to learn driving maneuvers for an RC car. Demonstrating and learning such maneuvers is typically challenging due to highly non-linear tire slip dynamics which are difficult to model or predict. The demonstration state vector consists of the body velocities $\dot{x}_b$ and $\dot{y}_b$, heading rate $\dot{\psi}$, and wheel speed ω. Learned subgoals can thus be specified as a GP trained in this 4-dimensional state space. Actions consist of the steer angle

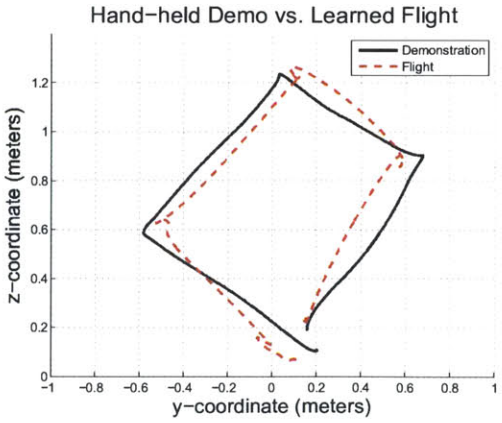
(a)



(b)



(c)



(d)

Figure 7-2: A human demonstrator motions with a disabled quadrotor (a), while an indoor motion capture system records and downsamples demonstration (b). The BNIRL algorithm with action comparison likelihood converges to a mode posterior with four subgoals, one at each corner of the demonstrated trajectory (c). Finally, an autonomous quadrotor takes the subgoals as waypoints and executes the learned trajectory in actual flight (d).

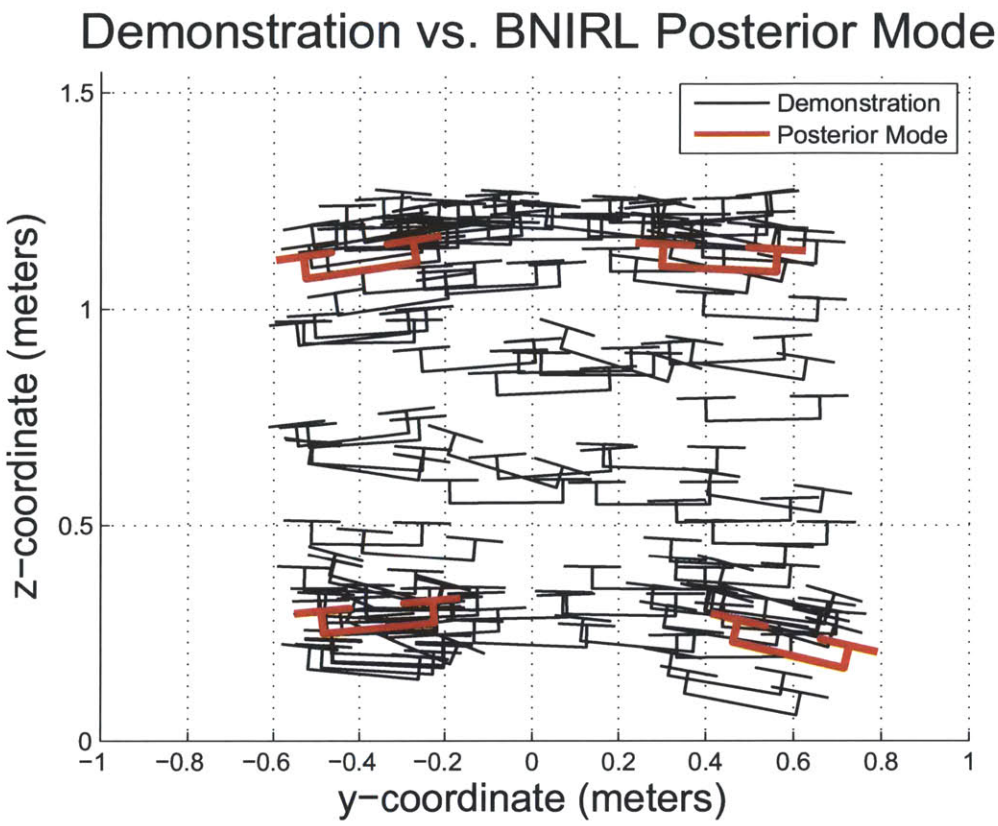


Figure 7-3: A cluttered trajectory in which the demonstrator moves randomly between the four corners of a square is shown in black. The BNIRL posterior mode is shown in red, which consists of four subgoals, one at each corner of the square as expected.

δ and commanded wheel speed ω_c (motor torque to the drive wheels is then set by an inner-loop PI controller on wheel speed). Figure 7-5 shows the RC car used in the experiment along with a diagram of states and actions. Demonstrations are performed via manual remote-controlled operation with a joystick. States $\dot{x}_b, \dot{y}_b$ and $\dot{\psi}$ are measured with a motion capture system sampled at 100Hz, and wheel speed ω is measured onboard with an optical encoder and transmitted wirelessly at 100Hz. The transition model f required for GPDP (Section 6.4) is taken from a basic model of car dynamics with wheel slip [95], with model parameters identified from test data.

The squared exponential kernel (2.14) is used for all GPs, with parameters optimized to local optima using gradient ascent of the log evidence. The CRP concentration parameter $\eta = 0.001$ is used for all experiments. While detailed discussion of

selecting this parameter is omitted, empirical findings indicate that subgoal learning results are not particularly sensitive to η.

Figure 7-6 (left) shows a 30-second demonstration which includes straight-line driving, standard left/right turns, and advanced drifting turns. The GPSRL algorithm is applied with $\alpha = 25$, and convergence to six learned subgoal rewards is attained within roughly 200 sampling iterations. Figure 7-6 (right) shows the six subgoal state locations, where arrows represent the body velocities $\dot{x}_b$ and $\dot{y}_b$, the rotation of the rectangle represents heading rate $\dot{\psi}$, and wheel speed ω is omitted for clarity.

The learned subgoals correctly identify the six basic maneuvers from the demonstration: stop, drive straight, left turn, right turn, left drifting turn, and right drifting turn. The trajectory is color-coded to show the partition assignments of the demo states to the six learned subgoals (Figure 7-6, left). Note that the Bayesian nonparametric model from Section 6.5 is not biased towards clustering contiguous trajectory segments, yet the posterior mode assignments shows that contiguous segments are indeed clustered into appropriate subgoals, as would be expected.

To explore the behavior of GPSRL as more demonstration data is added, ten more 30-second human-controlled demonstrations were recorded with the demonstrator instructed to include the same types of behavior as in Figure 7-6. Figure 7-7 shows the number of subgoals learned as each new demonstration is added, averaged over 25 trials, with the confidence parameter again set to $\alpha = 25$. The number of learned subgoals does not increase arbitrarily with more demonstration data, and stays within two standard deviations of six learned subgoals from the single demonstration case.

7.3.1 Confidence Parameter Selection and Expertise Determination

The confidence parameter α has a direct effect on the posterior distribution (6.3) and thus the number of subgoals learned from a given demonstration. Since α represents the expected degree to which the demonstrator is able to maximize reward, there is

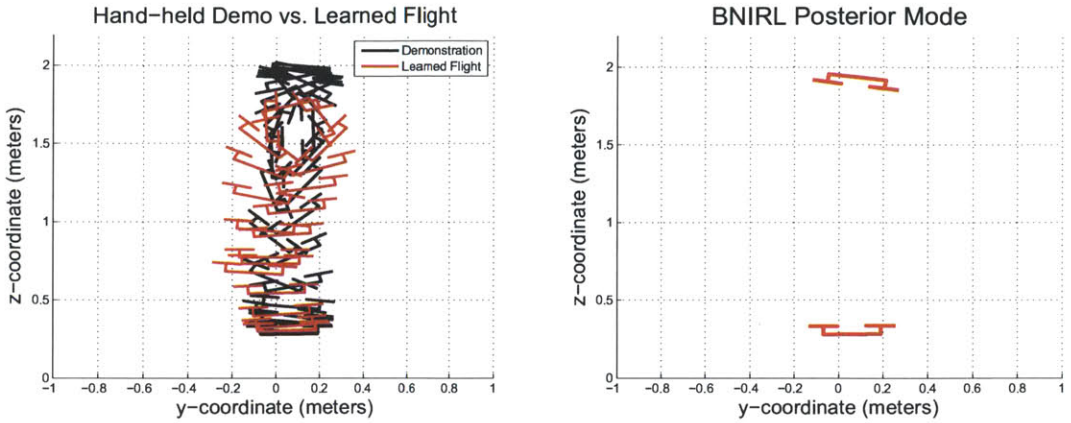
thus no analytical method for choosing the parameter. Even so, small or large values of alpha produce consistent trends in the number of learned subgoals. In the limit as $\alpha \to 0$, the demonstrator is assumed to choose arbitrary actions that do not attempt to maximize reward. This leads to the discovery of a single partition since the entire demonstration can be explained as noisy or suboptimal actions towards any arbitrary subgoal. In the limit as $\alpha \to \infty$, the demonstrator is assumed perfectly optimal. This leads to a larger number of learned subgoals since any noise or mistakes present in the demonstrated actions will be treated as completely intentional, and the resultant state will likely be added as a subgoal.

Figure 7-8 (left) shows a demonstration of the RC car that consists of a mixture of right-handed turns of three distinct turning radii. This demonstration was intentionally created so that there are three unambiguously "true" subgoals. Figure 7-8 (right) shows the number of learned subgoals for a logarithmic sweep of α averaged over 50 trials, with 1500 sampling iterations per trial. As expected, there is a range of $\alpha < 10$ through which only one subgoal is discovered. For $10 < \alpha < 60$ the number of subgoals discovered is within two standard deviations of the true value (three), showing that there is a relatively large range of suitable parameter settings. For $\alpha > 60$ the algorithm discovers more subgoals as expected, since noisy state-actions are interpreted as intentional.

The confidence parameter can also be used in the opposite way to quantify the level of "expertise" of the demonstrator. Consider instead if the demonstration in Figure 7-8 came from a demonstrator who was instructed to execute the same turn many times. If this were the case, the different turning radii would then be attributed to the sub-optimal execution of the maneuver. The numerical level of expertise of the demonstrator could then be found by sweeping the value of α until more than one subgoal is consistently discovered – in this case $\alpha = 10$. Aside from serving as an indicator of expertise, this value of α could then be used as a starting point to interpret future demonstrations which may contain more than one subgoal.

7.3.2 Autonomous Execution of Learned Subgoals

Once subgoal rewards are learned, they can be converted to options as presented in Section 6.6. Figure 7-9 shows a comparison of demonstrated maneuvers (from the original demonstration in Section 4.7.1) to the autonomous execution of the corresponding learned subgoal option for three of the six learned subgoals (colors correspond to Figure 7-6). The autonomously executed maneuvers match the original demonstration closely, even though *no additional learning* was performed beyond the GPDP step in Algorithm 5.



(a) (b)

Figure 7-4: A hand-held demonstrated quadrotor flip is shown in black (a). The BNIRL posterior mode in this case converges to two subgoals, one at the bottom and one (inverted) at the top of the flip trajectory (b). An autonomous quadrotor takes the subgoals as waypoints and executes the learned trajectory in actual flight, shown in red (a).

Figure 7-5: RC car used for experimental results (left) with optical encoder, battery, radio modem, and reflective markers for motion capture. Diagram of car state (right) with body velocities $\dot{x}_b$ and $\dot{y}_b$, heading rate $\dot{\psi}$, wheel speed ω, and steering command δ.

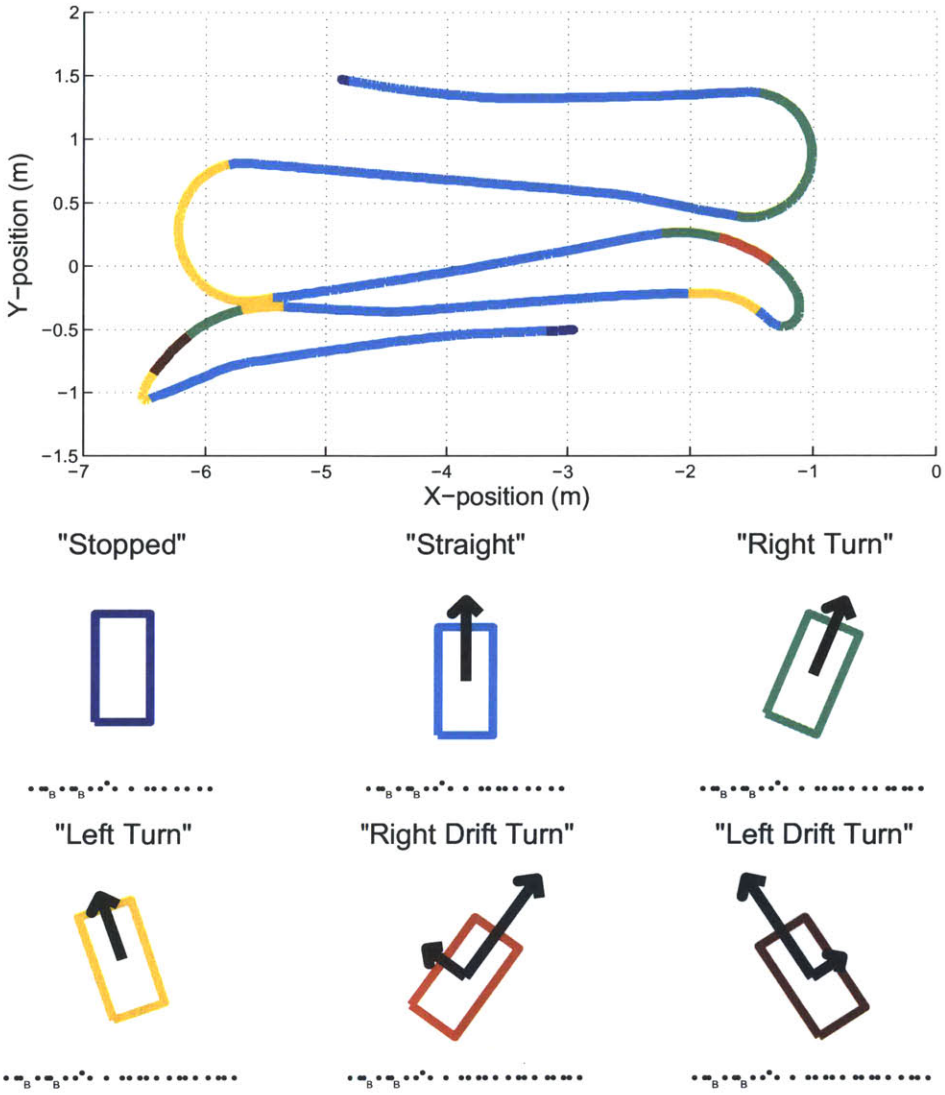


Figure 7-6: Thirty-second manually-controlled demonstration trajectory (top) starting in upper-left and ending in lower-middle. Six learned subgoal state locations (bottom), where arrows represent the body velocities $\dot{x}_b$ and $\dot{y}_b$, the rotation of the rectangle represents the heading rate $\dot{\psi}$, and the wheel speed ω is omitted for clarity. Learned subgoal labels ("Left turn", etc.) added manually.

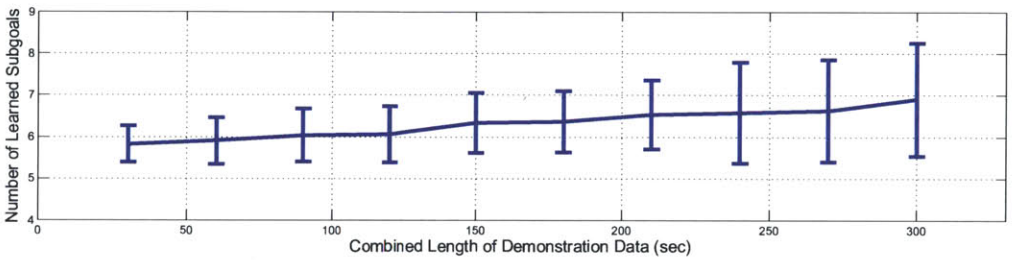


Figure 7-7: Number of subgoals learned versus the total length of demonstration data sampled, averaged over 25 trials. The number of learned subgoals does not increase arbitrarily with more demonstration data, and stays within 2-σ of the six learned subgoals from the single demonstration.

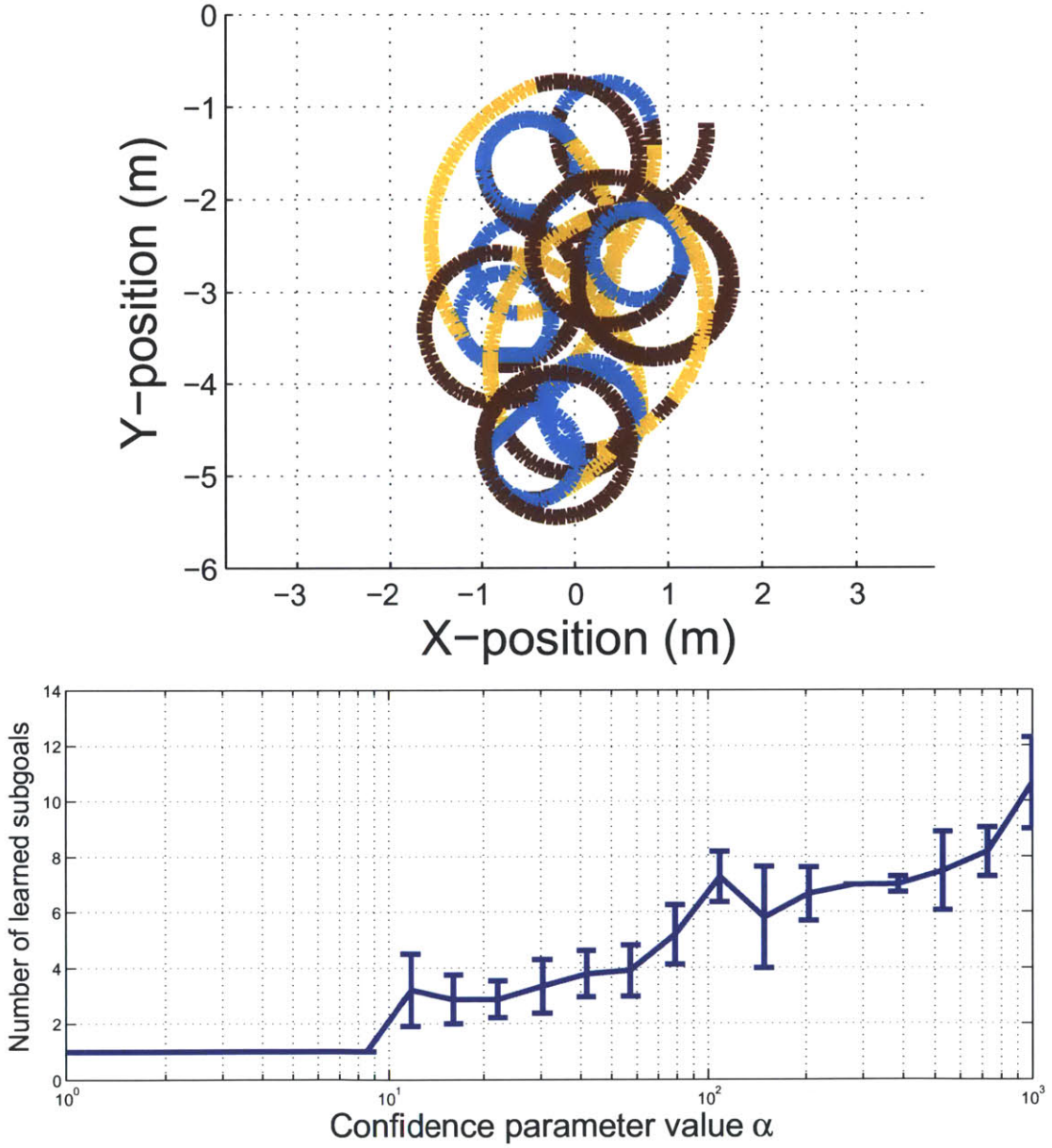


Figure 7-8: Demonstration with three distinct right-turning radii (top). Number of learned subgoals (50-trial average) versus the confidence parameter value α (bottom). The trajectory is color-coded with partition labellings for $\alpha = 25$, showing correct subgoal identification for the three turning radii.

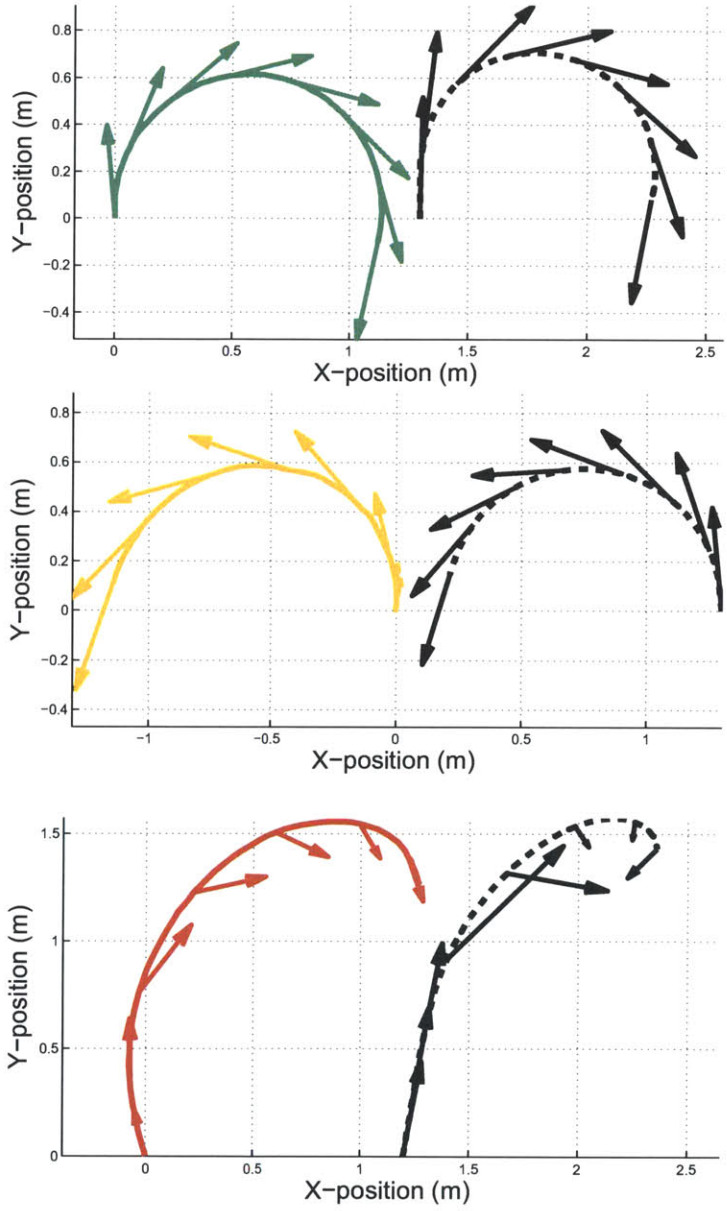


Figure 7-9: Comparison of demonstrated maneuvers (from Figure 7-6, shown here in black) to the autonomous execution of the corresponding learned subgoal options (colored to correspond with the subgoals in Figure 7-6.

Chapter 8

Conclusions and Future Work

This thesis has presented several contributions which improve existing reward learning from demonstration methods and developed new Bayesian nonparametric reward learning frameworks that enable scalable reward learning for real-world robotic systems.

In Chapter 3, several modifications to the Bayesian IRL algorithm were presented to improve its efficiency and tractability in situations where the state space is large and the demonstrations span only a small portion of it. Contributions in this chapter included:

- The identification of key limitations of the Bayesian IRL algorithm which hinder computational tractability for large domains.
- A fundamental improvement of the algorithm which takes as input a kernel function that quantifies similarity between states. The resulting algorithm is shown to have improved computational efficiency while maintaining the quality of the resulting reward function estimate.
- The development of a new acceptance probability similar to a cooling schedule in Simulated Annealing, enabling an effective trade-off between exploration and exploitation in the reward inference process. Use of the cooling schedule in the modified BIRL algorithm allows the MCMC process to first find areas of

high posterior probability then focus the samples towards them, speeding up convergence.

In Chapter 4, a method was developed that learns *multiple* reward functions from a single demonstration, enabling reward learning from unsegmented demonstrations containing multiple distinct tasks which are common in robot learning from demonstration. Chapter 5 offered several approximations to the demonstrator likelihood function to further improve computational tractability in large domains. Contributions in these chapters included:

- The Bayesian nonparametric inverse reinforcement learning (BNIRL) framework, which uses a Bayesian nonparametric mixture model to automatically partition the data and find a set of simple reward functions corresponding to each partition. Several computational advantages of the method over existing IRL are shown, namely the search over a finite (as opposed to infinite) space of possible rewards and the ability to easily parallelize the majority of the method's computational requirements.

- Simulation results are given which show the method's ability to handle cyclic tasks (where the agent begins and ends in the same state) that would break existing algorithms without modification due to the existence of multiple subgoal rewards in a single demonstration.

- The development of two approximations to the demonstrator likelihood function. In the first method, the Real-time Dynamic Programming (RTDP) framework is incorporated to approximate the optimal action-value function, and simulation results for a Grid World domain show order of magnitude speedups over exact solvers. In the second method, an existing closed-loop controller takes the place of the optimal value function, and simulation results are given for a pedestrian data set demonstrating the ability to learn meaningful subgoals using a very simple closed-loop control law.

In Chapter 6, the BNIRL framework was extended to general, continuous demonstration domains with the use of Gaussian process reward representations. Contributions in this chapter included:

- The Gaussian process subgoal reward learning (GPSRL) algorithm, which is the only learning from demonstration method able to learn multiple reward functions from unsegmented demonstration in general continuous domains. GPSRL does not require discretization of the continuous state space and focuses computation efficiently around the demonstration itself.

- Incorporation of the Markov decision process options framework to enable execution of the learned behaviors by the robotic system and provide a principled basis for future learning and skill refinement.

- The development of a method for choosing the key confidence parameter in the GPSRL likelihood function. The method can also be used to quantify the relative skill level of the original demonstration enabling comparison between multiple demonstrators.

Finally, Chapter 7 provided experimental results which validate the use of BNIRL and GPSRL for reward learning from demonstration on robotic hardware. Contributions in this chaper included:

- Application of BNIRL to learn Quadrotor flight maneuvers from a human demonstrator using only hand motions. Learned subgoal rewards (in the form of waypoints) are passed as commands to an autonomous quadrotor which executes the learned behavior in actual flight. The entire process from demonstration to reward learning to robotic execution takes on the order of 10 seconds to complete using a single computer, highlighting the ability of BNIRL to use data from a safe (and not necessarily dynamically feasible) demonstration environment and quickly learn subgoal rewards that can be used in the actual robotic system.

- Application of GPSRL to a robotic car domain. In the experiments, multiple difficult maneuvering skills such as drifting turns are identified from a single unsegmented demonstration. The learned subgoal rewards are then executed autonomously using MDP options and shown to closely match the original demonstration. Finally, the relative skill level of the demonstrator is quantified through *a posteriori* analysis of the confidence likelihood parameter.

8.1 Future Work

There are several extensions to the contributions presented in the thesis which could serve as areas of future work. These include improved posterior inference procedures, sparsifying demonstration trajectories to improve computational efficiency, learning more complex reward representations, and adding the ability to identify multiple demonstrators.

8.1.1 Improved Posterior Inference

Since both BNIRL and GPSRL rely on approximate inference of a Bayesian nonparametric mixture model, the inference procedure itself has a large effect on the computational properties of the algorithm. As presented in Chapters 4 and 6, straightforward Gibbs sampling is used for its algorithmic simplicity and ubiquity in the literature [45, 59, 60].

Several improvements to the Gibbs sampling method exist and can be applied to the reward learning algorithms in this thesis. These include the use of auxiliary variables for non-conjugate models [26] and efficient collapsed Gibbs sampling [68, 98], though the latter would require the derivation of a conjugate posterior distribution. A family of variational inference methods also exist which have been applied to Bayesian nonparametric models [15, 46, 91]. While these methods are promising, the reward learning posterior would have to be modified to make it compatible with variational methods. Future work could explore these options in an effort to improve the efficiency of the sampling step of the Bayesian nonparametric reward learning framework.

8.1.2 Sparsification of Demonstration Trajectories

Since the computational complexity of the sampling procedure scales with the amount of demonstration data to be analyzed, another method for improving algorithmic efficiency is to sparsify the demonstration trajectories. A straightforward method for reducing the amount of demonstration data would be to remove state-action pairs which are above some threshold of similarity. Such a process would resemble the method for reducing the number of subgoal candidates in Section 6.2, but applied to the entire demonstration set.

A more principled Bayesian method for reducing the amount of demonstration data could use the Gaussian process framework to represent trajectories instead of tabular storage of each observed state-action pair. This would require learning a Gaussian process which maps demonstration states to the observed actions taken. While such a representation would obviously be an approximation of the true observations, many GP sparsification methods exist [24, 52, 70] which could greatly reduce the memory and computational requirements of the reward learning framework in cases where there is a large amount of demonstration data.

8.1.3 Hierarchical Reward Representations

Throughout the thesis, subgoal reward functions are utilized as a basic representation which is easily learned and interpreted. The Bayesian nonparametric reward learning framework could be generalized to more complex representations. Of particular interest in the literature are hierarchical Bayesian nonparametric models [3, 14, 31]. While typically more difficult to infer, these models offer a method of learning a more complex hierarchical posterior structure which may better explain demonstrations with nested tasks. In analogy to constructing skill trees of MDP options [49], hierarchical representations could learn a tree of reward models, where the tree structure is inferred from the demonstration data.

8.1.4 Identifying Multiple Demonstrators

While much of the thesis has focused on learning multiple reward functions from a single demonstrator, the framework can also be extended to additionally identify *multiple demonstrators* from a set of observed trajectories. In this scenario, it is likely that individual demonstrators will perform the same overall tasks but do so in potentially different ways. The reward learning framework would then need the ability to not only learn the overall tasks, but differentiate between demonstrators within those tasks.

As a result, such an extension would likely leverage a hierarchical representation as discussed above. In the learned hierarchy, parent nodes would represent common tasks while the children of these nodes would represent the various ways which individual demonstrators perform the tasks. The learned structure would then provide insight into the commonalities and differences between different demonstrators solving the same task.

Bibliography

[1] Pieter Abbeel. *Apprenticeship learning and reinforcement learning with application to robotic control.* PhD thesis, Stanford, 2008.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. *Twentyfirst international conference on Machine learning ICML 04*, (ICML):1, 2004.

[3] Ryan Prescott Adams, Zoubin Ghahramani, and Michael I Jordan. Tree-Structured stick breaking for hierarchical data. *Statistics*, 23(1):16, 2010.

[4] R Aler, O Garcia, and J M Valls. Correcting and Improving Imitation Models of Humans for Robosoccer Agents. *2005 IEEE Congress on Evolutionary Computation*, pages 2402–2409, 2005.

[5] C Andrieu, N De Freitas, A Doucet, and Michael I Jordan. An Introduction to MCMC for Machine Learning. *Science*, 50(1):5–43, 2003.

[6] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[7] C.G. Atkeson and S. Schaal. Robot learning from demonstration. In *Machine Learning International Workshop*, number 1994, pages 12–20. Citeseer, 1997.

[8] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action understanding as inverse planning. *Cognition*, 113(3):329–349, 2009.

[9] Andrew G Barto, Steven J Bradtke, and Satinder P Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[10] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[11] Darrin C Bentivegna, Ales Ude, Christopher G Atkeson, and Gordon Cheng. Humanoid robot learning and game playing using PC-based vision. *IEEE/RSJ International Conference on Intelligent Robots and System*, 3:2449–2454, 2002.

[12] James O Berger. *Statistical Decision Theory and Bayesian Analysis (Springer Series in Statistics)*. Springer, 1985.

[13] D P Bertsekas and J N Tsitsiklis. *Neuro-Dynamic Programming*, volume 5 of *the Optimization and Neural Computation Series*. Athena Scientific, 1996.

[14] David M Blei, Thomas L Griffiths, and Michael I Jordan. The nested Chinese restaurant process and Bayesian nonparametric inference of topic hierarchies. *Journal of the ACM*, 57(2):1–30, 2007.

[15] David M Blei and Michael I Jordan. Variational methods for the Dirichlet process. *Twentyfirst international conference on Machine learning ICML 04*, 10(1):12, 2004.

[16] Stephen Boyd, L El Ghaoui, E Feron, and V Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *SIAM Studies in Applied Mathematics*. SIAM, 1994.

[17] Sylvain Calinon and Aude Billard. Incremental learning of gestures by imitation in a humanoid robot. *Proceeding of the International Conference on Human Robot Interaction HRI 07*, page 255, 2007.

[18] Antonio Chella. A posture sequence learning system for an anthropomorphic robotic hand. *Robotics and Autonomous Systems*, 47(2-3):143–152, 2004.

[19] J Chen and A Zelinsky. Programing by Demonstration: Coping with Suboptimal Teaching Actions. *The International Journal of Robotics Research*, 22(5):299–319, 2003.

[20] Sonia Chernova and Manuela Veloso. Confidence-based policy learning from demonstration using Gaussian mixture models. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems AAMAS*, volume 5. ACM, 2007.

[21] Sonia Chernova and Manuela Veloso. Multi-thresholded approach to demonstration selection for interactive robot learning. *Proceedings of the 3rd international conference on Human robot interaction HRI 08*, page 225, 2008.

[22] William S Cleveland and Clive Loader. Smoothing by Local Regression : Principles and Methods. *Technical Report, AT&T Bell Laboratories*, 1995.

[23] Adam Coates, Pieter Abbeel, and Andrew Y Ng. Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97, 2009.

[24] Lehel Csato and Manfred Opper. Sparse Online Gaussian Processes. *Neural Computation*, 14(3):641–668, 2002.

[25] Mark Cutler and Jonathan P How. Actuator Constrained Trajectory Generation and Control for Variable-Pitch Quadrotors. In *Conference on Guidance, Navigation and Control*, 2012.

[26] P Damien, J Wakefield, and S Walker. Gibbs sampling for Bayesian non-conjugate and hierarchical models by using auxiliary variables. *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, 61(2):331–344, 1999.

[27] Marc Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian Process Dynamic Programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

[28] N Delson and H West. Robot programming by human. demonstration: Adaptation and inconsistency in constrained motion. In *Proceedings of IEEE International conference on Robotics and Automation*, volume 1, pages 30–36. IEEE, 1996.

[29] J L Drury, J Scholtz, and H A Yanco. Awareness in human-robot interactions. In *IEEE International Conference on Systems Man and Cybernetics*, volume 1, pages 912–918. Ieee, 2003.

[30] M D Escobar and M West. Bayesian density estimation using mixtures. *Journal of the American Statistical Association*, 90(430):577– 588, 1995.

[31] Michael D Escobar and Mike West. Computing nonparametric hierarchical models. *Practical nonparametric and semiparametric Bayesian statistics*, 133(1992):1–22, 1998.

[32] R Fiebrink and P R Cook. The Wekinator : A System for Real-time, Interactive Machine Learning in Music. *ISMIR*, 4(3):2005–2005, 2010.

[33] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[34] T Fong, Jean Scholtz, Julie A Shah, L Fluckiger, Clayton Kunz, David Lees, John Schreiner, Michael Siegel, Laura M Hiatt, Illah Nourbakhsh, Reid Simmons, Brian Antonishek, Magda Bugajska, Robert Ambrose, Robert Burridge, Alan Schultz, and J Gregory Trafton. A Preliminary Study of Peer-to-Peer Human-Robot Interaction. *2006 IEEE International Conference on Systems Man and Cybernetics*, 4:3198–3203, 2006.

[35] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. Nonparametric Bayesian Learning of Switching Linear Dynamical Systems. *Electrical Engineering*, 21(1), 2008.

[36] H Friedrich and R Dillmann. Robot programming based on a single demonstration and user intentions. In *3rd European Workshop on Learning Robots at ECML'95*, 1995.

[37] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian Data Analysis*, volume 2 of *Texts in statistical science*. Chapman & Hall/CRC, 2004.

[38] Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984.

[39] M A Goodrich and Dan R Olsen JR. Seven principles of efficient human robot interaction. In *IEEE International Conference on Systems Man and Cybernetics*, volume 4, pages 3942–3948. IEEE, 2003.

[40] Daniel H Grollman and Odest Chadwicke Jenkins. Dogged Learning for Robots. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2483–2488, 2007.

[41] Daniel H Grollman and Odest Chadwicke Jenkins. Sparse incremental learning for interactive robot control policy estimation. *2008 IEEE Int. Conf. on Robotics and Automation*, 2008.

[42] J P How, B Bethke, A Frank, D Dale, and J Vian. Real-time indoor autonomous vehicle test environment, 2008.

[43] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning Rhythmic Movements by Demonstration Using Nonlinear Oscillators. In *In Proceedings of the IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS2002)*, pages 958—-963, 2002.

[44] Tetsunari Inamura, Masayuki Inaba, and Hirochika Inoue. Acquisition of probabilistic behavior decision model based on the interactive teaching method. *Learning*, pages 523–528, 1999.

[45] Michael I Jordan. Dirichlet Processes, Chinese Restaurant Processes and All That. *Tutorial presentation at the NIPS Conference*, 2005.

[46] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 233(2):183–233, 1999.

[47] S Kiesler. Fostering common ground in human-robot interaction. *IEEE International Workshop on Robot and Human Interactive Communication*, 2005:729–734, 2005.

[48] J Zico Kolter, Pieter Abbeel, and Andrew Y Ng. Hierarchical Apprenticeship Learning , with Application to Quadruped Locomotion. In J C Platt, D Koller, Y Singer, and S Roweis, editors, *Advances in Neural Information Processing Systems*, volume 1. MIT Press, 2008.

[49] G Konidaris, S Kuindersma, R Grupen, and A Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2011.

[50] Y Kuniyoshi, M Inaba, and H Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822, 1994.

[51] S Lauria. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3-4):171–181, 2002.

[52] Neil Lawrence, Matthias Seeger, and Ralf Herbrich. Fast Sparse Gaussian Process Methods: The Informative Vector Machine. *Advances*, 15:609–616, 2003.

[53] Manuel Lopes, Francisco Melo, and Luis Montesano. Active Learning for Reward Estimation in Inverse Reinforcement Learning. *Machine Learning and Knowledge Discovery in Databases*, pages 31–46, 2009.

[54] Barbara Majecka. *Statistical models of pedestrian behaviour in the Forum.* Msc dissertation, University of Edinburgh, 2009.

[55] Amy McGovern and Andrew G Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *Machine Learning*, volume 85. Citeseer, 2001.

[56] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. *Proc. 13th European Conf. on Machine Learning*, 2002.

[57] Bernard Michini and Jonathan P How. Bayesian Nonparametric Inverse Reinforcement Learning. In *European Conference on Machine Learning*, Bristol, United Kingdom, 2012.

[58] J Nakanishi. Learning from demonstration and adaptation of biped locomotion. *Robotics and Autonomous Systems*, 47(2-3):79–91, 2004.

[59] Radford M Neal. Probabilistic Inference Using Markov Chain Monte Carlo Methods. *Intelligence*, 62(September):144, 1993.

[60] Radford M Neal. Markov Chain Sampling Methods for Dirichlet Process Mixture Models. *Journal Of Computational And Graphical Statistics*, 9(2):249, 2000.

[61] Gergely Neu and Csaba Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proc. UAI*, 2007.

[62] A Y Ng and S Russell. Algorithms for inverse reinforcement learning. In *Proc. of the 17th International Conference on Machine Learning*, pages 663–670, 2000.

[63] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. *International Symposium on Experimental Robotics*, 21:1–10, 2004.

[64] R Parasuraman, T B Sheridan, and C D Wickens. A model for types and levels of human interaction with automation. *IEEE transactions on systems man and cybernetics*, 30(3):286–297, 2000.

[65] Jim Pitman. *Combinatorial Stochastic Processes.* Springer-Verlag, Berlin, 2006.

[66] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[67] P K Pook and D H Ballard. Recognizing teleoperated manipulations. In *1993 Proceedings IEEE International Conference on Robotics and Automation*, pages 578–585. IEEE, 1993.

[68] Ian Porteous, David Newman, Arthur Asuncion, and Max Welling. Fast Collapsed Gibbs Sampling For Latent Dirichlet Allocation Categories and Subject Descriptors. *Work*, page 569, 2008.

[69] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, volume 10 of *Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics*. John Wiley & Sons, Inc., 1994.

[70] Joaquin Quinonero Candela and Carl Edward Rasmussen. A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6(3):1939–1959, 2005.

[71] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *IJCAI*, 2007.

[72] Carl Edward Rasmussen and Christopher K I Williams. *Gaussian processes in machine learning*. Lecture Notes in Computer Science. The MIT Press, 2006.

[73] Nathan Ratliff, David Bradley, and JA Bagnell. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems 19*, 2007.

[74] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. *Proc. of the 23rd International Conference on Machine Learning*, pages 729–736, 2006.

[75] G O Roberts and S K Sahu. Updating schemes, correlation structure, blocking and parameterisation for the Gibbs sampler. *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, 59(2):291–317, 1997.

[76] P E Rybski and R M Voyles. Interactive task training of a mobile robot through human gesture recognition. *Proceedings 1999 IEEE International Conference on Robotics and Automation Cat No99CH36288C*, 1(May):664–669, 1999.

[77] C Sammut, S Hurst, D Kedzier, and D Michie. Learning to Fly. *Proceedings of the Ninth International Conference on Machine Learning*, 19(4):385–393, 1992.

[78] J Saunders, C L Nehaniv, and K Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. *Proceeding of the 1st ACM Conference on Human Robot Interaction HRI*, pages 118–125, 2006.

[79] Bernhard Scholkopf and Alexander J Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* The MIT Press, 2001.

[80] O Simsek and A G Barto. Learning skills in reinforcement learning using relative novelty. *Abstraction Reformulation And Approximation Proceedings*, 3607:367–374, 2005.

[81] Ozgur Simsek, Alicia P Wolfe, and Andrew G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proc. 22nd Int. Conf. on Machine Learning*, 2005.

[82] William D Smart. *Making Reinforcement Learning Work on Real Robots.* PhD thesis, Department of Computer Science, Brown University, 2002.

[83] J J Steil, F Röthling, R Haschke, and Helge Ritter. Situated robot learning for multi-modal instruction and imitation of grasping. *Robotics and Autonomous Systems*, 47(2-3):129–141, 2004.

[84] Martin Stolle and Doina Precup. Learning Options in Reinforcement Learning. *Abstraction Reformulation and Approximation*, 2371:212–223, 2002.

[85] Kristen Stubbs, Pamela J Hinds, and David Wettergreen. Autonomy and Common Ground in Human-Robot Interaction: A Field Study. *IEEE Intelligent Systems*, 22(2):42–50, 2007.

[86] Erik B Sudderth. Graphical Models for Visual Object Recognition and Tracking by. *Thesis*, 126(1):301, 2006.

[87] R S Sutton and A G Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[88] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

[89] John D Sweeney and Rod Grupen. A model of shared grasp affordances from demonstration. *2007 7th IEEERAS International Conference on Humanoid Robots*, pages 27–35, 2007.

[90] Umar Syed and R E Schapire. A Game-Theoretic Approach to Apprenticeship Learning. *Advances in Neural Information Processing Systems 20*, 20:1–8, 2008.

[91] YW Teh, K Kurihara, and M Welling. Collapsed Variational Inference for HDP. *Advances in Neural Information Processing Systems 20*, 20(20):1481–1488, 2007.

[92] Sebastian Thrun. Toward a Framework for Human-Robot Interaction. *Human-Computer Interaction*, 19(1):9–24, 2004.

[93] C P Tung and A C Kak. Automatic learning of assembly tasks using a DataGlove system. In *Proceedings 1995 IEEERSJ International Conference on Intelligent Robots and Systems Human Robot Interaction and Cooperative Robots*, volume 1, pages 1–8. IEEE Comput. Soc. Press, 1995.

[94] H Veeraraghavan and M Veloso. Teaching sequential tasks with repetition through demonstration. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1357–1360. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, 2008.

[95] Efstathios Velenis, Emilio Frazzoli, and Panagiotis Tsiotras. Steady-State Cornering Equilibria and Stabilization for a Vehicle During Extreme Operating Conditions. *International Journal Of Vehicle Autonomous Systems*, 8(2/3/4):217–241, 2010.

[96] Richard M Voyles and P Khosla. A multi-agent system for programming robotic agents by human demonstration. *Integrated ComputerAided Engineering*, 8(1):59–67, 1998.

[97] Thomas J Walsh, Michael L Littman, and Carlos Diuk. Generalizing Apprenticeship Learning across Hypothesis Classes. *Learning*, pages 1119–1126, 2010.

[98] Han Xiao and Thomas Stibor. Efficient Collapsed Gibbs Sampling For Latent Dirichlet Allocation. *Asian Conference on Machine Learning*, 13:63–78, 2010.

[99] Brian D Ziebart, Andrew Maas, and J Andrew Bagnell. Human Behavior Modeling with Maximum Entropy Inverse Optimal Control. *AAAI Spring Symposium on Human Behavior Modeling*, pages 92–97, 2009.

[100] Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum Entropy Inverse Reinforcement Learning. In *Proc AAAI*, pages 1433–1438. AAAI Press, 2008.