AUTONOMOUS ROBOT SKILL ACQUISITION

A Dissertation Presented

by

GEORGE DIMITRI KONIDARIS

Submitted to the Graduate School of the

University of Massachusetts Amherst in partial fulfillment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2011

Department of Computer Science

Report Documentation Page	*Form Approved OMB No. 0704-0188*

1. REPORT DATE **MAY 2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**
4. TITLE AND SUBTITLE **Autonomous Robot Skill Acquisition**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Massachusetts Amherst,Department of Computer Science ,Amherst,MA,01003**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		

14. ABSTRACT

Among the most impressive of aspects of human intelligence is skill acquisition?the ability to identify important behavioral components, retain them as skills, refine them through practice, and apply them in new task contexts. Skill acquisition underlies both our ability to choose to spend time and effort to specialize at particular tasks, and our ability to collect and exploit previous experience to become able to solve harder and harder problems over time with less and less cognitive effort. Hierarchical reinforcement learning provides a theoretical basis for skill acquisition, including principled methods for learning new skills and deploying them during problem solving. However, existing work focuses largely on small, discrete problems. This dissertation addresses the question of how we scale such methods up to high-dimensional continuous domains, in order to design robots that are able to acquire skills autonomously. This presents three major challenges; we introduce novel methods addressing each of these challenges. First, how does an agent operating in a continuous environment discover skills? Although the literature contains several methods for skill discovery in discrete environments, it offers none for the general continuous case. We introduce skill chaining, a general skill discovery method for continuous domains. Skill chaining incrementally builds a skill tree that allows an agent to reach a solution state from any of its start states by executing a sequence (or chain) of acquired skills. We empirically demonstrate that skill chaining can improve performance over monolithic policy learning in the Pinball domain, a challenging dynamic and continuous reinforcement learning problem. Second, how do we scale up to high-dimensional state spaces? While learning in relatively small domains is generally feasible, it becomes exponentially harder as the number of state variables grows. We introduce abstraction selection, an efficient algorithm for selecting skill-specific, compact representations from a library of available representations when creating a new skill. Abstraction selection can be combined with skill chaining to solve hard tasks by breaking them up into chains of skills, each defined using an appropriate abstraction. We show that abstraction selection selects an appropriate representation for a new skill using very little sample data, and that this leads to significant performance improvements in the Continuous Playroom, a relatively high-dimensional reinforcement learning problem. Finally, how do we obtain good initial policies? The amount of experience required to learn a reasonable policy from scratch in most interesting domains is unrealistic for robots operating in the real world. We introduce CST, an algorithm for rapidly constructing

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**163**	

AUTONOMOUS ROBOT SKILL ACQUISITION

A Dissertation Presented

by

GEORGE DIMITRI KONIDARIS

Approved as to style and content by:

Andrew G. Barto, Chair

Neil E. Berthier, Member

Roderic A. Grupen, Member

Sridhar Mahadevan, Member

Andrew G. Barto, Department Chair
Department of Computer Science

for my family

ACKNOWLEDGMENTS

Coming to UMass to work with Andy Barto was one of the most difficult decisions of my life: I chose to join a university I had not heard much about; in a small town far from the city life I was used to; in a country I found alternately puzzling, seductive and infuriating; all specifically to work with someone I had never met. I still remember feeling great trepidation stepping off the bus to Amherst, after three hours of being driven through dense forest. But ever since that day I have found myself continually in awe of Andy's incredible breadth of knowledge, his ability to see the underlying theme behind countless small technical results, his endless patience with my zany ideas, and his total commitment to granting me the freedom to pursue them. He is the researcher I aspire to be. Thank you, Andy. I am glad I came.

Sridhar Mahadevan's immense knowledge and insight have been sometimes intimidating but always inspiring, and his advice has gone a long way toward making me a better researcher. Rod Grupen—beyond letting me use his awesome robots—has inspired much of this work by his steadfast insistence on studying the right things. Neil Berthier has done much to lighten my burden when his own was heavy, for which I am very grateful. And, of course, many thanks to Leeanne and Gwyn, for making this all happen.

I have profoundly enjoyed working in the Autonomous Learning Lab, on such interesting problems with such cool people: Colin Barringer, Bruno Castro da Silva, Will Dabney, Kim

Ferguson, Mohammad Ghavamzadeh, Tom Helmuth, Jeff Johns, Anders Jonsson, Armita Kaboli, Peter Krafft, Scott Kuindersma, Bo Liu, Vicky Manfredi, Vimal Mathew, Scott Niekum, Khash Rohanimanesh, Ilya Scheidwasser, Ashvin Shah, Özgür Şimşek, Andrew Stout, Phil Thomas, Chris Vigorito, Chang Wang, and Pippin Wolfe. I would especially like to thank Sarah Osentoski, for many years of camaraderie; Jeff Johns and Özgür Şimşek, for setting such great examples; Ash Shah, for being a bridge; and Scott Niekum and Phil Thomas for reinvigorating the lab (without making me feel too old).

Shortly after defending my thesis proposal, I attended a conference where I described my plans to a senior researcher in the field over lunch. Upon hearing that I aimed to get skill acquisition working on a robot he replied, somewhat incredulously: *you and what army?* I would like to thank that army: all of the members of the Laboratory for Perceptual Robotics, who designed and built the uBot. I am continually amazed by their sheer technical wizardy. I would especially like to thank those present in the lab in the last few months of my thesis work: Yun Lin, Dirk Ruiken, Shiraj Sen, Grant Sherrick, and Dan Xie. Most importantly, I would like to thank Scott Kuindersma, who has been utterly critical to this work, and was a complete pleasure to work with. It is no exaggeration to say that it would have taken another year, at least, to complete this thesis without his help; even then, it would've been much less interesting. Thanks dude. Don't forget your slughorn.

I would also like to thank two people who did not contribute technically to this thesis, but without whom it would simply not have happened.

Eliza Nelson's love, enthusiasm and endless patience made all the difference when, in the blackest depths of culture shock, the urge to grab my laptop and my guitar and take the first bus to the nearest airport was nearly overwhelming; and in all the times between then and now, when the going got tough. Thank you, E. I could not have done it without you. Not even a little bit.

Bruno Ribeiro was the first person I met in Amherst, and has been by my side, through thick and thin, ever since. I long ago lost count of the times that Bruno came through for me, rescued me from myself, or, through simple generosity, made my life better. Thank you *hapa—meu irmão*.

I would also like to extend my thanks to each of my former housemates: Bruno Ribeiro, Gene Novark, Phil Kirlin, Steve Murtagh, Sarah Osentoski, Emily Horrell, Marc Cartright, Ilene Magpiong, Scott Kuindersma, Dirk Ruiken, Nisha Kini, and David Feierabend; and all of the wonderful friends who made my life in Amherst so much better. My improbable survival over all these years in the face of my own occasionally daunting cluelessness is down to you.

This has been a long journey, far from home. I would like to thank everyone who helped me along the way—from Edinburgh: Gill Hayes, Chris Malcolm, John Hallam, Tim Taylor, Joanna Bryson and Sethu Vijayakumar; from Wits: George Christelis, Dylan Shell, Sarah Rauchas, Lex Holt, Ian Sanders, Scott Hazelhurst, David Lubinsky, Philip Machanick, Paul Crawford, Nicolette Roussos, and Ebrahim Momoniat. And, of course, the friends that I wish I could still see every day instead of once a year: George, Lucy and Clara; Dyl; Andrea and Joanna; Gregor and Becky; Paul and Anjie; Asher; Fisco; Piet, Wes, James and Blaski; Lauren and Megan; Wendy, Sandy, Nardine and Ran; Sarah, Lex, Hannah and Matthew; Skull and Shun; Fats; Greg and Patrick; Rouven and Neelima; Matteo and Flavia. One day, somehow, we will all be together again. It will be *wild*.

Finally, I would like to thank my family. When I announced to my father, at the tender age of fourteen, that I wanted to be a computer scientist (a profession that he—and I—knew absolutely nothing about); when to my mother's horror I locked myself in my room for weeks at a time, sleeping all day and programming all night, eating nothing but cereal; when I "borrowed" my sister's university textbooks, and gleefully used her credit card to buy technical books on Amazon.com and mail them to South Africa; when I ran up frighteningly high phone-bills using my 14.4kbs modem; when I decided that to study further I needed to go overseas *for eight years*; and when not a single person in my family except me understood in the slightest what I was doing, or why; there was never a word of discouragement—not once: always encouragement. A silent conspiracy to get me what I needed, whatever it happened to be, however it could be found, even when we so obviously couldn't afford it. So: thank you—dad, mom, and Tai—for this amazing life.

This research was supported by the National Science Foundation under grant CCF-0432143, and the Air Force Office of Scientific Research under grant FA9550-08-1-0418. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or the Air Force Office of Scientific Research.

ABSTRACT

AUTONOMOUS ROBOT SKILL ACQUISITION

MAY 2011

GEORGE DIMITRI KONIDARIS

B.Sc., UNIVERSITY OF THE WITWATERSRAND

B.Sc. Hons., UNIVERSITY OF THE WITWATERSRAND

M.Sc., UNIVERSITY OF EDINBURGH

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew G. Barto

Among the most impressive of aspects of human intelligence is skill acquisition—the ability to identify important behavioral components, retain them as skills, refine them through practice, and apply them in new task contexts. Skill acquisition underlies both our ability to choose to spend time and effort to specialize at particular tasks, and our ability to collect and exploit previous experience to become able to solve harder and harder problems over time with less and less cognitive effort.

Hierarchical reinforcement learning provides a theoretical basis for skill acquisition, including principled methods for learning new skills and deploying them during problem solving. However, existing work focuses largely on small, discrete problems. This dissertation addresses the question of how we scale such methods up to high-dimensional, continuous domains, in order to design robots that are able to acquire skills autonomously. This presents three major challenges; we introduce novel methods addressing each of these challenges.

First, *how does an agent operating in a continuous environment discover skills?* Although the literature contains several methods for skill discovery in discrete environments, it offers none for the general continuous case. We introduce *skill chaining*, a general skill discovery method for continuous domains. Skill chaining incrementally builds a skill tree that allows an agent to reach a solution state from any of its start states by executing a sequence (or chain) of acquired skills. We empirically demonstrate that skill chaining can improve performance over monolithic policy learning in the Pinball domain, a challenging dynamic and continuous reinforcement learning problem.

Second, *how do we scale up to high-dimensional state spaces?* While learning in relatively small domains is generally feasible, it becomes exponentially harder as the number of state variables grows. We introduce *abstraction selection*, an efficient algorithm for selecting skill-specific, compact representations from a library of available representations when creating a new skill. Abstraction selection can be combined with skill chaining to solve hard tasks by breaking them up into chains of skills, each defined using an appropriate abstraction. We show that abstraction selection selects an appropriate representation for a new skill using very little sample data, and that this leads to significant performance improvements in the Continuous Playroom, a relatively high-dimensional reinforcement learning problem.

Finally, *how do we obtain good initial policies?* The amount of experience required to learn a reasonable policy from scratch in most interesting domains is unrealistic for robots operating in the real world. We introduce *CST*, an algorithm for rapidly constructing skill trees (with appropriate abstractions) from sample trajectories obtained via human demonstration, a feedback controller, or a planner. We use CST to construct skill trees from human demonstration in the Pinball domain, and to extract a sequence of low-dimensional skills from demonstration trajectories on a mobile robot. The resulting skills can be reliably reproduced using a small number of example trajectories.

Finally, these techniques are applied to build a mobile robot control system for the uBot-5, resulting in a mobile robot that is able to acquire skills autonomously. We demonstrate that this system is able to use skills acquired in one problem to more quickly solve a new problem.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

One of the great scientific challenges of our time is that of understanding the natural phenomenon of human intelligence, and engineering artificial systems that display aspects of that intelligence. Among the most impressive of these aspects is *skill acquisition*—the ability to identify important behavioral components, retain them as skills, refine them through practice, and apply them in new task contexts.

This dissertation addresses the question of how we might build artificial agents (especially robots) that are able to acquire skills autonomously, as part of a larger question that is core to Artificial Intelligence: rather than repeatedly engineering systems that can solve specific problems, how can we create learning agents that are flexible, adaptive and open-ended? This work falls under the broad heading of *reinforcement learning* (Sutton and Barto, 1998)—a machine learning paradigm concerned with the problem of *learning how to act*—and builds on the recent development of hierarchical reinforcement learning methods (outlined in Chapter 2). Existing work on hierarchical reinforcement learning has provided a theoretical basis for skill acquisition, including principled methods for learning new skills and deploying them during problem solving, but has focused on relatively small discrete problems. This dissertation greatly extends the reach of hierarchical reinforcement learning by developing new methods for autonomous skill acquisition in high-dimensional, continuous problems, leading to a mobile robot capable of acquiring new skills autonomously.

1.1 Skill Acquisition in Humans

Skill acquisition lies at the heart of two of the distinguishing characteristics of human intelligence. First, humans are able to perpetually improve their solutions to difficult control tasks through practice, moving from inefficient, planned movements that require a great deal of attention, to smooth, optimized movements that are executed efficiently without conscious thought. This type of learning underlies much of human achievement because it supports our unique ability to specialize at tasks by devoting time and effort to them.

Second, through the retention and refinement of solutions to important subproblems, humans become able to solve increasingly difficult problems over time. From shortly after birth we begin assembling a vast library of motor and cognitive skills that gradually enables us to engage in progressively more complex activities with progressively less cognitive effort. This ability to learn to become more efficient problem solvers is one of the key reasons human intelligence is so flexible.

Thus, acquired skills have a profound impact on our ability to interact with the world. Figure 1.1 presents some examples that illustrate their utility to human development and intelligence.

Human infants are virtually helpless at birth, but over the course of their first few years of life acquire motor skills of increasing complexity according to a developmental schedule—for example, they are able to successfully reach toward and grasp objects at approximately

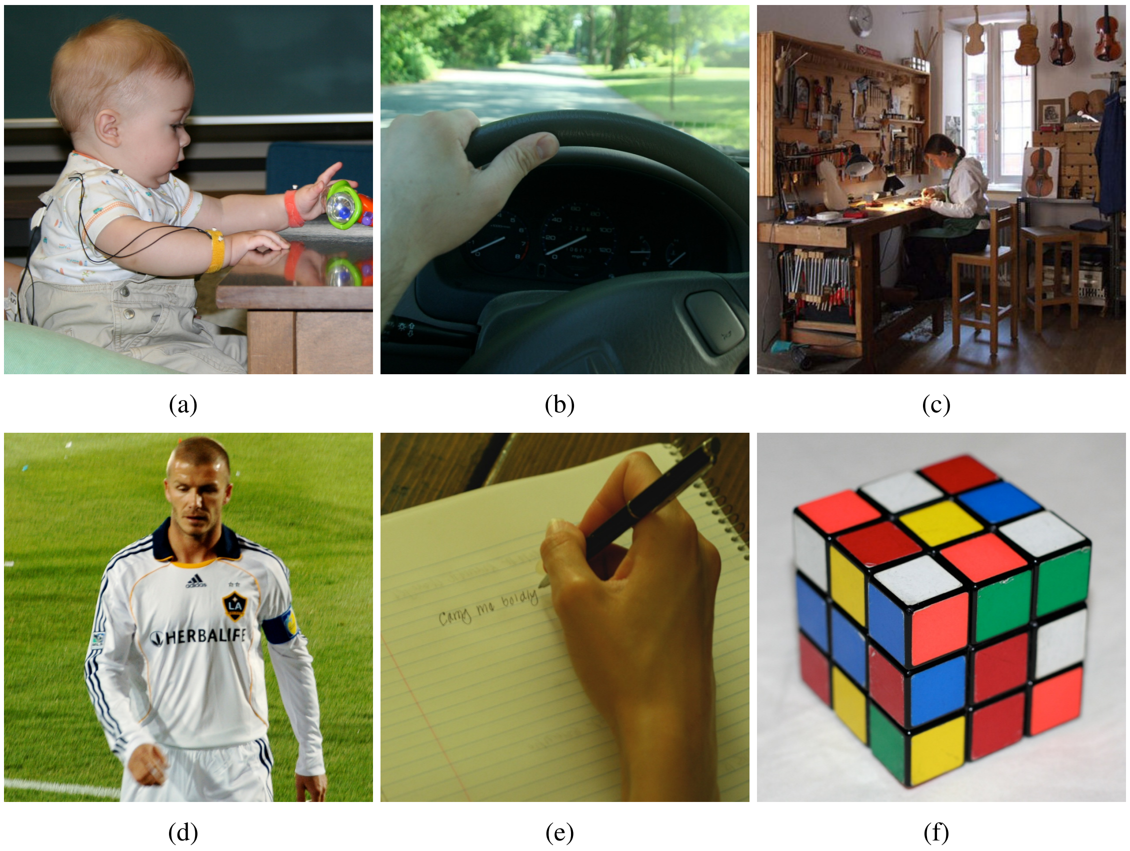


Figure 1.1. Examples of acquired skills. Some motor skills widen the scope of the learner's behavior: A 6-month old baby reaches for a toy (a) (picture copyright Eliza Nelson, used with permission); a young adult learns to drive a car (b). Specialized motor skills can require great effort to refine, and serve as the basis for an occupation: A luthier (c) (picture by Hildegard Dodel, released into the public domain), and David Beckham, professional footballer (d) (picture copyright Rajeev Patel, Creative Commons Attribution 2.0 Generic Licence). Finally, acquired skills can enhance performance in more abstract, cognitive tasks: the ability to write is a learned motor skill that underlies much of intelligent human behavior (e), while Rubik's Cube is an example of a puzzle in which expert play is distinguished by the use of acquired skills that transform it from a hard search problem to a much easier one (f).

four months of age (Berthier and Keen, 2006). Each motor skill widens the scope of the infant's behavior, endowing it with additional behavioral competence that in turn allows it to acquire more complex behaviors. This process continues throughout its lifetime; for example, American teenagers typically learn to drive at age 16, acquiring a new set of motor skills which significantly broadens their ability to travel on their own—and thus go on dates, attend college, and commute to work.

Later in life, adolescent and adult humans can choose to expend time and effort acquiring and refining specific sets of skills through practice and use them as the basis for a professional occupation. This refinement of acquired skills underlies the human ability to specialize beyond the level of skill an amateur might achieve and is consequently foundational to human technological progress and achievement.

An extreme example is provided by modern sportsmen. David Beckham, an English professional footballer, has achieved worldwide fame and a significant fortune due to his specific ability at two particular types of kicks common in football games: free kicks and crosses. Sir Alex Ferguson, Beckham's manager when he was at Manchester United, said:

> "David Beckham is Britain's finest striker of a football not because of God-given talent but because he practices with a relentless application that the vast majority of less gifted players wouldn't contemplate." (Ferguson, 1999)

Although literally *any* healthy whole-bodied human over the age of five can kick a ball, a combination of natural talent and decades of focused training on this specific set of motor skills have resulted in a player with unique ability far beyond that of an amateur. More generally, a combination of innate talent, years of experience, and a practice regime deliberately designed to gradually improve expertise seem to be required to reach expert-level performance in most domains (Ericsson, 2006).

So far our discussion has revolved around motor skills, without any reference to higher cognitive function. But acquired skills also underly human cognitive capabilities. The process of learning to write is a motor skill acquisition process that is fundamental to human education and higher cognitive development; someone who fails to master it is unlikely to complete their schooling.

Acquired skills can also be useful in more abstract contexts, as illustrated by the popular Rubik's Cube puzzle. Rubik's Cube is a difficult puzzle with roughly 4×10^{19} states, although the median optimal solution length is only about 18 moves (Korf, 1997). Despite this large search space, expert Rubik's Cube players are typically able to solve randomly permuted cubes in under a minute; they accomplish this by memorizing a set of "algorithms", or move sequences that make changes to some parts of the cube while leaving the remainder unchanged, or changed in a predictable way. (One such set of "algorithms" is given by Singmaster (1981).) Thus, the right set of acquired skills can transform a difficult abstract puzzle into a relatively simple one that can be solved using minimal cognitive effort.

1.2 Achieving Autonomous Robot Skill Acquisition

Our ultimate aim is to build agents—more specifically, robots—that are able to acquire skills autonomously, with the goal of creating robots that have the same behavioral characteristics that skill acquisition gives humans. Thus, we would like to be able to design a robot system that, while solving a problem, can identify subproblems that may become important in the future; capture them as skills and frame each skill as a learning problem; use some form of policy learning to improve each skill policy over time; and finally, when faced with a new problem, deploy its existing skills as necessary to find a solution to that problem faster that it would have been able to without them.

A robot that achieves autonomous skill acquisition in this manner would be a significant advance over the state of the art in intelligent robotics. General methods for autonomous skill acquisition could aid greatly in overcoming the obstacles facing robotics and intelligent systems today by circumventing the immense engineering effort of creating specialized controllers for each and every task a robot may encounter. More generally, a computational account of skill acquisition and an understanding of how to synthesize such agents should ultimately shed some light on the processes underlying human intelligence.

As we shall see in Chapter 2, reinforcement learning has a rich literature that already contains much of the technology we need to implement skill acquisition: several reinforcement learning methods for policy learning in moderate-sized continuous domains exist, and hierarchical reinforcement learning provides a clean, principled (though not yet complete) theoretical basis for skill acquisition and deployment, although it has so far only been applied in fairly small discrete state spaces.

We face three key challenges if we are to scale such methods up to high-dimensional, continuous domains (like robots):

1. *How do we discover skills in continuous state spaces?* All of the existing skill discovery methods for general reinforcement learning domains are only applicable to small, discrete domains, and are not easily extensible to continuous domains. We require a general algorithm for skill discovery in continuous state spaces.

2. *How do we scale up to high-dimensional state spaces?* Although we can perform policy learning in moderate-sized continuous state spaces, no existing methods are able to effectively scale up to larger problems (problems with more than roughly 15 state variables are generally beyond the reach of existing policy learning methods). In most cases, we cannot even *represent* general policies for large problems without making strong, problem-specific assumptions for the problem at hand—usually in the form of a state space and function approximator carefully designed by a human expert. We require a general method that can find small, relevant state spaces for the skills a robot may decide to acquire without advance knowledge of the nature of those skills.

3. *How do we find a reasonable initial policy in a feasible amount of time?* Even in problems that are sufficiently small to be represented and learned, finding a satisficing policy—one that achieves the goal of the task, however inefficiently—from scratch can take more time than is feasible for an agent operating in the real world. We need an effective method for creating a reasonable initial policy to circumvent the infeasibility of finding one from scratch.

1.3 Contributions

This dissertation makes four contributions. The first three address each of the major challenges discussed above, and the fourth is a demonstration of a working mobile robot system that performs autonomous skill acquisition:

1. *Skill chaining*, a general skill discovery method for continuous reinforcement learning domains. When applied to a continuous domain, skill chaining incrementally builds a skill tree that allows the agent to reach a solution state from any of its start states by executing a sequence (or chain) of acquired skills. Each skill is constructed so that the result of executing it is that the agent may execute its successor skill. This breaks the solution to the domain into a sequence of subproblems that can be represented and refined in isolation. We describe skill chaining and empirically demonstrate improved performance in a challenging dynamic and continuous domain in Chapter 3.

2. *Abstraction selection*, an efficient algorithm for selecting skill-specific, compact representations from a library of available representations when creating a new skill. A key challenge in reinforcement learning is scaling up to high-dimensional state spaces. One approach is to attempt to discard state features that are irrelevant to solving the problem, leaving only a small number of relevant state features; unfortunately, many problems are intrinsically high-dimensional when approached monolithically. Nevertheless, many real-world problems can be broken down into subproblems, each of which are individually feasible to solve using a small set of relevant features. Abstraction selection can be combined with skill chaining to solve hard tasks by breaking them up into chains of skills, each defined using an appropriate abstraction. We describe abstraction selection, demonstrate empirically that it selects an appropriate representation when creating a new skill using very little sample data, and show that this leads to significant performance improvements in Chapter 4.

3. The combination of skill chaining and abstraction selection can in principle be used to acquire a set of skills, each with a skill-specific abstraction, in high-dimensional continuous domains. However, their application is limited because skill chaining builds skills sequentially, and is therefore slow. Chapter 5 introduces *CST*, a method that incrementally builds entire skill chains from sample trajectories (as might be obtained from human demonstration, a feedback controller, or a planner), and merges chains from multiple trajectories into skill trees. Each skill is created with an initial policy matching that of the sample trajectories, thus bootstrapping policy learning. We show that CST can build skill trees from human demonstration in a challenging dynamic continuous domain, where the resulting trees result in significant performance benefits. We also show that CST is useful as an algorithm for robot learning from demonstration, that it creates appropriate trees from trajectories obtained by human control of a mobile robot, and that the resulting skills can be refined using reinforcement learning.

4. Finally, Chapter 6 uses CST to build a robot control system that is able to acquire skills autonomously. We demonstrate that this system is able to use skills acquired in one problem to more quickly solve a new problem.

First, however, Chapter 2 introduces the background necessary to understand the remainder of this dissertation and briefly outlines related research.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter surveys reinforcement learning and robotics, providing a necessarily brief overview of both fields. We thereafter outline existing research that is related to the high-level goal of skill acquisition. Research more specifically related to each of the individual algorithms developed in this thesis is discussed in the relevant chapter.

2.1 Reinforcement Learning

Reinforcement learning (Sutton and Barto, 1998) is a machine learning paradigm concerned with the problem of learning to maximize a numerical reward signal over time in a given environment. As a reinforcement learning agent interacts with its environment it receives a scalar reward for each action taken, and its goal is to learn to act so as to maximize the cumulative reward it receives over time.

When the agent's environment consists of a finite number of discrete states, it is usually modeled as a finite *Markov Decision Process* or MDP (Puterman, 1994) described by a tuple:

$$M = \langle S, A, P, R \rangle,$$

where S is the set of environmental *states* that the agent may find itself in; A is the set of *actions* it may execute; $P(s'|s,a)$ describes the probability of moving from state $s \in S$ to a state $s' \in S$ given action $a \in A$; and $R(s,a)$ returns the scalar *reward* received for executing action $a \in A$ in state $s \in S$.

In many application domains (including robotics), we are instead faced with states described by a vector of real-valued numbers. This is modeled as a continuous-state MDP, where a state s $\in S$ is a point in n-dimensional state space $S \subseteq \mathbb{R}^n$.

The agent chooses its actions according to a *policy* π which maps states to actions.[1] Given policy π and start state s, we define *return*, which measures the agent's cumulative discounted reward for executing policy π from a given state s, as:

$$R_\pi(s) = \sum_{t=0}^{\infty} \gamma^t r_{t+1},$$

where r_{t+1} is the reward obtained by executing π at time t and $0 < \gamma \leq 1$ is a *discount rate* that expresses the extent to which the agent prefers immediate reward over delayed reward. Since the goal of a reinforcement learning agent is to maximize return, the agent aims to find the *optimal policy* π^* that corresponds to a maximum value for R_{π^*} from its set of start states.

[1]This definition permits only deterministic policies. We may obtain stochastic policies by defining $\pi(s,a)$ as the probability of executing action a in state s.

2.1.1 Value Functions and Value Function Approximation

Given policy π, a reinforcement learning agent may learn a *value function* V_π mapping each state to the expected return for executing π starting from that state. If the agent is given or learns models of P and R, it may improve its performance by updating its policy using *policy iteration*:

$$\forall s, \pi(s) = \arg\max_a \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V_\pi(s')]. \tag{2.1}$$

Policy iteration is usually performed implicitly: the agent simply defines its policy as equation 2.1. Under certain conditions (Sutton and Barto, 1998), policy iteration is guaranteed to converge to the optimal policy.

When the agent does not have access to P and R, it may instead learn a state-action value function Q, which maps state-action pairs to expected return. Given Q, the agent performs policy iteration by modifying its policy at each state s so that it selects the action a the maximizes $Q(s, a)$. Although we phrase the remainder of this section in terms of value functions, state-action value functions can be learned and represented using similar methods.

In finite state spaces, value functions can be represented exactly using a table that directly stores the value of each individual state. In continuous state spaces (or very large discrete state spaces), we face two important difficulties. First, we must find a way to compactly represent a value function defined on a continuous space. Second, that representation must facilitate *generalization*: in a continuous state space we may never see the same state twice and must instead generalize from experiences in nearby states when encountering a novel one.

The most common approximation scheme (and the one used in this thesis) is *linear function approximation* (Sutton and Barto, 1998). Here, V_π is approximated by the weighted sum of a vector of *basis functions* $\mathbf{\Phi}$:

$$\bar{V}_\pi(\mathbf{s}) = \mathbf{w} \cdot \mathbf{\Phi}(\mathbf{s}) = \sum_{i=1}^{n} w_i \phi_i(\mathbf{s}). \tag{2.2}$$

Thus learning entails obtaining a weight vector $\mathbf{w}$ that results in an accurate approximation of V_π. Since $\bar{V}_\pi$ is linear in $\mathbf{w}$, when the basis functions in $\mathbf{\Phi}$ are orthogonal then there is exactly one $\mathbf{w}$ that produces the best approximation to V_π; however, we may represent complex value functions this way because each ϕ_i may be arbitrarily complex.

In this thesis we use the Fourier Basis, where each feature is defined as:

$$\phi_{\mathbf{c}}(\mathbf{x}) = \cos(\pi \mathbf{c} \cdot \mathbf{x}), \tag{2.3}$$

where $\mathbf{c} = (c_1, ..., c_m)$, $c_i \in \{0, ..., n\}$ for an order n basis on m state variables. Each basis function has a frequency along dimension i given by c_i; the order parameter n is an upper bound on that frequency. Thus, the Fourier Basis provides an easy way to model value functions using components up to a given frequency (and therefore level of detail). When a basis function has multiple elements of $\mathbf{c}$ that are non-zero, that function varies across each of the corresponding state variables, with frequency specified by that dimension's element of $\mathbf{c}$. Thus, the Fourier Basis models all possible interactions between state variables, up to a given order. More information can be found in Konidaris and Osentoski (2008).

The most common family of reinforcement learning methods are *temporal difference methods* (Sutton and Barto, 1998). When an agent takes action a from state $\mathbf{s}$, we can evaluate the difference between the return it eventually receives and that predicted by $\bar{V}$ to obtain a prediction error. However, since a measure of return is not immediately available (and may take some time to obtain), we can instead use $\bar{V}$'s own estimate of return at $\mathbf{s}'$, the state immediately following $\mathbf{s}$. This leads to an error estimate known as a temporal difference error:

$$E_t = V(\mathbf{s}) - [r_{t+1} + \gamma V(\mathbf{s}')]. \tag{2.4}$$

After each action we may perform a gradient descent step on the weights of our function approximator to reduce this error:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[r_{t+1} + \gamma V(\mathbf{s}') - V(\mathbf{s})]\nabla_{\mathbf{w}_t} V(\mathbf{s}), \tag{2.5}$$

where α is a step size parameter sometimes known as the *learning rate*. Since we are using linear function approximation, this becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[r_{t+1} + \gamma V(\mathbf{s}') - V(\mathbf{s})]\phi(\mathbf{s}). \tag{2.6}$$

This is a variant of the TD (temporal difference) family of algorithms. An extension of this method, called TD(λ), maintains an *eligibility trace* vector $\mathbf{e}$ (initially 0) that represents the discounted past activation of each element in $\mathbf{w}$:

$$\mathbf{e}_{t+1} = \gamma^{\tau} \lambda^{\tau} \mathbf{e}_t + \phi(\mathbf{s}), \tag{2.7}$$

where $\lambda \in [0, 1]$ can be set to control the amount of "history" included in the update (setting it 0 reverts to the one-step TD algorithm; setting it to 1 is equivalent to Monte Carlo value function sampling (Sutton and Barto, 1998)). $\bar{V}$ can then be updated as follows:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[r_{t+1} + \gamma V(\mathbf{s}') - V(\mathbf{s})]\mathbf{e}. \tag{2.8}$$

TD(λ) is in general more efficient that TD because it continuously updates visited states with data from new experiences. If experience is expensive—as is especially true in robot applications, where computation is much faster than action and repeated actions may result in mechanical wear—we may also use P and R to generate simulated trajectories and use them to perform offline learning. More details are available in Sutton and Barto (1998).

A related family of methods known as *least-squares temporal difference* (or LSTD) methods (Bradtke and Barto, 1996; Boyan, 1999) perform an explicit least-squares error minimization for V rather than iterative stochastic gradient descent. Given a sequence of t states $s_1, ..., s_t$, we can define eligibility trace $\mathbf{z_t} = \sum_{i=1}^{t} \lambda^{t-i}\Phi(s_i)$ and compute matrix $\mathbf{A_t}$ and vector $\mathbf{b_t}$:

$$\begin{aligned} \mathbf{A_t} = & \sum_{i=1}^{t} \mathbf{z_i}[\Phi(s_{i-1}) - \Phi(s_i)]^T \\ \mathbf{b_t} = & \sum_{i=1}^{t} \mathbf{z_i} r_i, \end{aligned} \tag{2.9}$$

from which we may obtain the $\mathbf{w}$ minimizing the least-squares projection error for V at time t as $\mathbf{w} = \mathbf{A_t^{-1}b_t}$.

LSTD methods have two important advantages over gradient-descent based temporal difference methods: they make more efficient use of sample data, and they are parameter free. However, they are not online methods, since $\mathbf{w}$ is not updated at every timestep. In addition, even though $\mathbf{A}$ and $\mathbf{b}$ can be computed incrementally (Boyan, 1999), LSTD methods are more computationally expensive than temporal difference methods, requiring $O(k^2)$ time per step and $O(k^3)$ to compute $\mathbf{w}$ for k basis functions, as opposed to the linear update time of gradient-descent temporal difference methods. Recursive least-squares methods can reduce the time required to compute $\mathbf{w}$ to $O(k^2)$, at the cost of a significant constant penalty.

Nevertheless LSTD methods are clearly appropriate for robot applications where data is expensive and free parameters present an obstacle to achieving autonomy.

2.1.2 Policy Search Methods

Performing reinforcement learning by estimating a value function and then deriving a policy from it is in some sense indirect. A more direct approach is to represent the policy explicitly, using its own function approximator. $\pi(s, a, \theta)$ thus returns the probability of selecting action a, given parameter vector θ and state s.[2] This approach leads to a class of reinforcement learning algorithms known as *policy search methods*.

Policy search methods provide a simple way to handle continuous actions, thus avoiding a potentially expensive search over a to maximize $Q(s, a)$, and similarly easily produce stochastic policies, which are especially useful in non-Markov problems. In addition, because the policy and its free parameters are directly specified, we can build in domain-specific structure or constraints that can greatly improve learning performance and help

[2]An alternative formulation (which we will not discuss further here, but for which similar properties hold) defines $\pi(s, \theta)$ as returning the selected action a, which is convenient when the agent can execute real-valued (rather than discrete) actions.

avoid dangerous or obviously bad policies. For these reasons, policy search methods are particularly well suited to robotics applications.

One approach to policy search is to directly search over the space of parameter vectors θ. Any standard optimization technique (such as a genetic algorithm) can be applied, but at present the most effective policy search method is cross-entropy (Mannor *et al.*, 2003), a variant of which has produced impressive performance when learning to play Tetris (Szita and Lörincz, 2006). However, because estimates of return given a policy parameter are noisy (due to stochasticity in the environment and in the policy), several policy evaluations are required for each candidate θ value, which often make direct search approaches impractical. However, when a simulator is available such methods can be made effective via a reduction of the variance of the estimates by fixing the sequence of random numbers used for each evaluation (Ng and Jordan, 2000).

Rather than directly searching over θ, *policy gradient methods* repeatedly compute or estimate a measure of the gradient of return with respect to the policy parameters $\frac{\partial R}{\partial \theta}$ at a given θ, and ascend it find a (locally) optimal policy. This gradient can be empirically estimated directly by sampling in the neighbourhood of a point. Kohl and Stone (2004) used such an approach to optimize a parametrized Aibo walk to obtain performance beyond that of a hand-tuned controller in about $1,000$ training episodes.

However, policy gradient methods more commonly make use of the *policy gradient theorem* (Sutton *et al.*, 2000):

$$\frac{\partial R}{\partial \theta} = \sum_s d^{\pi}(s) \sum_a \frac{\partial \pi(s,a)}{\partial \theta}(Q^{\pi}(s,a) - b(s)), \tag{2.10}$$

where d^π is the summed probability (discounted by time) of an agent following policy π reaching state s, and b (often called the *baseline* function) is any function defined over the state set. Although we do not typically know d^π, if we simply sum over transitions experienced following π we obtain an unbiased estimate of equation 2.10. The baseline function b may be set to zero everywhere, but the choice of b can substantially affect the performance of policy gradient algorithms by affecting the variance of the gradient estimator (Sutton *et al.*, 2000).

Sutton *et al.* (2000) showed that if the policy π is parameterized by θ, we can build a *compatible* linear function approximator for Q^π, defining each of its basis function as:

$$\phi_i = \frac{\partial \pi}{\partial \theta_i} \frac{1}{\pi(s, a, \theta)}. \tag{2.11}$$

The most common way to represent the policy is to use a Gibbs distribution over a linear combination of features Ψ, setting $\pi(s, a, \theta) = \frac{e^{\theta_a \cdot \boldsymbol{\Psi}_a(s)}}{\sum_b e^{\theta_b \cdot \boldsymbol{\Psi}_b(s)}}$, where the features active for action a are given by $\boldsymbol{\Psi}_a$ and their corresponding weight vector by θ_a. The compatible basis functions then become:

$$\phi_i = \frac{\partial \pi}{\partial \theta_i} \frac{1}{\pi(s, a, \theta)} = \boldsymbol{\Psi}_a(s) - \sum_b \boldsymbol{\Psi}_b(s). \tag{2.12}$$

Using this scheme we see that the value function is normalized to be mean zero at each state. This type of value function is known as an *advantage function*, defined as $A(s, a) = Q(s, a) - V(s) = Q(s, a) - \sum_a Q(s, a)$ (Sutton *et al.*, 2000; Peters *et al.*, 2003; Peters and Schaal, 2008). Advantage functions represent the relative advantage of taking action a instead of any other action from state s, and are easier to represent using function approximation (Baird, 1993). However, they cannot be learned directly using standard temporal difference learning algorithms (Peters *et al.*, 2003). Peters and Schaal (2008) describe a least-squares method where $Q(s, a)$ is represented and learned using state-action and state-only basis functions and where $A(s, a)$ can be obtained by omitting the state-only basis functions. This effectively sets $b(s) = V(s)$ while allowing us to use temporal difference learning methods.

Given an estimate of $\frac{\partial R}{\partial \theta}$, we may simply ascend the gradient by setting $\theta = \theta + \alpha \frac{\partial R}{\partial \theta}$. However, Kakade (2002) showed that this gradient descent rule is non-covariant; it uses a distance metric in the space of θ that is representation-specific and does not guarantee the steepest ascent. Instead, we should ascend the *natural gradient* $F^{-1}\frac{\partial R}{\partial \theta}$, where F is the Fischer Information Matrix. Peters and Schaal (2008) showed that the resulting update to the policy weights then becomes:

$$\theta = \theta + \alpha \mathbf{w}, \tag{2.13}$$

where $\mathbf{w}$ is the weight vector of the state-action basis functions representing $Q(s, a)$ and α is the learning rate. Although (like all policy gradient methods) it is only guaranteed to converge to a local optimum, the resulting learning algorithm (called *Natural Actor-Critic*) combining this update rule with their method for learning $A(s, a)$ has shown impressive performance in many real-world domains and is the method of choice for robot learning applications.

2.1.3 Hierarchical Reinforcement Learning using the Options Framework

The standard reinforcement learning framework assumes that actions take a single unit of time; however, in many application scenarios (in particular robotics), actions may be temporally extended—they may take an extended (and possibly varying) amount of time to run. The *options framework* (Sutton *et al.*, 1999) is a set of formalisms and methods for learning and planning using temporally extended actions called *options*.

An option o is a closed loop control policy unit consisting of three components (Sutton *et al.*, 1999): an *option policy* π_o mapping state-action pairs over which the option is defined to an execution probability; an *initiation set* indicator function I_o, which is 1 for states where the option can be executed and 0 elsewhere; and a *termination condition* β_o, giving the probability of the option terminating at each state in which it is defined. The option policy may be defined in its own state space S_o, which is usually a subset of the space in which the primary reinforcement learning problem is posed.

A reinforcement learning problem with options can be modeled as a Semi-Markov Decision Process (Precup, 2000), where the transition and reward functions model temporally extended actions. $P(s', \tau|s, o)$ now describes the probability of ending in state s' after time τ when executing option o from state s; $R(s, o)$ now describes the discounted reward accumulated before o completes execution. Given a set of options O, the temporal difference error resulting from executing option o for time τ from state s to state s' and accumulating reward r is then:

$$E_t = V(s) - [r_{t+\tau} + \gamma^{\tau} V(s')]. \quad (2.14)$$

Any temporal difference algorithm can therefore be adapted to handle SMDPs by discounting γ appropriately (Bradtke and Duff, 1995).

Thus, given a set of options O we are able to learn or plan to use them to maximize reward. In addition, if we are given an option reward function R_o, then learning that option's policy π_o is just another reinforcement learning problem, and can be solved using standard reinforcement learning techniques. We are thus able to build agents that are able to learn and plan using options, and also learn the policies behind those options.

An important research goal in hierarchical reinforcement learning is to take this one step further: to develop agents that not only learn their own option policies but also *acquire new options autonomously*. Option acquisition algorithms must specify when to create an option, how to define its initiation set, how to define its termination condition, and describe its reward function.

Creation and termination are usually performed by the identification of subgoal states, with an option created to reach a subgoal state and terminate when it has been reached. This type of option is known as a *subgoal option* (Precup, 2000), and will be the only type considered in this thesis. A subgoal is expressed as a synthetic option reward function R_o consisting of an action penalty or discount factor and a positive reward for reaching the option's goal. The initiation set is then the set of states from which the goal state is reachable, and the termination condition becomes either successfully reaching the goal or reaching a state from which success is unlikely.

Previous research has selected goal states by a variety of methods: visit frequency and reward gradient (Digney, 1998), visit frequency on successful trajectories (McGovern and Barto, 2001), variable change frequency (Hengst, 2002), relative novelty (Şimşek and Barto, 2004), salience (Barto *et al.*, 2004; Singh *et al.*, 2004), clustering algorithms and value gradients (Mannor *et al.*, 2004), local graph partitioning (Şimşek *et al.*, 2005), global max-flow/min-cuts (Menache *et al.*, 2002), causal decomposition (Jonsson and Barto, 2005), sample trajectory analysis using Dynamic Bayesian Networks (Mehta *et al.*, 2008), betweenness centrality (Şimşek and Barto, 2008) and sample trajectory analysis based on frequency, length and abstraction affordance of the extracted options (Zang *et al.*, 2009). Other research has focused on extracting options by exploiting commonalities in collections of policies over a single state space (Thrun and Schwartz, 1995; Bernstein, 1999; Perkins and Precup, 1999; Pickett and Barto, 2002).

Once R_o is available, then the option policy can be obtained by learning the corresponding value function V_o. Learning is generally performed by an off-policy reinforcement learning algorithm so that the agent can update all option policies simultaneously after taking an action (Sutton *et al.*, 1998).

Although the options framework has received a great deal of attention, almost all of the research on acquiring new options has focused on small, discrete domains. Part of the contribution of this thesis is the development of skill acquisition methods for continuous reinforcement learning domains.

2.2 Robotics

Robotics became an important area of artificial intelligence research due to the view (popularized by Brooks (1991)) that *artificial intelligence is a robot problem*. This view holds that intelligence exists to facilitate intelligent control, and cannot be separated from the problem of learning and acting in real time in the real world (Brooks, 1991; Arkin, 1998;

Pfeifer and Scheier, 1999; Murphy, 2000). The best way to study intelligence is thus to construct complete agents that must interact with complex environments in real time.

Although the isolated study of hypothesised components of intelligence is characteristic of artificial intelligence as a field, it carries several risks. It is not always clear that the chosen partitioning is correct, and so we may be solving the wrong problems. The ability of those components to interact may diminish over time as researchers focus narrowly, and so we may be producing useless solutions. The knowledge structures and constraints that result from a layered control architectures and real robot control are not usually considered, and so we may be solving harder problems than necessary; conversely, we may abstract out essential characteristics of a component, and so we may be solving easier problems than necessary.

By contrast, the embodied intelligence view emphasises real time robot control architecture research with a focus on the interaction of the components of an intelligent system. The emphasis on reactivity and competence in complex environments led to a shift away from *deliberative architectures*, where interactions with the world are carefully modelled and planned internally, to *reactive (*or *behavior-based) architectures*, where multiple control layers are built out of simple *behaviors* run in parallel, each directly connecting sensing and action (Murphy, 2000).

Although no standard definition exists, a behavior is usually described as a module mapping sensor data to control. A behavior contains all that is necessary for its own execution, including behavior-specific sensor processing, internal state (if any), and mechanisms to recognise when it can execute and when it should cease execution.

The earliest behavior-based robots had small numbers of behaviors, which made hand-engineered behavior coordination mechanisms feasible. For example, Brooks' subsumption architecture (Brooks, 1991) used the suppression of lower-level behaviors by higher-level behaviors as its primary coordination method, and Arkin's schema-based architecture (Arkin, 1989) used vector summation to add control signals from active behaviors. Apart from a great deal of design effort, these methods required significant designer insight into the task the robot is required to complete; consequently robots with a large number of behaviors and dynamic goals require more sophisticated behavior coordination methods.

Starting with Maes and Brooks (1990), a major research theme in robotics has been that of *action selection*: how should a robot with a set of behaviors use them to achieve a given goal? Most modern robot architectures employ a separate layer for action selection, using either a planning or learning system to decide which behaviors to execute and when. These architectures are considered *hybrid architectures* because they fuse reactive behaviors with higher level decision making elements.

Skill acquisition is complementary to action selection: rather than sequencing existing behaviors to achieve a goal, skill acquisition focuses on creating a new behavior that can be stored and deployed in later tasks. The need to accumulate skills is based on the idea that *an intelligent robot should be living a life, not solving a problem*—that it may need to solve a range of tasks over its operational lifetime, and that it should therefore use its experience solving some tasks to improve its performance on others. This view has recently been revived by the *developmental robotics* community, which emphasises the autonomous development of intelligent robots through lifelong learning (Weng *et al.*, 2000).

Reinforcement learning is naturally suited to both of these learning problems. Starting with Mahadevan and Connell (1992), robot action selection research has widely assumed goals specified via a reward function, and employed reinforcement learning methods to learn to achieve them. More recently, research in developmental robotics has begun to make use of the options framework for skill learning (Barto *et al.*, 2004; Stout *et al.*, 2005), since an option can be viewed as a formal model of a behavior.

Reinforcement learning and the options framework thus provide a principled theoretical framework, not only for behavior sequencing, but also for *learning new behaviors* and integrating them into the agent's behavioral repertoire. The goal of this thesis is to extend the options framework with new methods that can handle the kind of high-dimensional, continuous control problems posed by real robots.

2.3 Related Work

This section briefly covers research related to the high-level goal of robot skill acquisition; research more specifically related to each of the individual algorithms developed in this thesis is covered in the relevant chapter.

2.3.1 Instances of Real Time Robot Skill Learning

To the best of our knowledge, the earliest instance of robot skill learning was the seminal use of reinforcement learning to learn box-pushing behaviors by Mahadevan and Connell (1992). Although this work employed a fixed modular architecture and pre-designated target skills, it showed that reinforcement learning could be used to learn individual skills in real time, and that those skills could be deployed as behaviors in a modular control architecture.

Schaal *et al.* (2000) used locally weighted regression (Atkeson *et al.*, 1997) to build systems capable of learning complex robot control tasks (such as devil sticking and pole balancing) using very short training times (seconds or minutes). Using data from a human demonstration, their robot was able to learn to balance a pole in only a single trial (Schaal, 1997). An extension of this method (locally weighted projection regression) is able to deal with large numbers of irrelevant inputs (Vijayakumar and Schaal, 2000). However, although these systems are impressive, they are not reinforcement learning systems—learning is used to build a forward dynamic model that is combined with a hand-designed, task-specific controller. In addition, they make use of human demonstrations of the target task, and they use given goals and hand-engineered state spaces.

Ng *et al.* (2003, 2004) employed a locally linear learned model of the dynamics of remote-control helicopter flight and an offline policy search method called PEGASUS (Ng and Jordan, 2000) to learn policies for hovering, inverted hovering, and various maneuvers on a remote-controlled helicopter. Although the controllers did not autonomously determine which skills to learn and made use of significant domain knowledge, they provide convincing evidence that reinforcement learning methods can solve extremely hard control problems using feasible amounts of data (in this case, gathered from a few minutes of flight time). This work was later extended to use a trajectory-following reward function obtained by human demonstration to learn helicopter controllers able to perform aerobatics, outperforming trained human experts (Abbeel *et al.*, 2006).

A notably successful application of real-time skill learning has been that of gait learning for Sony AIBO robots. Here, the best performance achieved was through the use of a policy gradient algorithm (Kohl and Stone, 2004), which proved better than existing hand-tuned gaits. However, although this work was able to perform learning in real time, it was used to optimize the parameters of a highly application-specific policy.

Other robot systems (Morimoto and Doya, 2000; Konidaris and Hayes, 2005) have achieved real time learning, but only through the use of a hand-coded and task-specific hierarchy that rendered learning a high-dimensional control policy feasible.

2.3.2 Intrinsically Motivated Reinforcement Learning

Intrinsic motivation in reinforcement learning refers to the augmentation of a standard (*extrinsic*) reward function with additional internally generated reward signals designed to facilitate task-general learning. Intrinsically motivated agents are thus rewarded for activities that they may not find immediately useful or extrinsically rewarding, in the hope that such activities will result in the acquisition of knowledge that will prove useful in future tasks. Work in this area differs primarily according to the type of activities designated intrinsically rewarding, and the exact intrinsic reward mechanism used.

The earliest work on self-motived learning and curiosity focused on rewarding agents for learning progressively more accurate models of their environment (Schmidhuber, 1991a,b); more recent work has extended this broad approach to robotics (Oudeyer *et al.*, 2007). These approaches do not use acquired skills; by contrast, Vigorito and Barto (2008) employed intrinsically motivated exploration to efficiently learn a structured model that can be used to derive a skill hierarchy. However, it is not clear that the general approach of learning a model of the environment will scale up; good models of complex robots in real environments are extremely difficult to learn, even with a great deal of training data.

Another broad approach focuses on the intrinsically motivated acquisition of skill hierarchies (Singh *et al.*, 2004; Barto *et al.*, 2004; Bonarini *et al.*, 2006), and has been identified as particularly well suited to developmental robotics (Stout *et al.*, 2005; Soni and Singh, 2006). In this work the intrinsic motivation component rewards the agent for efficiently learning skill policies. This can be considered a mechanism for encouraging efficient exploration when learning a skill, and has been explicitly studied as such (Şimşek and Barto, 2006). We can thus consider this view of intrinsic motivation as complementary to skill acquisition: it provides an efficient exploration process that will aid in skill policy learning once a skill has been initially identified.

2.3.3 Qualitative State Representations

Qualitative representations of state have their roots in spatial map learning, where similar methods have been successfully applied to learn action models, control laws, and topological maps (Pierce and Kuipers, 1997).

A qualitative state representation of a single continuous variable is obtained when that variable is partitioned using a set of landmark values; the variable is then discretized into a finite set of state values corresponding to the resulting partitions. This process transforms a continuous state space into a discrete state space.

Most of this work has been concerned with autonomously building qualitative state representations from continuous features, and using the resulting discrete state space as an appropriate state space for reinforcement learning. Mugan and Kuipers (2007) describe a system that learned a qualitative state representation for a continuous domain using landmarks generated at runtime. A forward model of a form similar to a Dynamic Bayesian Network (or DBN) for each qualitative state was learned using active exploration. Stober and Kuipers (2008) introduced a method that discovers the features themselves in a pixel-based video system, by identifying moving objects using background subtraction and tracking them using their static features. Each object's dynamic features (and further features obtained using derivative and difference operators) were used as state features, upon which a fixed set of landmarks was then applied to obtain a qualitative state abstraction for policy learning.

More recently, Mugan and Kuipers (2009) extended their earlier work to create new options to reach newly discovered quantitative states, and leveraged their learned DBNs to obtain an abstraction for each option. This work showed successful learning on a very simple simulated robot platform, but although it is a fully autonomous skill acquisition system, its commitment to first learning a model of the environment from which options can be derived makes it unlikely to scale up to more interesting robots.

2.3.4 Grupen's Motor Control Hierarchy

Grupen's motor control hierarchy (Brock *et al.*, 2005) is a well developed architecture for hierarchical control and learning, primarily applied to grasping tasks on Dexter, a 29 degree of freedom humanoid robot.

The first level of the hierarchy, the *control basis* (Huber *et al.*, 1996), is a set of control-theoretic motor programs designed to provide a basis for further learning by providing stable (if suboptimal) controllers for achieving anything the robot wishes to do. Each controller can be bound to a sensorimotor coupling—a set of sensor inputs and motor outputs—and then applied to achieve control. Controllers can be combined and run simultaneously given a priority ordering, where lower priority controllers operate in the null space of higher priority controllers (Platt *et al.*, 2002). A task-specific controller can then be constructed by creating a controller assemblage for that task.

If a task is to be learned, the next level of the hierarchy uses reinforcement learning over a *discrete dynamic event system* or DEDS (Huber and Grupen, 1997; Huber, 2000), where each task-relevant controller is treated as a discrete feature that is either currently running, converged, or unable to run. Thus although control is by a set of controllers in a continuous space, learning can take place in a small, discrete state space where decisions must be made only when a controller state changes. This method is clearly strongly related to the options framework, although it was developed earlier (Huber and Grupen, 1997).

Although the use of a DEDS leads to fast learning, it can result in inefficient task performance because the controllers are typically constructed to cancel out the dynamic properties of the robot's actuators. Recent work has examined using inefficient controller trajectories to bootstrap more direct learning that harnesses the dynamic properties of the actuator (Rosenstein and Barto, 2004), though not explicitly in the context of a DEDS.

The primary limitation of constructing a DEDS is that learning is only feasible when a small number of controllers is present, since the size of the state space increases exponentially with the number of controllers present. *Schema structured learning* (Platt *et al.*, 2006) introduces the use of action schemas that represent abstract policies; learning then consists of matching a task to a candidate schema through experience and then using the schema to constrain learning. Recent work has also focused on relational representations for learning when a controller is likely to succeed (Hart *et al.*, 2005; Hart, 2009b).

Hart (2009a) combined the control basis, DEDS, and an intrinsic motivation mechanism to teach Dexter hierarchical manipulation skills. This work comes closest (to the best of our knowledge) to full autonomous skill acquisition: the robot was equipped with an intrinsic reward function (Hart, 2008) and learned to sequence a small set of controllers to achieve intrinsically rewarding events; learned controllers were then added to the robot's behavioral repertoire. However, the intrinsic reward (achieved when a controller converged to an external reference) was task-specific, and this framework relies on the presence of a teacher or programmer who both designs a developmental schedule for the robot and restricts the controllers available to it to make learning feasible. In addition, while the robot applied reinforcement learning to learn to sequence existing controllers, it did not do any low-level motor learning, and so once the controllers are sequenced the resulting policies are never improved.

2.3.5 Robot Learning by Demonstration

In robot learning by demonstration—surveyed by Argall *et al.* (2009)—a robot must extract a motor skill from a given set of successful skill trajectories. The trajectories are typically gained either by watching a human teacher demonstrate a specific task or through human teleoperation of the robot itself. In most cases the robot employs a supervised learning algorithm to obtain a policy from the example trajectories without further refinement, although in some cases the controller is subsequently improved through reinforcement learning using a hand-engineered reward function (Smart, 2002; Guenter *et al.*, 2007; Peters and Schaal, 2008).

Rather than extract a single demonstrated skill, some systems derive multiple skills by segmenting a single demonstration using either a learned Hidden Markov Model (HMM) (Pook and Ballard, 1993; Hovland *et al.*, 1996; Dixon and Khosla, 2004) or clustering (Jenkins and Matarić, 2004). Such work directly addresses the skill acquisition problem, but does not provide a mechanism for autonomously generating goals or creating initiation sets. Skills acquired this way can therefore be replayed, but their policies cannot be improved through experience.

2.4 Summary and Conclusions

Although robot learning is an active research area, there are three primary reasons why no existing robots can be considered to have achieved fully autonomous skill acquisition. First, in all cases the skill specification is given—no existing robot autonomously identifies target skills. Second, successful skill learning almost always relies heavily on designer insight to identify a task-specific state space, and in most cases upon the provision of suitable solution trajectories demonstrated by a human expert. Finally, in almost all cases skills are learned in isolation and never integrated back into a more general control architecture. Grupen's motor control hierarchy is the only robot control architecture of which we are aware that is capable of something like fully autonomous skill acquisition, but in that work skill learning consists of finding sequences of existing skills with pre-specified goals using a hand-engineered task-specific hierarchy.

Thus, the question of how to achieve autonomous robot skill acquisition, clearly a key component of an embodied artificial intelligence, is still open. In this chapter we have argued that hierarchical reinforcement learning and the options framework provide a principled theoretical framework for the skill acquisition problem. In the following chapters, we extend the options framework by developing new methods that overcome the remaining barriers to autonomy, and we use those methods to create a mobile robot capable of performing skill acquisition fully autonomously.

CHAPTER 3

SKILL DISCOVERY IN CONTINUOUS DOMAINS

The first important problem of robot skill acquisition is the problem of skill discovery—what method should a robot use to discover useful skills through interaction with its environment? In the context of the options framework, when should a robot create a new option, and what should its policy, termination condition and initiation set be? As we have shown in Chapter 2, the hierarchical reinforcement learning literature already describes several methods for discovering options in discrete domains. However, none of these methods are immediately extensible to or have been successfully applied in continuous domains.

This chapter introduces skill chaining, a skill discovery method for continuous domains. Skill chaining produces chains of skills, each leading to one of a list of designated target events, where the list can simply contain the end-of-episode event or more sophisticated heuristic events (e.g., intrinsically interesting events (Singh *et al.*, 2004; Barto *et al.*, 2004)). The goal of each skill in the chain is to enable the agent to reach a state from which its successor skill can be successfully executed.

In the following sections we briefly discuss skill discovery in continuous domains and formally define skill chaining. We then describe the implementation of skill chaining in the Pinball domain—a challenging continuous reinforcement learning domain—and demonstrate experimentally that Pinball agents using skill chaining significantly outperform agents that do not. We then cover related work and discuss the implications of the results presented here.

3.1 Skill Discovery in Continuous Domains

Although the problem of skill discovery in discrete domains is well studied and its benefits well understood, skill discovery in continuous domains remains relatively unexplored. In this section we argue that skill discovery offers additional benefits in continuous domains but poses greater challenges.

In discrete domains, the primary reason for creating an option to reach a subgoal state is to make that state *prominent* in learning: a state that may once have been difficult to reach can now be reached using a single decision (to invoke the option). This effectively modifies the connectivity of the MDP by connecting the option's subgoal to every state in its initiation set. Another reason for creating options is transfer: if options are learned in an appropriate space they can be used in later tasks to speed up learning. If the agent faces a sequence of tasks in the same state space, then options learned in it are portable (Thrun and Schwartz, 1995; Bernstein, 1999; Perkins and Precup, 1999; Pickett and Barto, 2002); if it faces a sequence of tasks in different but related state spaces, then the options must be learned using features common to all the tasks (Konidaris and Barto, 2007).

In continuous domains, there is a further reason for creating new options. An agent using function approximation to solve a task must necessarily obtain an approximate solution. Creating new options that each have their own function approximator concentrated in a subset of the state space may result in better overall policies by freeing the primary value function from having to simultaneously represent the complexities of the individual option policies.

Thus, skill discovery offers an additional representational benefit in continuous domains. However, several difficulties that are absent or less apparent in discrete domains become important in continuous domains.

Target regions. Most existing skill discovery methods identify a single state as an option target. In continuous domains where the agent may never see the same state twice, this must be generalized to a target region. However, simply defining the target region as a neighborhood about a point will not necessarily capture the goal of a skill. For example, many of the above methods aim to generate target states that are difficult to reach—a too-small neighborhood may make the target nearly impossible to reach; conversely, a too-large neighborhood may include regions of the state space that are not difficult to reach at all. Similarly, we cannot easily compute statistics over state space regions or over state-region connectivity graphs without first defining the size and shape of those regions, which is a non-trivial aspect of the problem.

Initiation sets. While in discrete domains it is common for an option's initiation set to expand arbitrarily as the agent learns a policy for successfully executing the option, in continuous spaces that is not desirable. In discrete domains without function approximation a policy to reach a subgoal can always be exactly represented; in continuous domains with function approximation (or even discrete domains with function approximation), it may only be possible to represent such a policy locally. We are thus required to determine the extent of a new option's initiation set either analytically or through trial-and-error learning.

Representation. An option policy in both discrete and continuous domains should be able to consistently solve a simpler problem than the overall task using a simpler policy. A value table in a domain with a finite state set is a relatively simple data structure, and updates to it take constant time. Thus, in a discrete domain it is perfectly feasible to create a new value table for each learned option of the same dimension as the task value table. In continuous domains with many variables, however, value function approximation may require hundreds of even thousands of features to represent the overall tasks value function, and updates are usually linear time. Therefore, lightweight options that use fewer features than needed to solve the overall problem are desirable in high-dimensional domains, or when we may wish to create many skills.

Characterization. Şimşek and Barto (2008) characterize useful subgoals as those likely to lie on a solution path of the task the agent is facing. Options that are useful across a collection of problems should have goals that have high probability of falling on the solution paths of some of those problems (although not necessarily the one the agent is currently solving). In a discrete domain where the agent faces a finite number of tasks, one characterization of an options usefulness may be obtained by treating the MDP as a graph and computing the likelihood that its goal lies on a solution path. Such a characterization is much more difficult in a continuous domain.

In the following section we develop an algorithm for skill discovery in continuous domains by addressing these challenges.

3.2 Skill Chaining

Since a useful option lies on a solution path, it seems natural to first create an option to reach the tasks goal. The high likelihood that the option can only do so from a local neighborhood about this region suggests a follow-on step: *create an option to reach the states where the first option can be successfully executed.*

This section describes skill chaining, a method that formalizes this intuition to create chains of options to reach a given target event by repeatedly creating options to reach options created earlier in the chain. First, we describe how to create an option given a target event.

3.2.1 Creating a Skill to Trigger a Target Event

Given an episodic task defined over state space S with reward function R, we assume we are given a target event trigger function T:

$$T(s) = \begin{cases} 1, & \text{if } s \text{ is in the target region;} \\ 0, & \text{otherwise,} \end{cases} \tag{3.1}$$

where $s \in S$. T is simply an indicator function that is 1 for states in the goal region and 0 otherwise. To create an option, o_T, to trigger T (reach a state s where $T(s) = 1$), we must define o_T's termination condition β_{o_T}, reward function R_{o_T} (and thereby implicitly policy π_{o_T}), and initiation set I_{o_T}.

Since o_T's goal is to reach an s where $T(s) = 1$, we can use T as o_T's termination condition directly and set $\beta_{o_T} = T$. Thus, o_T terminates when it triggers T.

We use the task reward function R as o_T's reward function but add an option completion reward c for triggering T:

$$R_{o_T}(s) = R(s) + cT(s). \tag{3.2}$$

We consider all states where T is 1 to be absorbing. We can then allocate o_T a suitable number of basis functions, and use a standard reinforcement learning algorithm to learn its policy π_{o_T}.

Obtaining o_T's initiation set I_{o_T} is more difficult because it should consist of the states from which executing π_{o_T} triggers T, once π_{o_T} has been initialized. Therefore, we cannot specify I_{o_T} in advance because it depends on π_{o_T} (which must itself be learned).

We can treat this as a standard classification problem by executing π_{o_T} and collecting labelled examples of states where it succeeds in triggering T and states where it fails. We consider an option execution failed when it has not reached a termination state within s_{max} steps. These states can be used as training examples for a classifier suited to a potentially nonstationary classification problem with continuous features. Once such a classifier has been learned, we can use its output as I_{o_T}. Thus, once we have an initial policy π_{o_T} we can proceed to learn I_{o_T} through experience.

3.2.2 Creating Skill Chains

Given an initial target event T_0, which for the purposes of this discussion we consider to indicate terminal states of the underlying task (i.e., T_0 is 1 for terminal states and 0 elsewhere), the agent creates a chain of skills as follows. First, the agent creates option o_{T_0} to trigger T_0, learns a good option policy, and obtains a good estimate, $\hat{I}_{T_0}$, of its initiation set. We then add new event $T_1 = \hat{I}_{o_{T_0}}$ to the list of target events, so that when the agent triggers T_1 it will create a new option o_{T_1} to reach it (o_{T_1} is defined as discussed above, using T_1 as a trigger event). Repeating this procedure results in a chain of skills leading from any state in which the agent may start to task's goal region, as depicted in Figure 3.1.

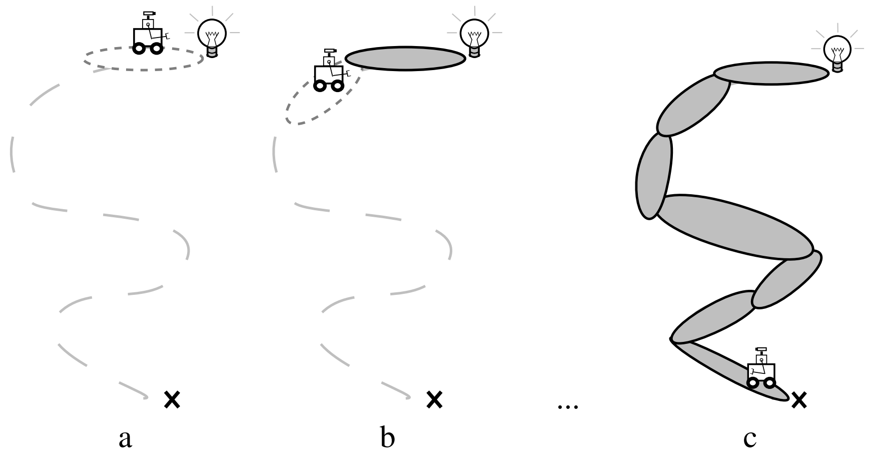


Figure 3.1. An agent creates skills using skill chaining. (a) First, the agent encounters a target event and creates an option to reach it. (b) Entering the initiation set of this first option triggers the creation of a second option whose target is the initiation set of the first option. (c) Finally, after many trajectories the agent has created a chain of options to reach the original target.

Note that although the options lie on a chain, the decision to to execute each option is part of the agent's overall learning problem. Thus, they may not necessarily be executed sequentially; in particular, if an agent has learned a better policy for some parts of the chain, it may learn to skip some options.

3.2.3 Creating Skill Trees

The procedure above can create more general structures than chains. More than one option may be created to reach a target event if that event remains on the target event list after the first option is created to reach it. Each "child" option then creates its own chain, resulting in a skill tree, depicted in Figure 3.2. This will most likely occur when there are multiple solution trajectories (e.g., when the agent has multiple start states), or when noise or exploration create multiple segments along a solution path that cannot be covered by just one option.

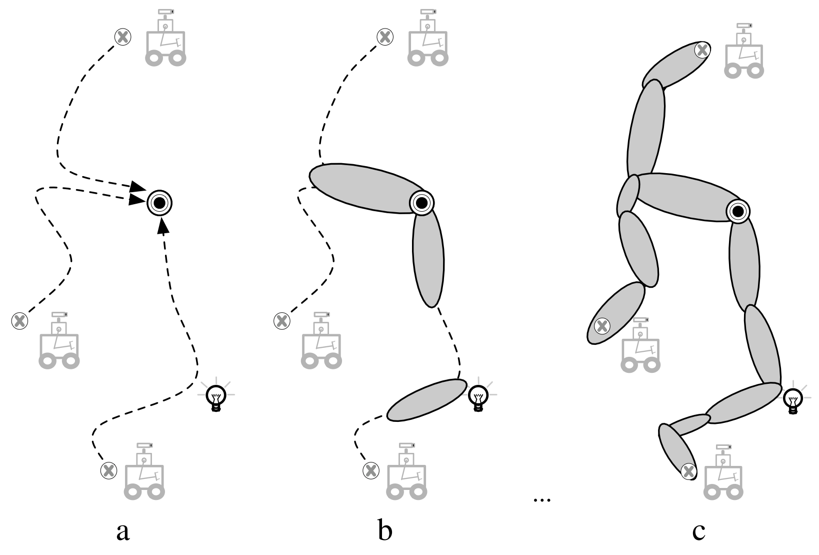


Figure 3.2. An agent creating a skill tree. (a) A skill chaining agent in an environment with multiple start states and two initial target events. (b) When the agent initially encounters target events it creates options to trigger them. (c) The initiation sets of these options then become target events, later triggering the creation of new options so that the agent eventually creates a skill tree covering all solution trajectories.

To control the branching factor of this tree, we need to place three further conditions on option creation. First, we do not create a new option when a target event is triggered from a state already in the initiation set of an option targeting that event. Second, we require that the initiation set of an option does not overlap that of its siblings or ancestors.[1] Finally, we may find it necessary to set a limit b_{max} on the branching factor of the tree by removing an event from the target event list once it has b_{max} options targeting it.

3.2.4 More General Target Events

Although we have assumed the task's end-of-episode trigger function is the only initial target event, we are free to start with *any* list of target events. We may thus include measures of novelty or other intrinsically motivating events (Singh *et al.*, 2004; Barto *et al.*, 2004) as triggers, heuristic triggers, events that are interesting for domain-specific reasons (e.g., physically meaningful events for a robot) or more general skill discovery techniques that can identify regions of interest before the goal is reached.

3.3 The Pinball Domain

The Pinball domain is a continuous, dynamic navigation task.[2] It is an appropriate continuous domain for skill discovery because its dynamic aspects, sharp discontinuities, and extended control characteristics make it difficult for control and for function approximation—

[1]Although these conditions seem computationally expensive, they can be implemented using at most one execution of each initiation classifier set per visited state—which is required for action selection anyway.

[2]The Java source code for Pinball is available for download under the GNU General Public License, at: `http://www-all.cs.umass.edu/~gdk/pinball`

much more difficult than a simple navigation task, or typical benchmarks like Acrobot. While a solution with a flat learning system is possible, there is scope for acquiring skills that could result in a better solution. Our experiments use two instances of the Pinball domain, shown in Figure 3.3.

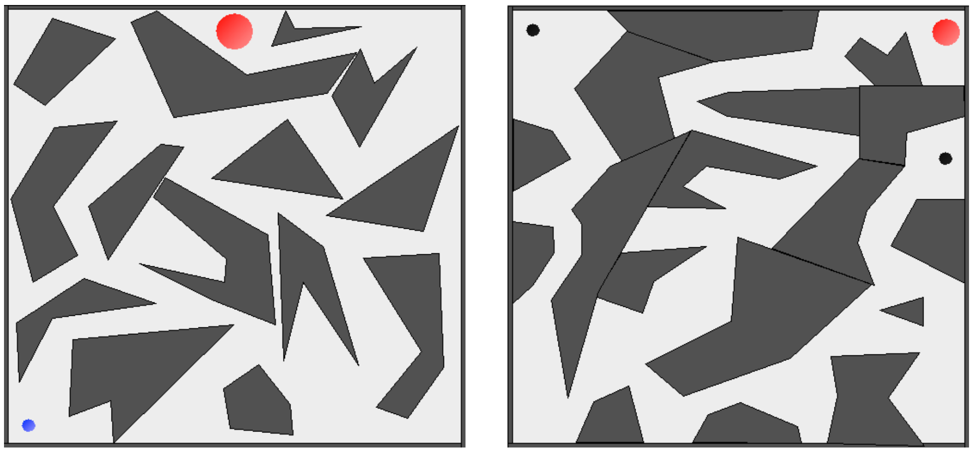


Figure 3.3. The two Pinball domain instances used in our experiments.

The goal of Pinball is to maneuver the small ball (which always starts in the same place in the first instance, and in one of two places in the second) into the large red hole. The ball is dynamic (drag coefficient 0.995), so its state is described by four variables: x, y, $\dot{x}$ and $\dot{y}$. Collisions with obstacles are fully elastic and cause the ball to bounce, so rather than merely avoiding obstacles the agent may choose to use them to efficiently reach the hole. There are five primitive actions: incrementing or decrement $\dot{x}$ or $\dot{y}$ by a small amount (which incurs a reward of -5 per action), or leaving them unchanged (which incurs a reward of -1 per action); reaching the goal obtains a reward of $10,000$.

3.3.1 Implementation Details

To learn to solve the overall task for both standard and option-learning agents, we used Sarsa ($\gamma = 1, \epsilon = 0.01$) with linear function approximation, using a 4th-order Fourier Basis (Konidaris and Osentoski, 2008) (625 basis functions per action) with $\alpha = 0.001$ for the first instance and a 5th-order Fourier Basis (1296 basis functions per action) with $\alpha = 0.0005$ for the second (in both cases α was systematically varied and the best performing value used).

Option policy learning was accomplished using Q-learning ($\alpha_o = 0.0005, \gamma = 1, \epsilon = 0.01$) with a 3rd-order Fourier Basis (256 basis functions per action). Off-policy updates to an option for states outside its initiation set were ignored (because its policy does not need to be defined in those states), as were updates from unsuccessful on-policy trajectories (because their start states were then removed from the initiation set).

Although most prior research has initialized new option policies using experience replay (McGovern and Barto, 2001), Jong *et al.* (2008) point out that these extra updates may be experimentally confounding. Therefore, in order to initialize the option policy before attempting to learn its initiation set, a newly created option was first allowed a "gestation period" $g = 10$ episodes where it could not be executed and its policy was updated using only off-policy learning.

After its gestation period, the option was added to the agent's action repertoire. For new option o, this requires expanding the overall action-value function Q to include o and assigning appropriate initial values to $Q(s, o)$. We therefore sampled the Q values of transitions that triggered the option's target event during its gestation, and initialized $Q(s, o)$ to the maximum of these values. This method reliably resulted in an optimistic but still fairly accurate initial value that encouraged the agent to execute the option.

Each option's initiation set was learned by a logistic regression classifier (Bishop, 2006), initialized to be true everywhere, using 2nd order polynomial features, learning rate $\eta = 0.1$, and $k = 100$ sweeps for each new data point. To balance positive and negative examples, each positive example was weighted by f_+ (the fraction of positive examples in the data set) and each negative example by f_- (the fraction of negative examples in the data set). When the agent executed the option, states in trajectories that reached its goal within $s_{max} = 250$ steps were used as positive examples, and the start states of trajectories that did not were used as negative examples. We considered an option's initiation set learned well enough to be added to the list of targets when its weights changed on average less than $\epsilon_I = 0.15$ per episode for two consecutive episodes. Since the Pinball domain has such strong discontinuities, to avoid over-generalization after this learning period we additionally constrained the initiation set to be true only for points within a Euclidean distance of $\delta_{max} = 0.1$ of a positive example. Each target event was assigned a maximum branching factor $b_{max} = 3$.

Table 3.1 summarizes the parameter settings used for the Pinball implementation of skill chaining, and organizes the parameters into groups. The parameters relating to task and option value function learning are unavoidable for any skill acquisition method, although parameter-free reinforcement learning methods are a topic of active research.

Skill chaining itself results in three parameters: maximum branching factor for the skill tree b_{max}, gestation period g and maximum skill length s_{max}. We expect that b_{max} may not be necessary for real applications (and can therefore be set to ∞), but if it is, it can be easily

Description	**Parameter**	**Value**
Task Value Function Learning		
Number of basis functions	$\|\Phi\|$	625 (1296)
Learning rate	α	0.001 (0.0005)
Exploration rate	ϵ	0.01
Option Value Function Learning		
Number of basis functions	$\|\Phi_o\|$	256
Learning rate	α_o	0.0005
Exploration rate	ϵ_o	0.01
Skill Chaining Parameters		
Maximum skill tree branching factor	b_{max}	3
Gestation period (episodes)	g	10
Maximum skill length (steps)	s_{max}	250
Initiation Set Classifier Parameters		
Number of features	$\|\Phi_I\|$	81
Learning rate	η	0.1
Number of sweeps	k	100
Convergence threshold	ϵ_I	0.15
Maximum positive example distance	δ_{max}	0.1

Table 3.1. Parameters for the Pinball domain implementation of skill chaining. When a different parameter value is used for the second instance it is given in parentheses.

set to a small integer. Similarly our informal experiments showed learning to be robust to changes in s_{max}. The gestation period g is a consequence of using off-policy learning for option policy initialization. It may be possible to eliminate g using another initialization method, or using some measure of value function error. However, again from informal experimentation g can be set to any small integer larger than two without greatly affecting our results.

The largest group of parameters are those related to the logistic regression classifier used to learn option initiation sets. These parameters were found using a few hours' worth of informal experiments in the domain. However, any appropriate classifier can be employed for initiation set learning, and more advanced classifiers (e.g., support vector machines (Schölkopf and Smola, 2001)) have fewer parameters, at the cost of increased algorithmic and computational complexity. We expect that for a particular domain an implementor will select an appropriate classifier by considering the tradeoff between the number of parameters and the complexity of the classifier.

3.4 Results

Figure 3.4 shows the performance (averaged over 100 runs) in the first Pinball instance for agents using a flat policy (without options) against agents employing skill chaining, and agents starting with given (pre-learned) options. Pre-learned options were obtained using skill chaining over 250 episodes in the same Pinball task.

Figure 3.4 shows that the skill chaining agents performed significantly better than flat agents by 50 episodes, and obtained consistently better solutions by 250 episodes, whereas the flat agents did much worse and were less consistent. Agents that started with given options did very well initially—with an initial episode return far greater than the average solution eventually learned by agents that start without options—and proceed quickly to the same quality of solution as the agents that discover the options themselves. This shows that the options acquired, and not some by-product of acquiring them, were responsible for the increase in performance.

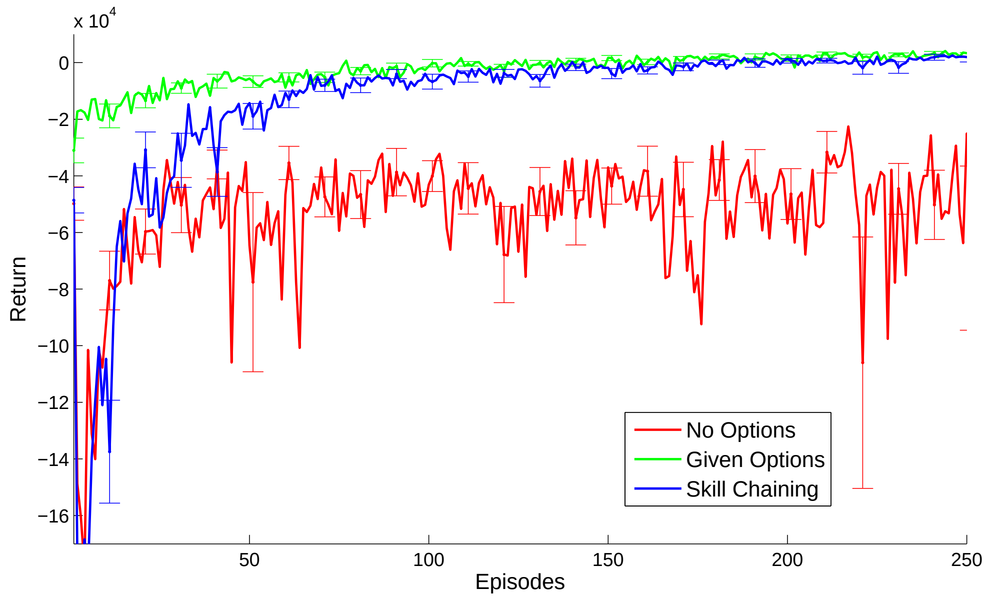


Figure 3.4. Performance in the first Pinball instance (averaged over 100 runs) for agents employing skill chaining, agents with given options, and agents without options.

Figure 3.5 shows a sample solution trajectory from an agent performing skill chaining in the first Pinball instance, with the options executed shown in different colors. The figure illustrates that this agent discovered options corresponding to simple, efficient policies covering segments of its sample trajectory. It also illustrates that in some places (in this case, the beginning of the trajectory) the agent has learned to bypass a learned option—the black portions of the trajectory show where the agent employed primitive actions rather than a learned option. In some cases this occured because poor policies were learned for those options. In this particular case, the presence of other options freed the overall policy (with its more complex function approximator) to represent the remaining trajectory segment better than an option could (with its less complex function approximator). Figure 3.6 shows the initiation sets and three sample trajectories from the options used in the trajectory shown in

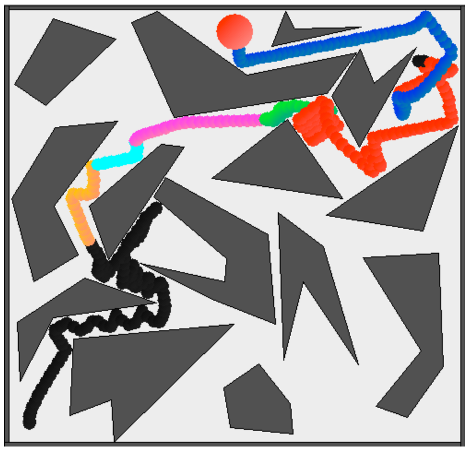


Figure 3.5. A sample solution trajectory for the first Pinball instance. Acquired options executed along the sample trajectory are shown in different colors; primitive actions are shown in black.

Figure 3.5. These learned initiation sets show that the discovered option policies are only locally valid, even though they are represented using Fourier basis functions, which have global support.

Figures 3.7, 3.8 and 3.9 show similar results for the second Pinball instance, although Figure 3.7 shows a slight and transient initial penalty for skill chaining agents, before they go on to obtain far better and more consistent solutions than flat agents. The example trajectory in Figure 3.8 and initiation sets in Figure 3.9 show that a skill tree has been successfully formed.

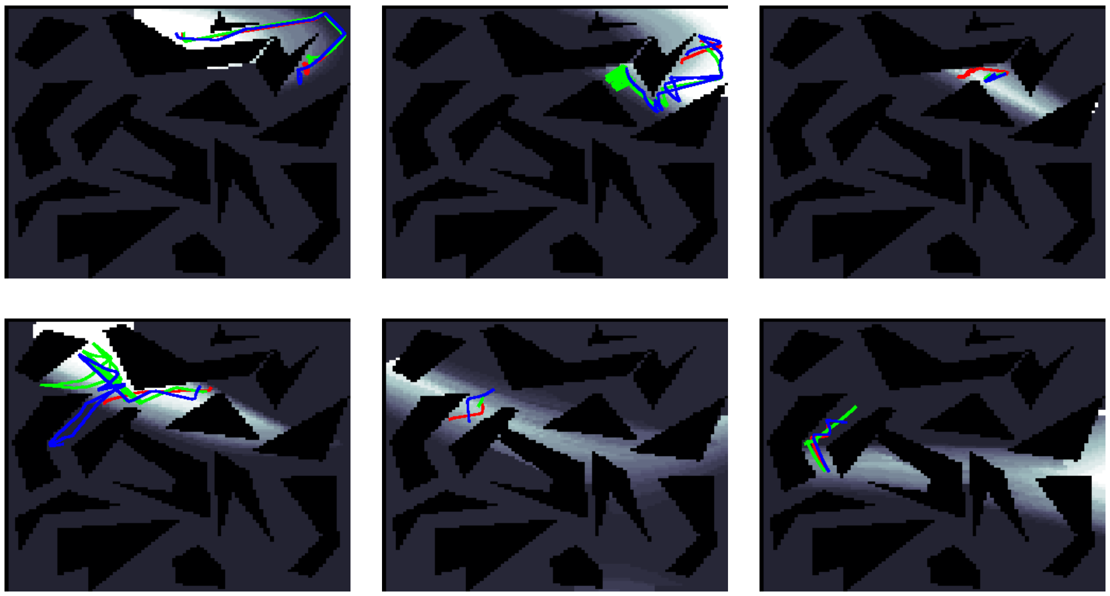


Figure 3.6. Initiation sets and sample policy trajectories for the options used in Figure 3.5. Each initiation set is a density plot, with lightness increasing proportionally to the number of points in the set for a given (x, y) coordinate, sampling $\dot{x}$ and $\dot{y}$ over $\{-1, -\frac{1}{2}, 0, \frac{1}{2}, 1\}$.

3.5 Related Work

As we have already seen in Chapter 2, there has been a great deal of research on skill acquisition in discrete domains. By contrast, we know of very little work on skill acquisition in continuous domains where the skills or action hierarchy are not designed in advance.

Mugan and Kuipers (2009) used learned qualitative models of a continuous state space to derive options. A qualitative model of a continuous variable discretizes it using a set of important values (called landmarks) that partition the variable into a finite number of states. The agent learns a forward model (in the form of a dynamic Bayesian network, or DBN) for each qualitative state, and uses it to build an option to reach that qualitative state. The option policies can then be obtained using dynamic programming. The use of a

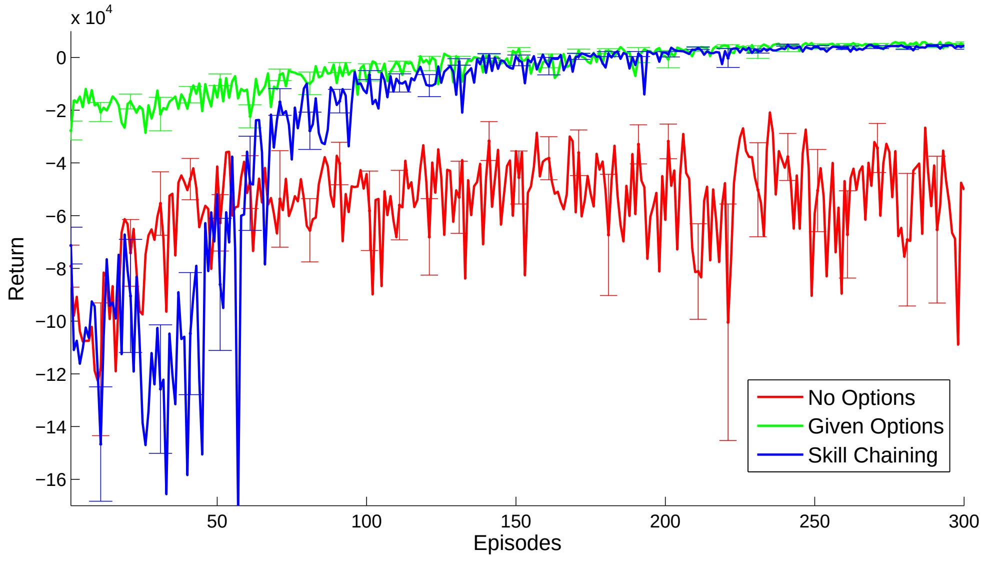


Figure 3.7. Performance in the second Pinball instance (averaged over 100 runs) for agents employing skill chaining, agents with given options, and agents without options.

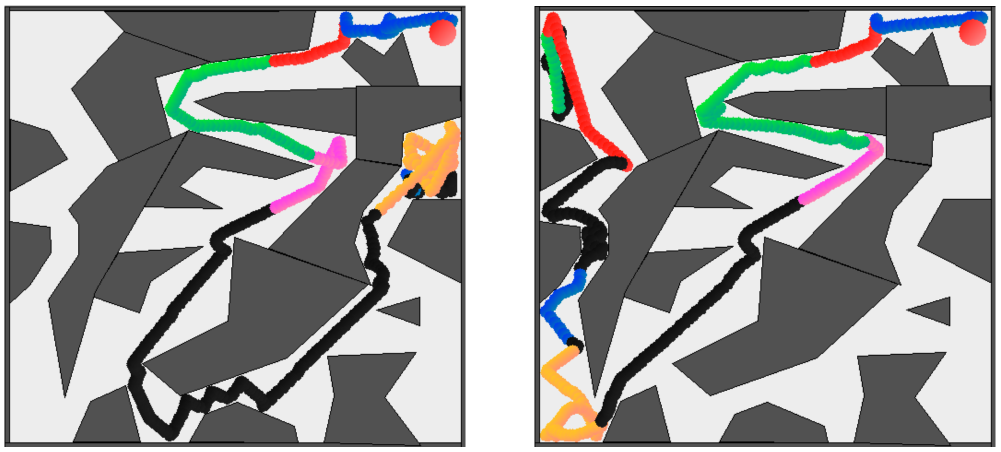


Figure 3.8. Sample solution trajectories from different start states for the second Pinball instance. Acquired options executed along the sample trajectory are shown in different colors; primitive actions are shown in black.

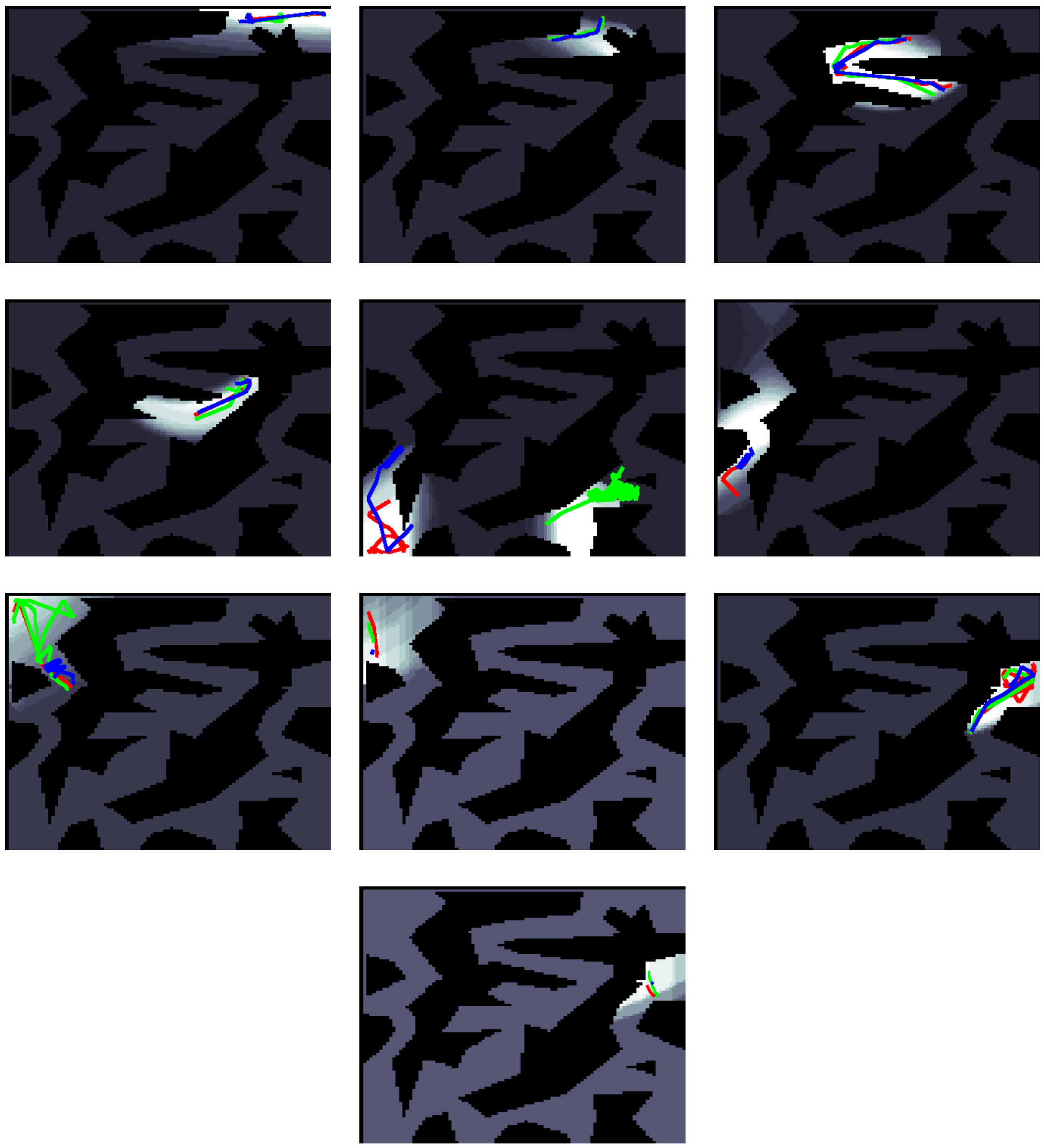


Figure 3.9. Initiation sets and sample policy trajectories for the options used in Figure 3.8.

DBN has the additional advantage of determining the relevant state variables and initiation sets for the options. However, this approach requires a domain where creating options to reach discretized values of individual state variables makes sense; such an assumption is similar to but stronger than that underlying Factored MDPs (Boutilier *et al.*, 1995). In addition, it remains unclear whether learning such models will be feasible for high-dimensional problems.

Neumann *et al.* (2009) propose a method similar to ours, where an agent learns to solve a complex task by sequencing motion templates. The motion templates are parametrized policies (modeled as options) designed specifically for the task. The agent must learn both the motion template parameters and which motion templates to execute for each state, although its choices are constrained. In addition, because primitive actions are not available and the agents must learn the correct parameters for each template, although their eventual performance is better than that of flat learning, their initial performance is significantly worse.

The notion of arranging controllers so that the execution of one allows another be executed has been present in robotics for a long time, known as pre-image backchaining or sequential composition (Lozano-Perez *et al.*, 1984; Burridge *et al.*, 1999). In such work the controllers and their pre-images (initiation sets) are typically given; our work can be thought of as learning controllers (and their initiation sets) that are suitable for sequential composition.

The most recent and most similar work in this line is by Tedrake (2009), who builds a similar tree to ours in the model-based control setting, where the controllers are locally valid LQR (linear quadratic regulator) controllers and their regions of stability (similar in function to initiation sets) are Lyapunov functions which can be computed using convex optimization. The tree is built by selecting random starting points in the state space, using motion planning to obtain a trajectory from the starting pointing to the nearest region of stability, and then covering the trajectory with local LQR controllers. New starting points are selected until the state space is probabilistically covered. This method is able to obtain nonlinear but certifiably robust policies. By contrast, our work does not require a model and may find superior (optimized) policies but does not make any formal guarantees.

3.6 Discussion and Conclusions

The performance gains demonstrated in this chapter show that skill chaining (at least using an end-of-episode initial target event) can significantly improve the performance of a reinforcement learning agent in a challenging continuous domain, by breaking the solution into subtasks and learning lower-order option policies for each one.

Further benefits could be obtained by including more sophisticated initial target events: *any* target events can be substituted for (or added to) the end-of-episode one used here. We expect that a method that identifies regions likely to lie on the solution trajectory before a solution is found would result in the kinds of very early performance gains sometimes seen in discrete skill discovery methods (e.g., Şimşek *et al.* (2005)).

In the experiment reported here, the primary benefit of skill chaining was that it reduces the burden of representing the task's value function, allowing each option to focus on representing its own local value function and thereby achieving a better overall solution. This implies that skill acquisition is best suited to high-dimensional problems where a single policy cannot be well represented using a feasible number of basis functions in reasonable time. In tasks where a good solution can be well represented using a low-order function approximator we do not expect to see any benefits when acquiring skills using skill chaining.

Similar benefits may be obtainable using representation discovery methods (Mahadevan, 2008), where new basis functions are constructed to compactly represent complex value functions. We expect that such methods will prove most effective for extended control problems when used in conjunction with a skill acquisition algorithm, where they can tailor a separate representation for each skill rather than a single representation for the entire problem.

In this chapter we used a a "lightweight" function approximator to represent option value functions. In very complex domains such as robotics where the state space may contain hundreds or even thousands of state variables, we require a more sophisticated approach that exploits the notion that although a difficult and extended control task may not be reducible to a feasibly sized state space, it can often be split into subtasks that are.

A key advantage of hierarchical reinforcement learning for extended control problems in high-dimensional spaces is the ability of each option to use not just its own function approximator, but also its own abstraction. In the next chapter, we introduce *abstraction selection*, an approach where an agent with a library of abstractions can take advantage of the experience used to initialize a new option to additionally select a suitable abstraction with which to efficiently learn that option.

CHAPTER 4

SELECTING SKILL-SPECIFIC ABSTRACTIONS

In the previous chapter we introduced a skill discovery method that can be applied to the robot skill acquisition problem to identify target skills through interaction with the environment. Once such a skill has been identified, we can in principle apply any reinforcement learning method to learn its policy; thereafter, we can learn its initiation set and use it, in turn, to create new skills. However, real-time policy learning in high-dimensional continuous reinforcement learning domains (like robots) remains extremely challenging.

A key approach to solving such problems is the use of an *abstraction* that reduces the number of state variables used to solve the problem by discarding irrelevant variables. It is typically difficult to find a single abstraction that applies to all of a complex problem: the problem itself may simply be intrinsically high-dimensional and therefore hard to solve monolithically. Nevertheless, it may at the same time consist of several subproblems—which we can capture as skills—each of which can be learned more efficiently using only a small (but skill-specific) set of state variables.

Thus, a complex human task such as driving to work that seems infeasible to learn as a single overall problem might be broken into a series of small skills (unlocking the car, starting the car, navigating to work, parking, walking inside, etc.), each of which is manageable on its own. This is a key advantage of skill hierarchies: we can break complex problems into a series of subproblems, each of which can be solved using its own abstraction.

The question then becomes: *how can we obtain skill-specific abstractions?* In principle, each skill could build its own abstraction from scratch during learning. However, for large enough state spaces (e.g., robots), building such an abstraction may impose infeasible sample, computation and memory costs. In this chapter we propose an intermediate solution: that an agent should have a *library* of abstractions available to it, from which it can select an appropriate abstraction when learning a new skill. We introduce an algorithm for *abstraction selection*, where an agent uses sample trajectories (such as those obtained during the gestation process in the previous chapter) to select an appropriate abstraction for a new skill from a library of pre-built abstractions.

In the following sections we formally define the abstraction selection problem, frame it as a model selection problem, and introduce an incremental abstraction selection algorithm. We evaluate the algorithm empirically in the Continuous Playroom, a reasonably large continuous reinforcement learning domain, and show that it selects an appropriate abstraction using very little sample data, thereby improving skill learning performance and rendering the overall learning task feasible.

4.1 Abstraction Selection

Humans have many sensory inputs and degrees of freedom, which viewed naively represent a very large state space. Although such a state space seems too large for feasible learning, specific sensorimotor skills almost always involve a small number of sensory features. One of the ways in which we might draw inspiration from human learning is the extent to which humans learning motor skills seem to ignore most of the sensor and motor features in their environment.

In reinforcement learning, the use of a smaller set of variables to solve a large problem is modeled using the broad notion of *abstraction*. For this work, we consider an abstraction M to be a pair of functions (σ, τ), where

$$\sigma : S \rightarrow S'$$

is a mapping from the overall state space S to a smaller state space S' (often simply a projection onto a subspace spanned by a subset of the variables in S), and

$$\tau : A \rightarrow A'$$

is a mapping from the full action space A to a smaller action space A' (often simply a subset of A). In addition, we assume that each abstraction has an associated vector of basis functions Φ defined over S' which we can use to define a value function.

When performing policy learning using an abstraction, the agent's sensor input is filtered through its sensor abstraction, σ, and the policy π maps from S' (the output set of σ) to A' (the input set of a motor abstraction, τ). The motor abstraction τ then maps inputs in A' to primitive actions in A. This is depicted in Figure 4.1.

Typically, a reinforcement learning agent tries to build a single abstraction for the entire problem; however, in the hierarchical reinforcement learning setting it may in principle try

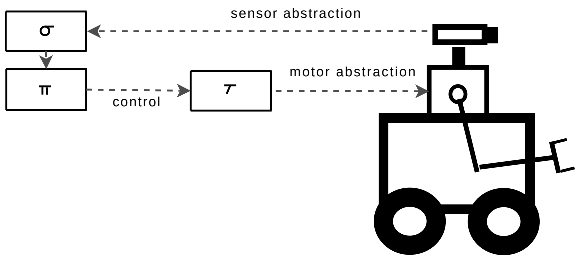


Figure 4.1. Policy learning using an abstraction. The agent's sensor inputs are filtered through its sensor abstraction σ, over which a policy π is defined mapping the abstract sensor signals to motor outputs using the motor abstraction τ.

to build as many abstractions as it has skills. An agent that must solve many problems in its lifetime may thus accumulate a *library* of abstractions that it can later deploy to solve new problems.

We propose that when an agent creates a new option it should create it with an accompanying abstraction, and that if it has a library of abstractions available it can select from among them, refining the selected abstraction through experience if necessary.

An agent using a skill discovery method only creates an option with the goal of entering a particular state region after first reaching that region. Thus, we have a sequence of transitions that ends at the new subgoal, and we may consider it a sample trajectory for the option. Assuming the trajectory has m steps, it consists of a sequence of m state-action pairs and resulting rewards:

$$T = \{(\mathbf{s}_1, a_1, r_1), (\mathbf{s}_2, a_2, r_2), \ldots, (\mathbf{s}_m, a_m, r_m)\}.$$

Given a library of abstractions, we can apply abstraction i to the sample trajectory and obtain:

$$T_i = \{(\mathbf{s}_1^i, a_1^i, r_1), (\mathbf{s}_2^i, a_2^i, r_2), \ldots, (\mathbf{s}_m^i, a_m^i, r_m)\},$$

where $(\mathbf{s}_k^i, a_k^i, r_k) = (\sigma_i(\mathbf{s}_k), \tau_i(a_k), r_k)$ is a state-action-reward tuple obtained from abstraction i describing the kth state-action pair in the trajectory.

4.1.1 Abstraction Selection as Model Selection

If we are given the entire trajectory at once (or can store it all in memory), we can compute Monte Carlo state-value samples (s_i, R_i) for each $1 \leq i \leq m$, where $R_i = \sum_{j=i}^{m} \gamma^{j-i} r_j$. Learning a value function is then simply a regression problem where we must learn a mapping from states to observed return. Selecting the appropriate set of basis functions in regression is known as *model selection* (Bishop, 2006); we can thus cast our problem as a model selection problem with the sets of basis functions corresponding to each abstraction as candidate models.

A common model selection criterion is the *Bayesian Information Criterion* (or BIC) (Schwarz, 1978), which states that:

$$\ln p(D|M_i) \approx \ln p(D|\theta_{MAP}, M_i) - \frac{1}{2}|M_i| \ln m, \tag{4.1}$$

where D is the data, M_i is abstraction i, $p(D|\theta_{MAP}, M_i)$ is the likelihood of D given the maximum a posteriori value function parameters θ_{MAP} for abstraction i, $|M_i|$ is the number of parameters in abstraction i and m is the sample size.

BIC has two important properties that make it well suited to our purposes. First, it controls for different sized abstractions (through the second term on the right hand side of Equation 4.1). Second, if we wish we can use the results of Equation 4.1 and Bayes Rule to obtain a probability that each abstraction is the correct one for a particular skill, naturally incorporating prior beliefs the agent may have about suitable abstractions.

We are now faced with the task of building a suitable statistical model for the data. Since we are using linear function approximation, a linear regression model is a natural fit. In statistical models of linear regression we typically make two assumptions: that our samples are independent, and that they are identically distributed with the target variable (in our case the value function) drawn from a Gaussian distribution with mean $\mathbf{w} \cdot \Phi_j(s)$ at state s (where Φ_j is a set of basis functions—in our case those associated with abstraction j) and variance β^{-1}. This leads to a likelihood function of the form:

$$p(D|M_i, \mathbf{w}, \beta) = \prod_{i=1}^{m} \mathcal{N}(\mathbf{w} \cdot \Phi_j(s_i) - R_i, \beta^{-1}), \tag{4.2}$$

and corresponding log-likelihood function

$$\ln p(D|M_i, \mathbf{w}, \beta) = \frac{m}{2} \ln \beta - \frac{m}{2} \ln 2\pi - \frac{1}{2} \sum_{i=1}^{m} \beta(\mathbf{w}.\Phi_j(s_i) - R_i)^2. \tag{4.3}$$

Notice that we can maximize this equation by minimizing $\sum_{i=1}^{m}(\mathbf{w}.\Phi_j(s_i) - R_i)^2$. Thus, the maximum likelihood solution for the standard statistical model of regression is obtained by minimizing the sum of squared errors over the sample data points.

Although this model is appealing, in the case of the Monte Carlo samples of return in a Markov Decision Process both assumptions that we make to obtain it are violated. First, since the return samples are obtained along the same trajectory, they are not independent. However, since the independence assumption is crucial to obtaining a model with a closed form solution and avoid having to use an expensive iterative approximation algorithm such as EM (Bishop, 2006), we retain it.

Second, in the case of an MDP Monte Carlo sample, all of the sample points do *not* have the same variance; a sample point's variance increases with the length of the trajectory following it. For simplicity, we model this increasing variance geometrically, modeling the increase in variance of step i using a multiplicative factor ρ_i^{-1}, $0 \leq \rho_i \leq 1$. Thus, sample i in a trajectory of length m is assumed to have variance $(\beta\bar{\rho}_i)^{-1}$, where:

$$\bar{\rho}_i = \prod_{j=1}^{m} \rho_i. \tag{4.4}$$

The resulting log likelihood for model M_i is:

$$\ln p(D|M_i, \mathbf{w}, \beta) = -\frac{\beta}{2} e_i + \frac{m}{2}(\ln \frac{\beta}{2\pi}) + \frac{1}{2}\sum_{i=1}^{m} \ln \bar{\rho}_i, \tag{4.5}$$

where β^{-1} is the variance, $\mathbf{w}$ is the function approximation weight vector, and

$$e_i = \sum_{j=1}^{m} \bar{\rho}_j [\mathbf{w} \cdot \Phi_i(\mathbf{s}_j) - R_j]^2 \tag{4.6}$$

is the summed weighted squared error. Thus, this corresponds to a weighted least squares regression model with sample i assigned weight $\bar{\rho}_i$. Note again that since maximizing Equation 4.5 with respect to $\mathbf{w}$ is equivalent to minimizing the (weighted) value function error (Equation 4.6), the resulting $\mathbf{w}$ vector can still be used as an initial value function parameter.

Since we wish to do selection with very little data if possible, we must avoid overfitting. We therefore regularize by assuming a zero-mean Gaussian prior with precision α for each element of $\mathbf{w}$ (this can be discarded if desired by setting $\alpha = 0$, which results in no regularization). The maximum a posteriori (MAP) parameters are then:

$$\begin{aligned} \beta = &\ e_i^{-1} m \\ \text{and} \quad \mathbf{w} = &\ (A + \eta I)^{-1} b, \end{aligned} \tag{4.7}$$

where

$$\begin{aligned} \mathbf{A} = &\ \textstyle\sum_{j=1}^{m} \bar{\rho}_j \Phi_i(\mathbf{s}_j) \Phi_i^T(\mathbf{s}_j) \\ \text{and} \quad \mathbf{b} = &\ \textstyle\sum_{j=1}^{m} \bar{\rho}_j R_j \Phi_i(\mathbf{s}_j), \end{aligned} \tag{4.8}$$

with $\eta = \alpha(\beta^{-1})$. We can also rewrite e_i as:

$$e_i = \mathbf{w}^T \mathbf{A} \mathbf{w} - 2\mathbf{w} \cdot \mathbf{b} + R_c, \tag{4.9}$$

with $\mathbf{A}$ and $\mathbf{b}$ defined as before, and $R_c = \sum_{j=1}^{m} \bar{\rho}_j R_j^2$.

4.1.2 Incremental Abstraction Selection

Thus far we have assumed that we are given the entire trajectory at once (or can store it all in memory), and can thereby compute the Monte Carlo return R_i for each sample (s_i, r_i). However, this is neither desirable nor necessary. From Equations 4.7 and 4.9, we see that in order to do selection incrementally, we are only required to compute sufficient statistics

$\mathbf{A}, \mathbf{b}$ and R_c incrementally in order to solve for the model parameters $\mathbf{w}$ and β and weighted error e_i, and thereby compute the BIC approximation to log likelihood (Equation 4.1, via Equation 4.5).

Following Boyan (1999), we can accomplish this by the incremental (least-squares weighted TD(1)) algorithm given in Figure 4.2. The algorithm is run simultaneously for each abstraction while the agent is interacting with the environment, and then computes the BIC for each abstraction in one step when a selection decision is required; the agent selects the abstraction with the highest BIC.

More than one sample trajectory may be available, or may be required to produce robust selection. Given p samples, the algorithm can be modified to run lines 1 to 14 (initialization and incoming sample processing) separately for each sample trajectory, resulting in separate sample statistics $\mathbf{A}_i$, $\mathbf{b}_i$ and R_{ci} for each sample trajectory $i \in [1, p]$. We can the run lines 15 to 21 (computing the parameters and returning the BIC measure) using summed statistics $\mathbf{A} = \sum_{i=1}^{p} \mathbf{A}_i$, $\mathbf{b} = \sum_{i=1}^{p} \mathbf{b}_i$, and $R_c = \sum_{i=1}^{m} R_{ci}$. This performs a fit and computes the BIC measure over all p trajectories simultaneously.

The algorithm uses $O(q_i^2)$ memory, $O(q_i^2)$ time at each step and $O(q_i^3)$ time at selection for each sensorimotor abstraction i using a function approximator with $q_i = |\Phi_i|$ features. Since we most likely wish for selection to be fast relative to learning, we can use a function approximator of the same type as used for learning but with fewer terms for selection, and then upgrade it for learning. This additionally reduces the sample complexity for successful selection, as we show in the following section.

```
Initialization
A = zero_matrix(|Φ_i|, |Φ_i|)
b = zero_vector(|Φ_i|)
z = zero_vector(|Φ_i|)
$R_c = R_z = g = 0$
Iteratively handle incoming samples—update sufficient statistics
for each incoming sample ($\mathbf{s}_t$, $a_t$, $r_t$, $\rho_t$) do
    $\Phi_t = \Phi_i(\mathbf{s}_t)$
    $\mathbf{A} = \rho_t(\mathbf{A} + \Phi_t\Phi_t^T)$
    $\mathbf{b} = \rho_t(\mathbf{b} + r_t\mathbf{z} + r_t\Phi_t)$
    $\mathbf{z} = \gamma\rho_t(\mathbf{z} + \Phi_t)$
    $g = \rho_t(\gamma^2 g + 1)$
    $R_c = \rho_t R_c + g r_t^2 + 2\rho_t r_t R_z$
    $R_z = \gamma(\rho_t R_z + g r_t)$
Compute weights, error and variance after $m$ samples
$\mathbf{w} = (\mathbf{A} + \eta\mathbf{I})^{-1}\mathbf{b}$
$e = \mathbf{w}^T\mathbf{A}\mathbf{w} - 2\mathbf{w}\cdot\mathbf{b} + R_c$
$\beta = \frac{m}{e}$
Compute log likelihood and BIC, ignoring quantities constant across abstractions
$ll = -\frac{\beta}{2}e + \frac{m}{2}\ln\beta$
return $ll - \frac{1}{2}|\Phi_i|\ln m$
```

Figure 4.2. An incremental algorithm for computing the BIC value of an abstraction i given a successful sample trajectory.

4.2 The Continuous Playroom Domain

The Continuous Playroom is a real-valued version of the Playroom domain (Singh *et al.*, 2004). It consists of three effectors (an eye, a marker, and a hand), five objects (a red button, a green button, a light switch, a bell, a ball, and a monkey) and two environmental variables (whether the light is on, and whether the music is on).

The agent is in $1x1$ room, and may move any of its effectors 0.05 units in one of the usual four directions. When both its eye and hand are over an object it may additionally interact with it, but only if the light is on (unless the object is the light switch). Figure 4.3 shows an example configuration.

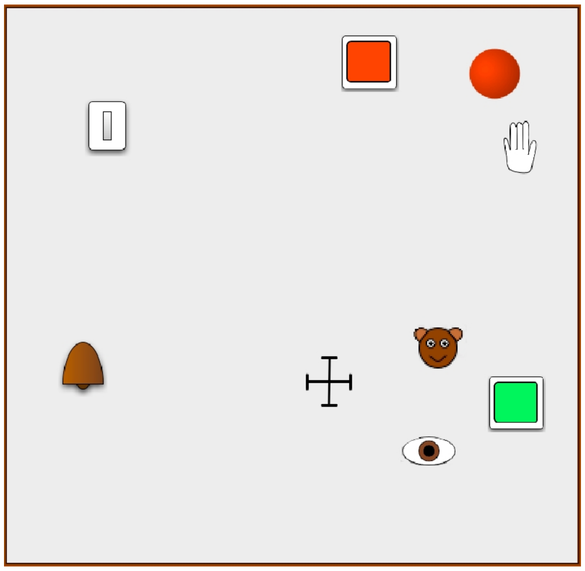


Figure 4.3. An example Continuous Playroom.

Interacting with the green button switches the music on, while the red button switches the music off. The light switches toggles the light. Finally, if the agent interacts with the ball and its marker is over the bell, then the ball hits the bell. Hitting the bell frightens the monkey if the light is on and the music is on and causes it to squeak, whereupon the agent receives a reward of $100,000$ and the episode ends. All other actions cause the agent to receive a reward of -1. At the beginning of each episode the objects are arranged randomly in the room so that they do not overlap.

The agent has 13 possible actions (3 effectors with 4 actions each, plus the interact action), and a full description of the Playroom requires 18 state variables—x and y pairs for three effectors and five objects (since we may omit the position of the monkey) plus a variable each for the light and the music. Since the domain is randomly re-arranged at the beginning of each episode, the agent must learn the relationships between its effectors and each object, rather than simply the absolute location for its effectors. Moreover, the settings of the light and music are crucial for decision making and must be used in conjunction with object and effector positions. Thus, for task learning we use 120 state variables—for each of the four settings of the lights and music we use a set of 30 variables representing the difference between each combination of object and effector (Δx and Δy for each object-effector pair, so 5 objects $\times 3$ effectors $\times 2$ differences $= 30$).

The Continuous Playroom is a good example of a domain that should be easy—and is easy for humans–but is made difficult by the large number of variables and the interactions between them (e.g., between Δx and Δy values for an object-effector pair) that cannot all be included in the overall task function approximator: a 1st order Fourier Basis over 120 variables that does not treat each variable as independent has 2^{120} features. Thus, it is a domain in which options can greatly improve performance, but only if those options are themselves feasible to learn.

4.2.1 Target Options and Abstraction Library

Originally, the playroom domain assumed that the agent was given 15 initial options, one for moving each effector over each object (Singh *et al.*, 2004). In this work, we assume that the agent starts with primitive actions only, and that some option discovery method creates a new option for moving each effector over each object when the agent first successfully does so. Our task is then to efficiently learn these option policies using abstraction selection.

We equip the agent with an abstraction library consisting of 17 abstractions. Since the domain consists of objects and effectors, we include abstractions for each of the 15 object-effector pairs, with each abstraction consisting of just two variables: Δx and Δy for the object-effector pair. We also include a random abstraction (two state variables with values selected uniformly at random over $[0, 1]$ at each step), and a null abstraction, which uses all 120 variables.

This set of options and abstractions are difficult to perform abstraction selection over because while the features of some abstractions (those referencing completely distinct objects-effector pairs) are not correlated with the features of the correct abstraction and are therefore easy to differentiate, the features of some (where either the effector or object is shared) *are* correlated with the features of the correct abstraction. This correlation makes them hard to differentiate from the correct abstraction. Figure 4.4 illustrates the value functions that might arise in this case.

In domains where the feature values of the incorrect abstractions are uncorrelated to return (e.g., when the features are random or uniform over the sample trajectory), we can expect to perform abstraction selection with high accuracy using very few sample trajectories–perhaps even just one. When there are correlations (such as those intentionally included in the Playroom domain) selection becomes much harder.

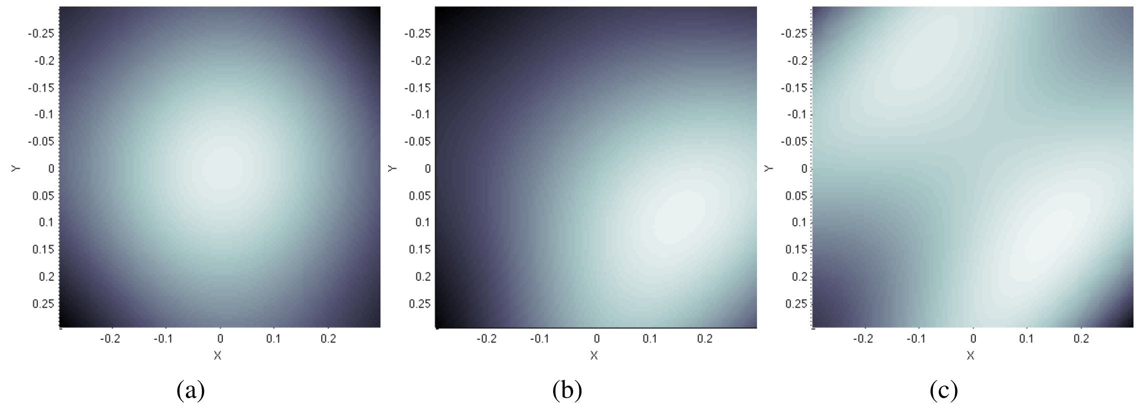


Figure 4.4. Value functions produced by the correct abstraction and a correlated abstraction in the Playroom domain. The correct abstraction has a high value only when the effector is directly over the object (a). However, *in the same configuration of objects and effectors*, an abstraction that shares either an object or an effector with the correct abstraction will also see a systematic relationship between its features and return (b), according to the relative position of its object-effector pair and the correct abstraction's object-effector pair. The difference only becomes clear when the agent experiences a new Playroom configuration (c), where the peak for the correlated abstraction moves. This temporarily introduces a second peak into the value function, until new samples are obtained near the original peak. These conflicting samples will introduce error and lower the the abstraction's probability of being selected.

4.2.2 Implementation Details

For overall task learning in the Playroom domain, we employ a 10th-order independent Fourier Basis over the 120 task features (1320 basis functions per action), with Sarsa(λ) ($\alpha = 0.00005$, $\gamma = 1$, $\lambda = 0.9$, $\epsilon = 0.01$).

When learning using an option with its own abstraction, we used a full 10th-order Fourier Basis (121 basis functions per action) with Sarsa(λ) ($\alpha = 0.0025$, $\gamma = 1$, $\lambda = 0.9$, $\epsilon = 0.01$). For option learning without an abstraction, we discard the lights and music variables and learn using the 30 difference variables, using a 10th-order independent

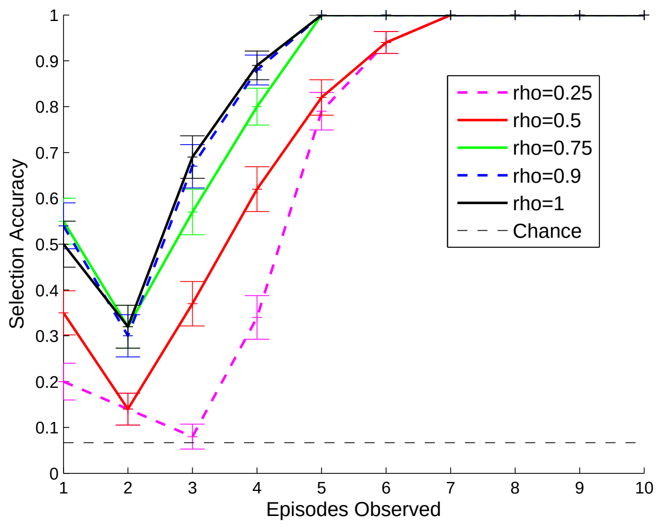
Fourier Basis (330 basis functions per action) and Sarsa(λ) ($\alpha = 0.001$, $\gamma = 1$, $\lambda = 0.9$, $\epsilon = 0.01$). The learning rate and function approximation order parameters were chosen for best performance in each case.

We use regularization parameter $\eta = 0.0001$ when performing abstraction selection. When performing abstraction selection while learning a policy for the entire Playroom task, agents initially created options when first encountering the a target region, without using an abstraction. Abstraction selection was performed when the agent had experienced at least 5 episodes, and had a confidence above 95% that it has the correct abstraction. The agent could later switch abstractions if it decided on a different abstraction with at least 95% confidence given more data.

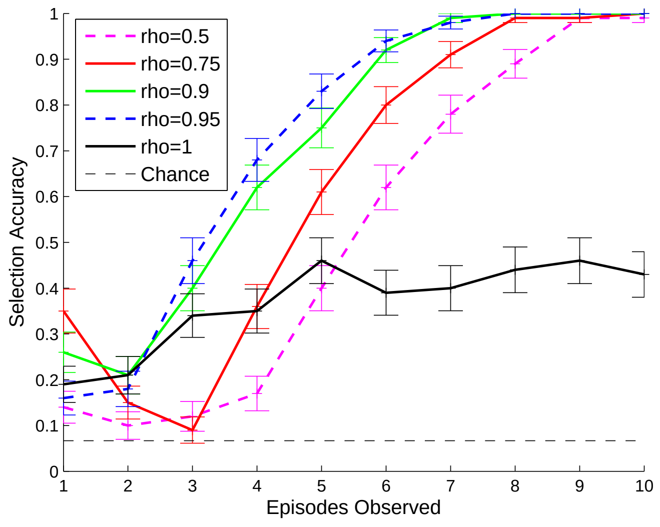
Although the algorithm given in Figure 4.2 supports the use of a distinct ρ value for every transition, for simplicity we assume a constant ρ, and determine the best value empirically.

4.3 Results

We first examine the effect of various values of ρ on the number of episodes required to make an accurate selection. Plots showing the correct fraction of abstraction selections (averaged over 100 runs) made for varying numbers of sample trajectories are given in Figure 4.5. We show graphs for trajectories obtained from the optimal option policy and trajectories obtained by selecting randomly from the 13 available actions at each timestep—since reinforcement learning agents typically start with a random policy and progress toward an optimal policy over time, the trajectories we see in practice will lie somewhere between these two extremes.



(a)



(b)

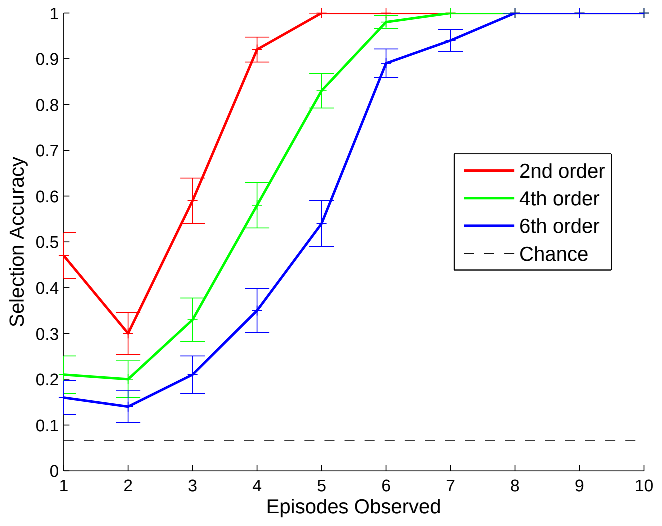
Figure 4.5. Selection accuracy as a function of the number of sample episodes observed for various values of ρ. The sample episodes are generated using either optimal (a) and random (b) policies.

As expected, abstraction selection performs best with $\rho = 1$ when using optimal sample trajectories (Figure 4.5a), and its performance degrades with decreasing ρ. This is because the underlying policy is not stochastic and does not make errors, and hence the variance of each value function sample along the trajectory is the same. However, we see in Figure 4.5b that, although increasing ρ generally improves abstraction selection performance when using trajectories from a random policy, setting $\rho = 1$ obtains very poor results. This reflects the fact that the variance of the value function samples increases with trajectory length when the policy generating that trajectory is imperfect.

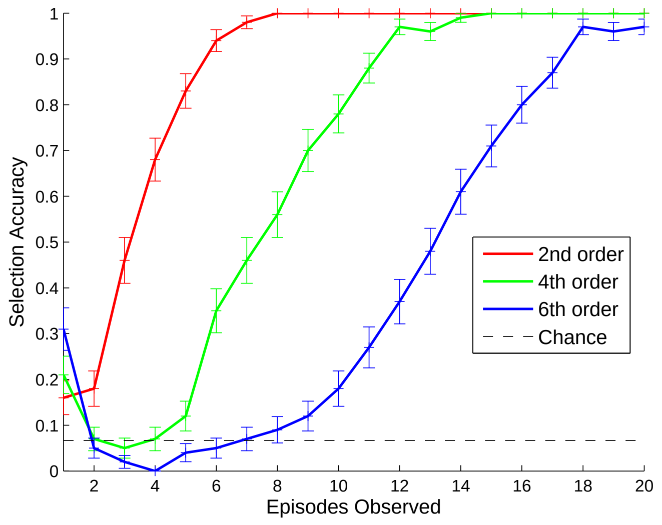
Since we obtain very good performance for both types of policies with $0.9 \leq \rho < 1$, we use $\rho = 0.95$ for the remainder of our experiments. Given this setting, Figure 4.5 shows that we require between 5 and 8 independent sample trajectories to select the correct abstraction 100% of the time, and between 4 and 6 samples to select the correct abstraction roughly 85% of the time.

Similar curves are shown in Figure 4.6 for various orders of function approximator. These curves clearly show that increasing the order of the function approximator also increases the amount of data required to accurately select an abstraction. This difference can be partially ascribed to the higher number of degrees of freedom in the higher-order function approximators, but may also be because their greater flexibility also allows them to model multiple peaks (as shown in Figure 4.4), and thereby requires more samples to introduce error for abstractions that are incorrect but correlated to the correct abstraction.

Given this data, we should use as low an order function approximator as is reasonable for the task. We therefore us a 2nd-order Fourier Basis for selection in the remainder of this



(a)



(b)

Figure 4.6. Selection accuracy as a function of the number of sample episodes observed for various orders of function approximator. The sample episodes are generated using either optimal (a) and random (b) policies.

section (a 1st-order Fourier Basis is not sufficient to represent a reasonable policy for this problem, even for selection).

Figure 4.7 plots agent selection confidence (again averaged over 100 runs) for the correct abstraction, given trajectories from either an optimal or random policy. It shows that the agent's confidence increases as it obtains more sample trajectories, and that this confidence closely tracks the probability of the agent selecting the correct abstraction (compare Figure 4.7 with the curves correspond to using a 2nd-order basis in Figure 4.6). Thus, we can use BIC to compute a selection confidence and make a principled decision about when to commit to an abstraction.[1]

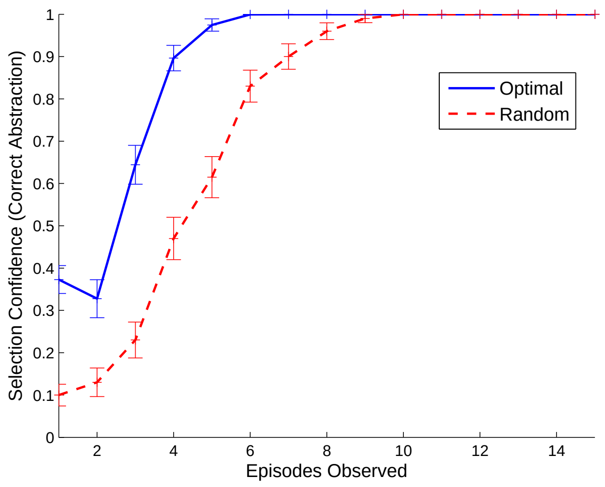


Figure 4.7. Mean agent selection confidence over time for the correct abstraction, given increasing numbers of sample trajectories obtained using either an optimal or a random policy.

[1]In practice, we find it best to additionally require a small minimum number of samples, since BIC can occasionally assign high confidence to the wrong abstraction given very little sample data. We believe this occurs due to the numerical instability of inverting $\mathbf{A}$ when a paucity of sample data causes it to be nearly singular.

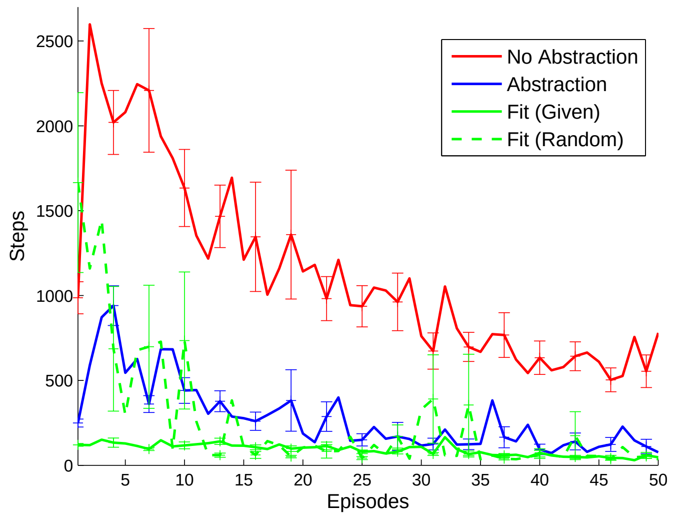


Figure 4.8. Skill learning curves for agents that do not use an abstraction, agents that use an appropriate abstraction, and agents that use the initial policy fit obtained via abstraction selection using 20 sample trajectories from either optimal or random policies.

We show learning curves for an individual skill for agents that do not use an abstraction, agents that use an appropriate abstraction, and agents that use the initial policy fit obtained via abstraction selection in Figure 4.8. These results show that using an appropriate abstraction results in a significant performance boost over not using one—agents that do not employ an abstraction are unable to learn good policies for the skill after 100 episodes. Furthermore, using the value function fit obtained via abstraction selection to initialize the skill value function leads to significant performance improvements, though these depend on the quality of the sample trajectories used for abstraction selection. As expected, initializing the skill value function using the fit obtained from an optimal policy results in very fast learning—the skill policy is learned virtually immediately—while using an initial value function from a random policy initially results in slightly slower learning than learning from scratch with an appropriate abstraction. However, the penalty is temporary and its performance improves, overtaking learning from scratch within a few episodes. This is likely because the value function fit obtained from random trajectories will be noisy and inaccurate in places, causing value function errors that must be learned away before the accurate parts of the value function provide a useful boost in performance.

Finally, Figure 4.9 compares the overall learning curves in the Playroom domain for agents that use abstraction selection against those that do not use abstractions when learning option policies. We also include curves for agents that are given pre-learned optimal option policies, and agents that are immediately given the correct abstraction so that we can compare to the ideal case.

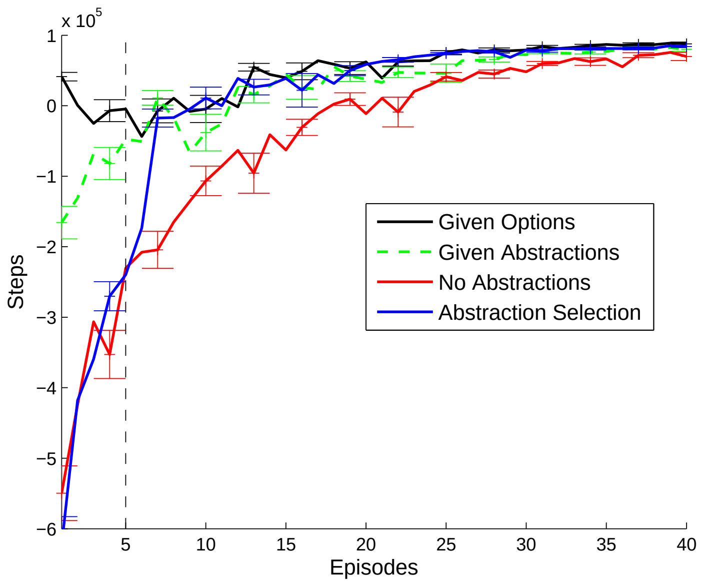


Figure 4.9. Overall learning curves for the agents given optimal option policies, agents given the correct abstraction in advance, agents using no abstractions and agents that perform abstraction selection. The dashed line at 5 episodes indicates the first episode where abstraction selection was allowed; before this line, agents using abstraction selection learned option policies without abstractions.

Figure 4.9 shows that agents performing abstraction selection perform identically to those that do not use abstractions initially, until about 5 episodes, whereafter the agents are allowed to perform abstraction selection and their performance improves relative to agents that do not use abstractions, matching the performance of agents that are given the correct abstractions in advance by 10 episodes. The difference between agents that do not use abstractions and agents that perform abstraction selection is substantial—roughly $100,000$ steps per episode.

4.4 Related Work

Most existing reinforcement learning research on state space reduction takes one of two broad approaches. In *state abstraction* (surveyed by Li *et al.*, 2006), a large state space is compressed by performing variable removal or state aggregation while approximately preserving some desirable property. However, without further information about the state space we cannot examine the effects of abstraction on the properties we are interested in—values or policies—without an existing value function, and so these methods have high computational and sample complexity.

The major alternative approach is to initially assume that *no* states or state variables are relevant, and then introduce *perceptual distinctions* (McCallum, 1996) by including them when it becomes evident that they are necessary for learning the skill. This requires a significant amount of data and computation to determine which variable to introduce, and then introduces them one at a time, which may require too much experience to be practical for large continuous state spaces.

Recently, a new state-space construction paradigm called *representation discovery* (Mahadevan, 2008) has received a great deal of attention in the reinforcement learning community. Representation discovery aims to construct a set of basis functions that can compactly represent the value function of a given task through experience. The two major representation discovery methods that have been applied to continuous domains are Proto-Value Functions (Mahadevan *et al.*, 2006) and Predictive Projections (Sprague, 2009). Both methods have shown promising results but have sample and time complexity demands that make them infeasible (at least for now) for real-time learning.

The notion of of selecting, rather than constructing, representations in reinforcement learning problems has also recently received some attention. van Seijen *et al.* (2007) describe a method where an agent with multiple representations is augmented with actions to switch between them. This fits neatly into the reinforcement learning framework, but does not appear likely to scale up to large numbers of representations since each new representation adds a new action to every state.

Diuk *et al.* (2009) introduce efficient algorithms for the Adaptive k-Meteorologist problem, where an agent is given a set of classifiers and must select the best predictor from among them. This is applied to a reinforcement learning task to learn a forward model of an action using at most D sensor features (where D is given in advance), which is accomplished by creating all possible classifiers of size D or less and selecting the one with the lowest error. This method could be used to select feature subsets for value function approximation, in which case it would result in an algorithm similar to ours (though developed independently); it would have higher computational complexity but provide probabilistic performance guarantees.

Finally, feature selection algorithms (Kolter and Ng, 2009; Johns and Mahadevan, 2009; Johns *et al.*, 2010) select features from a feature dictionary to compactly but accurately represent a value function. These algorithms were developed after the work reported here and have only been applied to relatively small problems. They are expected to face scaling difficulties for larger problems since they perform selection at the *feature level* rather than the state-variable level (or the sets-of-state-variables level), and the size of the feature dictionary increases exponentially with the number of state variables.

State abstraction, perceptual distinction and representation discovery methods are usually applied monolithically to a single large problem. However, some hierarchical reinforcement learning methods make use of skill-specific abstractions. Most prominently, the MAXQ hierarchical reinforcement learning formalism (Dietterich, 2000) assumes a hand-designed abstraction for each level in the skill hierarchy. On the other end of the spectrum, Jonsson and Barto (2001) have shown that each option can learn its own abstraction from scratch in discrete domains using a perceptual distinction method.

Until recently, hierarchical reinforcement learning methods with learned or selected abstractions have been used only in relatively small discrete domains (e.g., Jong and Stone, 2005; Luo *et al.*, 2008). Recent independent work by Mugan and Kuipers (2009) and Vigorito and Barto (2009) has shown that learning a Dynamic Bayes Net (DBN) is feasible for small factored continuous domains; a set of skills, each with their own abstraction, can be extracted from such a structure.

4.5 Discussion and Conclusions

For our purposes, the key advantage abstraction selection provides is that it shifts the state-space design problem out of the agent's control loop: instead of having to design a relevant abstraction for each skill as it is discovered, we can provide a library of abstractions and have the agent select a relevant one for each new skill. As we have seen, this can significantly bootstrap learning in high-dimensional state spaces, and it allows us to easily incorporate background knowledge about the problem into the agent.

However, providing a library of abstractions to the agent in advance requires both extra design effort and significant designer knowledge of the environment the agent is operating in. This immediately suggests that the abstraction library should be *learned*, rather than given. Just as an agent should learn a library of applicable skills over its lifetime, so should it also learn a library of suitable abstractions over its lifetime—because an agent's abstraction library determines which skills it can learn quickly. We expect that learning an abstraction library is feasible using unsupervised learning techniques, but only on a much longer timescale than learning an individual skill. One approach might be to seed the library with a set of basic abstractions, which the agent refines (perhaps using a representation discovery algorithm) during skill learning. The resulting refined abstractions could then be added back into the abstraction library, resulting in a fairly direct form of *representation transfer* (Ferguson and Mahadevan, 2006; Taylor and Stone, 2007).

Abstraction selection and feature selection aim to solve the same problem, but differ in the level at which selection is performed. The primary advantage that abstraction selection has over selection at the feature level is that by pre-packaging features into groups, we avoid having to deal with all the interaction terms between all state variables. Recall that a

kth-order Fourier basis defined over n state variables results in $(k+1)^n$ basis functions, of which all but $(k+1)n$ are interaction terms. Feature selection in this space rapidly becomes infeasible as the number of task variables grows because increasing n is an increase in the exponent. By contrast, if we have l abstractions, each of size at most $m \ll n$, then each abstraction has at most $(k+1)^m$ basis functions, as we do not consider interaction terms between variables not in the same abstraction. This results in $l(k+1)^m$ total features, and so increasing the number of abstractions only adds multiples of $(k+1)^m$ features. Moreover, since each abstraction can be evaluated completed independently, given enough processors we can perform abstraction selection using only the time required to deal with $(k+1)^m$ features. Of course, this relies on having sufficient background knowledge about the task to remove most combination of variables from consideration; if we have no such knowledge and must consider all combinations of variables as valid abstractions then we immediately obtain exponentially many abstractions.[2]

In the Playroom domain, abstraction selection results in performance benefits when compared to not using an abstraction. However, in this case the target options were specifically chosen so that it is possible to learn the individual option policies *without* using an abstraction: each option policy can first minimize the x distance to the target and then, independently, minimize the y distance. An option without an abstraction (that therefore uses a function approximator that treats each feature as independent, as we are forced to do for even moderate numbers of features) can thus still learn a reasonable option policy; this allows us to make a comparison between agents that do and do not use abstractions.

[2]A potentially interesting compromise might be to perform selection at the *state variable* level, although it remains unclear how interaction terms might be handled.

However, in many domains feature interactions are essential for learning option policies. In such domains abstraction selection may transform a problem that cannot be solved into one that can, rather than merely resulting in performance benefits.

The most significant disadvantage abstraction selection has when compared to feature selection is that the agent's skill acquisition abilities are constrained by the abstractions available to it—if it has unsuitable abstractions then it may not be able to acquire any task-relevant skills at all. We expect that in such situations it may be best to attempt skill acquisition using the most suitable abstraction available and gradually refine it over time using a representation discovery method; although skill learning will be slow it should be no slower than learning the abstraction from scratch. Moreover, once the abstraction has been created it can be added to the abstraction library for use in learning later skills.

In the skill chaining method described in the previous chapter, the initiation set for each option was learned using trial-and-error. When using abstraction selection, we expect that the available abstraction library will dominate the shape of each skill's initiation set, because each abstraction will only be able to support a policy locally. Although this may initially seem like a drawback, we believe it to be a feature: when using both skill chaining and abstraction selection, we can obtain a sequence of skills broken up by relevant abstraction, which is a natural way to segment extended policies. Ultimately, we expect that this kind of abstraction-based skill acquisition will provide a natural and simple way of scaling up control learning methods to high-dimensional problems.

CHAPTER 5

CONSTRUCTING SKILL TREES FROM DEMONSTRATION

Chapter 3 introduced skill chaining, a general method for skill discovery in continuous domains. When combined with abstraction selection (introduced in Chapter 4), skill chaining can in principle be used to acquire skills with their own abstractions in high-dimensional, continuous domains like robotics. These methods should allow us to scale up to such domains by adaptively segmenting complex policies that are high-dimensional when represented monolithically into sequences of much lower-dimensional skills with policies that are easier to represent and learn.

However, performing skill chaining iteratively is slow: it creates skills sequentially, and requires several episodes to learn a new skill's policy followed by a further several episodes to learn by trial and error where it can be successfully executed. While this is reasonable for many problems, in domains where experience is expensive (such as robotics) we require a faster method.

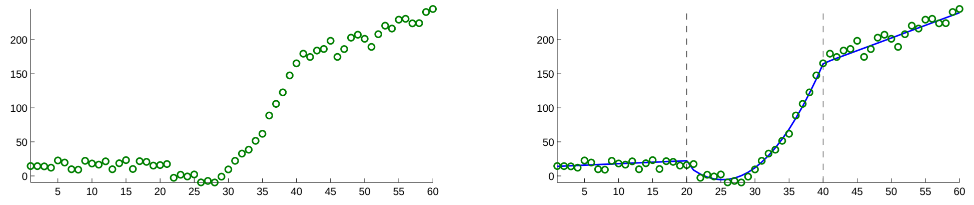
Moreover, there is a growing realization that learning policies for such domains completely from scratch in reasonable time is infeasible, primarily because of the extremely high cost of initially finding any successful policy at all through trial and error. Therefore, we require methods that are able to bootstrap learning by providing a reasonable initial policy from which learning can proceed. The most prominent such methods mirror the way that humans initially learn new control tasks, either through imitating others (Argall *et al.*, 2009), or by initially using a slow (and possibly computationally expensive) controller via kinematic planning or feedback control (Rosenstein and Barto, 2004).

One way to rapidly create skills would be to obtain a sample trajectory from a bootstrapped initial policy, and then segment that trajectory into skills. The question then becomes: *given a sample trajectory, how many options exist along it, and where do they begin and end?* In skill chaining, we segment a policy by creating a new option when either the most suitable abstraction changes, or when the value function (and therefore policy) becomes too complex to represent with a single option. We therefore require a principled way to similarly segment a trajectory.

This is known as a *multiple changepoint detection* problem (Fearnhead and Liu, 2007). In this chapter, we introduce a new skill acquisition method called CST, which uses an incremental MAP changepoint detection method to segment a sample trajectory into a chain of skills (each with an appropriate abstraction); the skill chains resulting from multiple sample trajectories are then merged into a skill tree. CST is incremental and efficient, with a time complexity that is kept to a constant per step using a particle filter. We show that CST can construct a skill tree from demonstration trajectories in Pinball, and that the resulting skills can be refined using reinforcement learning. We further show that it can be used to segment trajectories obtained from a human demonstrator controlling a mobile robot into chains of skills, where each skill is assigned an appropriate abstraction.

5.1 Changepoint Detection

In changepoint detection we are given observed data and a set of candidate models, and assume that the data are segmented such that the data within a segment are generated by a single model. We are to infer the number and positions of the *changepoints*—the points where the data switches from being generated by one model to being generated by another—in the data, and select and fit an appropriate model for each segment. Figure 5.1 shows a simple example.



(a) (b)

Figure 5.1. Data with multiple segments. The observed data (a) are generated by three different models (b; solid line, changepoints shown using dashed lines) plus noise. The first and third segments are generated by a linear model, whereas the second is quadratic.

Unlike the standard regression setting, in reinforcement learning our data is sequentially but not necessarily spatially segmented, and we would like to perform changepoint detection online—processing transitions as they occur and then discarding them. Fearnhead and Liu (2007) introduced online algorithms for both Bayesian and MAP changepoint detection; we use the simpler method that obtains the MAP changepoints and models via an online Viterbi algorithm.

Their algorithm proceeds as follows. We observe data tuples $(\mathbf{x}_t, y_t)$, for times $t \in [1, T]$, and are given a set of models Q with prior $p(q \in Q)$. We model the marginal probability of a segment length l with PMF $g(l)$ and CDF $G(l) = \sum_{i=1}^{l} g(i)$. Finally, we assume that we can fit a segment from time $j+1$ to t using model q to obtain the probability of the data $P(j, t, q)$ conditioned on q.

This results in a Hidden Markov Model where the hidden state at time t is the model q_t and the observed data is y_t given $\mathbf{x}_t$. The hidden state transition probability from time i to time j with model q is given by $g(j-i-1)p(q)$ (reflecting the probability of a segment of length $j-i-1$ and the prior for q). The probability of an observed data segment starting at time $i+1$ and continuing through j using q is $P(i, j, q)(1 - G(j-i-1))$, reflecting the probability of fitting model q to the segment, and the probability of a segment of at least $j-i-1$ steps. Note that a transition between two instances of the same model (but with different parameters) is possible. This model is depicted in Figure 5.2.

We can thus use an online Viterbi algorithm to compute $P_t(j, q)$, the probability of the changepoint previous to time t occuring at time j using model q:

$$P_t(j, q) = (1 - G(t-j-1))P(j, t, q)p(q)P_j^{MAP}, \tag{5.1}$$

and

$$P_j^{MAP} = \max_{i,q} \frac{P_j(i, q)g(j-i)}{1 - G(j-i-1)}, \tag{5.2}$$

for each $j < t$.

At time j, the i and q maximizing Equation 5.2 are the MAP changepoint position and model for the current segment, respectively. We then perform this procedure for time i, repeating until we reach time 1, to obtain the changepoints and models for the entire sequence.

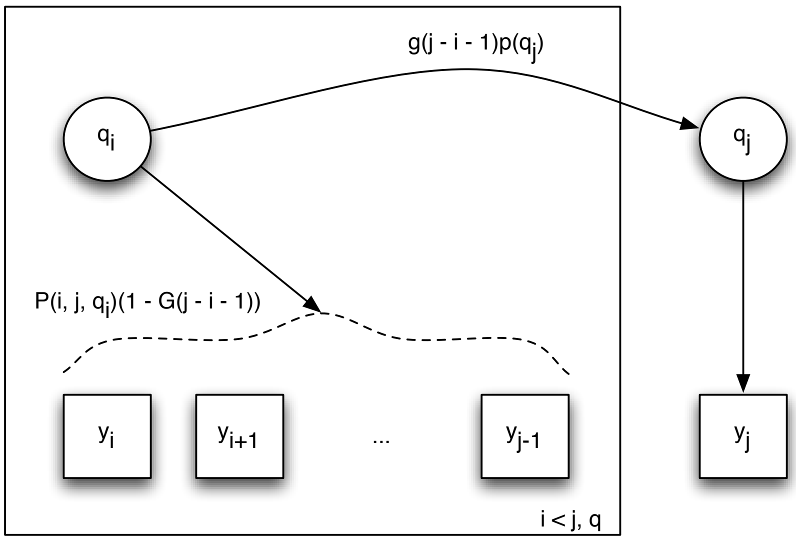


Figure 5.2. The Hidden Markov Model modeling changepoint detection. The model q_t at each time t is hidden, but produces observable data y_t. Transitions occur when the model changes, either to a new model or the same model with different parameters. The transition from model q_i to q_j occurs with probability $g(j-i-1)p(q_j)$, while the emission probability for observed data $y_i, ..., y_{j-1}$ is $P(i,j,q_i)(1-G(j-i-1))$. These probabilities are considered for all times $i < j$ and models $q_i, q_j \in Q$.

Thus, at each time step t we compute $P_t(j,q)$ for each model q and changepoint time $j < t$ (using P_j^{MAP}) and then compute P_t^{MAP} and store it.[1] This requires $O(T)$ storage and $O(TL|Q|)$ time per timestep, where L is the time required to compute $P(j,t,q)$.

Since most $P_t(j,q)$ values will be close to zero, we can employ a particle filter to discard most combinations of j and q and retain a constant number per timestep. Each particle then stores j, q, P_j^{MAP}, sufficient statistics and its Viterbi path. We use the Stratified Optimal Resampling algorithm of Fearnhead and Liu (2007) to filter down to M particles whenever the number of particles reaches N.

[1]In practice all equations are computed in log form to ensure numerical stability.

In addition, we can reduce L to a constant for most models of interest by storing a small sufficient statistic and updating it incrementally in time independent of t, obtaining $P(j,t,q)$ from $P(j,t-1,q)$. This results in a time complexity of $O(NL)$ and storage complexity of $O(Nc)$, where there are $O(c)$ changepoints in the data. Pseudo-code for the resulting algorithm is given in Figure 5.3.

5.2 Constructing Skill Trees from Demonstration Trajectories

In the following sections, we describe the application of changepoint detection to segmenting a single trajectory into a skill chain, and then the process of merging multiple skill chains into a single skill tree.

5.2.1 Constructing a Skill Chain from a Sample Trajectory

To segment a trajectory into skills, we propose performing changepoint detection using the sets of basis functions associated with each abstraction as models, and return R_t (sum of discounted reward) from each time t as the target variable. This provides a natural mapping to reinforcement learning because we are effectively performing changepoint detection on the value function sample obtained from the trajectory; segmentation thus breaks that value function up into simpler value functions, or detects a change in model (and therefore abstraction).

We must thus select an appropriate model of expected skill (segment) length, and an appropriate model for fitting the data. We assume a geometric distribution for skill lengths with

```
1  Initialization
2  particles = $\emptyset$
3  Process each incoming data point
4  for t = 1:T do
5      Compute fit probabilities for all particles
6      for $p \in$ particles do
7          p_tjq = (1 - G(t - p.pos - 1)) $\times$ p.fit_prob() $\times$ model_prior(p.model) $\times$ p.prev_MAP
8          p.MAP = p_tjq $\times$ g(t - p.pos) / (1 - G(t - p.pos - 1))
9      Filter if necessary
10     if $|particles| >= N$ then
11         particles = particle_filter(particles, particles.MAP, $M$)
12     Determine the Viterbi path
13     if t == 1 then
14         max_path = []
15         max_MAP = 1/$|Q|$
16     else
17         max_particle = $\max_{\mathrm{p}}$ p.MAP
18         max_path = max_particle.path $\cup$ max_particle
19         max_MAP = max_particle.MAP
20     Create new particles for a changepoint at time t
21     for $q \in Q$ do
22         new_p = create_particle(model = q, pos = t, prev_MAP = max_MAP, path =
           max_path)
23         particles = particles $\cup$ new_p
24     Update all particles
25     for $p \in$ particles do
26         p.update_particle($\mathbf{x}_t$, $y_t$)
27 Return the most likely path to the final point.
28 return max_path
```

Figure 5.3. Fearnhead and Liu's online MAP changepoint detection algorithm.

parameter p, so that $g(l) = (1-p)^{l-1}p$ and $G(l) = (1-(1-p)^l)$. This gives us a natural way to set p since $p = \frac{1}{k}$, where k is the expected skill length.

Since reinforcement learning in continuous state spaces usually employs linear function approximation, it is natural to use a linear regression model with Gaussian noise as our model of the data. Following Fearnhead and Liu (2007), we assume conjugate priors: the Gaussian noise prior has mean zero, and variance with inverse gamma prior with parameters $\frac{v}{2}$ and $\frac{u}{2}$.[2] The prior for each weight is a zero-mean Gaussian with variance $\sigma^2\delta$. Integrating the likelihood function over the parameters obtains:

$$P(j,t,q) = \frac{\pi^{-\frac{n}{2}}}{\delta^m} |(\mathbf{A}_q + \mathbf{D})^{-1}|^{\frac{1}{2}} \frac{u^{\frac{v}{2}}}{(y_q+u)^{\frac{n+v}{2}}} \frac{\Gamma(\frac{n+v}{2})}{\Gamma(\frac{v}{2})}, \tag{5.3}$$

where $n = t - j$, q has m basis functions, Γ is the Gamma function, $\mathbf{D}$ is an m by m matrix with δ^{-1} on the diagonal and zeros elsewhere, and:

$$\mathbf{A}_q = \sum_{i=j}^{t} \Phi_q(\mathbf{x_i})\Phi_q(\mathbf{x_i})^T \tag{5.4}$$

$$y_q = (\sum_{i=j}^{t} R_i^2) - \mathbf{b}_q^T(\mathbf{A}_q + \mathbf{D})^{-1}\mathbf{b}_q, \tag{5.5}$$

where $\Phi_q(\mathbf{x_i})$ is a vector of the m basis functions associated with q evaluated at state $\mathbf{x_i}$, $R_i = \sum_{j=i}^{T} \gamma^{j-i} r_j$ is the return obtained from state i, and $\mathbf{b}_q = \sum_{i=j}^{t} R_i \Phi_q(\mathbf{x_i})$.

Note that we are using each R_t as the target regression variable in this formulation, even though we only observe r_t for each state. However, to compute Equation 5.3 we need

[2]These parameters may seem cryptic, but they can be set indirectly using an expected variance σ_v^2 and scaling parameter β_v. The scaling parameter controls how sharply the distribution is peaked around σ_v^2; values closer to zero indicate a flatter distribution. We can then set $u = \sigma_v^2 + \beta_v$ and $v = \frac{\beta_v}{\sigma_v^2} - 1$.

only retain sufficient statistics $\mathbf{A}_q$, $\mathbf{b}_q$ and $(\sum_{i=j}^{t} R_i^2)$ for each model. Each can be updated incrementally using r_t (the latter two using traces). Thus, the sufficient statistics required to obtain the fit probability can be computed incrementally and online at each timestep, without requiring any transition data to be stored. The algorithm for this update is given in Figure 5.4.

```
input : $\mathbf{x}_t$: the current state
        $r_t$: the current reward

1  Initialization
2  if t == 0 then
3      $\mathbf{A}_q$ = zero_matrix(q.m, q.m)
4      $\mathbf{b}_q$ = zero_vector(q.m)
5      sum_r$_q$ = 0
6      $\mathbf{z}_q$ = zero_vector(q.m)
7      tr_1$_q$, tr_2$_q$ = 0;
8  Compute the basis function vector for the current state
9  $\Phi_t = \Phi_q(\mathbf{x_t})$
10 Update sufficient statistics
11 $\mathbf{A}_q = \mathbf{A}_q + \Phi_t \Phi_t^T$
12 $\mathbf{z}_q = \gamma \mathbf{z}_q + \Phi_t$
13 $\mathbf{b}_q = \mathbf{b}_q + r_t \mathbf{z}_q$
14 $\text{tr\_1}_q = 1 + \gamma^2 \text{ tr\_1}_q$
15 $\text{sum\_r}_q = \text{sum\_r}_q + r_t^2 \text{ tr\_1}_q + 2\gamma r_t \text{ tr\_2}_q$
16 $\text{tr\_2}_q = \gamma \text{ tr\_2}_q + r_t \text{tr\_1}_q$
```

Figure 5.4. Incrementally updating the changepoint detection sufficient statistics for model q.

Note that $\mathbf{A}_q$ and $\mathbf{b}_q$ are the same matrices used for performing a least-squares fit to the data using model q and R_t as the regression target. They can thus be used to produce a value function fit (equivalent to a least-squares Monte Carlo estimate) for the skill segment if so desired; again, without the need to store the trajectory.

In practice, segmenting a sample trajectory should be performed using a lower-order function approximator than is to be used for policy learning, since we see merely a single trajectory sample rather than a dense sample over the state space.

5.2.2 Merging Skill Chains into a Skill Tree

Using this model we can segment a single trajectory into a skill chain; given multiple skill chains from different trajectories, we would like to merge them into a skill tree by determining which pairs of trajectory segments belong to the same skills and which are distinct.

Since we wish to build skills that can be sequentially executed, we can only consider merging two segments when they have the same target—which means that the segments immediately following each of them have been merged. Since we assume that all trajectories have the same final goal, we merge two chains by starting at their final skill segments. For each pair of segments, we determine whether or not they are a good statistical match, and if so merge them, repeating this process until we fail to merge a pair of skill segments, after which the remaining skill chains branch off on their own. This process is depicted in Figure 5.5. A similar process can be used to merge a chain into an existing tree by following the chain with the highest merge likelihood when a branch in the tree is reached.

Since $P(j, t, q)$ as defined in Equation 5.3 is the integration of the likelihood function of our model given segment data over its parameters, we can reuse it as a measure of whether a

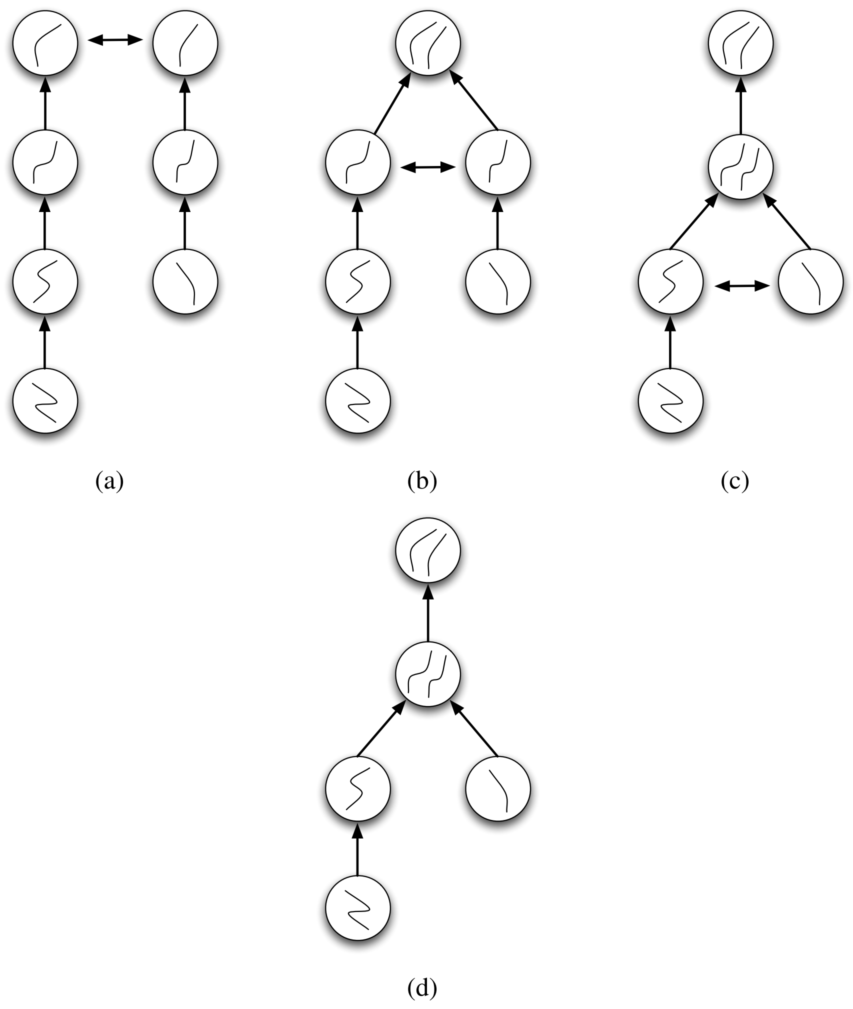


Figure 5.5. Merging two skill chains into a skill tree. We begin by considering merging the final trajectory segment in each of the chains (a). If these segments use the same model, overlap, and can be well represented using the same function approximator, we merge them and move on to the second segment in each chain (b). We continue merging until we find a pair of segments that should not be merged (c). Merging then halts and the remaining skill chains form separate branches of the tree (d).

pair of trajectories are better modeled as one skill or as two separate skills. Given sufficient statistics $\mathbf{A}_a, \mathbf{b}_a$ and sum of squared return R_a from segment a (having n_a transitions) and $\mathbf{A}_b, \mathbf{b}_b$ and R_b from segment b (having n_b transitions), we can compute the probability of both data segments given a single skill model by evaluating Equation 5.3 using the sum of these quantities: $\mathbf{A}_{ab} = \mathbf{A}_a + \mathbf{A}_b$, $\mathbf{b}_{ab} = \mathbf{b}_a + \mathbf{b}_b$, $R_{ab} = R_a + R_b$, and $n_{ab} = n_a + n_b$. Note that this model uses the same number of basis functions (m) as either model in isolation.

To evaluate the probability of data segments a and b coming from two different skill models, we evaluate Equation 5.3 using $R_{ab} = R_a + R_b$ and $n_{ab} = n_a + n_b$ as before, but with:

$$\mathbf{A}_{ab} = \begin{bmatrix} \mathbf{A}_a & 0 \\ 0 & \mathbf{A}_b \end{bmatrix}, \tag{5.6}$$

and

$$\mathbf{b}_{ab} = \begin{bmatrix} \mathbf{b}_a \\ \mathbf{b}_b \end{bmatrix}. \tag{5.7}$$

This is equivalent to using a larger set of basis functions, $\Phi_{ab} = [\Phi_a, \Phi_b]^T$, where the basis functions from each segment are each non-zero only in their own segments. Thus, although we may find a better fit using the two sets of basis functions independently, since we have a higher number of basis functions (m is twice as large), we obtain a higher probability only when the two segments really are much better fit separately.

When considering a merge between more than two segments (as occurs when merging a chain into a tree at points where the tree has already split), we perform a similar operation but evaluate the total probability of the data in all segments given that a pair of segments have merged and the remainder are independent, evaluated for all candidate merging pairs and the case where no merge occurs. This is necessary so that the probabilities obtained from Equation 5.3 for each case are over the same data, and therefore comparable.

Before merging, we perform a fast test to ensure that the trajectory pairs actually overlap in state space—if they do not, we will often be able to represent them both simultaneously with very low error and hence this metric may incorrectly suggest a merge.

If we are to merge skills obtained over multiple trajectories into trees we require the component skills to be aligned, meaning that the changepoints occur in roughly the same places. This will occur naturally in domains where changepoints are primarily caused by a change in relevant abstraction. When this is not the case, they may vary since segmentation is then based on function approximation boundaries, and hence two trajectories segmented independently may be poorly aligned. Therefore, when segmenting two trajectories sequentially in anticipation of a merge, we may wish to include a bias on changepoint locations in the second trajectory. We model this bias as a Mixture of Gaussians, centering an isotropic Gaussian at each location in state-space where a changepoint previously occurred. We can include this bias during changepoint detection by multiplying Equation 5.1 with the resulting PDF evaluated at the current state.

Note that, although segmentation is performed using a lower order function approximator than skill policy learning, merging should be performed using the same function approximator used for learning. This necessitates the maintenance of two sets of sufficient statistics during segmentation; fortunately, the major computational expense is computing $P(j, t, q)$, which during segmentation is only required using the lower-order approximator.

5.3 Acquiring Skills from Human Demonstration in Pinball

In this section, we evaluate the performance benefits obtained using a skill tree generated from a pair of human-provided solution trajectories in Pinball. We use the Pinball domain instance shown in Figure 5.6 with 5 pairs (one trajectory in each pair for each start state) of trajectories obtained from a human expert.

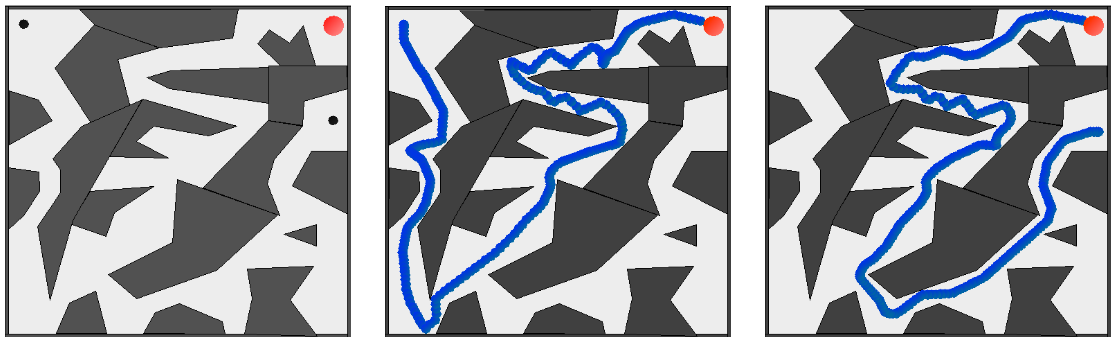


Figure 5.6. The Pinball instance used in our experiments, and a representative solution trajectory pair.

5.3.1 Implementation Details

All skill chaining parameters were as in Chapter 3; for CST agents, we used only the training examples from the demonstration trajectories for learning initiation sets (positive examples are those in the skill segment, negative examples all others). After training they were considered converged and updated thereafter as in Chapter 3.

We used an expected skill length of $k = 100$, $\delta = 0.0001$, particle filter parameters $N = 30$ and $M = 50$, and a first-order Fourier Basis (16 basis functions) for segmentation. We used expected variance mean $\sigma_v^2 = 15^2$ and scaling parameter $\beta_v = 0.0001$ for the noise prior. After segmenting the first trajectory we used isotropic Gaussians with variance 0.5^2

to bias the segmentation of the second. The full 3rd-order Fourier basis representation was used for merging. To obtain a fair comparison with skill chaining, we initialized the CST skill policies using 10 episodes of experience replay of the demonstrated trajectories, rather than using the sufficient statistics to perform a least-squares value function fit.

5.3.2 Results

Trajectory segmentation was successful for all demonstration trajectories, and all pairs were merged successfully into skill trees when the alignment bias was used to segment the second trajectory in the pair (two of the five could not be merged due to misalignments when the bias was not used). Example segmentations and the resulting merged trajectories are shown in Figure 5.7, and the resulting initiation sets are shown in their tree structure in Figure 5.8.

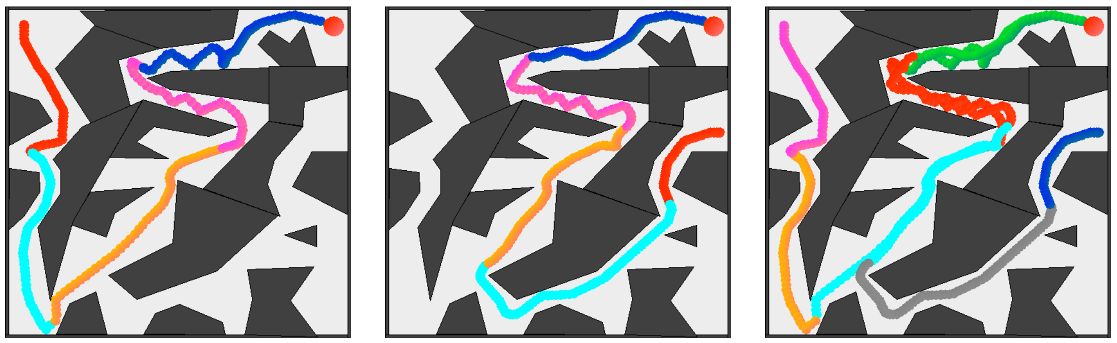


Figure 5.7. Segmented skill chains from the sample pinball solution trajectories shown in Figure 3.3, and the trajectory assignments obtained when the two chains are merged.

The learning curves obtained using the resulting skill trees to reinforcement learning agents, averaged over 100 runs (20 runs using each demonstrated skill tree) are shown in Figure 5.9. These results compare the learning curves of CST agents, agents that perform skill

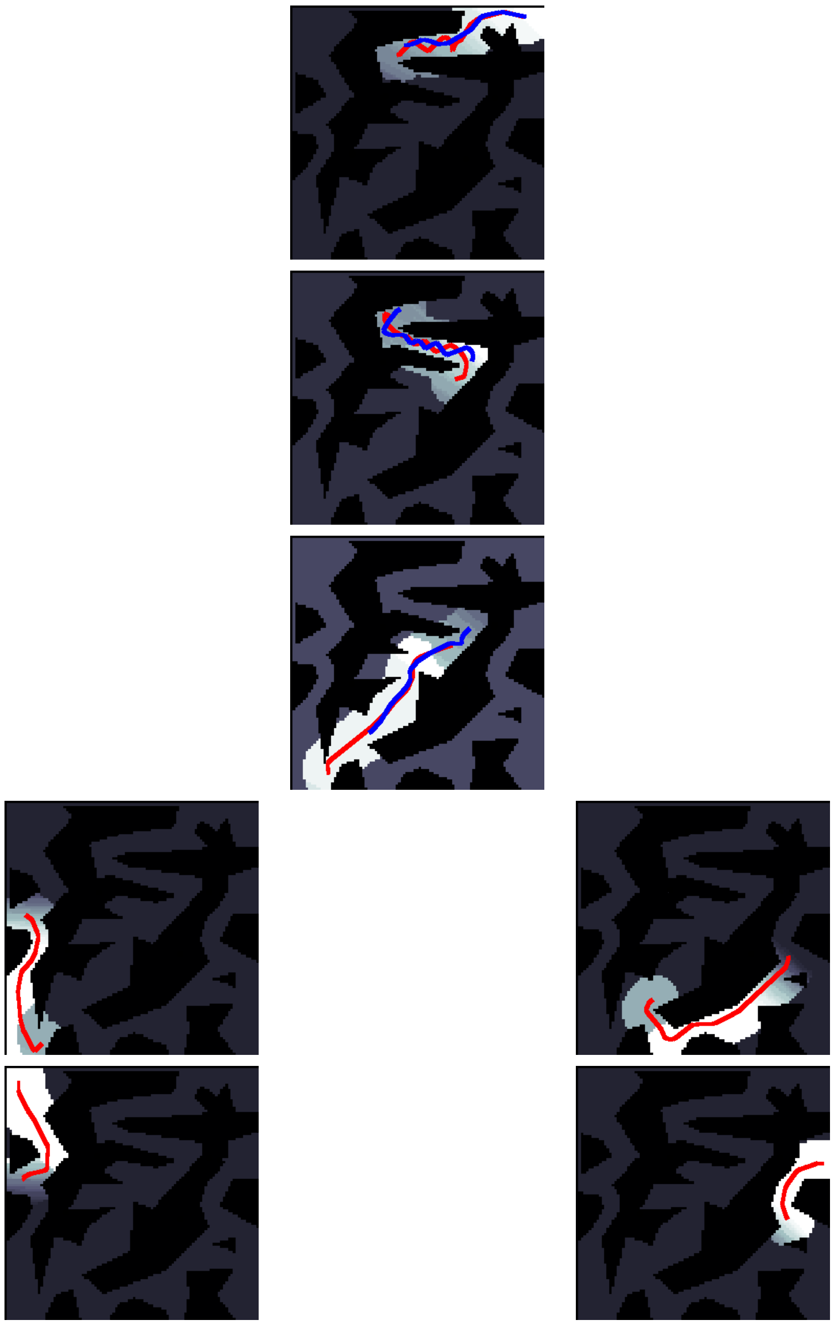


Figure 5.8. The initiation sets for each option in the skill tree shown in Figure 5.7.

chaining from scratch, and agents that are given fully pre-learned skills (obtained over 250 episodes of skill chaining). They show that the CST policies are not good enough to use immediately, as the agents do worse than those given pre-learned skills for the first few episodes (although they immediately do better than skill chaining agents). However, very shortly thereafter—by the 10th episode—the CST agents are able to learn excellent policies, immediately performing much better than skill chaining agents, and shortly thereafter actually temporarily exceeding the performance of agents with pre-learned skills. This is likely because the skill tree structure obtained from demonstration has fewer but better skills than that learned incrementally by skill chaining agents, resulting in a faster initial startup while the agents given pre-learned skills learn to correctly sequence them.

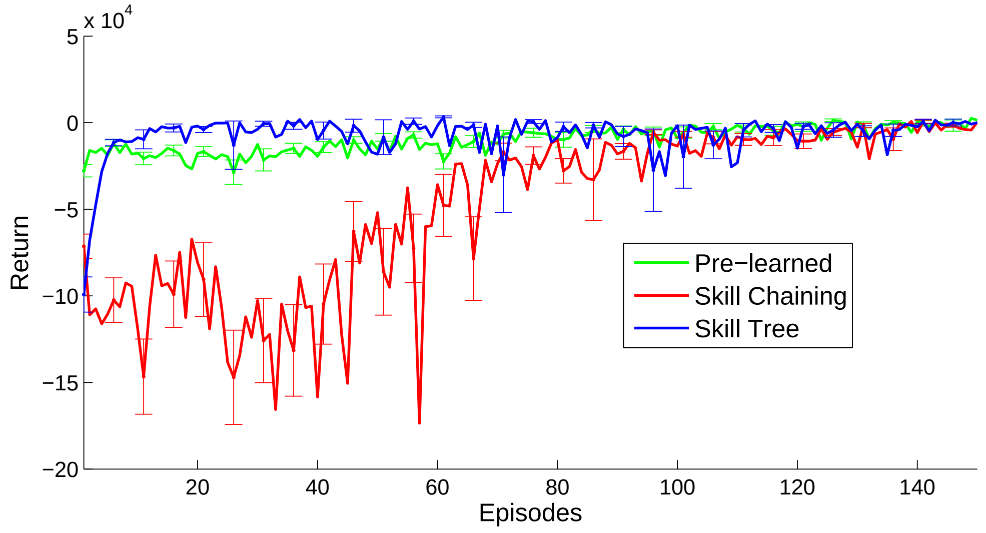


Figure 5.9. Learning curves in the PinBall domain, for agents employing skill trees created from demonstration trajectories, skill chaining agents, and agents starting with pre-learned skills.

In addition, segmenting demonstration trajectories into skills results in much faster learning than attempting to acquire the entire demonstrated policy at once. The learning curve for agents that first perform experience replay on the overall task value function and then proceed using skill chaining (not shown) is virtually identical to that of agents performing skill chaining from scratch.

5.4 Acquiring Skills from Human Demonstration on the uBot

In the previous section, we showed that CST is able to segment demonstration trajectories in Pinball, and merge them into tree suitable as a basis for further learning. In this section, we show that CST can scale up by applying it to creating skill chains from human demonstration on the uBot-5, a dynamically balancing mobile robot.

The robot's task in this section is to enter a corridor, approach a door, push the door open, turn right into a new corridor, and finally approach and push on a panel (illustrated in Figure 5.10). Twelve demonstration trajectories were obtained from an expert human operator.

5.4.1 Implementation Details

To simplify perception, we placed purple, orange and yellow colored circles on the door and panel, beginning of the back wall, and middle of the back wall, respectively, as perceptually salient markers indicating the centroid of each object. The distances (obtained using stereo vision) between the uBot to each marker were computed at 8Hz and filtered. The uBot was able to engage one of two motor abstractions at a time: either performing end-point position control of its hand, or controlling the speed and angle of its forward motion.

Using these features, we constructed six sensorimotor abstractions, one for each pairing of salient object and motor command set. When a marker was paired with the arm endpoint,

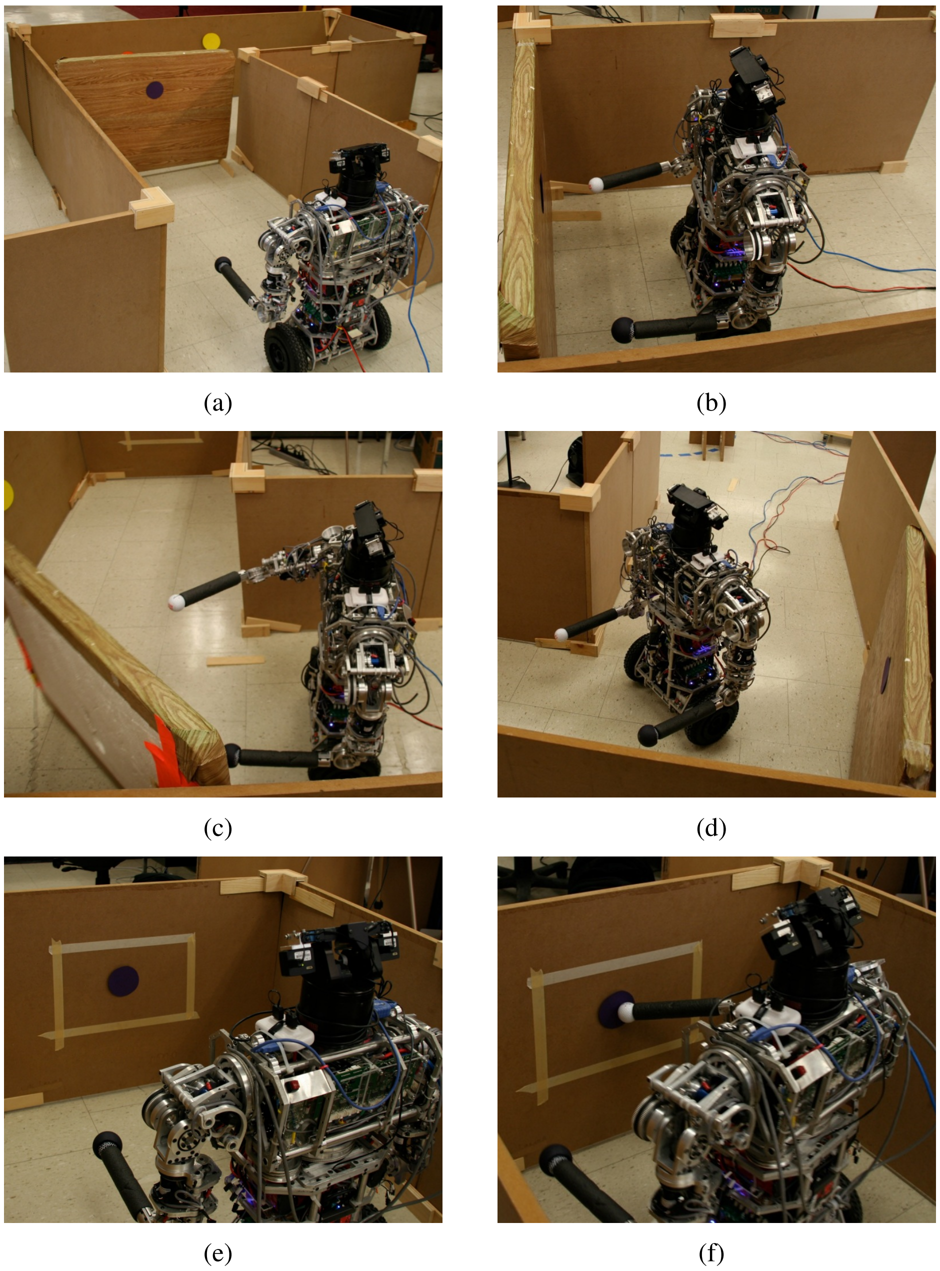


Figure 5.10. The task demonstrated on the uBot-5. Starting at the beginning of a corridor (a), the uBot approaches (b) and pushes open a door (c), turns through the doorway (d), then approaches (e) and pushes a panel (f).

the abstraction's state variables consisted of real-valued difference between the endpoint and the marker centroid in 3 dimensions. Actions were real-valued vectors moving the endpoint in 3 dimensions. When a marker was paired with the robot's torso, the abstraction's state variables consisted of two real values representing the distance and angle to the marker centroid. Actions were real-valued vectors controlling the speed and direction of the robot torso using differential drive. We assumed a reward function of -1 every 10th of a second.

Particles were generated according to the currently executing motor abstraction, and a switch in motor abstraction always caused a changepoint.[3] The parameters used for performing CST on the uBot were $k = 50$, $M = 60$, $N = 120$, $\sigma_v^2 = 8^2$ and $\beta_v = 0.00001$, using a 1st order Fourier basis. For merging, we used a 5th order Fourier basis with $\sigma_v^2 = 90^2$ and $\beta_v = 0.00001$.

When performing policy regression to fit the segmented policies for replay, we used ridge regression over a 5th order Fourier basis to directly map to continuous actions. The regularization parameter λ was set by 10-fold cross-validation, using a test set extracted from the same trajectories as those used to perform the fit. Policy replay testing was performed by varying the starting point of the robot by hand and used hand-coded stopping conditions that corresponded to the subsequent skill's initiation set.

[3]Informal experiments with removing this restriction did not seem to change the number or type of skills found but in some cases changed their starting and stopping positions by a few timesteps.

5.4.2 Results

Of the 12 demonstration trajectories gathered from the uBot, 3 had to be discarded because the perceptual features were too noisy. Of the remaining 9, all segmented sensibly and 8 were able to be merged into a single skill chain.[4]

Figure 5.11 shows a segmented trajectory obtained using CST, with Table 5.1 providing a brief description of the skills extracted along with their selected abstractions, and the number of sample trajectories required for each skill to be replayed successfully 9 times out of 10.

#	Abstraction	Description	Trajectories Required
a	torso-purple	Drive to door.	2
b	endpoint-purple	Push the door open.	1
c	torso-orange	Drive toward wall.	1
d	torso-yellow	Turn toward the end wall.	2
e	torso-purple	Drive to the panel.	1
f	endpoint-purple	Press the panel.	3

Table 5.1. A brief description of each of the skills extracted from the trajectory shown in Figure 5.11, along with their selected abstractions, and the number of example trajectories required for accurate replay.

Thus, CST is able to segment trajectories obtained from a robot platform, select an appropriate abstraction in each case, and then replay the resulting policies using a small number of sample trajectories.

[4]In four of these trajectories, a delay starting the robot moving after opening the door caused a "wait skill" to appear in the segmentation, marking out a period of time when the robot did nothing. We excised these skills before merging.

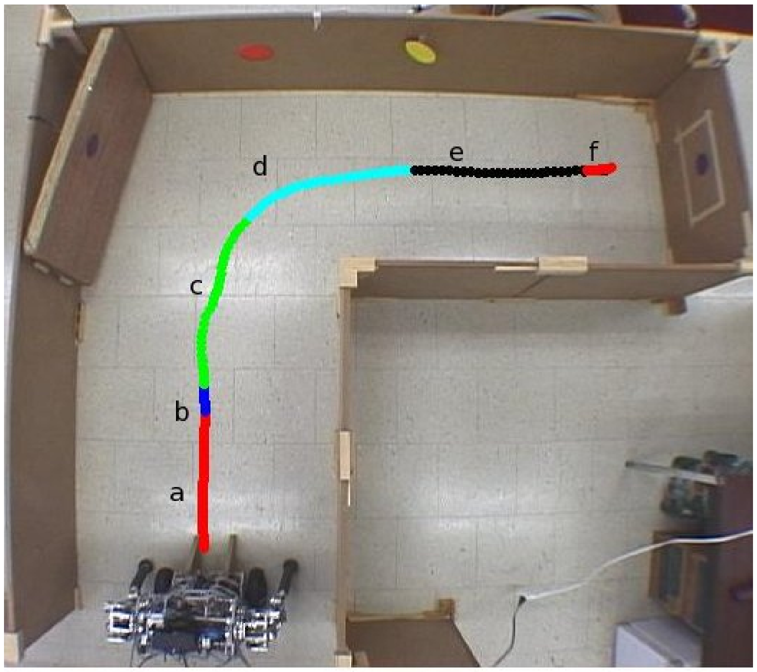


Figure 5.11. A demonstration trajectory from the uBot-5 segmented into skills.

The different numbers of example trajectories required to accurately replay the demonstrated skills occurred because of the varying difficult of each skill. Pushing the door open proved relatively easy—a general forward motion by the endpoint toward the purple circle will almost always succeed. By contrast, pushing the panel required greater precision, since the endpoint was required to be within the panel area when it touched the wall, even when the uBot was facing the wall at an angle. Finally, approaching a target turned out to be difficult in some cases because small mistakes in its angular velocity when the uBot neared the target could cause the target to drift out of the robot's fairly narrow field of view.

5.5 Related Work

The most relevant related work on skill acquisition in reinforcement learning is the algorithm introduced by Mehta *et al.* (2008), which induces a task hierarchy from a single demonstration trajectory, assigning an appropriate abstraction to each subtask. However, this method is only applicable to discrete domains and assumes a factored MDP with given dynamic Bayes network action models.

A sequence of policies represented using linear function approximators may be considered a switching linear dynamical system. Methods exist for learning such systems from data (Xuan and Murphy, 2007; Fox *et al.*, 2008); these methods are able to handle multivariate target variables and models that repeat in the sequence. However, they are consequently more complex and computationally intensive than the much simpler changepoint detection method we use, and they have not been used in the context of reinforcement learning or skill acquisition.

A great deal of work exists in robotics under the general heading of learning from demonstration (Argall *et al.*, 2009), where control policies are learned using sample trajectories obtained from a human, robot demonstrator, or a planner. Most methods learn an entire single policy from data, although some perform segmentation—for example, Jenkins and Matarić (2004) segment demonstrated data into *motion primitives*, and thereby build a motion primitive library. They perform segmentation using a heuristic specific to human-like kinematic motions; more recent work has used more principled statistical methods (Grollman and Jenkins, 2010; Butterfield *et al.*, 2010) to segment the data into multiple models as a way to avoid perceptual aliasing in the policy. Other methods use demonstration to provide an initial policy that is then refined using reinforcement learning—e.g., Peters and

Schaal (2008). Prior to our work, we know of no existing method that both performs trajectory segmentation and results in motion primitives suitable for further learning.

5.6 Discussion and Conclusions

CST makes several key assumptions. The first is that the demonstrated skills form a tree, when in some cases they may form a more general graph (e.g., when the demonstrated policy has a loop). A straightforward modification of the procedure to merge skill chains could accommodate such cases. We also assume that the domain reward function is known and that each option reward can be obtained from it by adding in a termination reward. A method for using inferred reward functions (e.g., Abbeel and Ng (2004)) could be incorporated into our method when this is not true. However, this requires segmentation based on *policy* rather than *value function*, since rewards are not given at demonstration-time. Because policies are usually multivariate, this would require a multivariate changepoint detection algorithm, such as that by Xuan and Murphy (2007). Finally, we assume that the best model for combining a pair of skills is the model selected for representing both individually. This may not always hold—two skills best represented individually by one model may be better represented together using another (perhaps more general) one. Since the correct abstraction would presumably be at least competitive during segmentation, such cases can be resolved by considering segmentations other than the final MAP solution when merging.

In the context of robot learning from demonstration, segmenting trajectories into skills has several advantages over methods that try to learn the entire policy monolithically. Each skill is allocated its own abstraction, and therefore can be learned and represented efficiently—

potentially allowing us to learn higher dimensional, extended policies. During learning, an unsuccessful or partial episode can still improve skills whose goals where nevertheless reached. Confidence-based learning methods (Chernova and Veloso, 2007) can be applied to each skill individually. Finally, skills learned using agent-centric features (such as in our uBot task) can be retained and transferred to new problem settings (Konidaris and Barto, 2007), and thereby detached from a specific problem to be more generally useful by allowing the robot to more quickly solve new tasks. These advantages may prove crucial to scaling up robot learning by demonstration methods, and making them practical for extended control tasks on high-dimensional robots.

CHAPTER 6

AUTONOMOUS SKILL ACQUISITION ON A MOBILE ROBOT

Although CST can be applied to demonstration trajectories regardless of their source, the previous chapter focused on learning from demonstration trajectories provided by a human expert. In this chapter, we use CST as a component of the control system of a mobile robot—the uBot-5—that uses reinforcement learning to find a solution to an instance of the Red Room Domain. It thus generates its own demonstration trajectories, thereby achieving autonomous skill acquisition. We show that this system is able to use skills acquired in that instance to improve its performance in another.

6.1 The Red Room Domain

The Red Room Domain places the uBot-5 in a small artificial room that contains various objects. The uBot is equipped with a set of innate controllers which it must learn to sequence to complete its task.

6.1.1 The uBot-5

The uBot-5 (briefly introduced in Chapter 5, and shown in Figure 6.1) is a dynamically balancing, 13 degree of freedom mobile manipulator (Deegan *et al.*, 2006). Dynamic balancing is performed using an LQR controller that keeps the robot upright and compensates for forces exerted upon it during navigation and manipulation. The uBot-5 has two arms, each terminated by a small ball that can be used for basic manipulation tasks.[1]

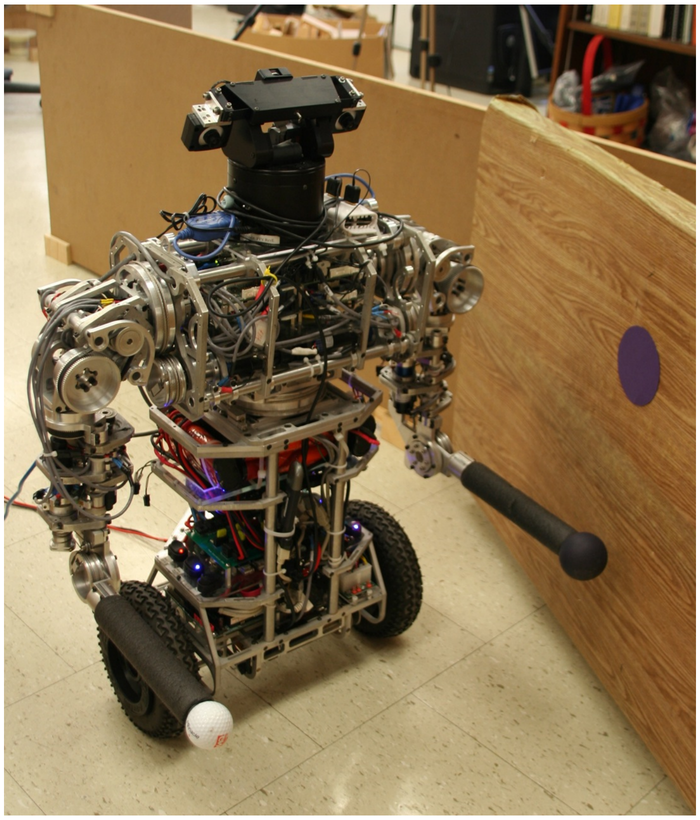


Figure 6.1. The uBot-5, a 13-DoF dynamically balancing mobile manipulator.

The uBot's control system is implemented primarily in Microsoft Robotics Developer Studio (Johns and Taylor, 2008), and allows differential control of its wheels and position control of each of its arm endpoints (though we only use the right arm in this work). Perception is performed using the uBot's twin cameras, and the ARToolkit augmented reality

[1]A grasping hand prototype is expected to be working shortly.

tag system (Kato and Billinghurst, 1999), whereby augmented reality tags (ARTags) that can be reliably detected using the robot's cameras are placed in the environment to mark important features of the task. The ARTags are placed in known configurations around important features, allowing the robot to localize each feature even when only a single ARTag is visible.

The robot is equipped with both navigation and manipulation controllers. Given a target feature, the navigation controller will first align the robot with that feature's normal, and then turn the robot to face the feature and approach it. This guarantees that the robot is in a good position to manipulate the target object when it arrives. The uBot also has controllers that move its endpoint to one of seven positions: withdrawn (allowing it to execute a navigation action), extended, and then extended and moved to the left, right, upwards, downwards, or outwards. Each of these controls the position of the endpoint relative to the centroid of the target feature.

Before starting a navigation, the robot performs a visual scan of the room to determine the positions of visible objects in the room. It then runs an orientation controller which turns it away from the location it is currently facing and toward its intended target. This controller is crucial in keeping the robot safely away from the walls of the domain and is thus not learned or including in segmentation. When the robot navigates towards or is trying to manipulate an object, it also runs a tracker which controls its pan-tilt head to keep one of its target's ARTags centered in its field of view.

As in Chapter 5, the uBot is given a library of abstractions. Each abstraction pairs one of its motor modalities (wheels or arms) with a task feature. However, in this case both types of abstractions are three-dimensional. Abstractions using the robot's endpoint contain state variables expressing the difference between its position and target feature's centroid in three dimensional space. Abstractions using the robot's torso and a target feature contain state variables expressing the distance from the robot's torso to the feature, the distance from the robot's torso to the wall upon which the feature is attached, and the angle between the robot's torso and the feature's normal.

When performing task-level learning, the uBot builds a discrete model of the task as an MDP. This model allows the uBot to plan using dynamic programming (Sutton and Barto, 1998) with learning rate $\alpha = 0.1$. The value function is initialized optimistically to zero for unknown state-action pairs, except for when the robot uses acquired skills, when the state-action pairs corresponding to basic manipulation actions are initialized to a time cost of three hours. This results in a robot that always prefers to use higher-level acquired skills when possible, but can also make use of lower-level innate controllers when all other options have been exhausted. Planning is performed online, in real time. The robot receives a reward of -1 for each second that passes until it has solved the task.

To learn a skill policy given a demonstration trajectory, we fit a linear spline model to the demonstrated trajectory. This allows us to replay the trajectory from new starting positions reliably, while the parameters of the model can be tuned using a policy gradient algorithm.

6.1.2 The Red Room

The Red Room Domain consists of two tasks. We use the first task as a training task: the uBot learns to solve it by sequencing its innate controllers, and extracts skills from the resulting solution trajectories. We then compare the time the uBot takes to initially solve the second task using its innate controllers against the time required when using acquired skills.

6.1.2.1 The First Task

The first task consists of a small room containing a button and a handle. When the handle is pulled after the button has been pressed, a door in the side of the room opens, allowing the uBot access to a compartment which contains a switch. The goal of the task is to press the switch. Sensing and control for the objects in the room are performed using touch sensors, with state tracked and communicated to the uBot via an MIT Handy Board (Martin, 1998). A schematic drawing and photographs of the first room are given in Figure 6.2.

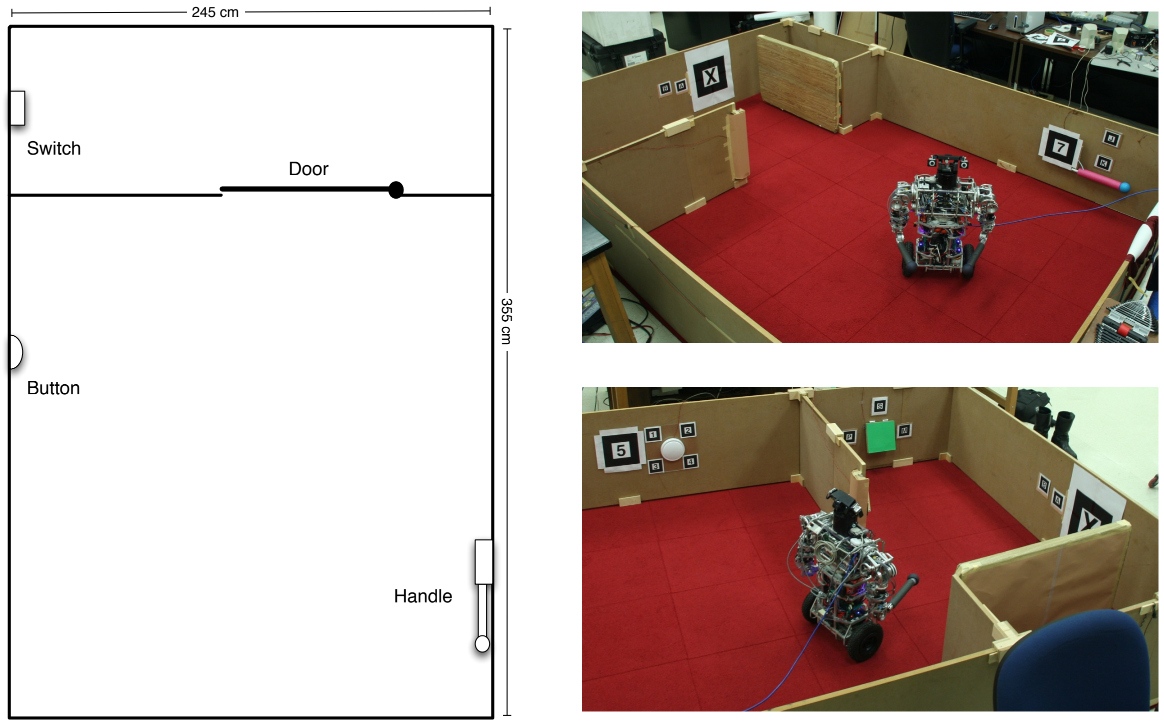


Figure 6.2. The first task in the Red Room Domain.

At the task level, the state of the first task at time i is described as a tuple $s_i = (r_i, p_i, h_i)$, where r_i is the state of the room, p_i is the position of the robot, and h_i is the position of its end-effector.

The state of the room at time i consists of four state bits, indicating the state of the button (pressing the button flips this bit), the state of the handle (this bit is only flipped once, and only when the button bit is set), whether or not the door is open, and whether or not the switch has been pressed (this bit is also only flipped once since the task ends when it is set).

The uBot may find itself at one of five positions: its start position, in front of the button, in front of the handle, through the door, and in front of the switch. Each of these positions is marked in the room using ARTags—a combination of small and large tags are used to ensure that each position is visible in the robot's cameras from all of the relevant locations in the room. The robot has a navigate action available to it that will move it from its current position to any position visible when it performs a visual sweep with its head. Thus, the robot may always move between the button and the switch, but can only move through the door entrance once it is open; only then can it see the switch and move towards it.

Finally, the robot's end-effector may be in one of seven positions: withdrawn (from where it can execute a navigation action), extended, and then extended and moved to the left, right, upwards, downwards, or outwards. The robot must always be facing an object to interact with it. In order to actuate the button and the switch, the robot must extend its arm and then move it outwards; in order to actuate the handle, it must extend its arm and then move it downwards.

6.1.2.2 The Second Task

The second Red Room task is similar to the first: the robot is placed in a room with a group of manipulable objects and a door. In this case, the robot must first push the switch, and then push the button to open the door. Opening the door hides a button in the second part of the room. The robot must then navigate to the second part of the room and pull a lever to close the door again, revealing the second button, which it must press to complete the task.

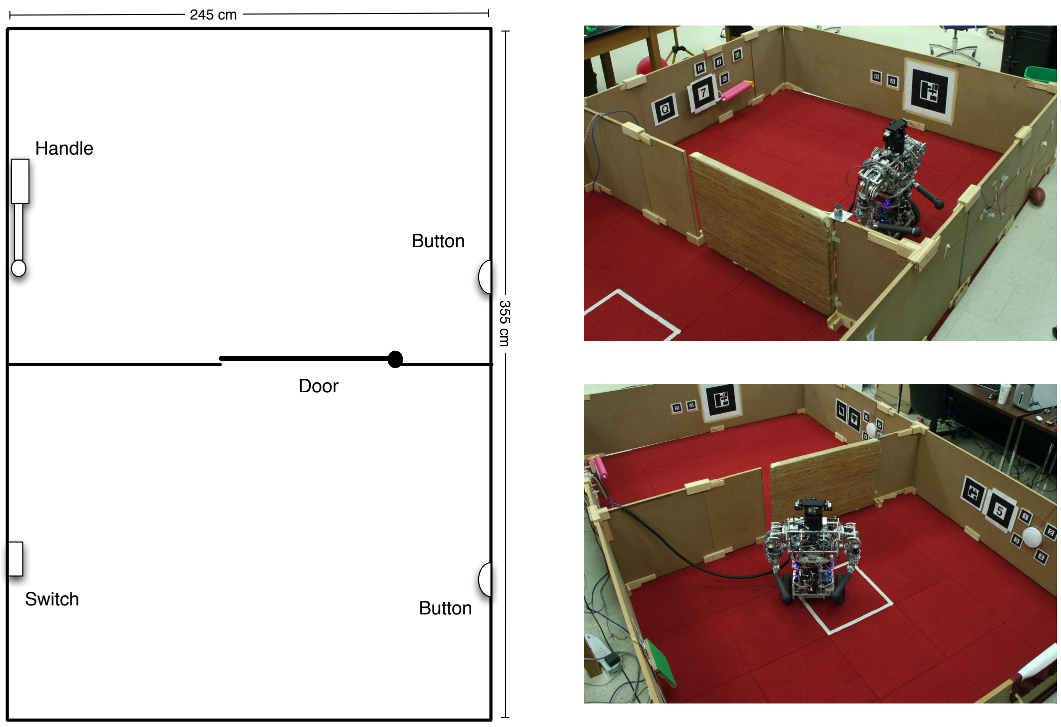


Figure 6.3. The second task in the Red Room Domain.

Note that this room contains the same types of objects as the first task, and so the robot is able to apply its acquired skills to manipulate objects when they are of a type it has encountered before. In general object classification could be done by visual pattern matching, but in this case we simply label the objects for the robot.

6.2 Results

The uBot's first task is to sequence its innate controllers to solve the first Red Room task. Since the robot starts with no knowledge of the underlying MDP, it must learn both how to interact with each object and in which order interaction must take place from scratch. Figure 6.4 shows the uBot's learning curve for the first Red Room task: it is able to find the optimal controller sequence after 5 episodes, reducing the time taken to solve the task from approximately 13 minutes to around 3.

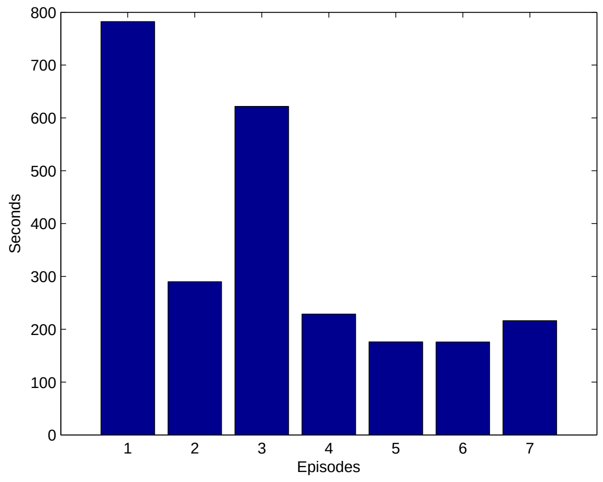


Figure 6.4. The uBot's learning curve in the first Red Room task. It executes the optimal sequence of controllers from the 5th episode on.

The resulting optimal sequence of controllers are then used to generate 5 demonstration trajectories for use in CST (using a first-order Fourier Basis, $k = 150$, $M = 60$, $N = 120$, $\sigma_v = 60^2$, and $\beta_v = 0.000001$). The resulting trajectories all segment into the same sequence of 10 skills, and are all merged successfully (using a 5th order Fourier Basis,

$\sigma_v = 500^2$, and $\beta_v = 0.000001$). An example segmentation is shown in Figure 6.5; a description of each skill along with its relevant abstraction is given in Table 6.1.

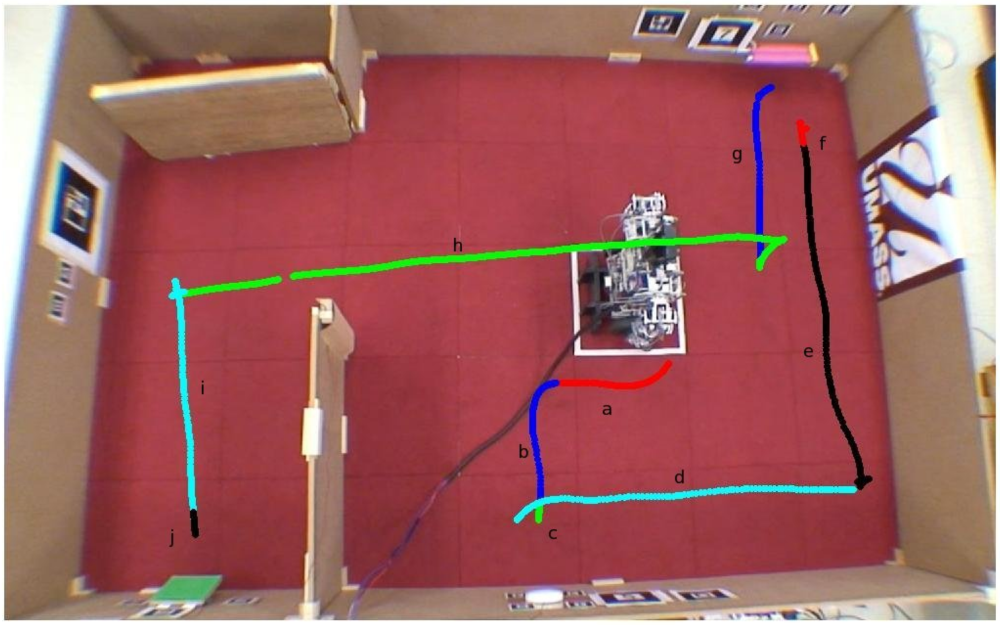


Figure 6.5. A trajectory from the learned solution to the first Red Room task, segmented into skills.

CST consistently extracted skills that corresponded to manipulating objects in the environment, and navigating towards them. In the navigation case, each controller execution was split into two separate skills. These skills correspond exactly to the two phases of the navigation controller: first, aligning the robot with the normal of a feature, and second, moving the robot toward that feature. We do not consider the resulting navigation skills further since they are room-specific and cannot be used in the second task.

In the object-manipulation case, a sequence of two controllers is collapsed into a single skill: for example, extending the arm and then extending it further forward is collapsed

#	Abstraction	Description
a	torso-button	Align with the button.
b	torso-button	Turn and approach the button.
c	endpoint-button	Push the button.
d	torso-handle	Align with the handle.
e	torso-handle	Turn and approach the handle.
f	endpoint-handle	Pull the handle.
g	torso-entrance	Align with the entrance.
h	torso-entrance	Turn and drive through the entrance.
i	torso-switch	Approach the switch.
j	endpoint-switch	Press the switch.

Table 6.1. A brief description of each of the skills extracted from the trajectory shown in Figure 6.5, along with their selected abstractions.

into a single skill which we might label *push the button*. We fitted the resulting policies for replay using a single demonstrated trajectory, and obtained reliable replay for all manipulation skills.

Figure 6.2 shows the results obtained when the uBot first attempts to solve the second Red Room task, given either its original innate controllers or, additionally, the manipulation skills acquired in the first Red Room task. We performed eight runs of each condition. The results show that using acquired manipulation skills, the uBot is able to initially complete the new task in on average a little more than half the time required when only innate skills were present. This difference is statistically significant (t-test, $p < 0.01$); moreover, the sample completion times for the two conditions do not overlap.

This data thus demonstrates that skills acquired in one task can be deployed to improve the robot's problem-solving abilities in second task.

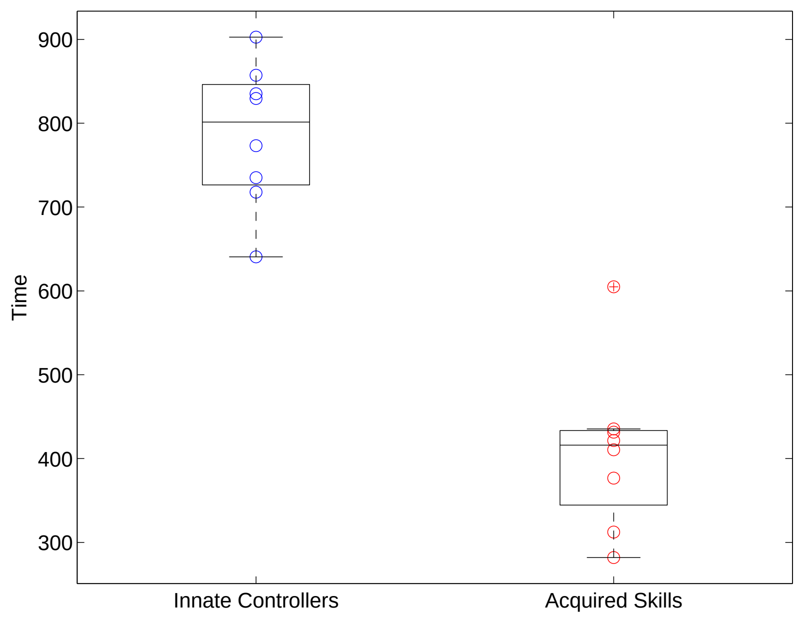


Figure 6.6. The time taken by the uBot-5 to first complete the second Red Room task, given innate controllers or acquired skills.

Note that one of the runs using skill acquisition is marked as an outlier (with a cross) in Figure 6.2. During this run, the robot explored virtually all transitions available in the MDP before finally finding the solution. This data point thus corresponds to a sample of the worst-case behavior of the uBot using acquired skills; it still requires less time (by about 30 seconds) than the fastest sample run using only innate controllers.

6.3 Related Work

The great deal of work on various aspects of robot skill acquisition has already been covered in Chapter 2; here we focus primarily on the few systems where acquired skills are used to more efficiently learn to solve new tasks. The most directly relevant work is by Hart (2009a), where a robot learns according to a developmental schedule whereby it acquires skills that are then used as primitives in later tasks. This work used an intrinsic reward function (Hart, 2008) which was task specific, and relies upon the presence of a teacher or programmer who both designs the developmental schedule for the robot and restricts the controllers available to it to make learning feasible. In addition, the skills acquired are chunked sequences of innate controllers and cannot be further improved via learning. Similar work by Huber (2000) is used to build directed locomotion controllers from learned component gaits.

6.4 Summary and Conclusions

We have demonstrated that CST is able to acquire skills using demonstration trajectories obtained from the robot's own solution to a problem, and that the resulting skills can be used to improve its performance in a new task.

It is worth dwelling on the implications of the results presented here. Although the uBot started off with the capacity to *learn* to, for example, push the button, this was accomplished through the laborious trial-and-error of running through many combinations of manipulation actions within a particular task. However, since this sequence of manipulation actions happened to be useful in *solving a problem*, it was extracted as a single action that can be deployed as a unit—requiring only a single action selection decision—when the robot encounters a new problem. Had the uBot attempted transfer its *entire* policy from the first Red Room to the second, it would have started off with a very poor policy. Instead, transfer was affected via the isolation and retention of skills—effectively *policy components*—that are suitable for reuse in later tasks.

Thus, the uBot was able to acquire procedural knowledge autonomously through interaction with its environment; its performance in the second Red Room task shows that acquiring such knowledge can allow robots to learn to solve problems more efficiently.

CHAPTER 7

DISCUSSION AND CONCLUSION

The goal of this thesis was to improve the state of the art in hierarchical reinforcement learning and skill acquisition, to the point where it is possible to create a robot that acquires new skills autonomously.

In the pursuit of this goal, we developed three new algorithms: skill chaining, the first (to the best of our knowledge) method for skill acquisition in general continuous reinforcement learning problems; abstraction selection, which allows an agent to select an appropriate abstraction from a library when acquiring a new skill, and hence aids in skill acquisition in high-dimensional problems; and CST, and algorithm that performs both skill chaining and abstraction selection efficiently and online, using demonstration trajectories.

Finally, we have described an example demonstration where a mobile robot has autonomously acquired a set of new skills through interaction with an environment. This system shows that the methods developed here are sufficient to realize autonomous skill acquisition on a mobile robot in at least one instance. Thus, this work was at least partially successful in achieving its goal.

However, each of the specific techniques developed here opens up several questions that merit further attention.

7.1 Future Work

While skill chaining provides a general method for skill acquisition in continuous reinforcement learning domains, it could be extended in several directions. Most obviously, general heuristics that can identify target events before the agent has first reached a goal state would significantly broaden the method's applicability. Future work might also examine ways to deal with the large numbers of skills that might result from the presence of multiple root target events. A possible solution might involve identifying target events where two skill trees approximately overlap—for example, skill trees to reach two different locations in the same room might both use opening the door to that room as an intermediate target event—and then merging those trees below the common target event.

Future work may also address the question of how model-free skill acquisition techniques—such as skill chaining—compare to the combination of a learned environmental model followed by a model-based skill acquisition technique—such as LQR-Trees (Tedrake, 2009)—and the circumstances under which each approach may be more efficient or more likely to succeed.

Abstraction selection assumes the existence of a library of sensorimotor abstractions; this raises the question of whether new techniques might make it feasible to instead simply build the relevant abstraction as necessary, while the agent learns the skill; if they cannot, future work might examine how an agent can acquire an abstraction library over its lifetime. Future work might also examine how an agent can acquire information about the probability of deploying an abstraction in various contexts, and so provide more informative priors to a selection algorithm.

All of these extensions would also be applicable to CST, since it combines the ideas underlying both skill chaining and abstraction selection. In addition, future work might consider more principled ways to combine the changepoint distributions that result from each sample trajectory, perhaps by maximizing likelihood over the entire set of trajectories rather than for each trajectory sequentially. Additionally, an important question that remains open in CST is that of safety: when can an agent determine that it has seen sufficiently many sample trajectories to be able to successfully and reliably execute a policy? The application of confidence-based methods (Chernova and Veloso, 2007) seems to be an appropriate initial direction for such work.

Finally, the robot demonstration in the previous chapter was fairly limited, and has significant scope for improvement. In addition to all of the extensions above, future work may also address improving its rather rudimentary perception, adding grippers to the robot and thus allowing for more interesting manipulation skills, creating a system that could choose to explicitly practice acquired skills—perhaps following initial work by Stout and Barto (2010)—and actively explore new environments to discover new skills.

7.2 Discussion

The techniques presented here, and especially the demonstration described in the preceding chapter, have several limitations. Most importantly, the robot demonstration is a fairly limited one, using an environment engineered for simplicity, a hand-designed abstraction library, and a small number of acquired skills. These limitations, especially when viewed in light of all of the extensions detailed in the previous section, demonstrate plainly that this work has only begun to scratch the surface of what is necessary to build robots that are can acquire skills in an open-ended, reliable, and completely autonomous fashion.

However, the techniques and demonstration presented here are sufficient to provide support for the two behavioral advantages described in Chapter 1: that skill acquisition allows an agent to both discover solutions to new problems faster than it would have been able to otherwise—as demonstrated in the Red Room in Chapter 6—and that it allows an agent to achieve performance improvements on hard control problems through improving the policies of its acquired skills—as demonstrated in the Pinball Domain in Chapter 3.

In addition to the behavioral advantages discussed in Chapter 1, we have also demonstrated that skill acquisition in continuous domains confers two additional *engineering advantages*. First, as demonstrated using skill chaining in Chapter 3, skill acquisition allows us to solve continuous problems that are too hard to solve monolithically by adaptively breaking them up into smaller problems, and then learning good policies for those subproblems. Thus, skill acquisition eases the burden of *learning* complex policies monolithically.

Second, as demonstrated via abstraction selection in Chapter 4, skill acquisition provides a natural way to solve high-dimensional problems that are not amenable to a solution using a state abstraction, by adaptively breaking them into sequences of smaller problems, each of which *can* be solved using an abstraction. Thus, skill acquisition eases the burden of *representing* complex policies monolithically.

Together, these two advantages may prove crucial in scaling up reinforcement learning methods to high-dimensional, continuous domains; more broadly, the ability to adaptively break a problem into subproblems that are easy to solve and then reassembling those solutions may underlie many aspects of human intelligence.

Beyond immediate extensions to the techniques presented in this thesis, the work here suggests some broader directions for future research. One of these is the need for *skill management*. If we are to build agents that acquire their own skills across a variety of different tasks and environments, we require methods that both restrict the set of available skills at any given moment—so that the robot's task is not made *harder* by too many available skills—while at the same time using the information contained in its skill library to acquire new skills more efficiently.

Several strategies might be useful for skill management. Prior knowledge of the context in which a skill is frequently deployed may rule it out or reduce the likelihood of its selection in many contexts. We may devise methods for recognizing when one skill is simply a (perhaps slightly perturbed) copy of another, and thereby both speed up (or indeed simply avoid) policy learning and avoid the acquisition of a duplicate skill. Alternatively, an agent might learn when a pair or sequence of policies are all distorted copies of each other, and thereby acquire a parametrized option. Using such methods, an agent could build a compact library of prototype skills that it could use for skill policy initialization when acquiring a new skill.

More generally, methods that are able to achieve inter-skill transfer may prove useful in achieving open-ended learning. Transfer in reinforcement learning (surveyed by Taylor and Stone (2009)) has received significant research attention, including some work on learning portable options for skill transfer (Konidaris and Barto, 2007). However, considering transfer in the context of a single agent trying to become more efficient at skill acquisition results in a new and interesting setting where common features between tasks are clear, and the problem of maintaining a compact library of prototype skills and then selecting among them when initializing a new skill policy becomes critical.

Another broad direction for future research is that of building true skill hierarchies. In all of the work described here, and more generally in almost all of the work using the options framework, only the first level of a hierarchy is constructed: new skills are simply added to the actions already available to the agent.

The other two major hierarchical reinforcement learning frameworks—MaxQ (Dietterich, 2000) and HAMs (Parr and Russell, 1997)—describe hierarchies with more than one level. However, attempts to acquire either type of hierarchy through interaction with the environment have met with only very limited success. In addition, in both frameworks *the size of the state space actually increases with the level of the hierarchy*, which is obviously undesirable. Thus, the question of how to acquire true skill hierarchies remains open.

Ideally, a method for acquiring a skill hierarchy would result in a hierarchy with at least the following properties: each level of the hierarchy is an easier problem (one with a smaller state space and fewer actions) than the level below it; each level would form an MDP, with all levels (except possibly the first) discrete; that MDP would be well-formed in the sense that executing an action in it would never result in the agent reaching a configuration that is not also a state at that level; at the highest levels, each state either has a symbolic interpretation or form the basis for one; and finally, each layer (except possibly the first) would contain sufficient information to allow planning at that level *without a model of the environment*. The development of such a method would be a significant advance in the state of the art of hierarchical reinforcement learning.

Finally, another broad direction of interest is the use of robots and intelligent agents as synthetic models of human skill acquisition. Although we have discussed human skill acquisition in this thesis, no serious attempt was made to accurately model the human brain, except at a very abstract level. Hierarchical reinforcement learning has recently received some attention as a model of human skill acquisition (Botvinick *et al.*, 2009), and synthetic models might provide a useful mechanism for testing such models.

7.3 Conclusion

If we are to succeed in developing truly intelligent artificial systems, we are faced with the problem of moving beyond well-engineered systems that are adept at a single, specialized task. A unique trait of human intelligence is that it is flexible, adaptive and open-ended: humans are able to learn to become proficient in tasks as diverse as playing tennis, driving a car, weather forecasting, playing the stock market, assembling a computer, flying a plane, designing a circuit, creating furniture from wood, and proving a theorem. They are indeed capable of going *further* than proficiency: a human who is trained to expert-level proficiency in a task is often capable of using that training to exceed the level of expertise of their trainers.

Thus, this research is concerned with the question of how to build agents that can use their experience in solving some problems to later solve new, harder problems more efficiently. We have chosen to focus on skill acquisition, which involves the acquisition of procedural knowledge—knowledge about how to act—through interaction with an environment, because the primary function of the brain is control.

This thesis reflects the belief that hierarchical reinforcement learning provides a principled theoretical approach to skill acquisition, and has developed new methods that extend the reach of skill acquisition algorithms to the point where they can begin to be applied to high-dimensional, continuous domains. This thesis also reflects the belief that progress in artificial intelligence is best achieved through the design of relatively complete, integrated agents, especially robot systems. In this case, such an approach has both emulated the hypothesized *behavioral* advantages of skill acquisition, and also shed light on some of the *engineering* advantages of it. Although much remains to be done before we can even begin to claim the ability to create flexible, adaptive and open-ended artificial agents, the research presented here represents a small but hopefully concrete step in that direction.

BIBLIOGRAPHY

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

Abbeel, P., Coates, A., Quigley, M., and Ng, A. (2006). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems 19*.

Argall, B., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, **57**, 469–483.

Arkin, R. (1989). Neuroscience in motion: The application of schema theory to mobile robotics. In J.-P. Ewert and M. Arbib, editors, *Visuomotor Coordination: Amphibians, Comparisons, Models, and Robots*, pages 649–672. Plenum Press, New York.

Arkin, R. (1998). *Behavior-Based Robotics*. MIT Press, Cambridge, Massachusetts.

Atkeson, C., Moore, A., and Schaal, S. (1997). Locally weighted learning for control. *Artificial Intelligence Review*, **11**(1-5), 75–113.

Baird, L. (1993). Advantage updating. Technical Report WL-TR-93-1146, Wright-Patterson Air Force Base Ohio: Wright Laboratory.

Barto, A., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the International Conference on Developmental Learning*.

Bernstein, D. (1999). Reusing old policies to accelerate learning on new MDPs. Technical Report UM-CS-1999-026, Department of Computer Science, University of Massachusetts at Amherst.

Berthier, N. and Keen, R. (2006). Development of reaching in infancy. *Experimental Brain Research*, **169**, 507–518.

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer, New York, NY.

Bonarini, A., Lazaric, A., Restelli, M., and Vitali, P. (2006). Self-development framework for reinforcement learning agents. In *Proceedings of the Fifth International Conference on Development and Learning*.

Botvinick, M., Niv, Y., and Barto, A. (2009). Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, **113**, 262–280.

Boutilier, C., Dearden, R., and Goldszmidt, M. (1995). Exploiting structure in policy construction. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1104–1113.

Boyan, J. (1999). Least squares temporal difference learning. In *Proceedings of the 16th International Conference on Machine Learning*, pages 49–56.

Bradtke, S. and Barto, A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, **22**(1-3), 33–57.

Bradtke, S. and Duff, M. (1995). Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems 7*, pages 393–400.

Brock, O., Fagg, A., Grupen, R., Platt, R., Rosenstein, M., and Sweeney, J. (2005). A framework for learning and control in intelligent humanoid robots. *International Journal of Humanoid Robotics*, **2**(3).

Brooks, R. (1991). Intelligence without representation. In J. Haugeland, editor, *Mind Design II*, pages 395–420. MIT Press, Cambridge, Massachusetts.

Burridge, R., Rizzi, A., and Koditschek, D. (1999). Sequential composition of dynamically dextrous robot behaviors. *International Journal of Robotics Research*, **18**(6), 534–555.

Butterfield, J., Osentoski, S., Jay, G., and Jenkins, O. (2010). Learning from demonstration using a multi-valued function regressor for time-series data. In *Proceedings of the Tenth IEEE-RAS International Conference on Humanoid Robots*.

Chernova, S. and Veloso, M. (2007). Confidence-based policy learning from demonstration using Gaussian mixture models. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*.

Deegan, P., Thibodeau, B., and Grupen, R. (2006). Designing a self-stabilizing robot for dynamic mobile manipulation. In *Proceedings of the Robotics: Science and Systems Workshop on Manipulation for Human Environments*.

Dietterich, T. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, **13**, 227–303.

Digney, B. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In R. Pfeifer, B. Blumberg, J. Meyer, and S. Wilson, editors, *From Animals to Animats 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Zurich, Switzerland. MIT Press.

Diuk, C., Li, L., and Leffler, B. (2009). The adaptive k-meteorologists problems and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 249–256.

Dixon, K. and Khosla, P. (2004). Learning by observation with mobile robots: a computational approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Ericsson, K. (2006). The influence of experience and deliberate practice on the development of superior expert performance. In K. Ericsson, N. Charness, P. Feltovich, and R. Hoffman, editors, *Cambridge handbook of expertise and expert performance*, chapter 13, pages 685–706. Cambridge University Press, Cambridge, UK.

Fearnhead, P. and Liu, Z. (2007). On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society B*, **69**, 589–605.

Ferguson, A. (1999). *Managing My Life: My Autobiography*. Hodder and Stoughton Ltd., London.

Ferguson, K. and Mahadevan, S. (2006). Proto-transfer learning in markov decision processes using spectral methods. In *Proceedings of the ICML Workshop on Structural Knowledge Transfer for Machine Learning*.

Fox, E., Sudderth, E., Jordan, M., and Willsky, A. (2008). Nonparametric Bayesian learning of switching linear dynamical systems. In *Advances in Neural Information Processing Systems 21*.

Grollman, D. and Jenkins, O. (2010). Incremental learning of subtasks from unsegmented demonstration. In *International Conference on Intelligent Robots and Systems*.

Guenter, F., Hersch, M., Calinon, S., and Billard, A. (2007). Reinforcement learning for imitating constrained reaching movements. *Advanced Robotics*, **21**(13), 1521–1544.

Hart, S. (2008). Intrinsically motivated hierarchical manipulation. In *Proceedings of the 2008 IEEE Conference on Robots and Automation*.

Hart, S. (2009a). *The Development of Hierarchical Knowledge in Robot Systems*. Ph.D. thesis, University of Massachusetts Amherst.

Hart, S. (2009b). An intrinsic reward for affordance exploration. In *Proceedings of the Eighth IEEE International Conference on Development and Learning*.

Hart, S., Grupen, R., and Jensen, D. (2005). A relational representation for procedural task knowledge. In *Proceedings of the 2005 American Association for Artificial Intelligence (AAAI) Conference*.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250.

Hovland, G., Sikka, P., and McCarragher, B. (1996). Skill acquisition from human demonstration using a Hidden Markov Model. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Huber, M. (2000). A hybrid architecture for hierarchical reinforcement learning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 3290–3295.

Huber, M. and Grupen, R. (1997). Learning to coordinate controllers - reinforcement learning on a control basis. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1366–1371.

Huber, M., MacDonald, W., and Grupen, R. (1996). A control basis for multilegged walking. In *Proceedings of the 1996 IEEE Conference on Robotics and Automation*, pages 2988–2993.

Jenkins, O. and Matarić, M. (2004). Performance-derived behavior vocabularies: data-driven acquisition of skills from motion. *International Journal of Humanoid Robotics*, **1**(2), 237–288.

Johns, J. and Mahadevan, S. (2009). Sparse approximate policy evaluation using graph-based basis functions. Technical Report UM-CS-2009-041, University of Massachusetts Amherst.

Johns, J., Painter-Wakefield, C., and Parr, R. (2010). Linear complementarity for regularized policy evaluation and improvement. In *Advances in Neural Information Processing Systems 23*.

Johns, K. and Taylor, T. (2008). *Professional Microsoft Robotics Developer Studio*. Wrox Press, Hoboken, New Jersey.

Jong, N. and Stone, P. (2005). State abstraction discovery from irrelevant state variables. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 752–757.

Jong, N., Hester, T., and Stone, P. (2008). The utility of temporal abstraction in reinforcement learning. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*.

Jonsson, A. and Barto, A. (2001). Automated state abstraction for options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems 13*, pages 1054–1060.

Jonsson, A. and Barto, A. (2005). A causal approach to hierarchical decomposition of factored MDPs. In *Proceedings of the Twenty Second International Conference on Machine Learning*.

Kakade, S. (2002). A natural policy gradient. In *Advances in Neural Information Processing Systems 14*.

Kato, H. and Billinghurst, M. (1999). Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*.

Kohl, N. and Stone, P. (2004). Machine learning for fast quadrapedal locomotion. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 611–616.

Kolter, J. and Ng, A. (2009). Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 521–528.

Konidaris, G. and Barto, A. (2007). Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*.

Konidaris, G. and Hayes, G. (2005). An Architecture for Behavior-Based Reinforcement Learning. *Adaptive Behavior*, **13**(1), 5–32.

Konidaris, G. and Osentoski, S. (2008). Value function approximation in reinforcement learning using the Fourier basis. Technical Report UM-CS-2008-19, Department of Computer Science, University of Massachusetts Amherst.

Korf, R. (1997). Finding optimal solutions to Rubik's Cube using pattern databases. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 700–705.

Li, L., Walsh, T., and Littman, M. (2006). Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*.

Lozano-Perez, T., Mason, M., and Taylor, R. (1984). Automatic synthesis of fine-motion strategies for robots. *The International Journal of Robotics Research*, **3**(1), 3–24.

Luo, Z., Bell, D., McCollum, B., and Wu, Q. (2008). Learning to select relevant perspective in a dynamic environment. In *Proceedings of the International Joint Conference on Neural Networks*, pages 666–673.

Maes, P. and Brooks, R. (1990). Learning to coordinate behaviors. In *Proceedings of the Eighth Annual Meeting of the American Association for Artificial Intelligence*, pages 796–802, Cambridge, MA. MIT Press.

Mahadevan, S. (2008). *Representation discovery using harmonic analysis*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan Claypool.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, **55**(2–3), 311–365.

Mahadevan, S., Maggioni, M., Ferguson, K., and Osentoski, S. (2006). Learning representation and control in continuous markov decision processes. In *Proceedings of the Twenty First National Conference on Artificial Intelligence*.

Mannor, S., Rubinstein, R., and Gat, Y. (2003). The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning*, pages 512–519.

Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty First International Conference on Machine Learning*, pages 560–567.

Martin, F. (1998). *The Handy Board Technical Reference*. MIT Media Lab, Cambridge MA.

McCallum, A. (1996). Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*.

McGovern, A. and Barto, A. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368.

Mehta, N., Ray, S., Tadepalli, P., and Dietterich, T. (2008). Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the Twenty Fifth International Conference on Machine Learning*, pages 648–655.

Menache, I., Mannor, S., and Shimkin, N. (2002). Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the Thirteenth European Conference on Machine Learning*, pages 295–306.

Morimoto, J. and Doya, K. (2000). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*.

Mugan, J. and Kuipers, B. (2007). Learning distinctions and rules in a continuous world through active exploration. In *Proceedings of the Seventh International Conference on*

Epigenetic Robotics.

Mugan, J. and Kuipers, B. (2009). Autonomously learning an action hierarchy using a learned qualitative state representation. In *Proceedings of the Twenty First International Joint Conference on Artificial Intelligence*.

Murphy, R. (2000). *Introduction to AI Robotics*. MIT Press, Cambridge, Massachusetts, 1st edition.

Neumann, G., Maass, W., and Peters, J. (2009). Learning complex motions by sequencing simpler motion templates. In *Proceedings of the 26th International Conference on Machine Learning*.

Ng, A. and Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence, Proceedings of the Sixteenth Conference*.

Ng, A., Kim, H., Jordan, M., and Sastry, S. (2003). Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*.

Ng, A., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. In *Proceedings of the International Symposium on Experimental Robotics*.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computing*, **11**(2), 265–286.

Parr, R. and Russell, S. (1997). Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049.

Perkins, T. and Precup, D. (1999). Using options for knowledge transfer in reinforcement learning. Technical Report UM-CS-1999-034, Department of Computer Science, University of Massachusetts Amherst.

Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, **71**(7-9), 1180–1190.

Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the Third IEEE-RAS International Conference on Humanoid*

Robotics.

Pfeifer, R. and Scheier, C. (1999). *Understanding Intelligence*. MIT Press, Cambridge, MA.

Pickett, M. and Barto, A. (2002). PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the Nineteenth International Conference of Machine Learning*, pages 506–513.

Pierce, D. and Kuipers, B. (1997). Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, **92**(1-2), 169–227.

Platt, R., Fagg, A., and Grupen, R. (2002). Nullspace composition of control laws for grasping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Platt, R., Grupen, R., and Fagg, A. (2006). Improving grasp skills using schema structured learning. In *Proceedings of the Fifth International Conference on Development and Learning*.

Pook, P. and Ballard, D. (1993). Recognizing teleoperated manipulations. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning*. Ph.D. thesis, Department of Computer Science, University of Massachusetts Amherst.

Puterman, M. (1994). *Markov Decision Processes*. Wiley.

Rosenstein, M. and Barto, A. (2004). Supervised actor-critic reinforcement learning. In J. Si, A. Barto, A. Powell, and D. Wunsch, editors, *Learning and Approximate Dynamic Programming: Scaling up the Real World*, pages 359–380. John Wiley & Sons, Inc., New York.

Schaal, S. (1997). Learning from demonstration. In *Advances in Neural Information Processing Systems 9*.

Schaal, S., Atkeson, C., and Vijayakumar, S. (2000). Real time robot learning with locally weighted statistical learning. In *Proceedings of the International Conference on Robotics and Automation*, volume 1, pages 288–293.

Schmidhuber, J. (1991a). Curious model-building control systems. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1458–1463.

Schmidhuber, J. (1991b). A possibility for implementing curiosity and boredom in model-building neural controllers. In I. A. Meyer and S. W. Wilson, editors, *From Animals to Animats: Proceedings of the International Conference on Simulation of Adaptive Behavior*, pages 222–227.

Schölkopf, B. and Smola, A. (2001). *Learning with Kernels*. MIT Press.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, **6**, 461–464.

Şimşek, Ö. and Barto, A. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758.

Şimşek, Ö. and Barto, A. (2008). Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems 22*.

Şimşek, Ö. and Barto, A. G. (2006). An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning*.

Şimşek, Ö., Wolfe, A., and Barto, A. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning*.

Singh, S., Barto, A., and Chentanez, N. (2004). Intrinsically motivated reinforcement learning. In *Proceedings of the 18th Annual Conference on Neural Information Processing Systems*.

Singmaster, D. (1981). *Notes on Rubik's 'Magic Cube'*. Enslow Publishers, Inc., Berkeley Heights, NJ.

Smart, W. (2002). *Making Reinforcement Learning work on Real Robots*. Ph.D. thesis, Brown University.

Soni, V. and Singh, S. (2006). Reinforcement learning of hierarchical skills on the Sony Aibo robot. In *Proceedings of the Fifth International Conference on Development and*

Learning.

Sprague, N. (2009). Predictive projections. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*.

Stober, J. and Kuipers, B. (2008). From pixels to policies: a bootstrapping agent. In *Proceedings of the 7th IEEE International Conference on Development and Learning*.

Stout, A. and Barto, A. (2010). Competence progress intrinsic motivation. In *Proceedings of the Ninth IEEE International Conference on Development and Learning*.

Stout, A., Konidaris, G., and Barto, A. (2005). Intrinsically motivated reinforcement learning: A promising framework for developmental robot learning. In *The AAAI Spring Symposium on Developmental Robotics*.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sutton, R., Precup, D., and Singh, S. (1998). Intra-option learning about temporally abstract actions. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 556–564.

Sutton, R., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112**(1-2), 181–211.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063.

Szita, I. and Lörincz, A. (2006). Learning Tetris using the noisy cross-entropy method. *Neural Computation*, **18**(12), 2936–2941.

Taylor, M. and Stone, P. (2007). Towards reinforcement learning representation transfer. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*.

Taylor, M. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, **10**, 1633–1685.

Tedrake, R. (2009). LQR-Trees: Feedback motion planning on sparse randomized trees. In *Proceedings of Robotics: Science and Systems*.

Thrun, S. and Schwartz, A. (1995). Finding structure in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. The MIT Press.

van Seijen, H., Bakker, B., and Kester, L. (2007). Reinforcement learning with multiple, qualitatively different state representations. In *Proceedings of NIPS 2007 Workshop on Hierarchical Organization of Behavior*.

Vigorito, C. and Barto, A. (2008). Autonomous hierarchical skill acquisition in factored MDPs. In *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT.

Vigorito, C. and Barto, A. (2009). Incremental structure learning in factored MDPs with continuous states and actions. Technical Report UM-CS-2009-029, University of Massachusetts Amherst.

Vijayakumar, S. and Schaal, S. (2000). Fast and efficient incremental learning systems for high-dimensional movement systems. In *Proceedings of the International Conference on Robotics and Automation*.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., and Thelen, E. (2000). Autonomous mental development by robots and animals. *Science*, **291**(5504), 599–600.

Xuan, X. and Murphy, K. (2007). Modeling changing dependency structure in multivariate time series. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*.

Zang, P., Zhou, P., Minnen, D., and Isbell, C. (2009). Discovering options from example trajectories. In *Proceedings of the Twenty Sixth International Conference on Machine Learning*, pages 1217–1224.

"Childe Roland to the Dark Tower came."