Skill Discovery from Multiple Related Demonstrators

Pravesh Ranchod

Supervised by

Prof. George Konidaris, Dr. Benjamin Rosman, and Prof. Clint van Alten

A thesis presented for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science and Applied Mathematics
University of the Witwatersrand
South Africa

December 2017

Abstract

An important ability humans have is that we can recognise that some collections of actions are useful in multiple tasks, allowing us to exploit these skills. A human who can run while playing basketball does not need to relearn this ability when he is playing soccer as he can employ his previously learned running skill.

We extend this idea to the task of Learning from Demonstration (LfD), wherein an agent must learn a task by observing the actions of a demonstrator. Traditional LfD algorithms learn a single task from a set of demonstrations, which limits the ability to reuse the learned behaviours. We instead recover all the latent skills employed in a set of demonstrations. The difficulty involved lies in determining which collections of actions in the demonstrations can be grouped together and termed "skills"? We use a number of characteristics observed in studies of skill discovery in children to guide this segmentation process – usefulness (they lead to some reward), chaining (we tend to employ certain skills in common combinations), and reusability (the same skill will be employed in many different contexts).

We use reinforcement learning to model goal directed behaviour, hidden Markov models to model the links between skills, and nonparametric Bayesian clustering to model reusability in a potentially infinite set of skills. We introduce nonparametric Bayesian reward segmentation (NPBRS), an algorithm that is able to segment demonstration trajectories into component skills, using inverse reinforcement learning to recover reward functions representing the skill ob-

jectives.

We then extend the algorithm to operate in domains with continuous state spaces for which the transition model is not specified, with the algorithm successfully recovering component skills in a number of simulated domains. Finally, we perform an experiment on CHAMP, a physical robot tasked with making various drinks, and demonstrate that the algorithm is able to recover useful skills in a robot domain.

Declaration

I, Pravesh Ranchod, declare that this Thesis is my own, unaided work. It is being submitted for the Degree of Doctor of Philosophy at the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other University.

Signature

15/12/2017

Date

Acknowledgements

When I decided to leave the business world and go back to academia, I really had no idea what I would work on, beyond the happy supposition that it was unlikely to be database driven web applications. I toyed with many ideas before settling on one, and the most important reason that robotics and machine learning stuck was my friend and supervisor, George Konidaris. George has that perfect combination of being tremendously well-informed, incredibly intelligent, and overwhelmingly good-natured. Despite us being the same age, George is the academic I want to be when I grow up, and I hope that I can be as influential and valuable to my students as George has been to me.

Another of my supervisors, Clint van Alten was also pivotal at this point. Together, we started the Artificial Intelligence and Machine Learning research group at Wits, which had previously not been active in this field at all. There is no way I would have been able to strike out in this new direction without his support and encouragement.

Benjamin Rosman joined my supervision team late, when it became clear that reinforcements were required. He injected a great deal of energy and enthusiasm into my research, and brought a pile of new insights with him. His leadership of the RAIL Lab has made all of us better academics. As a supervisor, colleague and friend, he has been a source of support in every sphere of my life, and I am deeply grateful.

One of the great things about an academic environment is that you get to inter-

act with so many cool people. I loved working with Dean Wookey in particular, who is driven by improving things. Whether it's training students for competitive programming, helping out with my lectures or developing systems and course materials, he always acts selflessly. And he's quite good at Warcraft too.

Shun Pillay is an unsung hero at Wits, as he always goes above and beyond to support anything and anyone interesting. Without Shun pushing robotics, everything would have moved WAY slower.

Scott Hazelhurst, Mike Mchunu and Turgay Celik, thank you for showing me how to be a teacher and a researcher. I would be pretty lousy at this if I didn't have such good targets.

Steven James and Richard Klein have been great partners in crime, and have always come through when I needed anything.

To all of the awesome people that Wits has allowed me (and for some reason paid me) to hang out with: Angelo, Max, Mitch, Craig, Sicelukwanda, Orry, Hima, Warwick, Phumlani and the entire RAIL Lab, you are why it's so cool to be at the University. I hope we'll keep hanging out until we're dead.

And for perhaps the most important characteristic that an academic can have, I'd like to thank my parents and my sisters. In this family, it's considered good manners to take a contrary position on everything. Sure, this can mean you sometimes have to lurch from anarchism to capitalism within the space of two sentences, and you certainly can't support the same football club as anyone else even though Arsenal is obviously the correct answer. But this taught me

that every viewpoint must be justified, that your viewpoints are not worthy because you're older or because you're louder, but because you have thought things through. It's a great example, and one of many reasons I hope that my son can one day view me the way I view you.

Which brings me to my own little family. Priya, I've lost count of the ways that you make my life more exciting, more fulfilling and easier. When my instinct is to stick to my routine, you make me want to explore. When I get overwhelmed by complexity, you show up with a pencil and simplify things. When I forget myself, you remind me.

And to my little guy, Dhruv, you are the strongest, cleverest, cutest little happiness generator. You're like a heater for the soul, and I can't wait to see what wonderful things you're going to do.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human beings interact with the world in tremendously impressive ways. We accomplish incredibly complex tasks in fields such as science, engineering, art, philosophy and entertainment. And yet, our ability to affect the world is contingent on us flexing and relaxing particular muscles in specific sequences. We cannot contemplate expressing the relatively simple task of saying a sentence at the level of actively choosing specific muscle activations, yet we manage the task with ease anyway. The key is that we are able to operate at different levels of abstraction, with abilities that become unconscious as we master them. For instance, at one level of abstraction we can formulate a sentence, and forward it to the lower level "speak" ability to translate into muscle activations.

An important advantage of this hierarchical structure is that it encourages the reuse of common actions, allowing us to identify and exploit commonalities in different tasks. When a basketball player decides to play baseball instead

and is faced with the need to move quickly, the player is able to recognise that the desired outcome can be achieved by using the running ability that already available to him. Without access to his skill library, the player would need to learn to accomplish the objective from scratch using only the actions available at the lowest level of abstraction (known as primitive actions), rendering even relatively simple tasks intractable.

The development of our skill library is a focus of our early years, as we learn to predict the consequences of our actions, and build up a library of basic skills that we can reuse in later tasks. The techniques we use to learn these skills include both experimentation – using trial and error, as well as learning by observing the behaviour of those around us.

The presence of a skill library is pivotal to our ability to accomplish complex tasks. It is thus desirable for artificial agents to have one, and experiments have validated that agents can learn much faster when given access to useful skills rather than operating at the level of primitive actions [Konidaris *et al.* 2012; Sutton *et al.* 1999]. One major difficulty is identifying these useful skills in the first place.

This thesis focusses on identifying skills from unstructured expert demonstrations. We aim to discover all the reusable behaviours employed by a demonstrator which provides only low level primitive actions, with no annotations regarding skill boundaries or objectives. This allows us to build the next level of the abstraction hierarchy, with higher level skills available to an agent when performing new tasks.

1.1 Skills

A number of interesting questions arise when attempting to model the skill discovery process. How do we recognise that certain sequences of primitive actions can be separated from the actions before and after them and grouped together into a skill? Could all such collections of actions be termed skills? A natural restriction to consider is that all skills must have an objective. This immediately rules out the vast majority of action sequences, allowing us to focus on the ones that accomplish something.

Are skills then fundamentally just sequences of actions? Consider the case of catching a ball. We could use vastly different action sequences to accomplish this objective, depending on situational variables such as the shape and flight of the ball, any obstructions that block our limbs, and how sure our footing is. Despite this, we recognise that all of them are in essence the same skill because they share a common objective. The idea that skills are defined by their objectives is central to the approaches developed in this thesis.

It is also worth questioning whether skills are useful things to discover at all. If we are mainly going to use these skills to learn to execute larger tasks, do we gain anything by having our agent learn these tasks in terms of their component skills rather than the low level actions themselves, especially given that this necessitates learning a mapping from skills to low level actions in the first place? There are a number of benefits to this approach:

- The required behaviour can be expressed using significantly shorter se-

quences of skills than primitive actions, potentially making it much easier to learn. A potential difficulty here is that the presence of a large skill library can make it difficult to select the useful skills from this set, causing a combinatorial explosion of skill combinations. This is known as the utility problem, and is an open question.

- Although learning the skills themselves can be expensive, the cost is mitigated by the ability to reuse skills as components of multiple tasks rather than relearning them.

- Skills can also be used to facilitate communication, as a human can specify desired behaviour in terms of high level skills rather than low level actions.

1.2 Acquiring skills

The use of skills allows for more efficient hierarchical methods of learning to accomplish tasks. However, current research in this area is not sufficiently advanced to allow artificial agents to acquire reusable skills on their own with sufficient robustness and generality.

Methods exist which allow us to learn to accomplish objectives using low level actions, and most are in the domain of reinforcement learning (RL). In the reinforcement learning paradigm, objectives are defined through the use of reward functions, which specify the conditions under which an agent receives a

reward. The agent then learns a policy, or behaviour, to maximize its expected long term reward. If the reward function is crafted to reward achieving a particular objective, this maximization requires the agent to learn to accomplish that objective.

RL agents learn by selecting actions and observing the results of executing them. This trial and error approach yields good results in many cases, but often requires a large number of observations before the generated policies perform well. This results in RL being infeasible for many tasks [Taylor and Stone 2009]. Research has shown that the use of hierarchical skills can dramatically speed up the process, and there are many competing frameworks including Options [Sutton *et al.* 1999] and MaxQ [Dietterich 2000] which allow an agent to use pre-learned skills.

The problem then becomes one of learning these skills. As discussed earlier, we take the approach that skills are characterised by their objectives. We thus aim to discover skill objectives that are reusable across multiple tasks, after which we can learn policies that accomplish these tasks using standard RL techniques.

Defining reward functions which characterise skill objectives is a complex task for an expert, as modifying a reward function can result in unexpected changes to the behaviour of an RL agent. In this context, the complexity is compounded by the fact that we would like to specify reward functions which encode skills rather than entire tasks. An expert would thus need to identify all necessary skills and provide a reward function for each one, which is unrealistic.

1.3 Learning from demonstration

Learning from demonstration (LfD) reduces the burden on an expert by allowing a robot to learn to perform a task purely by observing it demonstrated. This allows the required behaviour to be specified by a user with no knowledge of programming or robotics. It is often extremely difficult for even a skilled programmer to program complex behaviour such as walking, making demonstration a more attractive option.

LfD methods fall into two broad categories based on the approach taken to mimic the demonstrator. Methods either learn a policy which maps actions to states in a way that is consistent with the demonstrator, or attempt to learn a task description capturing the goal of the demonstrator. For the latter category, it is common to learn an objective or reward function, which we have identified as the defining characteristic of a skill. A further significant advantage of this approach, known as inverse reinforcement learning (IRL), is that a reward function is robust to changes in the environment and agent configuration, making it suitable for representing transferable skills.

LfD algorithms typically search for a policy or reward that explains the entirety of a demonstration, making the assumption that demonstrators are performing a single monolithic task. This is problematic for two reasons. Firstly, it imposes the unnatural constraint that the demonstrator must repetitively demonstrate a single skill with a defined start and end. Secondly, the discovery of complex global reward functions is suboptimal in the context of transfer, as a complex

task may be composed of skills that can be reused in other tasks. It is thus desirable for an algorithm to discover multiple skill reward functions from a single set of demonstrations.

1.4 Aim and Contributions

Until recently, LfD was not well suited to skill discovery because any demonstration produced a single skill. A more natural way to accomplish demonstration is to have the demonstrator demonstrate the skill in context, as part of a larger task in which the skill is employed. For example, if we wanted to demonstrate all of the distinct motor skills involved in baking a cake, it makes more sense to have the demonstrator bake a cake than it does to demonstrate pouring repetitively, followed by cracking an egg into a mixture repetitively, and so forth. The aim of this research is to recover all of these multiple skills from unstructured demonstrations in which the component skills are not annotated.

A major difficulty with this approach is figuring out which parts of a demonstration correspond to which underlying skills. Without any information about the skill objectives involved , skill switching dynamics, or even the number of skills contained in the demonstrations, it is difficult to differentiate skills from each other. For instance, it would be possible to end up with a single skill that encodes cracking an egg into a bowl followed by stirring the mixture rather than separating these tasks correctly. Correctly segmenting expert demonstrations according to their component skills is the focus of this research.

This thesis makes the following contributions:

- *Nonparametric Bayesian Reward Segmentation* – We introduce Nonparametric Bayesian Reward Segmentation (NPBRS), a novel method for segmenting expert demonstrations according to their underlying reward functions. The method needs no prior knowledge of the skills inherent in the demonstration or the timing of the expert's switches between them. It produces a set of reward functions and policies, with one produced for each skill in the demonstration. In addition, it produces a mapping which indicates which inferred skill the demonstrator was performing at every point in the demonstration. Chapter 3 describes the model and the sampling algorithm, and empirically demonstrates its ability to segment demonstrations in a number of domains.

- *Model Learning* – Most IRL methods require a transition model. This requirement severely limits their usefulness in robotic domains, as we commonly do not have access to a transition model. We integrate a model learning step into NPBRS which learns the transition model from the same samples later used to infer the skills. Because the input to NPBRS is a set of input trajectories encompassing multiple skill and tasks, model learning works better than it traditionally does as the presence of multiple skills leads to better coverage of the state space. In Chapter 4, we demonstrate that it is possible to learn a sufficiently good transition model from the skill demonstration trajectories, presenting results in both discrete and continuous domains.

- *Implementation using a robot* – Using these methods on a robot is challenging for a number of reasons. Model learning is often more difficult in robotic domains as states and actions are typically noisy. State spaces are typically high-dimensional and continuous, and sampling is expensive. In Chapter 5, we validate that the methods presented in this thesis are suitable for these domains by testing this on CHAMP, a mobile manipulation platform equipped with a 7-DOF arm. The algorithm successfully segments demonstrations of various drinks being mixed, producing skills which made subsequent demonstrations easier.

In the next chapter, we present background material that explains the approach followed and methods used, and summarises related research.

Chapter 2

Background

This chapter provides a survey of background material in the field of skill discovery. In Section 2.1, we discuss skill acquisition in human beings, providing an understanding of the techniques and processes children use to develop skills, and motivating some of the decisions made in the rest of the thesis. This provides context for the survey of methods used to model this process, providing an introduction to reinforcement learning to model goal-directed behaviour (Section 2.2), inverse reinforcement learning to accomplish learning from demonstration (Section 2.3), time-series analysis to model skill interactions (Section 2.4) and nonparametric Bayesian models to encourage reusable skills (Section 2.5).

2.1 Skill Development

This section presents a selection of results from cognitive development research, which focuses on the development of a child's ability to think and understand. In particular we will focus on the development of motor skills, as these are the most relevant to robotics and provide an insight into the skill discovery process in humans. The structure it is presented in is slightly unnatural, as it is chosen to facilitate a mapping to the techniques we will use to model this process in artificial agents, rather than to provide the clearest possible explanation of the field of cognitive development itself.

2.1.1 Goal directed behaviour

Piaget's theory of cognitive development [Piaget and Cook 1952] helps explain the development of skills in infants. Before the age of two, an infant focuses heavily on sensorimotor development. As a newborn, the child behaves reflexively, without any high level planning. However, they will occasionally randomly stumble upon basic actions that feel good or accomplish some objective, for example an infant might accidentally stick its hand in its mouth and find the sensation soothing. At this point, they will start to engage in basic goal directed activity by repeating these actions. This early period of development is characterised by a mix of seemingly random movements which the baby uses to explore the possible actions and their consequences, and frequent repeated actions where the baby exploits actions already known to be desirable.

2.1.2 Sequential Composition

In the next phase of development, the infant learns that it can combine skills it has discovered to accomplish different goals. For instance, it might realise that picking up and dropping an object makes a pleasing sound, and it will then be able to repeatedly perform this combination to achieve a single goal. According to Claxton *et al.* [2003], ten month old infants are able to plan two stage tasks ahead of time, with experiments showing that infants compose the combined action prior to beginning the first component action. By 18 months, infants are capable of long skill sequences, as evidenced in their ability to build tall towers of blocks [Keen 2011].

A taxonomy for this type of planning behaviour is presented by De Lisi [1987], describing four planning types.

- Type 1 plans are feedback based. These are goal directed sequences of actions geared toward a particular purpose. A good illustrative example from McCarty *et al.* [1999] is a child bringing food to its mouth. The plan is fundamentally "Manipulate spoon until food is in mouth". This sort of plan is purely defined by a goal and the child has no strategy or intermediate goals.

- Type 2 plans are also known as "plans of action", wherein the child deliberately forms sequences of actions which lead to the goal. The child constructs the plan immediately before executing it, with the plan specialised to the immediate context. There is a capacity for further plan-

ning during execution of the task, as there could be a need to backtrack. To continue the infant feeding example, during execution of this type of plan, an infant may discover that its grip on the spoon is incorrect, requiring a correction.

- Type 3 plans are fully formed plans, which are more general and become habitual solutions. This type of plan does not need to be reconstructed for each task instance. When the child needs to bring a spoon to its mouth, the plan already dictates what grip the child needs to have, and sequences the gripping action and transport to the mouth. These types of plans are able to deal with different contexts as they encompass a representation of the future states of the environment. At this level, the child builds up a library of plans and is able to invoke an appropriate plan as needed.

- Type 4 plans occur once a person is aware of the usefulness of planning. These types of plans encode solutions to hypothetical problems, where the plan may never be executed. An identifying feature here is that the problem may never have been observed, but the planner has anticipated that it may. An example here is making a will or planning for a natural disaster.

The ability to plan, to construct sequences of learned actions, is fundamental to our success at problem solving tasks.

2.1.3 Reusability

Type 3 plans are stored and reused, with children iteratively refining their plans to be applicable in more contexts. This process of adaptation has been characterised by Piaget and Cook [1952] as being made up of two related processes known as assimilation and accommodation.

Assimilation refers to the process of adapting existing cognitive structures to incorporate new information or stimuli. A commonly used example of this is a child's understanding of what a dog is [Bartolotta and Shulman 2013]. Initially, having observed only a single dog, the child will have an incorrectly specific perception of what a dog is. On encountering different types of dogs, the child will assimilate these by modifying the definition of a dog to encompass these. This assimilation process will typically result in more general definitions.

Accommodation results from the inability to fit some new information into existing cognitive structures, requiring the creation of entirely new internal representations. Continuing the example, if a child encounters a different animal, perhaps a donkey, it might attempt to assimilate this into the definition of a dog. This assimilation fails as it becomes clear that this animal is too different to be integrated, and the child creates a new representation for a donkey.

This process of adaptation results in the generalisation and reuse of existing skills where possible, and the creation of new ones where appropriate.

2.1.4 Learning from demonstration

While it is clear that children learn by imitating others, an interesting question that arises is: what do children imitate? Experiments indicate multiple levels of learning from demonstration which develop as the child ages [Want and Harris 2001].

Initially, children directly copy the demonstrator's actions, with no grasp of causal relationships or intentions, in a process called "mimicry". This is followed by a second level called "imitation". This process sees the child copying the actions of the demonstrator with an understanding of the intended goal. One way in which this phenomenon has been observed is that children will leave out parts of the demonstration that are not relevant to the inferred goal. The next level is known as "emulation", wherein the child copies the goal of the demonstrator, potentially using entirely different actions to achieve the same objective. It has also been observed that children are able to infer the goal of a demonstrator from failed demonstrations, and are able to use this information to construct successful plans even though they have never observed the demonstrator successfully complete the demonstration [Meltzoff 1995].

The understanding that what humans eventually imitate is the goal or intention of a demonstrator rather than the actions themselves is significant in the context of artificial agents, and is discussed further in Section 2.3.

2.1.5 Conclusion

From these studies of human skill development, we note a number of characteristics. Skills are goal directed, able to be sequenced, and reusable. We also note that they are often learned from demonstration, and that a desirable thing to learn from demonstration is the goal or intention of the demonstrator, although it may also be useful to learn the actions that result in the goal. In the next sections we present techniques used to model skills, indicating in italics the aspect each technique is used to model.

2.2 Reinforcement Learning

Goal Directed Behaviour

Reinforcement learning is a branch of machine learning wherein an agent must learn goal directed behaviour [Sutton and Barto 1998]. In this paradigm, actions taken by an agent result in changes to the state of the environment and can yield numerical rewards which encode the task goal. An agent must learn a policy for acting in its environment that maximizes the long term reward received.

The environment is most commonly formulated as a Markov Decision Process (MDP) [Puterman 1994], a control process wherein at each time step, the process is in a particular state, and the agent can choose an action from a set of available actions. This action will cause the environment to transition to a new state and give the agent a reward. Formally, an MDP is a tuple

$M = (S, A, T, R, \gamma)$, where :

- S is the set of states;
- A is the set of actions;
- T is the transition probability model, where $T(s, a, s')$ is the probability that taking action a from state s will result in a transition to state s';
- R is the reward function where $R(s, a, s')$ is the reward obtained by the agent for taking action a in state s, causing a transition to state s'. Note that because T is stochastic, an action taken by an agent in a particular state need not result in the same successor state each time; and
- γ is the discount rate, with $0 < \gamma <= 1$, indicating how strongly the agent prefers rewards to occur sooner rather than later.

The reinforcement learning agent acts according to a policy $\pi(s, a)$ which defines the probability of selecting action a in state s. The cumulative discounted reward that it receives is known as the return, defined as:

$$R^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^{t} r_{t}, \tag{2.1}$$

where r_t is the reward received from the environment at time t when acting according to policy π from state s. The goal of the agent is to learn the optimal policy π^* that achieves the maximum expected return.

This is often achieved by keeping track of the expected return from every state

s when executing a policy π, known as the value function $V^{\pi}(s)$:

$$\begin{aligned} V^{\pi}(s) &= E[R^{\pi}(s)] \\ &= \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')]. \end{aligned} \tag{2.2}$$

Equation (2.2) provides some insight into the nature of the value function, showing that the value of a state is the sum of the immediate reward received for transitioning from that state and the value of the successor state the agent transitions to. Since the agent and environment are stochastic, we calculate an expectation over all possible actions and successor states.

Once we have a value function, policy iteration [Howard 1960] is used to update the policy to maximise the value as follows:

$$\pi(s,a) = \begin{cases} 1, & \text{if } a = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')], \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

After updating its policy, the agent can then learn a new value function corresponding to the new policy using equation (2.2). Policy iteration entails repeatedly interleaving these two steps (learning the value function and updating the policy) until the policy converges.

In many real-world problems the agent does not have access to T, making it impossible to use equation (2.3) to update the policy from the value function. In these cases we can instead learn an action-value function $Q^{\pi}(s,a)$, representing the expected return from executing action a in state s and thereafter

following policy π. The policy is then similarly updated from Q to select the action a which maximises $Q(s,a)$ at each state s.

Continuous State Spaces

A reinforcement learning agent selects actions on the basis of the values of the states or state-action pairs available to it. When the states are discrete, it is possible to represent V or Q using a table. However, many real world problems involve continuous state spaces, which results in two major problems: there are uncountably infinitely many states, which means that we cannot store their values individually, and the agent is unlikely to encounter any state multiple times since there are an infinite number of them. The agent is thus required to generalise its value estimates to states it has never encountered before.

This problem is overcome through the use of function approximation, which is a technique whereby the values or action values of the states are compactly approximated. This allows the agent to keep estimates of the values of the entire infinite state space, and at the same time generalize to estimate the values of unseen states. The most common approximation is linear function approximation [Tsitsiklis and Van Roy 1997], which approximates V^{π} by representing it as the weighted sum of a vector ϕ of basis functions as shown below:

$$\hat{V}^{\pi}(s) = \mathbf{w}.\phi(s) = \sum_{i=1}^{n} w_i \phi_i(s),$$

where ϕ_i is the ith basis function. The weight vector $\mathbf{w}$ is learned in order to approximate V^π.

2.3 Inverse Reinforcement Learning

Learning from demonstration

In a traditional reinforcement learning task, the reward function is specified, and the agent must find a policy which maximises it. In the inverse reinforcement learning (IRL) setting [Abbeel and Ng 2004], the agent does not have access to the reward signal, and must infer it from demonstration trajectories provided by an expert. This ties in well with the process of emulation discussed in Section 2.1.4 as the agent must become aware of the goal of the demonstrator, and can use entirely different actions to achieve the same goal.

IRL algorithms take as input an MDP/R (an MDP in which the reward function is not specified), and a set of expert trajectories D. Each trajectory $D^i = \{(s_1^i, a_1^i), (s_2^i, a_2^i), ...\}$ represents a sequence of state-action pairs produced by the expert. The goal is to find a reward function R with optimal policy π^* that is consistent with the trajectories observed in D [Zhifei and Joo 2012].

This formulation is troublesome, in that the problem is ill-posed. There are many possible reward functions for which the observed trajectories are optimal, making the recovery of the expert's reward function impossible. In fact, a reward function in which every move is awarded a reward of 0 makes every possible trajectory optimal, leading to a trivial degenerate solution.

Various IRL methods propose different ways of dealing with this issue. In the original apprenticeship learning algorithm [Abbeel and Ng 2004], two additional criteria are added to identify good choices of R:

- The reward function should maximize the margin between the best solution and the second best solution. This penalizes uninformative reward functions such as the 0 reward since all solutions to the corresponding MDP are equally good.
- Introduce a bias towards simple rewards. This selects reward functions which have small rewards in most states, and large rewards in a few states.

Maximum Entropy IRL [Ziebart *et al.* 2008] instead resolves the dilemma by employing the principle of maximum entropy, which attempts to give the least biased estimate of the reward distribution that matches the given trajectories, meaning that the reward function should not lead to any preferences that are not present in the demonstrations.

Bayesian IRL [Ramachandran and Amir 2007] generalises these approaches by modelling the reward bias as a prior and the expert behaviour as evidence that is used to update the prior.

In continuous state spaces, the reward function is approximated using a weighted

linear combination of basis functions as follows:

$$R(s) = \sum_i w_i . \phi_i(s). \tag{2.4}$$

A number of IRL algorithms have achieved good results in domains such as learning helicopter manoeuvres [Abbeel *et al.* 2010], parking styles [Abbeel *et al.* 2008] and pedestrian behaviours [Chung and Huang 2010].

2.3.1 Hierarchical Reinforcement Learning

The MDP formulation commonly used to express reinforcement learning problems has a single reward function encoding a single goal, and traditional reinforcement learning algorithms can be used to learn an optimal policy for that goal. However, we have noted in Section 2.1.2 that humans view individual skills as goal-directed behaviours for policies can be learned, and we can then compose these skill policies to solve larger problems. This hierarchical approach is key to solving complex real-world problems.

Sutton *et al.* [1999] introduced the options framework, which integrates temporally extended actions into the reinforcement learning framework. In addition to primitive actions which take one time step to execute, the agent is given access to options, which are actions that execute over multiple time steps, analogous to skills.

Formally, an option is a tuple $O = (I_o(s), \pi_o(s,a), \beta_o(s))$, where

- $I_o(s)$ is the initiation set indicator, which is 1 if the option can be initiated in state s and 0 otherwise;
- $\pi_o(s,a)$ is the policy under which actions are selected during execution of the option; and
- $\beta_o(s)$ is the termination condition, giving the probability that the option will terminate in state s.

A reinforcement learning problem in which options are used can be modelled as a Semi-Markov Decision Process in which the agent can choose from the available primitive actions as well as the options for which the current state is in the initiation set. The agent then learns the option-value function $Q(s,o)$ in the same way it does for primitive actions.

Importantly, we can use reinforcement learning to learn the option policy if we instead specify an option reward function R_o. This allows us to define these skills by their objectives – what must be done rather than how to do it. This is an important property as it allows skills to be transferred between agents with different capabilities. While the agents will have to learn different option policies based on their capabilities and environments, they will have the same goals.

It has been demonstrated that access to good options can reduce the time taken to learn good policies [Sutton *et al.* 1999].

2.4 Time Series Analysis

Sequential Composition

The options framework is a useful formulation for planning using skills. An agent employing options can learn to behave optimally according to a particular reward function by sequencing various skills, each of which can be represented by its own reward function.

Trajectories recording the agent's behaviour can thus be generated by a sequence of switches between the agent's skills. These skill switching dynamics are not directly observable, but because each skill implies a different action selection probability, their effects can be observed in the trajectories. We would also expect that some skill switches would be more likely than others. For example, it is more likely that after kicking a ball we would choose to run rather than recite the periodic table of elements.

More generally, we have a sequence of probabilistic transitions between unobservable hidden variables z_i (the skills), each of which has a corresponding distribution from which our observations y_i are drawn. A common way of modelling this type of data is a Hidden Markov Model (HMM), shown in Figure 2.1.

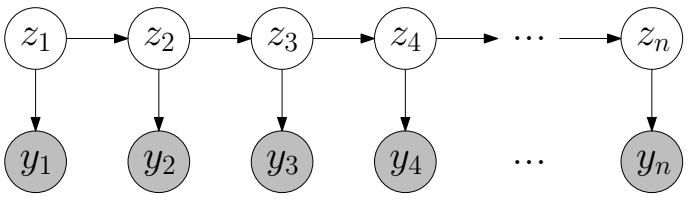


Figure 2.1: Graphical representation of a Hidden Markov Model.

In this model, the observations y_i are assumed to be generated by the latent variables z_i, generally referred to as modes, which represent skills in our context. The connections between the z variables in this system capture the dependencies between the skills, indicating that the probability of a particular skill being employed at time t is dependent on the skill employed at time $t-1$, but not on the entire history of skills employed previously. This is known as the Markov property.

The HMM is characterized by its transition probability matrix δ[1] and emission distribution $F(\cdot)$. The probability matrix δ gives the distribution of successor modes given the current mode. Each $\delta_{z\,t-1}$ provides the distribution of states at time t, i.e $p(z_t|z_{t-1})$. $F(\cdot)$ is the emission distribution $p(y_t|z_t)$, defining the probability of observation y_t given latent mode z_t. The generative model for an HMM is shown below:

$$z_t \sim \delta_{z_{t-1}}$$

$$y_t \sim F(\theta_{z_t}).$$

In the context of skill discovery, $F(\theta_{z_t})$ defines the probability of observing the state-action pair (s_t, a_t) given that we are employing skill z_t. The problem here is that the observation is dependent not only on the skill, but also on the observed state the agent found itself in at time $t-1$. This state action pair (s_{t-1}, a_{t-1}) is necessary to determine state s_t, and hence the optimal action chosen from the option policy as shown in equation (2.5).

$$a_t \sim \pi_{z_t}(s_t). \tag{2.5}$$

[1]Traditionally, π is used to refer to the transition probability matrix. We have departed from this notation to avoid confusion with the reinforcement learning policy π.

We thus need to introduce dependencies between successive observations, leading to a first order autoregressive Hidden Markov Model (AR-HMM), depicted in Figure 2.2.

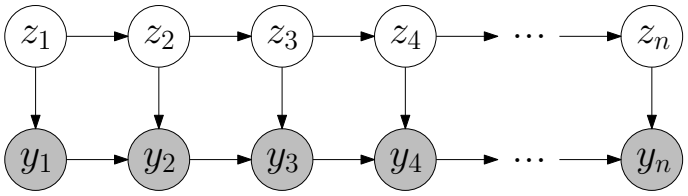


Figure 2.2: Graphical representation of a first order autoregressive Hidden Markov Model.

One significant weakness of this model in the context of skill discovery is that the number of modes K occurring in the demonstrations must be prespecified. This is problematic as it is difficult to imagine a scenario in which we are unaware of the nature of the skills inherent in a demonstration while simultaneously knowing the number of skills present. This restriction can be removed by applying a Bayesian nonparametric treatment to the model.

2.5 Bayesian Nonparametrics

Reusability

One of the key decisions when modelling skills is determining the complexity of the model. In the context of skill discovery, the model complexity refers to the number of skills that should be inferred, or K in the discussion of Hidden Markov Models in Section 2.4. If K is underestimated, the HMM will be forced to incorrectly combine some skills, leading to a less accurate model. However, a

small K is also necessary to avoid overfitting the data, as it forces the discovery of more general skills which can accurately model more of the data. A large K would allow the model to create skills to represent small, possibly insignificant variations in behaviour, inhibiting generalisation.

Bayesian nonparametric methods allow the model complexity to be inferred from the data, as they allow the model size to grow as necessary, bounded only by the data size. In case of a Hidden Markov Model of unknown size, we use Bayesian nonparametric methods to model the potentially infinite transition distribution between skills.

A problem with allowing the number of skills to grow to best represent the expert's trajectories is that there is a degenerate solution available to us. If we use a different skill to represent the behaviour at each time step, we can easily find a trivial reward function that results in the action selected at that time step, and produce a model that is consistent with the demonstrator's behaviour while being completely uninformative. We can overcome this problem by biasing the skill assignment to favour reusable skills. In the sections that follow we introduce Bayesian nonparametric models in the specific context of modelling switches between skills characterised by reward functions. Good general tutorials on nonparametric methods can be found in the work of Niekum [2015] and Gershman and Blei [2012]

2.5.1 The Chinese Restaurant Process / Dirichlet Process

In order to model the transition distribution from a particular skill, we use the Chinese Restaurant Process (CRP), a stochastic process that models partitioning with a potentially infinite set of partitions.

The CRP analogy defines a procedure for seating customers at a Chinese restaurant which contains an infinite number of tables, each of which can seat an infinite number of people. Customers enter the restaurant at a rate of one per time step and must choose a table to sit at. In this analogy, customers represent data points and the tables represent the next skill selection.

The process begins by seating the first customer at the first table. Subsequent choices are parametrized by a concentration parameter α, which controls the tendency of customers to prefer unoccupied tables. Customer i will choose to sit at the first unoccupied table with probability $\frac{\alpha}{i-1+\alpha}$, and an occupied table with probability $\frac{c}{i-1+\alpha}$ where c is the number of occupants at the table. Note that any occupied table will be selected with a probability proportional to the number of occupants it already has, causing a clustering effect. This encodes the idea that we are more likely to transition to skills we have transitioned to before.

The representation has a potentially infinite number of tables as there is always a non-zero probability that an arriving customer will choose to sit at an unoccupied table. However, note that the probability of sitting at an unoccupied table decreases with i, making it increasingly likely that an existing table will

be reused as the number of customers increases.

Chinese Restaurant Process Mixtures are constructed in the same way, but each table has a dish associated with it, selected when the first customer sits at a table. The dish represents a parameter θ, which parametrizes an emission distribution, from which data points are drawn. Whenever a customer selects a table j, a data point is drawn from the distribution $F(\theta_j)$.

Each draw from the original CRP selects a distribution from which data points are drawn, making the CRP mixture a distribution over distributions. This model is also known as the Dirichlet Process Mixture Model and its generative model is as follows:

$$G|\alpha, G_0 \sim DP(\alpha, G_0)$$

$$\theta_j|G \sim G$$

$$x_i|\theta_j \sim F(\theta_j).$$

In this formulation, the Dirichlet Process (DP) is parametrized by a base distribution G_0 which defines a distribution over possible θ values (in our case skill reward functions), as well as the concentration parameter α. G, drawn from this Dirichlet process, provides a probability distribution over discrete atoms, or individual skills from the infinite set of possible skills, defining the probability of transitioning to each skill. The data points x_j are then drawn from the emission distributions $F(\theta_j)$ of the assigned θ_j. In our context, the θ_j values are reward functions from which the trajectories are generated.

2.5.2 The Chinese Restaurant Franchise / Hierarchical Dirichlet Process

The CRP is a good way to model transitions from a particular skill. However, we have to model transitions from multiple skills, as each skill will have its own transition distribution. We could model the transition probabilities from every skill using the same model. The problem with this approach is that the next skill would always be drawn independently from the same distribution G. This is unrealistic as the next skill is likely to be correlated with the current one. We could instead model each skill using a separate CRP. However, recall that G is drawn from the infinite set of possible skills, meaning that two draws of G have 0 probability of selecting the same skills.

We would expect the transition probabilities encountered to be skill specific, which means that we would need a separate transition distribution for each skill, while still retaining global characteristics such as the set of skills that we can transition to, and the general popularity of particular skills. In order to model this situation, we use the Chinese Restaurant Franchise (CRF).

The CRF is similar to the CRP, but introduces a franchise which is made up of multiple restaurants which share a menu. Each customer is at a particular restaurant, corresponding to the current skill. The process for assigning customers to a table (or next skill) is the same as the process in the CRP, with customers choosing to sit at an occupied table with a probability proportional to the number of occupants the table has, and the probability of selecting an

empty table governed by the concentration parameter α.

The formulation differs in the way dishes (skill parameters or θ values) are selected, in that the dish selection is also influenced by a concentration parameter γ. When a new table is opened, a dish that has previously been selected at any restaurant in the franchise is selected with a probability proportional to the number of times it has been globally selected. A new dish is selected with a probability proportional to γ. This is equivalent to the Hierarchical Dirichlet Process (HDP), whose generative model is presented below.

$$
\begin{aligned}
G_0|\gamma, H &\sim DP(\gamma, H) \\
G_j|\alpha, G_0 &\sim DP(\alpha, G_0) \\
\theta_{ji}|G_j &\sim G_j \\
x_{ji}|\theta_{ji} &\sim F(\theta_{ji}).
\end{aligned}
$$

When the Hierarchical Dirichlet Process is applied to time series analysis to provide a prior distribution on the transition distribution, the resulting model is known as an HDP-HMM [Teh *et al.* 2005]. The HDP allows the representation to encompass a potentially infinite set of latent states, removing the need to specify the model complexity. The model is amenable to Bayesian inference, and has been successfully used in a variety of applications including scene recognition [Kivinen *et al.* 2007], music synthesis [Hoffman *et al.* 2008] and gene expression modelling [Beal and Krishnamurthy 2012].

2.5.3 Indian Buffet Process

One weakness of the HDP-HMM formulation is that the model does not differentiate between the input trajectories. Each trajectory is assumed to contain the same skills and transition dynamics. This assumption is not appropriate in the context of skill discovery, as the trajectories could be demonstrations of different tasks. While we would assume that the tasks do share skills, it is unlikely that all tasks would use the same skills and have the same skill transition dynamics. The model which allows us this flexibility is the Indian Buffet Process (IBP).

To apply the IBP to skill discovery, we use the analogy of trajectories as customers. In the IBP, the first customer enters the restaurant and selects Poisson(α) dishes. Each dish represents a skill, and the selection implies that the skills are present in the first trajectory. The ith customer selects previously selected dishes based on their popularity, selecting dish k with probability $\frac{m_k}{i}$, where m_k is the number of times the dish has been selected before. The customer then selects Poisson($\frac{\alpha}{i}$) new dishes. This is equivalent to the Beta process and provides a featural representation of skills, with each trajectory having a potentially infinite binary vector representing the presence of each skill in that trajectory. This facilitates the sharing of skills while not requiring every trajectory to use every skill.

2.5.4 The Beta Process Autoregressive Hidden Markov Model

The Beta process allows us to assign skills to trajectories. However, we need to assign skills to individual time steps within the trajectories. We would still like skill transitions to be biased towards frequently selected skills. The Beta Process Autoregressive Hidden Markov Model (BP-AR-HMM) [Fox 2009] has the properties that we need. The generative model for the BP-AR-HMM is shown below:

$$
\begin{aligned}
B|B_0 &\sim \mathrm{BP}(1, \beta_0) \\
X^{(i)}|B &\sim \mathrm{BeP}(B) \\
\delta_j^{(i)}|\boldsymbol{f}_i, \gamma, \kappa &\sim \mathrm{Dir}([\gamma, ..., \gamma+\kappa, \gamma, ...] \otimes \boldsymbol{f}_i) \\
z_t^{(i)} &\sim \delta^{(i)}_{z_{t-1}^{(i)}} \\
\mathrm{y}_t^{(i)} &= \sum_{j=1}^{r} A_{j, z_t^{(i)}} \mathrm{y}_{t-j}^{(i)} + \mathrm{e}_t^{(i)}(z_t^{(i)}).
\end{aligned}
$$

In this model, the Beta process is parametrized by a base measure β_0 containing the potentially infinite set of skill parameters. The draw B then selects a weight ω_k for each skill k, representing the global probability of the presence of skill. A subset of the skills are selected from a Bernoulli process parametrized by B, and for each trajectory i, a binary feature vector f_i is drawn, with f_{ik} ~Bernoulli(ω_k). The skills and their corresponding weights form the draw $X^{(i)}$.

Now that it is known which skills occur in trajectory i, the skill transition distribution $\delta_j^{(i)}$ is drawn from a Dirichlet distribution in which skills that do not occur have a 0 probability of being selected. The distribution also has a self

transition bias, indicating that if a skill is employed at time t, it has a greater probability of being selected at time $t+1$. The skill employed at each time step is drawn from the transition distribution of the skill employed at the previous time step.

Finally, because the BP-AR-HMM was developed to model switching linear dynamical systems, the observation y_t is computed from y_{t-1} using the linear transformation A corresponding to z_t and mode dependent noise e. In our case, z_t defines an option policy, and the observation will be generated from the action selected at time t as described in Section 3.3.

2.5.5 Inference

Generative models give us a clear description of how data would be generated from the model, drawing variables from their defined distributions and eventually drawing the data from the resulting distributions. However, when doing data analysis, we are given the data and would like to discover the model parameters and hidden variables that led to its generation. This process is known as inference, and is most commonly accomplished using Markov chain Monte Carlo (MCMC) sampling [Neal 1993].

MCMC methods construct a Markov chain for which the distribution which best explains the data is the equilibrium distribution. This functions as a directed search over the space of model parameters. While there are many MCMC algorithms, many follow the general procedure of beginning with a random set

of parameters, and then selecting moves with probability determined according to the relative likelihood of the resulting model. Two such algorithms, the Metropolis-Hastings algorithm [Hastings 1970] and Gibbs sampling [Geman and Geman 1984] are discussed here.

Algorithm 1 Metropolis Hastings Algorithm (adapted from Niekum [2013])

Given: Proposal distribution $g(\mathbf{y}|\mathbf{x})$ defining the distribution of next model parameters given current model parameters
Choose initial model parameters $\mathbf{x}^{(0)}$ arbitrarily
for each iteration i **do**
  Sample $\mathbf{y} \sim g(\mathbf{y}|\mathbf{x}^{(i-1)})$
  Set acceptance probability $A(\mathbf{y}, \mathbf{x}^{(i-1)}) = min\left(1, \frac{p(\mathbf{y})g(\mathbf{x}^{(i-1)}|\mathbf{y})}{p(\mathbf{x}^{(i-1)})g(\mathbf{y}|\mathbf{x}^{(i-1)})}\right)$
  Sample $u \sim$uniform(0,1)
  if $A(\mathbf{y}, \mathbf{x}^{(i-1)}) > u$ **then**
    $\mathbf{x}^{(i)} = \mathbf{y}$
  else
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)}$
  end if
end for

Algorithm 1 presents the Metropolis-Hastings algorithm, which begins with an arbitrary initial vector of parameters x_0. In practice, this choice is not entirely arbitrary as it affects the speed of convergence. The algorithm requires a proposal distribution $g(y|x)$ from which a proposed parameter set y can be drawn from the current set x. After the parameter is drawn, the likelihood of the proposed model is evaluated. If it is greater than that of the current model, the proposal is accepted. If it is smaller, the proposal is accepted with a probability based on the likelihood of the proposed parameters relative to the current ones.

Gibbs sampling, outlined in Algorithm 2 is a simpler procedure that works by

cycling through all model variables x_j, sampling each one in sequence while holding all the others constant. It is thus conditioned on $\mathbf{x}_{-\mathbf{j}}$, a vector of every model variable in $\mathbf{x}$ apart from x_j itself, and thus requires that we can sample from the conditionals $p(x_j|\mathbf{x}_{-j})$ for all i.

Algorithm 2 Gibbs Sampling (adapted from Niekum [2013])

Given: Conditionals $p(x_j|\mathbf{x}_{-\mathbf{j}})$ for all j
Choose initial model parameters $\mathbf{x}^{(0)}$ arbitrarily
for each iteration i **do**
  for each model parameter x_j **do**
    Sample $x_j^{(i)} \sim p(x_j|\mathbf{x}_{-j}^{(i-1)})$
  end for
end for

The BP-AR-HMM deals with the potentially infinitely sized set of model parameters by using Metropolis Hastings sampling for the set of available skills and Gibbs sampling to infer the rest of the model parameters given the skill distribution. The initial set β_0 has one skill. The proposals in this scheme are birth and death moves which increase or decrease the number of skills. This means that the sampler only has to work with a finite number of skills at any time. In practice the number of skills is limited by the data, as proposals are only accepted if they fit the data. In real applications, the clustering effects of the prior distributions create a bias towards smaller models.

2.6 Robot Learning from demonstration

In this work, we focus on the idea that what a robot should learn from a demonstrator is the set of goals the demonstrator is optimising for. It is important to note that this is not the only approach to LfD, or even the most common. In this section we present some approaches to the problem.

LfD in robots has been accomplished by learning a policy directly, instead of learning a reward function. This is a supervised learning problem wherein an agent with access to demonstration trajectories generalises the state and action pairs it observes to learn a mapping that is defined for states that were not observed in the demonstrations. Various supervised learning approaches have been employed to learn to perform particular tasks; for example autonomous driving using Gaussian Mixture Models [Chernova and Veloso 2007], airplane flight using decision trees [Sammut *et al.* 1992], and egg flipping using Hidden Markov Models and k-Nearest Neighbours [Pook and Ballard 1993]. Because these methods directly learn the experts mapping between states and actions, they cannot accommodate for differences in the capabilities of the demonstrator and the robot (known as the correspondence problem). Furthermore, these methods do not attempt to recover multiple policies.

Robots have successfully learned to mimic expert behaviour by recovering the expert's reward function using IRL. In Abbeel *et al.* [2007], an agent learns to perform aerobatic manoeuvres on a helicopter using the apprenticeship learning algorithm [Abbeel and Ng 2004]. IRL has subsequently been used to teach

robots to navigate parking lots [Abbeel *et al.* 2008], grasp unknown objects [Boularias *et al.* 2012] and play table tennis [Muelling *et al.* 2014]. While these behaviours were learned using methods that recover a single reward function, methods that can recover multiple reward functions have also shown promise.

A similar problem to the one addressed in this thesis is addressed in previous research [Babes *et al.* 2011; Dimitrakakis and Rothkopf 2011]. In this problem, the IRL method must recover multiple reward functions, with the restriction that each demonstrated trajectory is generated from a single reward function. The methods presented can discover good reward functions in this context. However, the restriction that each demonstration trajectory must be characterised by a single reward function limits its usefulness in skill discovery as the trajectories cannot be decomposed.

Recent work has addressed the problem of decomposing trajectories into their component skills [Konidaris *et al.* 2012; Michini *et al.* 2015; Hughes *et al.* 2012]. We compare our methods against these approaches, and thus present more detail on these methods in Section 3.2.

2.7 Conclusion

In this chapter, we presented research on cognitive development in humans to discover the properties of skills. Reinforcement learning was introduced as the setting within which we aim to discover skills. Inverse reinforcement learn-

ing was presented as the means by which we can recover reward functions or objectives from demonstrated trajectories. Hidden Markov Models were presented as a method for partitioning time series data according to some latent states, which we will be using to model skills. Next, nonparametric Bayesian inference was introduced to illustrate how we will allow the number of skills to be determined from the data in a Bayesian manner. In the following chapter, these concepts will be tied together into a method for segmenting trajectories according to the latent skills inherent in the demonstrations.

Chapter 3

Nonparametric Bayesian Reward Segmentation

3.1 Introduction

Our overall goal is to discover all the skills used by a demonstrator by observing its behaviour and segmenting that behaviour into component skills. Part of the difficulty in this process lies in determining where the skill boundaries lie. This is a segmentation process, and is complicated because must determine which skill the demonstrator is employing at every point in the a trajectory without knowledge of that skill's policy, initiation set, or termination condition.

In this chapter, we introduce nonparametric Bayesian reward segmentation (NPBRS), a method for discovering latent skills from unstructured demonstra-

tion trajectories. It models a skill as having a single reward function that generates its policy, and then the segments the trajectories by inferring both the hidden reward functions and the states at which the agent switches between distinct reward functions. Because we do not know how many skills are employed in a set of demonstration trajectories, we use a Bayesian nonparametric approach to segmentation, allowing the number of skills to be determined by the data. We also do not know that the trajectories have the same skill transition dynamics, as outlined in Section 2.5.3, as different task demonstrations may use different sequences of common skills. We use the Beta Process Hidden Markov Model (BP-HMM) [Fox *et al.* 2014] outlined in Section 2.5.3 as it provides a mechanism for dealing with this complication.

NPBRS is able to identify multiple skills in unstructured demonstrations, identifying skill boundaries and recovering the reward functions associated with them. We show that it extracts suitable skills given demonstration trajectories in two domains, the car driving domain and the quadcopter gate domain.

Note that this work also appears in Ranchod *et al.* [2015].

3.2 Related Work

A number of approaches have been used in the past to extract skills from unstructured demonstrations. Many of these model skills as behaviours which attempt to reach particular desirable states, while others model behaviour as

parametrized linear dynamical systems. These two broad categories are described in this section.

3.2.1 Subgoal State Methods

Subgoal state methods describe a skill by first finding its termination condition, and then discovering a method for reaching it. The Bayesian nonparametric inverse reinforcement learning algorithm [Michini *et al.* 2013; Michini 2013] is capable of learning multiple reward functions from unstructured demonstrations. The method has been extended with a number of optimizations making it suitable for large state spaces [Michini *et al.* 2015], but is restricted to reward functions representing the goal of reaching particular subgoal states. These reward functions return a positive reward in a single state, and zero elsewhere.

This restriction is also present in CST [Konidaris *et al.* 2012], which segments trajectories using changepoint detection. The value function is restricted to being linear in one of a given set of abstractions (which are collections of basis functions). Changepoints occur when an abstraction changes, or when a trajectory is too complex to represent using a single linear value function. The method produces options which can be chained together with the termination set of an option corresponding to the initiation set of its successors. The discovered options have the goal of reaching these common subgoal states.

3.2.2 Switching Dynamical Systems

Algorithms which model expert trajectories as a series of switches between dynamical systems have achieved good results in the context of human motion segmentation [Chiappa and Peters 2010; Niekum *et al.* 2012; Alvarez *et al.* 2010]. These have been successfully modelled using hidden Markov models, with each mode representing a dynamical system. Methods exist which can discover the most likely sequence of modes, but commonly require that either the number of modes be prespecified (which is not realistic for unstructured demonstrations) or do not allow for modes to be repeated or shared across trajectories.

The Beta Process Hidden Markov Model (BP-HMM) [Fox *et al.* 2014] outlined in Section 2.5.3 addresses these problems by placing a beta process prior on the mode sequence. This allows for potentially infinitely many modes, with each time series exhibiting a subset of the total modes. Each time series can then switch between the modes it exhibits. This representation also encourages the sharing of modes between trajectories. The appropriate number of modes can be inferred from the data instead of being prespecified. In the Beta Process Autoregressive Hidden Markov Model (BP-AR-HMM), emissions are produced from a mode specific vector autoregressive (VAR) process.

The generative model for the BP-AR-HMM is

$$
\begin{aligned}
B|B_0 &\sim \mathrm{BP}(1, \beta_0) \\
X_i|B &\sim \mathrm{BeP}(B) \\
\pi_j^{(i)}|\boldsymbol{f}_i, \gamma, \kappa &\sim \mathrm{Dir}([\gamma, ..., \gamma + \kappa, \gamma, ...] \otimes \boldsymbol{f}_i) \\
z_t^{(i)} &\sim \pi^{(i)}_{z_{t-1}^{(i)}} \\
\mathrm{y}_t^{(i)} &= \sum_{j=1}^{r} A_{j, z_t^{(i)}} \mathrm{y}_{t-j}^{(i)} + \mathrm{e}_t^{(i)}(z_t^{(i)}).
\end{aligned}
\tag{3.1}
$$

Equation (3.1) is discussed in greater detail in Section 2.5.3.

Inference is accomplished using a Markov Chain Monte Carlo (MCMC) algorithm using both Metropolis-Hastings proposals and Gibbs sampling [Hughes *et al.* 2012]. The sampler alternates between: sampling the feature vector and mode sequence given the data and mode specific HMM emission parameters; sampling the HMM emission parameters from a matrix normal inverse-Wishart distribution given the data and mode sequence; and proposing birth/death moves which introduce new features or remove existing ones from the global feature set.

Niekum *et al.* [2012] uses the BP-AR-HMM formulation to segment demonstration trajectories, and then uses Dynamic Movement Primitives to learn policies representing the skills present in each segment. The technique has been successfully used to segment trajectories on the basis of its latent policies. This work is the most similar to ours, but representing skills as policies rather than reward functions limits the ability to use the skills in agents with varying abil-

ities and in differing environments.

In the experiments presented in Section 3.4, we will compare the segmentations produced by our algorithm against those produced by subgoal state methods and by the BP-AR-HMM.

3.3 Nonparametric Bayesian Reward Segmentation

Our task is to extract repeated skills given a set of demonstration trajectories. Each trajectory may contain multiple skills, and common skills occur in multiple trajectories. We model these skills as reward functions, allowing us to capture more complex subgoals that cannot be represented as a single target state, as illustrated in Figure 3.1.

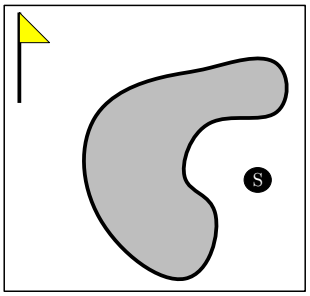


Figure 3.1: If the goal of the agent is to get from position S to the flag, we could represent the subgoal as a target state. However, a single target state cannot capture the situation where an agent must also avoid the swamp in the middle.

We introduce nonparametric Bayesian reward segmentation (NPBRS), a method for segmenting trajectories from an MDP according to latent reward functions. The method combines the BP-HMM and IRL so that the number of skills can be

determined by the data, and the reward function for each skill can be recovered.

The system is required to segment the demonstration trajectories into skills, with the important property that skills can be shared between trajectories, and that the number of skills is potentially infinite. We use the beta process and conjugate Bernoulli process in the same manner as the BP-HMM model, and draw features and mode sequences representing skills in the same way. However, we model the emissions as an MDP rather than as a VAR process, meaning that each mode (or skill) has an associated MDP.

Given the mode sequence $z_t^{(i)}$, we model the dynamics of the system as follows:

$$\begin{aligned} P(a)|z_t^{(i)} &= \frac{e^{\tau Q^{z_t^{(i)}}(y_{t-1}^{(i)},a)}}{\sum_a e^{\tau Q^{z_t^{(i)}}(y_{t-1}^{(i)},a)}} \\ a_t^{(i)} &\sim P(a)|z_t^{(i)} \\ y_t &\sim T(y_{t-1}, a_t^{(i)}). \end{aligned} \tag{3.2}$$

We thus replace the step wherein HMM emission parameters A and e are sampled. Instead, we use IRL to determine the skill reward function and accompanying policy, and calculate the transition probabilities from a combination of the policy and the known dynamics of the environment.

We use the birth and death reversible jump Markov Chain Monte Carlo sampler developed for BP-AR-HMM [Fox 2009]. The algorithm proposes skill birth and death moves and accepts or rejects them on the basis of the relative model likelihood. Skills can also be created using a split-merge procedure which in-

troduces new skills by either combining or splitting existing ones [Hughes *et al.* 2012]. The skill sequence is then block sampled using a variant of the forward-backward algorithm [Rabiner 1989].

Our algorithm then models the skill specific dynamics by learning the action-value function associated with each skill ($Q^{z_t^{(i)}}$ in equation (3.2)). We do this by grouping all subtrajectories by the skill assigned to them. We perform maximum entropy IRL on each skill, taking all of its subtrajectories as input paths, which yields a reward function for each skill. We then use value iteration to determine $Q^{z_t^{(i)}}$, allowing us to determine the likelihood of the demonstration trajectories.

It is worth noting here that the log likelihood of the trajectories D given policy π optimizing the reward function R is based purely on the action selection probabilities defined in the policy, without any reference to the transition probabilities $T(s_i, a_i, s_{i+1})$. It is calculated as $\log P(D|R) = \Sigma_i \Sigma_t \log \pi(a_t^{(i)}|s_t^{(i)})$. This makes it clear that the likelihood is not affected by the stochastic nature of T as the goal is to mimic the behaviour of the expert rather than the reaction of the environment.

This approach replaces the HMM emission model with an MDP, allowing us to use the segments to infer a policy to be used for control, rather than to recognise a change in dynamics. It does, however, incur a large cost as we must perform IRL multiple times at each iteration of the sampler.

The process for evaluating a birth proposal is outlined in Figure 3.2, giving a

high level view of the workings of the algorithm.

3.4 Experiments

We evaluate the effectiveness of segmentation based on extracted rewards by comparing against methods based on subgoal states and linear dynamical systems. In order to remove the effect of other design decisions, we keep the probabilistic model and inference procedure the same for all the methods, and vary the segmentation likelihood calculation to obtain algorithms based on reward functions, subgoal states and linear dynamical systems.

3.4.1 Trajectory Generation

For each of the two domains, we designed multiple reward functions representing skills. We then used each reward function to train an optimal agent using Q-Learning [Watkins and Dayan 1992]. In order to generate trajectories containing a mixture of skills, we switched between the action selections of the trained agents. This selection is biased to be "sticky", as a skill is likely to be employed for multiple sequential timesteps. We generated 16 trajectories for each domain, and recorded the true switching sequence for later comparison.

We used these trajectories as input to the segmentation methods. We ran the sampler 10 times for 40 minutes each, and selected the segmentation with the highest log likelihood.

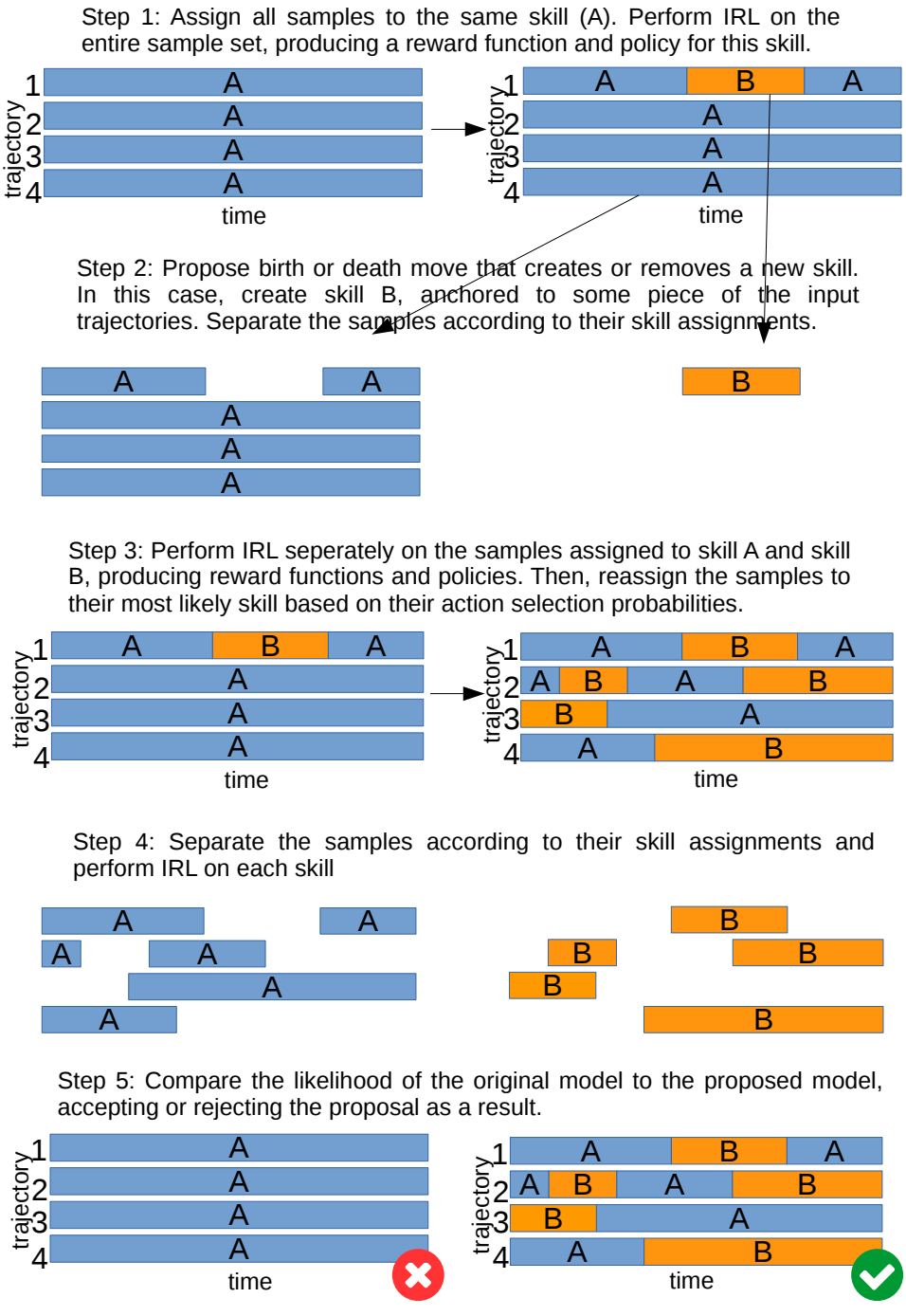


Figure 3.2: First iteration of the NPBRS algorithm

3.4.2 Driving Domain

For our first experiment, we use a modified version of the car driving domain, commonly used as an IRL benchmark problem [Abbeel and Ng 2004]. In this domain, the agent is driving on a three lane highway as shown in the screenshot in Figure 3.3, with each containing cars moving at a different speed.

The car controlled by the agent is moving faster than all other cars on the road and has three available actions: it can change lanes to the left, to the right, or stay in the current lane. The car is also able to drive on the verge to either side of the highway, in which there are no cars. Cars appear randomly in the distance, so it is possible for the highway to be completely blocked.

The state is described by 4 variables: the current lane, which includes the verge on either side, and 3 variables representing the distance to the closest car in each lane. If this distance is zero for the current lane, a collision has occurred. This distance was discretized to 6 distance values to simplify the learning process.

We designed three reward functions to encode different objectives:

- A: Hit as many cars as possible. The agent receives a reward of 500 for hitting a car, 0 otherwise.
- B: Drive in the right lane, but change lanes to avoid collisions. The agent receives a reward of 10 for being in the right lane, -500 for hitting a car and 0 otherwise.

- C: Drive in the left lane, but change lanes to avoid collisions. The agent receives a reward of 10 for being in the left lane, -500 for hitting a car and 0 otherwise.

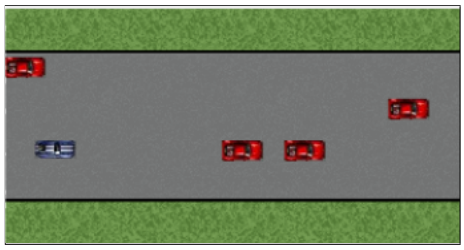


Figure 3.3: The car driving domain. The blue car is the learning agent which could be tasked with avoiding the red cars.

For this domain, we generated two sets of trajectories, of length 128 and 500, with a 2% chance of switching reward functions at each timestep. The choice of a 2% switching rate was made to cause the agent to display switching behaviour in both the short and long trajectories. The trajectory lengths of 128 and 500 are arbitrary and while we would not expect changes in the trajectory length to significantly change the outcome of the experiments, we chose different lengths as a preliminary test of this hypothesis.

Because this switching behaviour is generated randomly, the behaviour displayed by the trajectories in their entirety cannot be characterised by a single reward function. The methods must thus segment the demonstrated trajectories in order to recover the original reward functions A, B and C. They must do this without access to the original reward functions, and with no information about the reward switches.

These generated trajectories were segmented using the NPBRS, subgoal state

and linear dynamical systems (labelled LDS) samplers.

Note that the reward functions recovered by the method are unlikely to be superficially the same as the original reward functions, as the underlying IRL problem seeks only to mimic the behaviour of the expert, and this can be accomplished with many different reward functions, as discussion in Section 2.3. We thus use two characteristics to determine if the methods were successful. Firstly, if the skills recovered imply switches in the same places as the true skills, the recovered reward functions have allowed us to accurately perform segmentation, which is in itself a difficult task. Secondly, we measure the reward achieved by the recovered skills and inferred switching behaviour when reward is assigned according to the original reward function and switching behaviour. If the reward is high, this would indicate that the recovered skills induce behaviour that is equivalent to the original skills.

Figure 3.4 presents a typical result from this process. The skills produced are labelled based on their correspondence to skills in the true sequence. If a skill overlaps with a true skill for more than 50% of its occurrences, it is given the same letter. Thus, inferred skills B, B_1 and B_2 all received the label B because they mostly align with skill B in the true skill sequence across all trajectories. Skills which did not share more than 50% of their occurrences with any true skill are shown in grey. NPBRS produces segmentations very similar to the true mode sequence. It makes small errors at the boundaries between skills because the skills agree on their action selections in these small windows, making the skill boundary indistinguishable.

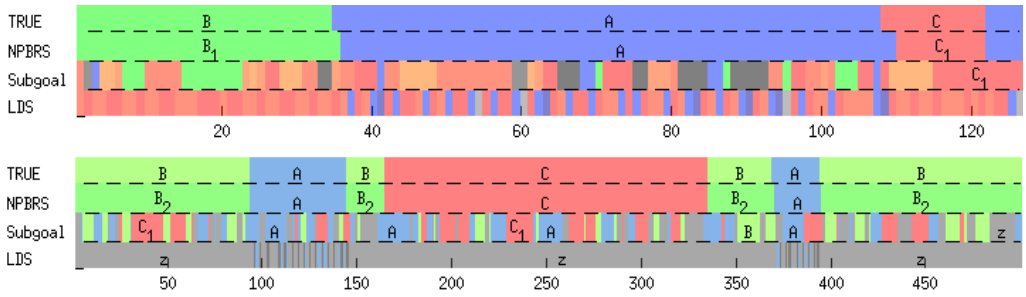


Figure 3.4: Segmentation for the car driving domain for trajectory lengths 128 (top) and 500 (bottom), showing using segmentation based on NPBRS, subgoal states, and linear dynamical systems.

The poor performance of the LDS sampler is expected as the method is not meant to model a system whose mode specific emissions are not a linear dynamical system. It is included here only to verify that different models are necessary in some domains, but does not indicate poor performance in more appropriate domains. Similarly, the poor performance of the subgoal state segmentation indicates that the domain cannot be modelled using subgoal state reward functions such as those used in Michini *et al.* [2015] and Konidaris *et al.* [2012]. This is an important result as it indicates that NPBRS can segment trajectories that existing methods cannot, but it should not be taken to mean that subgoal state methods fail in more appropriate domains.

	Expert	NPBRS	Subgoal	IRL
Reward	39093	37832	-5660	-14453
Skill Switches	7.13	9.2	45.11	0
Skills	3	7.8	3.4	1

Table 3.1: Results from NPBRS, Subgoal state and unsegmented IRL in the 500 length highway domain averaged over 10 runs. Rewards and skill switches are presented as an average per trajectory

In Table 3.1 we present aggregate results of using the inferred skills in the original trajectories according to the corresponding inferred segmentation. Since we are using these skills for control, we exclude LDS from the comparison, as the model is uncontrolled. We instead compare against unsegmented IRL. NPBRS achieves a much higher average reward than subgoal state segmentation or IRL, and is very close to that achieved by the original expert. It also switches skills at a similar frequency to the expert. It does, however, find 8 skills where the expert displayed 3, finding multiple skills corresponding to each expert skill. The subgoal state method has a negative average reward and switches skills frequently, indicating that it has not managed to find appropriate skills. Unsegmented IRL performs poorly in terms of reward received, meaning that it could not find a single reward function that explains the expert behaviour.

3.4.3 Quadcopter Gate Domain

In the quadcopter gate domain, a simulated quadcopter must make its way through an obstacle course. This was implemented using the TUM simulator [Engel *et al.* 2012], a 3D simulation of the AR.Drone quadcopter built in Gazebo [Koenig and Howard 2004] and integrated with ROS [Quigley *et al.* 2009].

The obstacles involved are square hoops placed in a corridor, as shown in Figure 3.5. The quadcopter is able to control its motion in the height and width dimensions using the actions up, down, left and right, and moves forward by

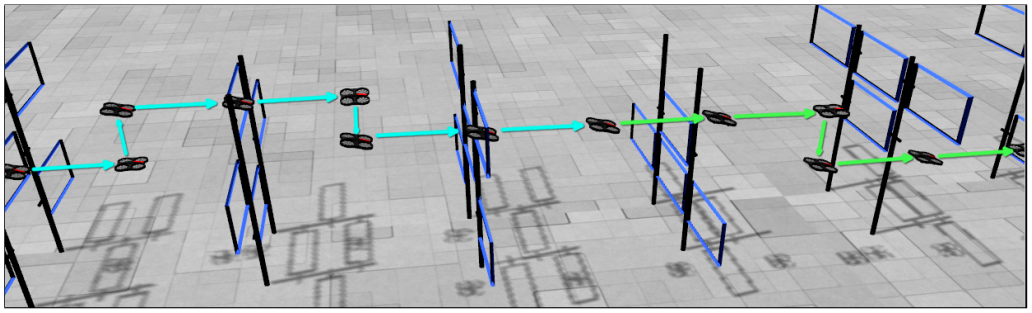


Figure 3.5: The quadcopter gate domain. The agent is initially flying through as many hoops as possible (cyan) but switches to avoiding the hoops (green).

a fixed amount at every timestep. The corridor is discretized into a $3 \times 3 \times 500$ grid, with each 3×3 slice having one of 6 hoop configurations placed in it. The available hoop configurations are shown in Figure 3.6.

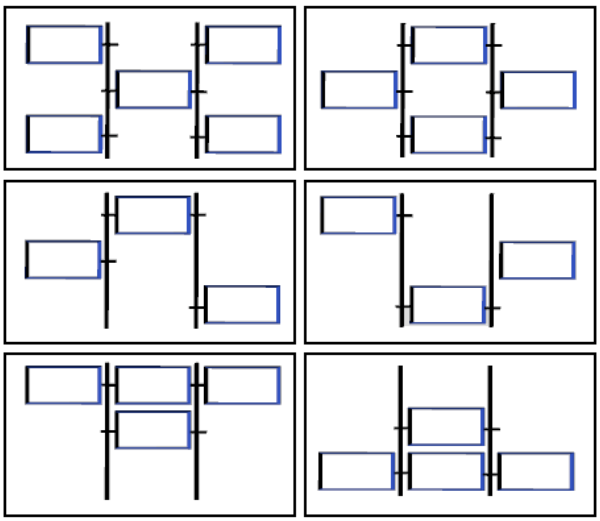


Figure 3.6: The hoop configurations available in the quadcopter gate domain.

The state is described by the coordinates of the quadcopter within the slice and the hoop configurations of the next four slices. We have designed three reward

functions in this domain:

- A: Fly through as many hoops as possible. The agent receives a reward of 500 for flying through a hoop and zero otherwise.
- B: Fly as close to the ceiling as possible, avoiding flying through hoops. The agent receives a reward of 10 for being in the top row, a reward of -500 for flying through a hoop, and zero otherwise.
- C: Fly as close to the ground as possible, avoiding flying through hoops. The agent receives a reward of 10 for being in the bottom row, a reward of -500 for flying through a hoop, and zero otherwise.

A typical segmentation for the quadcopter domain is shown in Figure 3.7. NPBRS again performs very well, with the segmentation approximating the true mode sequence. It does however sometimes find multiple skills where a single true skill appears, as in the case of B in Figure 3.7 which is split into B_1, B_2, and B_3. This happens because IRL is an ill-posed problem, with multiple reward functions able to explain a trajectory [Abbeel and Ng 2004].

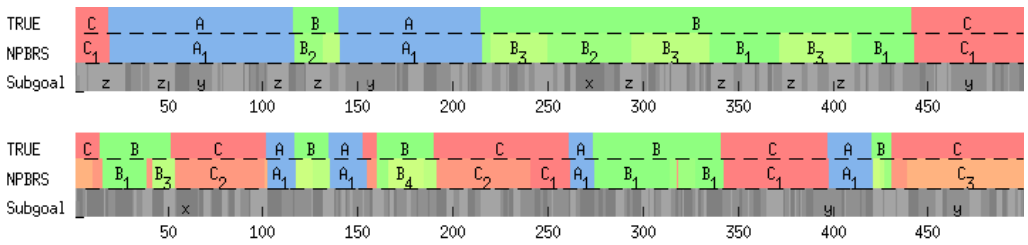


Figure 3.7: Segmentations for the quadcopter gate domain for trajectory length 500 showing using segmentation based on NPBRS and subgoal states.

	Expert	NPBRS	Subgoal	IRL
Reward	82616	79501	-30045	139
Skill Switches	6.81	9.47	88.95	0
Skills	3	8.2	2.6	1

Table 3.2: Results from NPBRS, Subgoal state and unsegmented IRL in the quadcopter gate domain averaged over 10 runs. Rewards and skill switches are presented as an average per trajectory.

Subgoal states perform very poorly, which is possibly a result of the stochasticity of the domain. The agent has no control over the hoop configurations which will be observed as these are randomly generated. It is thus impossible for an agent to formulate an effective policy to reach a particular subgoal state.

Table 3.2 presents aggregate results. Again, NPBRS achieves an average reward which is similar to the expert. Unsegmented IRL performed reasonably well in this domain, achieving a positive reward, and outperformed the subgoal state method by a large margin, indicating that the expert's behaviour in this domain is better modelled by a complex reward function rather than a combination of simple ones. NPBRS greatly outperforms both unsegmented IRL and the subgoal state method. It also indicates that the method is not data intensive, as it was able to model expert behaviour using only 16 input trajectories. The domains involved are fairly simple, with discrete state spaces that are small enough for a tabular representation to be feasible.

3.5 Conclusion

We have presented a method that segments unstructured demonstrations based on the skill being employed at each timestep, with skills represented by reward functions in an MDP. We have demonstrated that this method is better suited to some domains than representing skills as linear dynamical systems or subgoal states.

A limitation of this approach is that the method cannot distinguish between reward functions which induce the same behaviour, which caused errors in the skill boundaries due to the skills inducing the same behaviour at these points. In these experiments this was not a major problem as the skills were sufficiently different that the points of agreement were relatively small. However, when skill libraries are large, and contain many similar skills, the method is likely to come up with different skill combinations that mimic the expert behaviour rather than recovering the exact skills employed by the demonstrator. This would also imply different skill boundaries than the ones recorded by the demonstrator.

The method is capable of segmenting trajectories only in discrete domains for which a transition model is available. In complex domains, the transition model is often unknown. In the next chapter, we investigate the effect of learning the transition model from the expert trajectories.

Chapter 4

Scaling Skill Discovery to Robotic Domains

4.1 Introduction

In Chapter 3 we introduced the NPBRS algorithm, which uses an MCMC sampler to iteratively propose and test skill segmentations from unstructured demonstrations. In order to evaluate the likelihood of a proposal, the algorithm uses inverse reinforcement learning (IRL) to determine the reward functions associated with candidate skills and determines how likely it is that the behaviour segments were derived from those reward functions. The algorithm thus inherits the common IRL requirement that the transition model for the environment must be available *a priori*. This is particularly problematic in robotic domains,

where the transition model is difficult to specify upfront and must be learned.

In this chapter we extend the NPBRS algorithm, removing the need to specify the transition model. We investigate learning the environment's transition model from the provided trajectories, exploiting the fact that while the demonstrated skills have different reward functions, their transition dynamics are shared. We follow the model learning algorithm used in the Reinforcement Learning with Decision Trees algorithm [Hester *et al.* 2010] but do not actively explore the environment. Instead, we use the expert demonstrations as samples of the transition dynamics, resulting in an algorithm that requires only the expert demonstrations as input. In Section 4.3.1, we investigate the effectiveness of this algorithm in the driving domain and the quadcopter gate domain introduced in Chapter 3, finding that the use of a learned transition model does not significantly reduce accuracy or performance.

Another barrier to the use of the NPBRS algorithm in robotic domains is that it requires the use of exact value functions, making the algorithm feasible only in domains with small, discrete state spaces. Robotic domains are typically large, with continuous state variables. In order to remove this restriction, we use linear feature combinations to represent the reward functions and leverage Receding Horizon IRL [MacGlashan and Littman 2015] to recover the feature weights associated with latent skills. In Section 4.4.1 we investigate the impact of these modifications in a boat steering domain and a simulated robot arm domain, finding that the method accurately segments the demonstrations and recovers appropriate reward functions in continuous domains both with and

without a known transition model.

4.2 Background

4.2.1 Model free Inverse Reinforcement Learning

One way to accomplish model free skill discovery would be to replace the Maximum Entropy IRL algorithm with model free IRL methods. Two major algorithms that do this are the Cascaded Supervised Inverse Reinforcement Learning algorithm (CSI) [Klein *et al.* 2012] and the Relative Entropy for Inverse Reinforcement Learning (REIRL) [Boularias *et al.* 2011]. REIRL avoids solving the RL problem by using importance sampling, but this requires further samples from the environment, making it unsuitable for this research. CSI uses a two step approach in which a score function is first learned using a supervised approach associating actions to states. A regressor is then used to learn a reward function from the score function and a set of samples which are used to approximate the dynamics.

The CSI algorithm decouples the task of learning the transition model from learning the reward function. It uses a two step process, wherein learning the transition model is the task of a regression algorithm. This approach is a good fit for skill discovery as the algorithm performs IRL many times, but the transition model is unchanged across all iterations. Separating the task of learning the model allows us to reuse the learned model in all IRL iterations.

4.2.2 Model Learning

Model learning is a supervised learning task, where the agent must predict the next state given a state-action pair. R-MAX [Brafman and Tennenholtz 2002] takes the straightforward approach of taking every action from every state multiple times in order to calculate the transition model. However, if there is some structure to the transition model, we can reduce the number of samples required by generalizing across similar states. This has been accomplished using a wide variety of function approximators, such as neural networks [Venayagamoorthy *et al.* 2002], Gaussian processes [Deisenroth and Rasmussen 2011] and decision trees [Degris *et al.* 2006].

SPITI [Degris *et al.* 2006] learns a factored model using decision trees, an approach adopted by TEXPLORE's Reinforcement Learning with Decision Trees (RL-DT) algorithm [Hester and Stone 2009] used in this work. RL-DT learns the relative transition effects of actions rather than learning the absolute transitions, which improves its ability to generalize. For example, it may learn that the effect of the forward action is to increase the x variable by 1. This applies regardless of what the value of x was initially. RL-DT builds a decision forest for each state variable, with their predictions averaged to produce a stochastic transition model.

These learning methods have learned accurate models with very few samples, but are able to actively sample the domain in areas where the model is uncertain. In the LfD setting, we must learn using only the expert trajectories provided.

4.3 Model learning for recovering multiple reward functions

Our task is to learn multiple reward functions from a set of unstructured demonstration trajectories in an environment for which the transition model is unknown. We learn a transition model from the given trajectories using the model learning component of the RL-DT algorithm [Hester and Stone 2009]. We test whether the presence of multiple reward functions results in sufficient coverage of the environmental dynamics, as our algorithm cannot actively sample the transition dynamics.

In order to test this, we compare the accuracy of the models learned when samples are produced from a single expert policy with samples produced by a switching policy. We tested according to two correctness measures. The first is the proportion of successor states correctly predicted over the entire environment, and the second is the proportion of state variables correctly predicted.

Next, we segment the trajectories and learn the reward functions that explain them using the NPBRS algorithm. We give the underlying maximum entropy inverse reinforcement learning algorithm access to the learned transition model instead of the true model that was available in the experiments in Chapter 3. A system diagram of this process is presented in Figure 4.1.

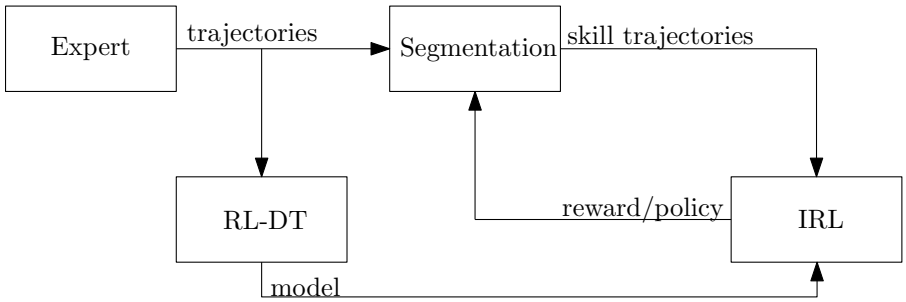


Figure 4.1: System diagram of NPBRS with a learned model.

4.3.1 Experiments

Trajectory Generation

We used the highway driving domain and the quadcopter date domain from Chapter 3, training optimal agents for each skill. We then generated 16 trajectories of length 500 for each skill, and another set of 16 for the mixed policy wherein the agent randomly switches between the available skills. The true switching behaviour for the mixed policy was recorded for validation.

Highway driving domain

The state in this domain is described by 5 variables: the current lane l, which includes the verge on either side, 3 continuous variables d_1, d_2 and d_3 representing the distance to the closest car in each lane, and c which is a boolean indicating a collision. The environment decreases the d_i variables at each timestep. When one of these variables hits 0, it is set to a random number between 0 and 6. This simulates the random generation of cars in each lane.

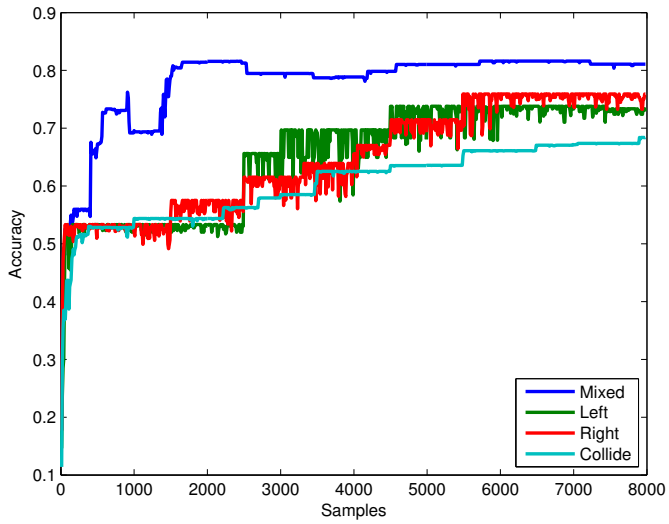
The lane variable l changes in response to action selection, and the c variable is set to true if the current lane has a distance that is less than 1. The d_i variables are discretized before being used by the model learning and skill discovery algorithms.

The three reward functions used in this domain encode the following behaviours: hit as many cars as possible (collide), drive in the right lane but change lanes to avoid collisions (right), and drive in the left lane but change lanes to avoid collisions (left). A fourth agent switches between these behaviours (mixed).

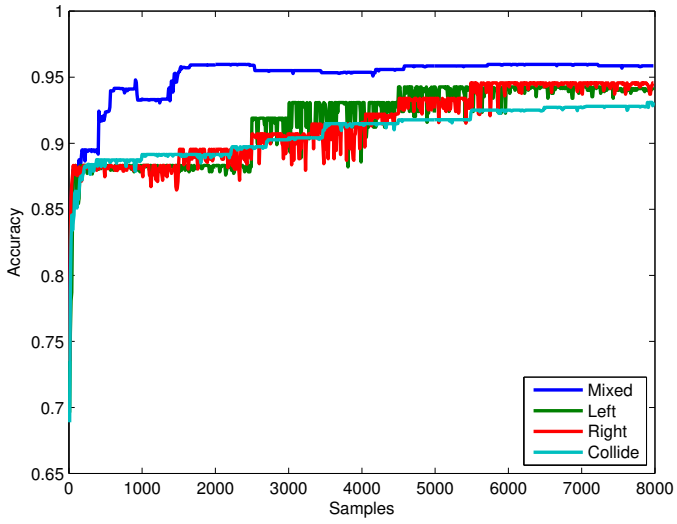
The trajectories generated by the left, right, collide and mixed agents are used to predict the environment dynamics, varying the number of samples available to the model learning system. The learned models were then compared to the true model by counting the proportion of states correctly predicted over the space.

Since the domain is discrete, we exhaustively test the successor state predictions for every state and action, comparing the prediction to the actual successor state. We present the average of the probability of correctly predicting the successor state over all possible initial states and actions.

In the figure we see that learning from the mixed policy results in a better model. This result is stronger when using fewer samples, but persists even when using all available samples. This indicates that the sort of samples encountered in a skill discovery setting are well suited to model learning. The model accuracy stabilizes at 0.8, indicating that a large proportion of the states



(a) Proportion of correctly predicted successor states. Predictions are only considered correct if all state variables are correctly predicted.



(b) Proportion of correctly predicted successor state variables. Each successor state variable is counted independently, meaning that we count partially correct predictions.

Figure 4.2: Model accuracy in the highway driving domain.

are still incorrectly predicted.

However, a state prediction can be useful even if it is incorrect, because some of the state variables could have been correctly predicted. These partially correct predictions can still help the agent avoid undesirable states. In Figure 4.2b, we compare the models by calculating the proportion of state variables correctly predicted over all states and actions in the space. This differs from the result presented in figure 4.2a in that the model is given partial credit for partially correct predictions. Again we see that the learning from the mixed model results in improved performance with fewer samples, with the graph having the same shape as Figure 4.2a, but with a much higher accuracy, stabilizing at around 0.96.

We then use the NPBRS algorithm to segment the mixed trajectories and recover the reward functions present in the unstructured trajectories. A typical segmentation is shown in Figure 4.3. In one instance we use the true dynamics model, and in the other we learn the model from the same expert trajectories that we are segmenting. In Table 4.1 we present aggregate results of using the inferred skills in the original trajectories according to the corresponding inferred segmentation. We compare the policies recovered by NPBRS given the true model and the estimated model.

The skills recovered when learning from the estimated model perform slightly worse than those recovered using the true model, but the reward achieved is similar. Both the true and estimated models result in more skills than the expert displayed, but analysis of the segmentations showed that both split some true

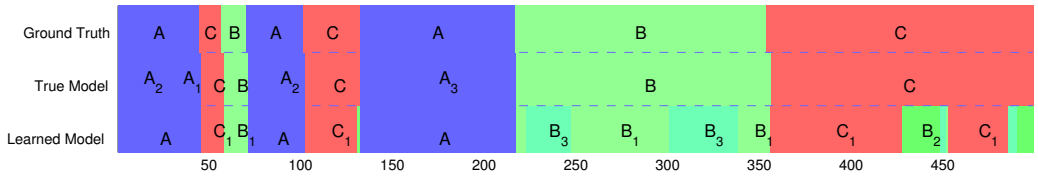


Figure 4.3: A trajectory segmentation in the car domain showing ground truth and the segmentations performed by NPBRS using the true model and the learned model.

	Expert	True Model	Learned Model
Reward	39093	37554	35418
Skill Switches	7.13	9.88	12.19
Skills	3	9.5	9

Table 4.1: Results from the NPBRS algorithm using the true model and learned model in the highway domain averaged over 10 runs. Rewards and skill switches are presented as an average per trajectory.

skills into multiple components rather than getting the segmentation totally wrong.

Quadcopter gate domain

The quadcopter gate domain is described by 6 variables. The position p in the 3×3 grid occupied by the drone, h_1, h_2, h_3, h_4 representing the configurations of the next four hoops, and c indicating a collision. At every time step, h_n takes the value of h_{n+1} to simulate the quadcopter moving forward in the domain, and h_4 is randomly generated. The position p changes according to the action selected and c is set to true if there is a hoop in position p in configuration h_1.

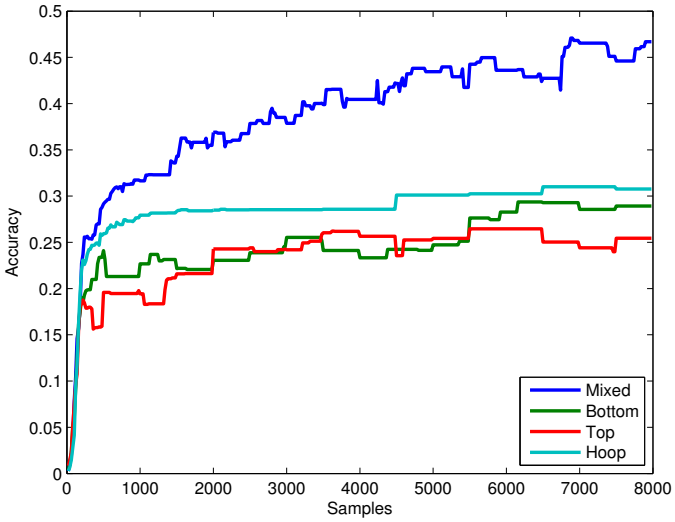
The reward functions used in this domain represent the following behaviours:

fly through the hoops (hoop), avoid hoops while flying as high as possible (top), and avoid hoops while flying as low as possible (bottom). A fourth agent switches between these behaviours (mixed).

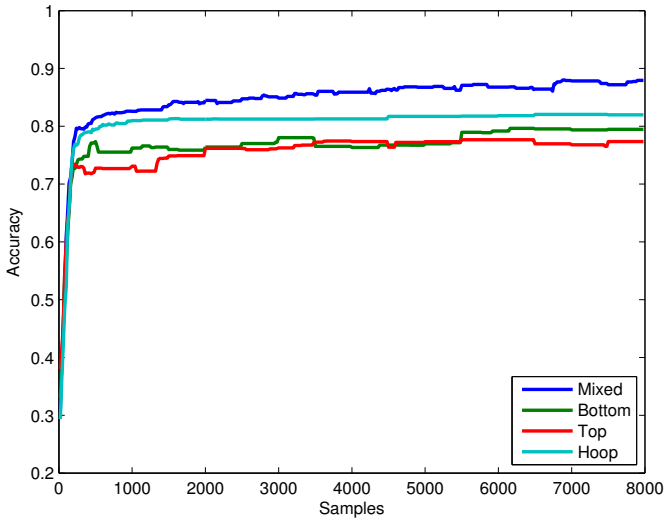
We again test the model accuracy when learning from the trajectories produced by each agent. Again, we exclude randomly generated values, in this case h_4, from the accuracy calculation. The accuracy of complete state predictions are presented in Figure 4.4a.

Again, learning from the mixed policy results in a better model. In this case the advantage persists throughout the experiment as the models do not converge within the 8000 samples available due to the larger state space. Even learning from the mixed policy only achieves an accuracy of around 0.47. However, when checking the accuracy of individual state variable calculations as shown in Figure 4.4b we find an accuracy of approximately 0.88. Again, learning from the mixed policy trajectories results in better model learning at all sample sizes.

A typical segmentation is presented in Figure 4.5 wherein we can see a slight inaccuracy in the segmentation produced by the learned model. Table 4.2 presents aggregate results. The skills recovered when segmenting using the learned model do very well in this domain, getting extremely close to the performance of the skills recovered using the true model. In this case, the skill discovery algorithm was given an inaccurate model and produced a good segmentation and good learned rewards, indicating that the skill discovery algorithm is resilient to inaccuracies in the model.



(a) Proportion of correctly predicted successor states. Predictions are only considered correct if all state variables are correctly predicted.



(b) Proportion of correctly predicted successor state variables. Each successor state variable is counted independently, meaning that we count partially correct predictions.

Figure 4.4: Model accuracy in the quadcopter gate domain.

4.4 Continuous State Spaces

Maximum Entropy Inverse Reinforcement Learning presents a challenge in that it is unable to cater for extremely large state spaces. In order to deal with the

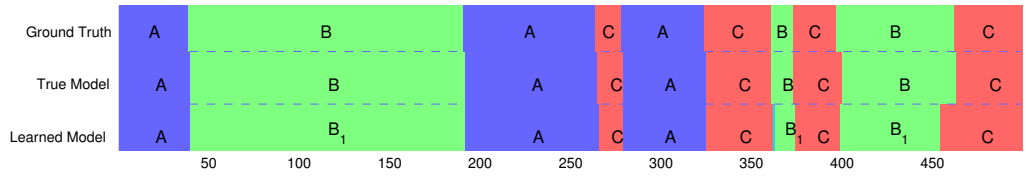


Figure 4.5: A trajectory segmentation in the quadcopter gate domain showing ground truth and the segmentations performed by NPBRS using the true model and the learned model.

	Expert	True Model	Learned Model
Reward	82616	79488	78235
Skill Switches	6.81	7.06	7.13
Skills	3	6	6.1

Table 4.2: Results from the NPBRS algorithm using the true model and learned model in the quadcopter gate domain averaged over 10 runs. Rewards and skill switches are presented as an average per trajectory

high-dimensional continuous domains commonly found in robotics, we use Receding Horizon IRL (RHIRL) [MacGlashan and Littman 2015] to recover reward functions and corresponding policies from the trajectories. This method uses a sparse sampling planner [Kearns *et al.* 2002], allowing control of the sampling depth in order to keep the computation tractable.

An important limitation of the planner is that it works best when it is likely to find states with a non-zero reward within its planning horizon. Reward functions are commonly sparse, making this a problematic restriction. To accommodate for this, we use radial basis functions with artificially large standard deviations to represent the reward features in our domains. This helps to direct the planner by providing a dense shaping reward function.

We investigate the impact of these modifications in a boat steering domain and a simulated robot arm domain, finding that the method accurately segments the demonstrations and recovers appropriate reward functions. Furthermore, we test the impact of using a learned transition model in continuous state spaces, noting that while accuracy is reduced, the skill segmentation produced still performs well.

4.4.1 Experiments

Boat domain

For our first experiment, we use a boat steering domain. The agent controls a boat moving down a river at a constant speed but with a controllable heading, with other boats moving at different speeds, as shown in the screenshot in Figure 4.6. This can be viewed as a continuous version of the commonly used car driving domain [Abbeel and Ng 2004].

The boat controlled by the agent is moving faster than all other boats on the river and has three available actions: it can rotate clockwise by 10 degrees, rotate anti-clockwise by 10 degrees, or maintain its heading. The angle is constrained to a maximum of 40 degrees.

The state is defined by the position and heading of the boat, with reward features consisting of Gaussians centered on every boat.

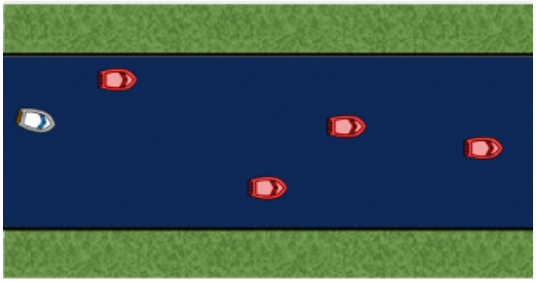


Figure 4.6: The boat steering domain. The white boat is the learning agent which could be tasked with avoiding the red boats.

We designed reward functions for the following objectives:

- A: Steer safely, preferring to stay far away from other boats
- B: Steer aggressively, deliberately colliding with other boats, while preferring the left side of the river
- C: Steer aggressively, deliberately colliding with other boats, while preferring the right side of the river

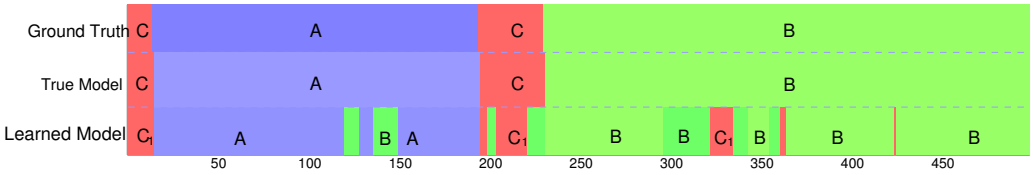


Figure 4.7: A trajectory segmentation in the boat domain showing ground truth and the segmentations performed by NPBRS using the true model and the learned model.

We trained an optimal agent for each reward function using a sparse sampling planner [Kearns *et al.* 2002] and generated trajectories containing a mixture of skills by switching between the action selections of the trained agents. This

selection was biased to be "sticky", as a skill is likely to be employed for multiple sequential timesteps. We generated 8 trajectories of length 500 and recorded the true switching sequence for later comparison.

We then used the NPBRS algorithm to segment the mixed trajectories and recover the reward functions present in the unstructured trajectories. A typical segmentation is shown in Figure 4.7. In one instance we use the true dynamics model, and in the other we learn the model from the same expert trajectories that we are segmenting. An inferred skill is given the same label and colour as a true skill if they overlap for more than half of its assigned samples over all trajectories.

In Table 4.3 we present aggregate results of using the inferred skills in the original trajectories according to the corresponding inferred segmentation. We compare the policies recovered by NPBRS given the true model and the estimated model to the expert policy. For example, the reward of 17724 listed under 'True Model' indicates that the skill sequence inferred when given access to the true transition dynamics achieves a reward of 17724 when the environment provides rewards according to the true skill sequence.

	Expert	True Model	Learned Model
Reward	20225	17724	16111
Skill Switches	3.75	14.20	12.13
Skills	3	12	14.2

Table 4.3: Results from the NPBRS algorithm using the true model and learned model in the boat domain averaged over 10 runs. Rewards and skill switches are presented as an average per trajectory.

The skills recovered when learning from the estimated model perform slightly worse than those recovered using the true model, but the reward achieved is similar. Both the true and estimated models result in more skills than the expert displayed, but analysis of the segmentations showed that both split some true skills into multiple components rather than getting the segmentation totally wrong.

The relatively small impact on performance of using a potentially flawed transition model is surprising given the large state space and small number of samples. It is possible that using the same samples for the model learning and the skill demonstrations is helpful, as the model is most accurate in regions the inferred reward function incentivizes.

Robot Arm Domain

In the robot arm domain, a simulated robot arm is used for target reaching tasks, as shown in Figure 4.8. This was implemented using a 3D simulation of the Baxter robot arm in Gazebo [Koenig and Howard 2004], controlled using ROS [Quigley *et al.* 2009]. We use the right arm in this experiment.

The agent has 16 actions available to it: it can increase or decrease the angle on each of the arm's 7 joints by 5 degrees, and it can open and close its gripper by a small amount. The state is described by the current angles on each of its joints, and the state of the gripper.

The agent has to learn to reach objects in the environment which are speci-

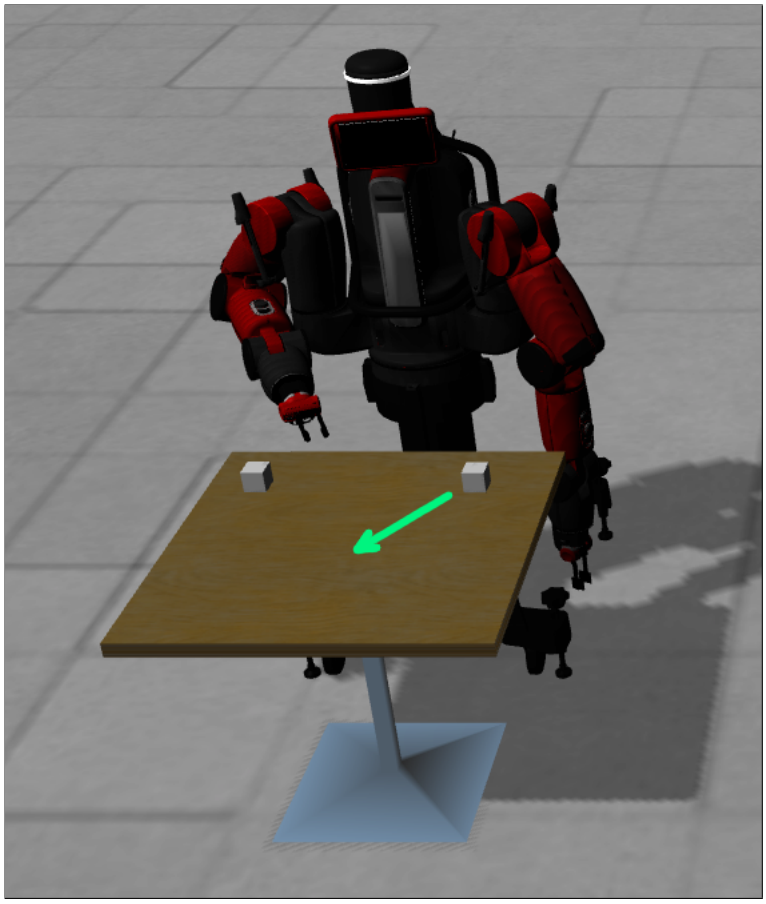


Figure 4.8: The robot arm domain. The robot alternates between picking up each of the two objects and taking them to the marked location

fied by Cartesian coordinates. The reward features are Gaussians centered on each object, with artificially large standard deviations set to allow the sparse sampling planner to more easily find regions with a non-zero reward. A further feature representing the gripper angle is included. Note that the reward function is specified in cartesian coordinates, requiring a forward kinematics computation in order to calculate the feature values.

The reward functions used in this domain represent the following behaviors:

- A: Move to location A
- B: Move to location B
- C: Move to location C
- D: Open Gripper
- E: Close Gripper

The demonstrations provided were meant to mimic a simplified coffee making task, with the robot picking up the object in locations A and B in turn and placing them in location C and then moving them back to their original locations. The task was chosen to be similar to the task of making instant coffee, where the robot would put some coffee grounds and sugar in turn in a pot. The algorithm would then need to be able to discover the skills of picking up items and moving them to the pot and then moving them back. The trajectories were generated using a sparse sampling planner switching tasks at goal locations.

A typical segmentation is presented in Figure 4.9 wherein we can see some inaccuracy in the segmentation produced by the models. Skill E is difficult to distinguish from Skill C, as its purpose is to close the gripper, but skill C also rewards a closed gripper as the objects must not be dropped on the way to location C. The underlying action sequence is thus likely under both reward functions.

Table 4.4 presents aggregate results. Again, the skill discovery algorithm was given a model learned with few samples in a large space and produced a good

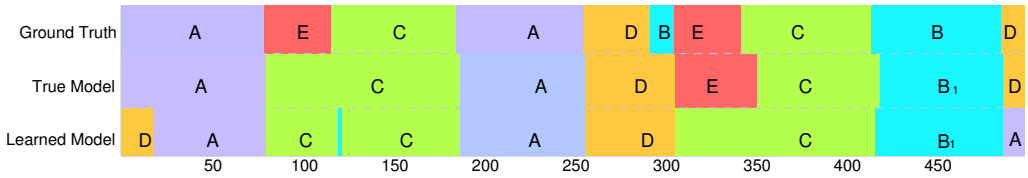


Figure 4.9: A trajectory segmentation in the robot arm domain showing ground truth and the segmentations performed by NPBRS using the true model and the learned model.

segmentation, but the resulting rewards display more of a penalty in this domain, indicating that the segmentation is more resilient to inaccuracies in the model than planning is. This suggests that the inferred dynamics are different enough to make planning difficult, but the demonstrated skills have clearly distinguishable characteristics even under the inaccurate dynamics, making it possible to segment them.

	Expert	True Model	Learned Model
Reward	82616	62314	22112
Skill Switches	9	9.12	8.6
Skills	5	15.4	10.8

Table 4.4: Results from the NPBRS algorithm using the true model and learned model in the robot arm domain averaged over 10 runs. Rewards and skill switches are presented as an average per trajectory.

4.5 Conclusion

We have presented evidence that model learning is well suited to the skill discovery setting. Models learned from unstructured expert trajectories contain-

ing multiple skills have been shown to be superior to trajectories containing a single skill. We have also demonstrated that a learned model can be successfully used for skill discovery.

We have also demonstrated that learning a model from the trajectories used for skill discovery is a successful strategy in such cases, and have shown that the method is feasible in high dimensional continuous domains where a shaping reward function is available. Model inaccuracies do have an impact on the accuracy of the recovered skill policies, but we are still able to achieve good results in the original tasks. Importantly, this work reduces the burden on an expert as the method does not require a model, and now requires only demonstrations and reward features.

These improvements facilitate the use of NPBRS in real-world robotic domains. In the next chapter, we implement this on a robot, showing that we are able to successfully recover multiple skills from unstructured expert demonstrations on a robot.

Chapter 5

Robot Experiment

The previous chapters have demonstrated the usefulness of NPBRS in simulated domains. Robotic domains present significant additional challenges as demonstrations are costly to capture, the environment's dynamics are more difficult to model, and sensor measurements and effector actions are noisy. In this chapter we conduct experiments on a robot, showing that the proposed method discovers useful options despite these challenges.

5.1 CHAMP

The experiments were run on the CSIR Hybrid Autonomous Manipulation Platform (CHAMP) [Van Eden *et al.* 2014]. CHAMP, shown in Figure 5.1, integrates the 7-DOF Barret Whole Arm Manipulator (WAM) with a four wheeled differ-

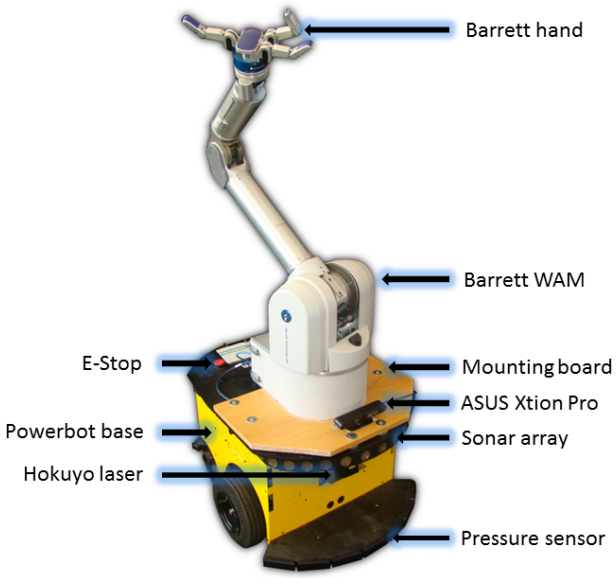


Figure 5.1: CSIR Hybrid Autonomous Manipulation Platform [Van Eden *et al.* 2014]

.

ential drive base and a number of sensors including a depth sensing camera and a laser scanner. CHAMP provides standard interfaces to its sensors and effectors through the Robot Operating System [Quigley *et al.* 2009], with joint states reported at a rate of 100Hz.

5.2 Drink-making domain

We used the task of making drinks to test our segmentation, with a similar structure to the simulated robot arm experiment presented in Section 4.4.1. There are three ingredients available in the domain: coffee, sugar, and cocoa, with each ingredient in a separate container. The robot can add these ingredients to a pot filled with water to produce various drinks. The environment is

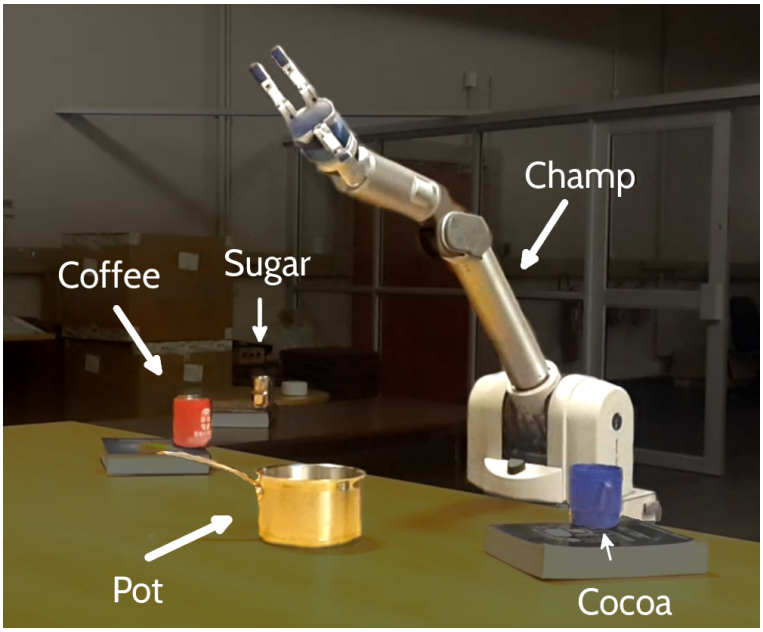


Figure 5.2: Drink-making domain.

shown in Figure 5.2.

We did not use the mobile base in our experiments, focussing instead on the WAM. The state space is composed of 8 variables, one for each joint angle, and another representing the state of the hand (either open or closed). These are controlled by 16 primitive actions which increase or decrease each of the 7 joint angles by 5 degrees, and open or close the hand.

We recorded a number of demonstrations via teleoperation, wherein a human made one of the following drinks using only the primitive actions controlled through keyboard commands:

- coffee with sugar (3 demonstrations),

- hot chocolate made up of cocoa and sugar (3 demonstrations), or
- a mocha made up of coffee, sugar and cocoa (2 demonstrations)

The demonstrations were annotated, recording which skill the human was employing. The skill annotations were as follows:

- A - Move to pot
- B - Pour
- C - Move to coffee
- D - Move to sugar
- E - Move to cocoa
- F - Close gripper
- G - Open gripper

The annotations were only used to visualize the results and were not available to the skill discovery algorithm.

5.3 Results

The demonstration trajectories were segmented using NPBRS, producing the segmentation shown in Figure 5.3. The inferred segments consistently switch

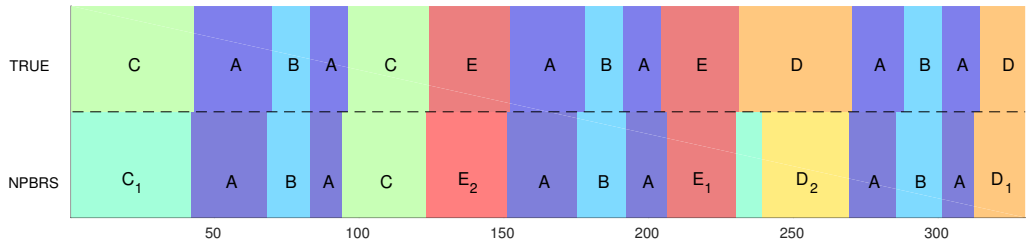


Figure 5.3: Hand-labelled vs inferred segments in the drink-making domain.

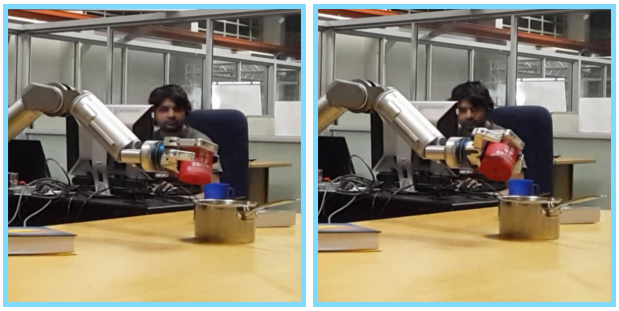
slightly earlier than the true skills, as they group gripper actions with the next skill rather than the previous one. This produces two skills for each of C, D and E, representing moving to the object with an open hand and a closed hand.

Another interesting difference between the true and inferred sequence is the presence of skill C_1 around timestep 240, between skills E_1 and D_2. This artifact was consistent across all segmentations, as the algorithm seemed to view coffee as a waypoint between sugar and cocoa.

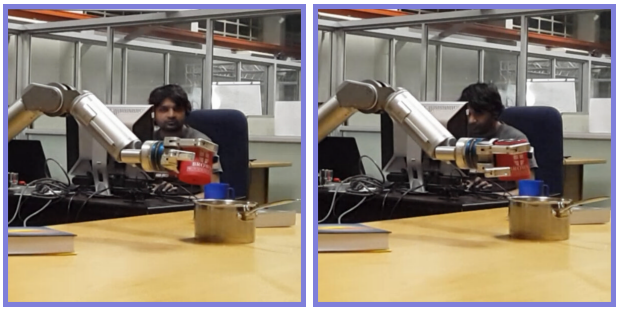
We show a time-series of robot states for the various inferred skills in Figure 5.4 to make it clearer what behaviour each segment captured. The time series shows a part of the task of making coffee with sugar, and shows the robot picking up coffee and pouring it into the pot before replacing it, picking up the sugar and taking the sugar to the pot. The border colours indicate the skill employed, and are chosen to match Figure 5.3.

(a) Skill C_1

(b) Skill A

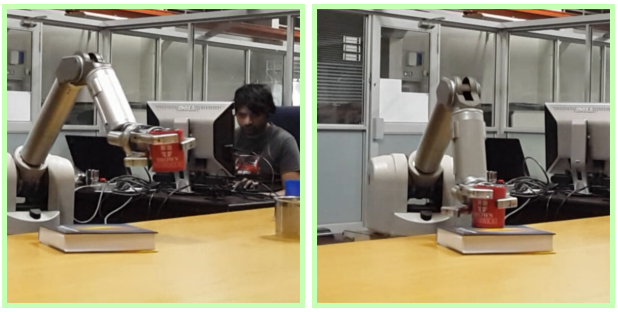


(c) Skill B

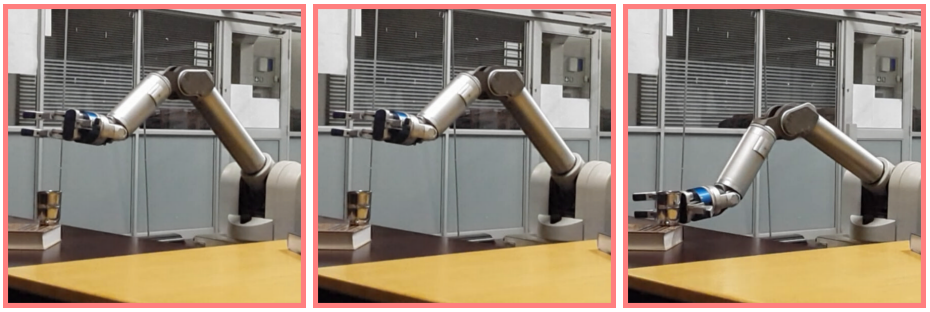


(d) Skill A

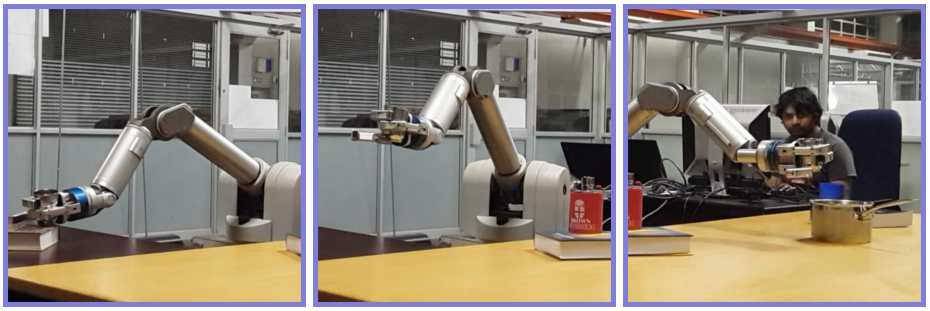
Figure 5.4: Skills in the drink-making domain.



(e) Skill C_2



(f) Skill E_2



(g) Skill A

Figure 5.4: Skills in the drink-making domain (continued).

We then added these skills to the palette of actions available to a human controlling the robot via teleoperation to view the effects of each of the inferred skills and to determine their usefulness. The use of skill C_1 as a waypoint between between skills E_1 and D_2 turned out to be necessary as the optimal action for D_2 when at the cocoa was to do nothing. This could be due to the sugar

being too far away, so that the sparse sampling planner could not reach any states with a positive reward within its planning depth.

We recorded the number of commands required to produce the various drinks using the inferred skills versus using only the primitive actions, shown in Table 5.1. This reduction in the number of steps required is significant for a number of reasons. Since human input is error prone, the chance of an error is greatly reduced as the number of input events is reduced. Furthermore, the tree of actions that must be explored in order to reach the goal is much smaller, with the increased width of the tree more than made up for by the decreased depth. For example, for the mocha, the tree would be of size 7^{336} without the use of skills and 15^{16} with them. This reduced space would also make it much easier for a reinforcement learning agent to discover an optimal policy using the discovered skills.

Drink	With Skills	Without Skills
Sugar water	5	101
Coffee With sugar	10	203
Cocoa with sugar	11	209
Mocha	16	336

Table 5.1: Number of successively correct commands required to produce various drinks with and without inferred skills

5.4 Conclusion

In this chapter, we demonstrated that the NPBRS algorithm can produce useful skills on a real robot. Challenges relating to sensor and effector noise, as well as planning in a large continuous domain, did not severely impair performance. The necessary reduction in demonstrations due to the cost and difficulty of recording trajectories on a real robot also did not cause a drop in performance. The algorithm discovered a minimal set of useful skills, which allowed a human to communicate solutions to various tasks more easily than before, showing that the inferred skills accomplish useful subgoals. This indicates that planning using these recovered skills will be significantly easier than planning using only primitive actions.

Chapter 6

Conclusion

The aim of this thesis was to facilitate hierarchical reinforcement learning by developing methods that enable an agent to discover transferable skills from unstructured demonstrations.

We developed the nonparametric Bayesian reward segmentation (NPBRS) algorithm for recovering the skills employed in a set of unstructured expert demonstrations, employing a number of biases observed in human skill discovery. The algorithm infers the number of skills, the skill reward functions, and the skill switching sequence of the demonstrated trajectories using a Markov chain Monte Carlo sampler performing inverse reinforcement learning. Our first set of experiments demonstrated its usefulness in discrete domains with a specified transition model. The algorithm represents skills as reward functions which generate behaviour, making the skills transferable to agents with different capabilities and operating in different environments to the demonstrator.

In order to make the algorithm usable in robotic domains, we adapted it to cater for continuous domains with unknown transition models by learning the model from the observations themselves. We used decision forests to model the relative transitions in each state variable, and were able to learn in continuous domains using Receding Horizon Inverse Reinforcement Learning MacGlashan and Littman [2015]. The method showed good results in a number of simulated domains, although there is some loss resulting from inaccuracy in the learned model.

Finally we demonstrated the effectiveness of the method on CHAMP, a physical mobile manipulation platform. The algorithm was able to decompose the tasks of making coffee and hot chocolate into component skills which were successfully replayed on the robot, allowing a user to specify novel behaviours with much shorter descriptions. This indicated that the method can be used on a robot acting in the real world, with the sensor and effector noise that this implies.

6.1 Future Work

The experiments conducted in this thesis showed that the algorithm recovers useful skills. The experiment conducted in Chapter 5 indicated that planning should be significantly easier when skills inferred by NPBRS are available. However, the costs involved may not be offset if the inferred skills are not useful in the target task. Planning using skills should thus be experimentally tested

in a variety of domains, with particular attention paid to the increased cost involved when the inferred skills are not useful for solving task goal. The work presented in this thesis did not test this important concern as the recovered skills were evaluated in their original domains. Existing work in transfer using options[Konidaris and Barto 2007] indicates that learned options can be useful in related domains, but more work is necessary in this area.

The NPBRS algorithm represents skills as reward functions which encode the skill objectives. This has the important advantage of being portable, as the reward function representation is not necessarily agent or environment specific. However, the reward features are currently designed by a human, meaning that care must be taken to select a feature set that is able to represent all the skills that may be inherent in the demonstrations. This is a significant challenge, and could perhaps be better performed using a feature discovery method. Recent work on deep inverse optimal control [Finn *et al.* 2016] avoids the need for a manually designed feature set by using deep neural networks to learn an arbitrary cost function and optimal policy. Integrating this work with the NPBRS algorithm is a promising avenue for improvement.

A further limitation is that the method cannot currently be used to construct a multi-level hierarchy of skills. The approach segments trajectories into the highest likelihood combination of skills, but currently this results in just one level of decomposition. More work is needed to allow the method to construct a multi-level tree wherein skills can be composed of other skills.

Given that the discovered skills are portable, we could envision the creation

of a global skill library from which agents can draw skills to help solve new tasks. A number of problems will need to be solved before this would be useful. One issue that arises when using reward functions to represent skills is that the agent must use reinforcement learning to discover an optimal policy for each skill in the library. As the library of known skills grows, this initialisation process in a new robot or environment could become very costly. Furthermore, learning the optimal policy for the task itself will be much slower and use more memory as the number of options available to the agent increases, as each option-value function must be learned. It may thus be useful to have some heuristic for the usefulness of various skills, perhaps using metadata regarding the goals of each skill. The agent could then use only the skills that are probably useful for the task it is intending to solve, deferring learning the full set of skills. This could be tackled using action priors [Rosman and Ramamoorthy 2012], which been successfully used to prune irrelevant options using information gathered in multiple domains [Abel *et al.* 2015].

Domains that require an agent to multitask would also be an interesting area for future research. An example of the need for this is in the Wargus real-time strategy game [1], wherein an agent may be interleaving battle strategy, resource collection and technology upgrades, causing the bias towards long sequences of individual skills to be unrealistic. NPBRS is able to cater for short sequences as the degree of bias is learned, but it is unable to deal with the interleaving of actions that multitasking would create. This could potentially be dealt with using multitask linearly solvable Markov decision processes [Saxe *et al.* 2017],

[1] http://wargus.sourceforge.net

which are able to represent simultaneously executing subgoals.

6.2 Conclusion

Machine learning methods are often inspired by the processes observed in human beings. An area requiring major attention is the hierarchical behaviour representation which allows humans to solve complex tasks. This hierarchy allows us to reuse our abilities in different contexts, making learning complex tasks tractable.

This thesis provides a promising method of emulating this ability, being able to produce transferable representations of higher level skills from demonstrations containing only primitive actions. This allows an agent to learn skills from demonstration, but could also be used to decompose the agent's own policies, as there is no reason the demonstrations shouldn't be trajectories generated by the agent itself.

While there are a number of challenges which require attention, we believe that this thesis represents a significant step towards general purpose robots which are able to quickly solve new tasks using hierarchical skills.

References

[Abbeel and Ng 2004] P. Abbeel and A.Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[Abbeel *et al.* 2007] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[Abbeel *et al.* 2008] Pieter Abbeel, Dmitri Dolgov, Andrew Y Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *Proceedings of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1083–1090. IEEE, 2008.

[Abbeel *et al.* 2010] P. Abbeel, A. Coates, and A.Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[Abel *et al.* 2015] David Abel, D Ellis Hershkowitz, Gabriel Barth-Maron,

Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *Proceedings of The 2014 International Conference on Automated Planning and Scheduling*, pages 306–314, 2015.

[Alvarez *et al.* 2010] Mauricio Alvarez, Jan R Peters, Neil D Lawrence, and Bernhard Schölkopf. Switched latent force models for movement segmentation. In *Advances in Neural Information Processing Systems*, pages 55–63, 2010.

[Babes *et al.* 2011] Monica Babes, Vukosi Marivate, Kaushik Subramanian, and Michael L Littman. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning*, pages 897–904, 2011.

[Bartolotta and Shulman 2013] T.E. Bartolotta and B.B. Shulman. *Language development: foundations, processes, and clinical applications*, chapter Child Development. Jones & Bartlett Publishers, 2013.

[Beal and Krishnamurthy 2012] Matthew Beal and Praveen Krishnamurthy. Gene expression time course clustering with countably infinite hidden markov models. *arXiv preprint arXiv:1206.6824*, 2012.

[Boularias *et al.* 2011] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[Boularias *et al.* 2012] Abdeslam Boularias, Oliver Krömer, and Jan Peters. Structured apprenticeship learning. In *Joint European Conference on Ma-*

chine Learning and Knowledge Discovery in Databases, pages 227–242. Springer, 2012.

[Brafman and Tennenholtz 2002] Ronen I Brafman and Moshe Tennenholtz. R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[Chernova and Veloso 2007] 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*, page 233. ACM, 2007.

[Chiappa and Peters 2010] Silvia Chiappa and Jan R Peters. Movement extraction by detecting dynamics switches and repetitions. In *Advances in Neural Information Processing Systems*, pages 388–396, 2010.

[Chung and Huang 2010] Shu-Yun Chung and Han-Pang Huang. A mobile robot that understands pedestrian spatial behaviors. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5861–5866. IEEE, 2010.

[Claxton *et al.* 2003] Laura J Claxton, Rachel Keen, and Michael E McCarty. Evidence of motor planning in infant reaching behavior. *Psychological Science*, 14(4):354–356, 2003.

[De Lisi 1987] Richard De Lisi. *Blueprints for thinking: The role of planning in*

cognitive development, chapter A cognitive-developmental model of planning. Cambridge University Press, 1987.

[Degris *et al.* 2006] Thomas Degris, Olivier Sigaud, and Pierre-Henri Wuillemin. Learning the structure of factored Markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 305–313, 2006.

[Deisenroth and Rasmussen 2011] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning*, pages 465–472, 2011.

[Dietterich 2000] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[Dimitrakakis and Rothkopf 2011] Christos Dimitrakakis and Constantin A Rothkopf. Bayesian multitask inverse reinforcement learning. In *European Workshop on Reinforcement Learning*, pages 273–284. Springer, 2011.

[Engel *et al.* 2012] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Camera-based navigation of a low-cost quadrocopter. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2815–2821, 2012.

[Finn *et al.* 2016] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In

Proceedings of the 2016 International Conference on Machine Learning, pages 49–58, 2016.

[Fox *et al.* 2014] E. Fox, M.C. Hughes, E.B. Sudderth, and M.I. Jordan. Joint modeling of multiple time series via the beta process with application to motion capture segmentation. *The Annals of Applied Statistics*, 2014.

[Fox 2009] E.B. Fox. *Bayesian Nonparametric Learning of Complex Dynamical Phenomena*. Ph.D. thesis, MIT, Cambridge, MA, 2009.

[Geman and Geman 1984] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

[Gershman and Blei 2012] Samuel J Gershman and David M Blei. A tutorial on bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12, 2012.

[Hastings 1970] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[Hester and Stone 2009] Todd Hester and Peter Stone. Generalized model learning for reinforcement learning in factored domains. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 717–724. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[Hester *et al.* 2010] T. Hester, M. Quinlan, and P. Stone. Generalized model

learning for reinforcement learning on a humanoid robot. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, pages 2369–2374. IEEE, 2010.

[Hoffman *et al.* 2008] Matthew D Hoffman, Perry R Cook, and David M Blei. Data-driven recomposition using the hierarchical dirichlet process hidden markov model. In *Proceedings of the 2008 International Computer Music Conference*, 2008.

[Howard 1960] Ronald A Howard. Dynamic programming and markov processes. 1960.

[Hughes *et al.* 2012] Michael C Hughes, Erik B Sudderth, and Emily B Fox. Effective split-merge Monte Carlo methods for nonparametric models of sequential data. In *Advances in Neural Information Processing Systems*, pages 1295–1303, 2012.

[Kearns *et al.* 2002] Michael Kearns, Yishay Mansour, and Andrew Y Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning*, 49(2):193–208, 2002.

[Keen 2011] Rachel Keen. The development of problem solving in young children: A critical cognitive skill. *Annual Review of Psychology*, 62(1):1–21, 2011. PMID: 20822435.

[Kivinen *et al.* 2007] Jyri J Kivinen, Erik B Sudderth, and Michael I Jordan. Learning multiscale representations of natural scenes using dirichlet processes. In *Proceedings of the 11th IEEE International Conference on Computer Vision*, pages 1–8. IEEE, 2007.

[Klein *et al.* 2012] Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. In *Advances in Neural Information Processing Systems*, pages 1007–1015, 2012.

[Koenig and Howard 2004] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 2149–2154. IEEE, 2004.

[Konidaris and Barto 2007] George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[Konidaris *et al.* 2012] G.D. Konidaris, S.R. Kuindersma, R.A. Grupen, and A.G. Barto. Robot learning from demonstration by constructing skill trees. *International Journal of Robotics Research*, 31(3):360–375, March 2012.

[MacGlashan and Littman 2015] James MacGlashan and Michael L Littman. Between imitation and intention learning. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 3692–3698, 2015.

[McCarty *et al.* 1999] Michael E McCarty, Rachel K Clifton, and Roberta R Collard. Problem solving in infancy: the emergence of an action plan. *Developmental psychology*, 35(4):1091, 1999.

[Meltzoff 1995] Andrew N Meltzoff. Understanding the intentions of others: Re-enactment of intended acts by 18-month-old children. *Developmental psychology*, 31(5):838, 1995.

[Michini *et al.* 2013] Bernard Michini, Mark Cutler, and Jonathan P How. Scalable reward learning from demonstration. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*, pages 303–308. IEEE, 2013.

[Michini *et al.* 2015] Bernard Michini, Thomas J Walsh, Ali-Akbar Agha-Mohammadi, and Jonathan P How. Bayesian nonparametric reward learning from demonstration. *IEEE Transactions on Robotics*, 31(2):369–386, 2015.

[Michini 2013] Bernard J Michini. *Bayesian Nonparametric Reward Learning From Demonstration*. PhD thesis, Massachusetts Institute of Technology, 2013.

[Muelling *et al.* 2014] Katharina Muelling, Abdeslam Boularias, Betty Mohler, Bernhard Schölkopf, and Jan Peters. Learning strategies in table tennis using inverse reinforcement learning. *Biological cybernetics*, 108(5):603–619, 2014.

[Neal 1993] Radford M Neal. Probabilistic inference using markov chain monte carlo methods. 1993.

[Niekum *et al.* 2012] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from

unstructured demonstrations. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5239–5246, 2012.

[Niekum 2013] S.D. Niekum. *Semantically grounded learning from unstructured demonstrations*. Ph.D. thesis, University of Massachusetts, Amherst, MA, 2013.

[Niekum 2015] Scott Niekum. A brief introduction to bayesian nonparametric methods for clustering and time series analysis. *Technical report CMU-RI-TR-15-02, Robotics Institute, Carnegie Mellon University*, 2015.

[Piaget and Cook 1952] Jean Piaget and Margaret Cook. *The origins of intelligence in children*, volume 8. International Universities Press New York, 1952.

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

[Puterman 1994] ML Puterman. Markov decision processes. 1994. *John Wiley & Sons, New Jersey*, 1994.

[Quigley *et al.* 2009] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[Rabiner 1989] Lawrence R Rabiner. A tutorial on hidden markov models

and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[Ramachandran and Amir 2007] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *Urbana*, 51(61801):1–4, 2007.

[Ranchod *et al.* 2015] Pravesh Ranchod, Benjamin Rosman, and George Konidaris. Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 471–477. IEEE, 2015.

[Rosman and Ramamoorthy 2012] Benjamin Rosman and Subramanian Ramamoorthy. What good are actions? accelerating learning using learned action priors. In *Proceedings of the 2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–6. IEEE, 2012.

[Sammut *et al.* 1992] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. In *Machine Learning Proceedings 1992*, pages 385–393. Elsevier, 1992.

[Saxe *et al.* 2017] Andrew M Saxe, Adam C Earle, and Benjamin S Rosman. Hierarchy through composition with multitask lmdps. 2017.

[Sutton and Barto 1998] R.S. Sutton and A.G. Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[Sutton *et al.* 1999] R. Sutton, D. Precup, and S. Singh. Between MDPs and

semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[Taylor and Stone 2009] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[Teh *et al.* 2005] Yee W Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Sharing clusters among related groups: Hierarchical dirichlet processes. In *Advances in neural information processing systems*, pages 1385–1392, 2005.

[Tsitsiklis and Van Roy 1997] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[Van Eden *et al.* 2014] Beatrice Van Eden, Benjamin S Rosman, Daniel J Withey, Terence Ratshidaho, Mogomotsi Keaikitse, Ditebogo Masha, Ashley Kleinhans, and Ahmed Shaik. Champ: A bespoke integrated system for mobile manipulation. 2014.

[Venayagamoorthy *et al.* 2002] Ganesh K Venayagamoorthy, Ronald G Harley, and Donald C Wunsch. Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator. *IEEE Transactions on Neural Networks*, 13(3):764–773, 2002.

[Want and Harris 2001] Stephen C Want and Paul L Harris. Learning from

other people's mistakes: Causal understanding in learning to use a tool. *Child development*, 72(2):431–443, 2001.

[Watkins and Dayan 1992] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Zhifei and Joo 2012] Shao Zhifei and Er Meng Joo. A review of inverse reinforcement learning theory and recent advances. In *Proceedings of 2012 IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[Ziebart *et al.* 2008] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008.