TEMPORAL ABSTRACTION
IN REINFORCEMENT LEARNING

A Dissertation Presented

by

DOINA PRECUP

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 2000

Department of Computer Science

TEMPORAL ABSTRACTION
IN REINFORCEMENT LEARNING

A Dissertation Presented

by

DOINA PRECUP

Approved as to style and content by:

____
Richard S. Sutton, Chair

____
Andrew G. Barto, Member

____
Roderic A. Grupen, Member

____
Christopher V. Hollot, Member

____
Leslie P. Kaelbling, Member

____
James F. Kurose, Department Chair
Department of Computer Science

To my family

ACKNOWLEDGMENTS

My deepest gratitude goes to Richard Sutton, my thesis advisor. He helped me tremendously to become a better researcher and a better writer, and provided constant inspiration, feedback and support, as well as a wealth of intriguing ideas. It was a great honor and a real pleasure for me to work with him, and I cannot imagine a better advisor. I also want to thank the members of my thesis committee, Andy Barto, Rod Grupen, Leslie Kaelbling and Chris Hollot, for their patience in reading drafts of my thesis, and their very useful feedback.

Thanks to Satinder Singh for many inciting conversations and good research ideas. His help with structuring some of the mathematical results included in this dissertation is greatly appreciated.

During my graduate studies I also had the opportunity to collaborate with other faculty members. I especially want to thank Paul Utgoff, for his guidance during the first part of my graduate studies; Eliot Moss, for useful conversations regarding the potential applications of my research; Andy Barto, for being a great resource and for his constant feedback on my research; Rod Grupen and Paul Cohen, for their constant support and for inspiring conversations. During my internship at AT&T Research Labs, Michael Kearns and Dave McAllester provided great comments on my research. I also want to thank Prof. Ioan Alfred Letia from the Technical University Cluj-Napoca, Romania, for inciting my interest in artificial intelligence and machine learning.

The ANW research group was a stimulating and fun research environment. I especially want to thank Ted Perkins and Amy McGovern for many inspiring conversations, for reading carefully through several preliminary drafts of the dissertation, and for their constant friendship and support. Thanks to Andy Fagg, Balaraman Ravindran, Leo Zelevinsky, Nathan Sitkoff, Anders Jonsson and Dan Bernstein for their interest in my work, and for providing useful comments on preliminary drafts of this dissertation. Thanks to Jitu Padhye for being a constant friend in good and bad times.

This research would not have been possible without the financial support that I received during my graduate studies from the Fulbright foundation, the University of Massachusetts, and AT&T Research Labs. Part of this research was supported by NSF grant ECS-9511805 and grant AFOSR-F49620-96-1-0254, both to Andrew Barto and Richard Sutton, and by NSF grant ECS-9980062 to Andrew Barto and John Moore.

Finally, a special thanks to my family, who made many sacrifices over time to help me reach this point. Their love, support and encouragement has been invaluable over the years. This dissertation is dedicated to them.

ABSTRACT

TEMPORAL ABSTRACTION IN REINFORCEMENT LEARNING

MAY 2000

DOINA PRECUP

B.Sc., TECHNICAL UNIVERSITY CLUJ-NAPOCA, ROMANIA

M.Sc., UNIVERSITY OF MASSACHUSETTS, AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Richard S. Sutton

Decision making usually involves choosing among different courses of action over a broad range of time scales. For instance, a person planning a trip to a distant location makes high-level decisions regarding what means of transportation to use, but also chooses low-level actions, such as the movements for getting into a car. The problem of picking an appropriate time scale for reasoning and learning has been explored in artificial intelligence, control theory and robotics. In this dissertation we develop a framework that allows novel solutions to this problem, in the context of Markov Decision Processes (MDPs) and reinforcement learning.

In this dissertation, we present a general framework for prediction, control and learning at multiple temporal scales. In this framework, temporally extended actions are represented by a way of behaving (a policy) together with a termination condition. An action represented in this way is called an *option*. Options can be easily

incorporated in MDPs, allowing an agent to use existing controllers, heuristics for picking actions, or learned courses of action.

The effects of behaving according to an option can be predicted using multi-time models, learned by interacting with the environment. In this dissertation we develop multi-time models, and we illustrate the way in which they can be used to produce plans of behavior very quickly, using classical dynamic programming or reinforcement learning techniques.

The most interesting feature of our framework is that it allows an agent to work simultaneously with high-level and low-level temporal representations. The interplay of these levels can be exploited in order to learn and plan more efficiently and more accurately. We develop new algorithms that take advantage of this structure to improve the quality of plans, and to learn in parallel about the effects of many different options.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . **v**

ABSTRACT . . . . **vii**

LIST OF FIGURES . . . . **xii**

CHAPTER

1. INTRODUCTION **1**

1.1 Markov Decision Processes and Reinforcement Learning 3
1.2 Kinds of Abstraction 4
1.3 Contributions 7
1.4 Outline 11

2. BACKGROUND **13**

2.1 Temporal Abstraction in Classical AI 13
2.2 Temporal Abstraction in Control 16
2.3 Reinforcement Learning (MDP) Framework 17
2.4 Prior Work on Temporal Abstraction in Reinforcement Learning . . . 20

3. OPTIONS **25**

3.1 Policies and Options 26
3.2 Specifying Options 29

3.2.1 Flat Options 29
3.2.2 Hierarchical Options 32

3.3 Value Functions for Options 39
3.4 Conclusions 41

4. SMDP METHODS FOR LEARNING AND PLANNING WITH OPTIONS **43**

4.1 SMDPs and Options 43
4.2 SMDP Planning 45
4.3 Illustration: Rooms Example 48
4.4 Illustration: Random Options 52
4.5 SMDP Value Learning 55
4.6 Conclusions 57

5. INTRA-OPTION LEARNING **58**

5.1 Intra-Option Model Learning 59
5.2 Illustration of Intra-Option Model Learning 63
5.3 Intra-Option Value Learning 64
5.4 Illustration of Intra-Option Value Learning 66
5.5 Conclusions 69

6. OFF-POLICY LEARNING: MONTE CARLO METHODS **71**

6.1 Policy Evaluation 72
6.2 Importance Sampling 73
6.3 Applying Importance Sampling to MDPs 74
6.4 Per-Decision Importance Sampling 76
6.5 Conclusions 79

7. OFF-POLICY LEARNING: TEMPORAL-DIFFERENCE METHODS **81**

7.1 One-Step TD Learning 81
7.2 Temporal-Difference Per-Decision Importance Sampling 83
7.3 Tree Backup Algorithm 86
7.4 Empirical Comparison 90
7.5 A Unified View of the Two Multi-Step TD Algorithms 92
7.6 Applying Multi-Step Off-Policy Learning to Intra-Option Learning . . 93
7.7 Conclusions 95

8. ADAPTING OPTIONS **96**

8.1 Interruption 96
8.2 Examples of Interruption 99
8.3 Termination Iteration 103
8.4 Illustration of Termination Iteration 107
8.5 Policy Iteration for Options 109
8.6 Conclusions 112

9. CREATING NEW OPTIONS FROM SUBGOALS **113**

9.1 Subgoals of Achievement 114
9.2 Subgoals and Transfer 116

10. CONCLUSIONS AND FUTURE WORK **119**

10.1 Contributions 119
10.2 Future Research Directions 121

BIBLIOGRAPHY **125**

LIST OF FIGURES

Figure		Page
4.1	The rooms example is a gridworld environment with stochastic cell-to-cell actions and room-to-room hallway options. Two of the hallway options are suggested by the arrows labeled o_1 and o_2. The labels G_1 and G_2 indicate two locations used as goals in experiments described below.	49
4.2	The policy underlying one of the eight hallway options.	49
4.3	Value functions formed over iterations of planning by synchronous value iteration with primitive actions and with hallway options. The hallway options enabled planning to proceed room-by-room rather than cell-by-cell. The area of the disk in each cell is proportional to the estimated value of the state, where a disk that just fills a cell represents a value of 1.0.	51
4.4	An example in which the goal is different from the subgoal of the hallway options. Planning here was done by SVI with options $\mathcal{O} = \mathcal{A} \cup \mathcal{H}$. Initial progress was due to the models of the primitive actions, but by the third iteration room-to-room planning dominated and greatly accelerated planning.	52
4.5	Empty gridworld task. Options allow the error in the value function estimation to decrease more quickly	53
4.6	Performance of SMDP Q-learning in the rooms example with various goals and sets of options. After 100 episodes, the data points are averages over groups of 10 episodes to make the trends clearer. The step size parameter was optimized to the nearest power of 2 for each goal and set of options. The results shown used $\alpha = \frac{1}{8}$ in all cases except that with $\mathcal{O} = \mathcal{H}$ and G_1 ($\alpha = \frac{1}{16}$) and that with $\mathcal{O} = \mathcal{A} \cup \mathcal{H}$ and G_2 ($\alpha = \frac{1}{4}$).	56
5.1	Learning curves for model learning by SMDP and intra-option methods.	64

5.2 The learning of option values by intra-option methods without ever selecting the options. The value of the greedy policy goes to the optimal value (left panel) as the learned values approach the correct values (as shown for one state, in the right panel). 67

5.3 Comparison of SMDP, intra-option and macro Q-learning. Intra-option methods converge faster to the correct values. 68

6.1 Different target and sampling distributions 73

6.2 Comparison of classical and weighted importance sampling on 100 randomly generated MDPs. On the left, the behavior policy chose 50-50 from the two actions. On the right, the behavior policy chose with 20-80 probabilities, exactly opposite to the target policy. In both cases, the weighted algorithm is faster and more stable. . . . 75

6.3 Comparison of Classical (Per-Return) and Per-Decision Monte Carlo Importance Sampling Algorithms 79

7.1 Backup tree for one-step TD 82

7.2 Comparison of One-Step TD and Monte Carlo Importance Sampling 82

7.3 Backup diagram for the tree backup algorithm 87

7.4 Comparison of all the off-policy learning algorithms on a testbed of 100 random MDPs. 91

7.5 Full trajectory tree for an MDP 92

7.6 Error curves for the reward predictions (left panel) and next-state predictions (right panel) for the SMDP, one-step intra-option learning, and intra-option learning with tree backup eligibility traces. 95

8.1 Interruption in navigating with landmark-directed controllers. The task (up) is to navigate from S to G in minimum time using options based on controllers that run each to one of seven landmarks (the black dots). The circles show the region around each landmark within which the controllers operate. The thin line shows the optimal behavior that uses only these controllers run to termination, and the thick line shows the corresponding interrupted behavior, which cuts the corners. The lower panels show the state-value functions for the SMDP-optimal and interrupted policies. Note that the latter is greater. 100

8.2 Phase-space plot of the SMDP and interrupted policies in a simple dynamical task. The system is a mass moving in one dimension: $x_{t+1} = x_t + \dot{x}_{t+1}$, $\dot{x}_{t+1} = \dot{x}_t + a_t - 0.175\dot{x}_t$ where x_t is the position, $\dot{x}_t$ the velocity, 0.175 a coefficient of friction, and the action a_t an applied force. This continuous system is controlled at a discrete time scale of 0.001. Two controllers are provided as options, one that drives the position to $x^* = 1$ and the other to $x^* = 2$. Whichever option is being followed at time t, its target position x^* determines the action taken, according to $a_t = 0.01(x^* - x_t)$. 101

8.3 The mission planning task and the performance of policies constructed by SMDP methods, interruption of the SMDP policy, and an optimal static re-planner that does not take into account possible changes in weather conditions. 102

8.4 Simple MDP in which adding more options can decrease the quality of the interrupted policy. 106

8.5 The result of termination iteration in the rooms environment, in the room containing the goal. The states in which the options terminate immediately are marked by small squares. The initiation sets of the options are shaded. The two options available in the room are terminated immediately if the option would take the agent away from the goal state. 108

8.6 The result of termination iteration in the rooms environment, in the north-west room. One of the options gets very fragmented. 109

9.1 Learning subgoal-achieving hallway options under random behavior. Shown on the left is the error between $Q_g(s,a)$ and $Q^*_g(s,a)$ averaged over $s \in \mathcal{I}$, $a \in \mathcal{A}$, and 30 repetitions. The right panel shows the learned values for two options at one state (maximum over action values) approaching their correct values. 115

9.2 Two different optimal policies for options given two different subgoal values at the target hallway. A subgoal value of +10 (left) results in a more direct policy than a subgoal of +1. 116

9.3 A subgoal to which a hallway option does *not* transfer. The option for passing from the lower-left room through to the state with subgoal value 10 no longer works because of the state with subgoal value −1. The original model of this option is overpromising with respect to the subgoal. 117

CHAPTER 1

INTRODUCTION

How should an intelligent agent represent its knowledge of the world in order to be able to reason and learn about different courses of action? This is a key, long standing question in artificial intelligence (AI), as well as in robotics and control engineering.

For instance, consider a robot docking with its battery charger. Should it plan at the lowest level of motor torques for its wheels? Or should it consider high-level actions such as "locate charger," "navigate to charger" and "precise docking?" Both levels have their advantages. Reasoning at the higher level allows for short, compact plans but might lack the detail necessary to carry out the plan or to optimize it further in specific situations. The lower level allows filling in these details but is too fine-grained to be used efficiently. Ideally, an intelligent agent should have the ability to use both levels, as required by the circumstances it faces. For instance, the agent could start with a high-level plan, then refine it into low-level actions. Moreover, the agent should be able to reason and acquire knowledge at both levels in parallel, from the real experience that it gets by acting in its environment.

Most approaches to action representation use hand crafted knowledge representations specialized for the specific task under consideration. Over the years, these representations have taken many different forms. The AI planning community has considered using macro-operators and, more recently, closed-loop sequences of actions as the basis for describing the agent's behavior. Roboticists and control engineers have considered methodologies for combining and switching between independently designed controllers or behaviors. All these approaches combine in some way the low-level actions in the domain into *courses of action* that can be temporally extended (i.e. take a variable, unknown amount of time). Some of these approaches address the issues of learning useful courses of actions, and learning how to combine existing courses of action in an efficient way.

In this dissertation we address the issue of representing knowledge about actions and courses of action within the context of reinforcement learning (RL) and Markov Decision Processes (MDPs). Our main goal is to provide a framework for representing, learning, and reasoning about courses of action that are temporally extended, stochastic and contingent of events. This framework should satisfy several criteria:

Expressivity: The representation should be able to include basic kinds of common sense knowledge, similar to that used by humans when reasoning at multiple temporal scales.

Clarity: The framework for describing actions should be mathematically grounded in primitive observations and actions. Many of the frameworks used in AI use vague descriptions, which are made to be understandable by humans but have no independent meaning for the agent.

Generality: The knowledge representations used to describe courses of actions should be relevant for a variety of different tasks, rather than having special-purpose representations for a specific task. The knowledge representation should also allow the use of the existing body of theory and algorithms from the RL and MDP literature, with minimal extensions.

Suitability for planning: The framework should specify methods for building predictive models for the actions that can be used to make decisions. Using these models, the agent should be able to build plans for acting in the environment. Rather than working at a fixed time scale, the models and planning methods used should allow intermixing and relating of different temporal scales.

Suitability for learning: The agent should have the ability to learn about the consequences of its courses of action, and to improve its strategy for picking actions over time. This is especially important for agents acting in stochastic environments, and having incomplete knowledge about the environment, which is the case we are considering. The knowledge representations adopted by the agent should be easy to change based on observations from the environment, in order to adapt to new situations, to changes in the environment, and to changes in the agent's own goals.

In this dissertation we elaborate a framework for representing and reasoning about temporally extended actions that satisfies these criteria. We do not address the issue of selecting a particular level of temporal abstraction for representing the actions. We are just providing the ability to represent and reason about actions with different time scales.

1.1 Markov Decision Processes and Reinforcement Learning

In this dissertation, we use Markov Decision Processes (MDPs) and reinforcement learning (RL) as a theoretical foundation for the study of temporal abstraction. Reinforcement learning is a computational approach to automating goal-directed learning and decision making (Sutton & Barto, 1998). It encompasses a broad range of methods for determining optimal ways of behaving in complex, uncertain and stochastic environments.

In reinforcement learning, MDPs are used as a formal framework for defining the interaction between the agent and its environment. MDPs are a standard, very general formalism for studying stochastic, sequential decision problems (Bellman, 1957; Howard, 1960). In an MDP, an agent interacts with its environment at some discrete, lowest-level time scale. On each time step, the agent perceives the state of the environment and on that basis chooses a primitive action. In response to this action, the environment produces one time step later a numerical reward, and the agent finds itself in a new next state. The goal of the agent is to find a way of behaving, a mapping of states into actions that maximizes the numerical reward signal received over time. Such a mapping is called a policy. The expected long-term reward associated with a policy is called a value function.

Many problems from artificial intelligence, control theory and operations research can be tackled using this framework. Reinforcement learning has been used successfully in a large variety of applications, ranging from elevator dispatching (Crites & Barto, 1996) to a world champion backgammon player (Tesauro, 1995), and from schedule optimization (Zhang & Dietterich, 1995) to robotic soccer players (Stone & Veloso, 1999). MDP planning has also been used successfully to tackle classical AI planning tasks (Boutilier et al., 1999).

MDPs and RL provide a good theoretical foundation for grounding the semantics of temporally extended actions. Such actions can be described in terms of closed-loop policies that are used to pick primitive actions for some duration of time. The effects of the actions can be related to the primitive observations and actions of the system. This allows the agent to learn about the effects of its courses of action by interacting with the environment, without human supervision. RL planning and learning algorithms can be used to build knowledge representations for the agent, which are clear and expressive. RL and MDPs will allow us to build a framework that satisfies the desired criteria enumerated above.

1.2 Kinds of Abstraction

Artificial intelligence typically tackles large, complex problems. Solving such problems requires adequate representations. *Abstraction* in classical AI refers to a broad range of techniques that attempt to provide a more compact representation for the problem at hand, both at the state level and at the action level. Once the problem is solved using the new representation, the solution is translated back into the original state and action space.

There are two main kinds of abstraction methods in AI. *State abstraction* methods attempt to find a different representation of the state space, which makes it easier to find a solution. Typically such representations ignore certain details of the original state information. *Temporal abstraction* methods, the subject of this dissertation, attempt to find a time scale that is adequate for describing the actions of an AI system and the evolution of the system as a result of its actions. Methods for modeling the evolution of physical systems at adequate time scales have been formally studied by mathematicians and engineers since the 1800s. The previous AI research on temporal abstraction methods has not focused as much on modeling an independently evolving system. The main focus has been to represent the actions available to an AI agent in a way that allows the agent to build reliable predictive models of the effects of its actions, and use these models efficiently to construct plans. A central idea that has emerged throughout this research is to combine the low-level actions available to the AI system into macro-actions or behaviors. Such macro-actions are typically executed during an extended time period. The use of higher-level, temporally extended actions allows AI systems to solve the tasks they face in a smaller number of action steps. But how should an AI agent decide which extended actions to choose given a particular problem? For how long should it execute an extended action? How could it find macro-actions that are useful? These questions can be asked in a very clear and precise way for systems based on RL.

Like many other AI systems, reinforcement learning systems often use abstraction techniques in order to solve large problems efficiently. State abstraction methods have been the focus of extensive theoretical research and empirical studies in RL systems because they allow RL agents to tackle domains with large discrete or continuous state spaces. In such domains, the issue is not only to represent the value function compactly (with low memory costs), but also to generalize a limited amount of experience in order to produce a good approximation of the value function over the entire state space. A standard approach is to use some form of function approximation to generate a new representation of the state space. The value function is then learned using this new representation. This approach has been used successfully in several large-scale applications (see, e.g. Crites & Barto, 1995; Lin, 1993; Tesauro, 1995).

Recent RL research has focused more on methods for temporal abstraction (Singh, 1992b; Singh, 1992a; Kaelbling, 1993b; Kaelbling, 1993a; Dayan & Hinton, 1993; Dean & Lin, 1995; Dietterich, 1998; Parr, 1998; Sutton, 1995; Precup & Sutton, 1998; Sutton et al., 1999a). All these methods are centered around the idea of defining new, temporally extended actions as closed-loop ways of behaving (policies) that terminate. Such extended actions can be defined by the designer of the system, based on prior knowledge of the task at hand, or can be acquired automatically by the RL agent itself. These actions can be used by the RL agent instead of (or in addition to) the primitive actions to generate behavior. The agent can also learn predictive models and value functions for extended actions, which it can then use to improve its behavior. The methods proposed so far differ mainly in the representation used for the temporally extended actions, and in the learning algorithms used to acquire knowledge about the effects of these actions. A detailed account of the existing methods is given in chapter 2.

The research we present in this dissertation also focuses on temporal abstraction. Our research builds on the work of Singh (1992b; 1992a), Sutton (1995) and Sutton & Pinnette (1985). It is most closely related to the recent research of Parr (1998) and Dietterich (1998). We define a simple and explicit representation of extended actions, as policies together with a termination condition. Models and value functions for such actions can be learned using minimal extensions of traditional RL algorithms. Because this representation is very simple, we are able to formulate algorithms that improve extended actions and that can learn efficiently about many actions from the same real experience.

Some of the frameworks mentioned above combine state and temporal abstraction (Dayan & Hinton, 1993; Dietterich, 1998) by constructing a new representation of the state space together with a new set of actions. Unlike these methods, we address the issue of temporal abstraction independently of the state space representation. Our approach can be combined with any existing methods for state abstraction and function approximation.

1.3 Contributions

In this dissertation we present a general framework for prediction, control and learning at multiple temporal scales. In this framework, temporally extended actions are represented by a (partial) policy together with a termination condition. An action represented in this way is called an *option*. Unlike other representations for temporally extended actions, options are a strict generalization of the primitive actions. This enables primitive actions and temporally extended actions to be used interchangeably by a learning agent. Options are a very general way to specify extended actions, and they can be used to describe both open-loop and closed-loop policies. In this dissertation, we show how hierarchical descriptions and other descriptions for extended actions introduced in RL research can be mapped into options.

In developing this new framework, we build on the existing theory of Semi-Markov Decision Processes (SMDPs). SMDPs are models of continuous-time, discrete-event systems. In an SMDP, actions can take variable, stochastic amounts of time. When introducing options in an MDP, we obtain a new, related SMDP. SMDP theory provides planning and learning methods similar to those used for MDPs, and we can use any of these methods to find an optimal policy in the higher-level SMDP defined by the options.

The advantage of transforming the initial MDP is that we usually obtain a considerably smaller SMDP, in which planning and learning can proceed faster. But the best long-term rewards that can be obtained in the SMDP are typically lower than those that can be obtained in the original MDP. These rewards are limited by the quality of the options that the learning agent can execute. So if an agent would have to choose between working at the MDP level or at the SMDP level, it would need to accept a trade-off between speed of computation and the quality of the solution obtained.

In this dissertation we take a different approach to this tradeoff. Instead of restricting the algorithms to act either at the SMDP level or at the MDP level, learning and planning can proceed at *both levels*. Leverage can be gained from inter-relating the different time scales at which the system is acting. For instance, consider a policy for finding a cup of coffee in an unknown building. Such a policy could use options for following corridors and going through doors, and it could be computed fairly quickly. But what if, while following a corridor, the agent sees the coffee pot? An SMDP solution would still follow the corridor until the end, and then of course the agent could choose to come back. Working between MDPs and SMDPs allows the agent instead to immediately interrupt the option, and head for the coffee. We elaborate an this idea in chapter 8. While following the corridor, the agent can also learn about different other destinations that could be of interest later (as we will show in chapter 5). Such learning is not possible if we only consider the SMDP level. Finally, the experience can be used to improve overall navigation performance and each of the individual options used.

We propose novel algorithms that work between the MDP and the SMDP level, in order to obtain better performance than the SMDP level alone, in a shorter time than working at either level.

One class of algorithms addresses the issue of learning about the consequences of options from experience. In the SMDP view, an agent can only learn about one option at a time, by executing that option, observing its consequences and adjusting its prediction accordingly. This kind of execution can be very expensive, especially for options that take a long time to execute, or options that can lead to failure outcomes. Our new learning algorithms (presented in chapter 5) do not require complete execution of an option before learning about it. Instead, we define learning rules that use the information obtained after each primitive action executed. In this way, we take advantage of the underlying MDP structure. The learning algorithms use the information obtained from the transitions that take place on every time step, in order to learn about the consequences of all the options that could have caused these transitions. For instance, our navigation robot can learn about navigating towards the coffee pot even from the first step it takes in the right direction, without waiting until it actually reaches its destination. And if the copier happens to be near the coffee pot, it can learn about navigating toward it while it is actually going for the coffee. There are two major advantages to this approach. First, the real experience gathered from the environment is used more efficiently, by learning about many options from the same data. Second, the options do not have to be executed to completion. This gives the learning agent the opportunity to consider many hypothetical options, without losing time to gather information about each of them.

Our learning algorithms are an example of a more general RL idea, called off-policy learning. Off-policy learning is learning about one way of behaving while following a different way of behaving. Prior to this work, off-policy learning was used mainly in control algorithms, such as Q-learning (Watkins, 1989), in which the agent learns how to behave optimally while taking many sub-optimal, exploratory actions. In this work, we use off-policy learning in order to learn about many different ways of behaving while following just one way of behaving. This poses a technical problem that has not been studied in depth before: can we use standard RL methods, such as temporal-difference learning (TD) and eligibility traces, to evaluate the long-term reward of a given way of behaving from data generated by a different way of behaving by using TD learning (Sutton, 1988) is one of the central ideas of RL, and it allows a learning agent to update its predictions about the long-term reward it will obtain based on the transitions observed on every time step. Eligibility traces are a mechanism that allows such updates to take into account not only immediate information, but also information from the more distant future. TD methods with eligibility traces have been extensively studied in the case in which the agent learns about its current behavior. A few algorithms have been proposed for the control problem of learning the optimal policy, but their theoretical properties are not currently known. In this dissertation, we present the first eligibility trace, temporal-difference, off-policy learning algorithms for evaluating policies, and prove convergence properties for these algorithms. We apply these algorithms mainly in the context of learning about options, but they can be applied for solving other RL problems as well.

Another class of algorithms that we propose and investigate concern changing the options available to the agent in order to obtain better performance. One such change is based on the simple idea that the agent should not be forced to execute an option to completion if a better alternative is available. We allow options to interrupt whenever a more promising option can be initiated. This change can be temporary, during the actual execution of the options, but it can also be used as part of a new approach for determining the best termination conditions for a set of options. Another similar algorithm can be used to change the internal way in which an option picks primitive actions.

1.4 Outline

In chapter 2 we review the literature on modeling physical systems at different time scales and on describing temporally extended actions in control research, in robotics, and in artificial intelligence. We discuss in detail the notion of macro-operators from classical AI and the different approaches to temporal abstraction in MDPs and RL, because these are especially relevant for our research.

In chapter 3 we introduce options, the main element of our framework for handling temporally extended actions. We show how options can be described explicitly through flat or hierarchical representations. We also develop the links between the options framework and other approaches to temporal abstraction described in the RL and AI literature.

In chapter 4 we establish the link between options and SMDP theory. In particular, we introduce predictive models of options, which can be used to model their consequences for planning purposes, and summarize basic SMDP learning and planning methods that can be used to approximate the values and models of options.

In chapter 5 we introduce a new class of algorithms for learning about options, called intra-option methods. Intra-option methods have been designed to make efficient use of the agent's experience in the environment. Unlike SMDP methods, intra-option methods can learn about many options from experience generated by just one way of behaving. As we mentioned in the previous section, these learning algorithms have significant advantages over SMDP learning methods in terms of speed and flexibility.

As we mentioned before, intra-option learning methods are examples of off-policy learning. In chapters 6 and 7 we study off-policy learning in depth, in a more general context. We focus on the problem of policy evaluation, in which the agent is trying to predict the long-term reward it can receive while following a specified way of behaving. We present the first TD algorithms with eligibility traces for solving this general problem, and we illustrate how these algorithms can be applied for learning about options.

In chapter 8, we return to the study of the options framework. Whereas in the rest of the dissertation, we assume that the set of options is fixed, in this chapter we consider changing the options to obtain better performance. We present algorithms that allow us to change both the termination conditions and the internal policies of the options. These changes can be performed on-line during behavior, to improve performance in particular situations.

In chapter 9 we discuss briefly the acquisition of new options. Options can be viewed as policies that achieve subgoals; we present one way in which such subgoals can be formulated that allows options and models to be transferred from one task to another. We discuss some of the problems that arise during transfer and how these problems can be avoided.

In chapter 10 we summarize the main contributions of the dissertation and the open issues.

CHAPTER 2

BACKGROUND

Reasoning and learning about temporally extended actions has been studied extensively in several fields. In this chapter we review the main lines of research on this topic from classical AI, control theory and reinforcement learning. We also review the conventional reinforcement learning framework, which is used as a theoretical basis throughout this dissertation.

2.1 Temporal Abstraction in Classical AI

The problem of using abstraction to facilitate planning has been a key focus of AI research since its early days (see e.g., Fikes, Hart, & Nilsson, 1972; Newell & Simon, 1972). The key idea was to replace the low-level actions available to solve a given task by *macro-operators*, open-loop sequences of actions that can achieve some subgoal.

Different forms of representation have been used for macro-operators. For instance, Sacerdoti (1977; 1974) used procedural nets to represent an action hierarchy. Each node represents an action both through a "declarative representation" (analogous to a model) and through a "procedural representation" (analogous to an internal policy). More recently, Levinson and Fuchs (1994) proposed a decomposition of macros into patterns and weights. The patterns are partially matched with states, while the weights put an upper bound on the distance to the goal. The pattern-weight pairs can then be used in a hill-climbing procedure to search for a solution to the planning task. The idea of having a representation for the way in which the macro-operator picks actions, as well as some predictive model for its consequences, is common to many approaches to macro-operators.

A key issue in the AI research is learning *useful* macro-operators, which can be re-used to solve different planning problems. Korf (1985; 1987) introduced the notion of independent and serializable subgoals, which provide a decomposition of a planning problem. Such subgoals can be solved individually, and then the corresponding macro-operators are combined to solve the larger planning problem. Korf proposed a generate-and-test approach for constructing the macro-operators. The SOAR system used a *chunking mechanism*, by which action sequences used to solve subtasks were memorized as macro-operators (see, e.g., Laird, Rosebloom and Newell, 1986) . Such macro-operators were then stored in a table, and no further modifications were made. The SOAR research identified a *utility problem*: a moderate number of macro-operators speeds up planning significantly, but when too many macro-operators exist, the cost of choosing the right ones becomes significant, and planning speed slows down, even below the performance of using low-level actions.

Minton (1988) and Knoblock (1990) addressed the learning of macro-operators in conjunction with the pre-conditions under which they succeed or fail. Their work identified conditions under which a solution obtained in an abstracted state and action space can be indeed executed. Iba (1989) designed a heuristic method for automatically acquiring macro actions. The heuristic assumes the use of a value function to determine the "goodness" of each state. Sequences of primitive actions that lead from one peak in the value function to another peak are grouped together to propose a macro, which can later be discarded, based on domain-specific filters.

Drescher (1991) advocated a constructive approach in which knowledge about the world is gradually acquired in the form of *schemas*, elementary models containing a context (state), an action and a result (new state). Schemas are built with the purpose of capturing "regularities" in the environment. Subsequently, they are also used to construct new, composite actions, by sequencing existing primitives.

More recent work focuses on methods that use closed-loop macros, as well as more reactive ways of executing the plans, and re-planning in case of failure. Pierce and Kuipers (1994) describe a method for creating an abstraction of the state and action spaces for an agent operating in a continuous world. The state space abstraction is based on features constructed from the raw input, using a generate-and-test method. In this discrete state space, the agent can use primitive actions, as well as open-loop and closed-loop behaviors. The closed-loop behaviors are constructed by generalizing the open-loop behaviors, based on information contained in the models of the primitive actions.

Nilsson (1994) proposed a teleo-reactive planning system in which actions with variable duration (potentially whole behaviors) are represented through their pre-images and their post-conditions, expressed in a logical form. These actions are used to construct plans, represented as trees with the goal state as the root. Plan execution is reactive, and the course of action can be changed based on the conditions in the environment. Ryan and Pendrith (1998) proposed a subsumption architecture in which low-level behaviors are learned in parallel, using reinforcement learning, and a teleo-operator approach is used for higher-level planning.

Some of the recent research even takes into account the assumption of a stochastic, changing environment, in which the plans have to be executed. Probabilistic and statistical methods are used to deal with such environments. For instance, Brafman and Moshe (1997) advocate the use of "mental models", a form of behavior modeling that includes beliefs, value functions and different decision criteria to improve over future behavior. Oates and Cohen (1996) use a statistical methods to detect correlations between variations appearing in multiple streams of data; this technique can potentially uncover the effects of actions over different time periods. Rosenstein and Cohen (1998) use dynamic maps as models for behaviors, both for prediction and for recognition purposes.

A line of research related to AI planning is qualitative reasoning about physical systems (see e.g., De Kleer and Seely Brown, 1984 ; Kuipers, 1979; DeJong, 1994; Say & Kuru, 1996) in which qualitative descriptions, based on discrete variables and events, replace differential equations. In this case, both spatial and temporal abstraction are used to ensure the desired degree of generality. These qualitative descriptions are designed a priori in order to capture the relevant dynamics of the system.

2.2 Temporal Abstraction in Control

Multiple time scales arise naturally in many physical systems, ranging from ensembles of mechanisms to fluid flows and plasma evolution. Therefore, the modeling and control of such systems have been addressed both by mathematicians and by engineers (see, e.g., Brackbill & Cohen, 1985). Multiple scale systems are often characterized by a fast motion superimposed over a slow motion. If the two motions do not influence each other, then the fast motion can be modeled and then eliminated to analyze the slow motion.

Multigrid methods for solving partial differential equations (see e.g. McCromick, 1989) address the issue of numerically solving partial differential equations for physical systems with large variations in scale. These methods define the resolution of the discretization adaptively, for different regions, depending on the speed of the variations.

In the control literature, the problem of controlling a system at multiple time scales has been addressed by singular perturbation methods (Kokotovic, Khalil & O'Reilly, 1986; Naidu & Rao, 1985 ; Naidu, 1988). These methods assume that the physical system to be controlled has state variables that have fast and slow variations respectively. Initially, the slow variations are ignored, and they are only taken care of after the fast variations have been accounted for. Each type of variation is modeled separately. This leads to a form of hierarchical control.

The control of hybrid systems represents another research area where temporally extended actions and models have been used extensively. Hybrid systems (see, e.g. Grossman et. al, 1993; Brockett, 1993; Godbole, Lygeros and Sastry, 1995) contain both digital and analog devices that interact with each other. The interaction takes place through some interface, which is able to translate the continuous state of the analog devices into "events" that can be handled by the digital devices. Most of the research in this area is looking at ways of automatically representing the events that are important for the system and the separate regimes in which it should be controlled. However, most of these methods are restricted in the sense that they assume extensive knowledge about the physical system to be controlled, such as the variables that are relevant for certain functioning regimes, and the differential equations underlying the system. This assumption is often not true for the kinds of tasks tackled by artificial intelligence. In particular, many reinforcement learning methods are designed to solve problems for which no models are available.

2.3 Reinforcement Learning (MDP) Framework

In this section we briefly review the conventional reinforcement learning framework of discrete-time, finite *Markov decision processes*, or *MDPs*, which forms the basis for our extensions to temporally extended courses of action. In this framework, a learning *agent* interacts with an *environment* at some discrete, lowest-level time scale $t = 0, 1, 2, \ldots$. On each time step the agent perceives the state of the environment, $s_t \in \mathcal{S}$, and on that basis chooses a primitive action, $a_t \in \mathcal{A}_{s_t}$. In response to each action, a_t, the environment produces one time step later a numerical reward, r_{t+1}, and a next state, s_{t+1}. It is notationally convenient to suppress the differences in available actions across states whenever possible; we let $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$ denote the union of the action sets. If $\mathcal{S}$ and $\mathcal{A}$, are finite, then the environment's transition dynamics are modeled by one-step state-transition probabilities,

$$p^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\},$$

and one-step expected rewards,

$$r^a_s = E\{r_{t+1} \mid s_t = s, a_t = a\},$$

for all $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}$ (it is understood here that $p^a_{ss'} = 0$ for $a \notin \mathcal{A}_s$). These two sets of quantities together constitute the *one-step model* of the environment.

The agent's objective is to learn an *optimal Markov policy*, a mapping from states to probabilities of taking each available primitive action, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, that maximizes the expected discounted future reward from each state s:

$$\begin{aligned} V^\pi(s) &= E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \;\middle|\; s_t = s, \pi\right\} && (2.1) \\ &= E\left\{r_{t+1} + \gamma V^\pi(s_{t+1}) \;\middle|\; s_t = s, \pi\right\} \\ &= \sum_{a \in \mathcal{A}_s} \pi(s,a)\left[r^a_s + \gamma \sum_{s'} p^a_{ss'} V^\pi(s')\right], && (2.2) \end{aligned}$$

where $\pi(s, a)$ is the probability with which the policy π chooses action $a \in \mathcal{A}_s$ in state s, and $\gamma \in [0, 1]$ is a *discount-rate* parameter. This quantity, $V^\pi(s)$, is called the *value* of state s under policy π, and V^π is called the *state-value function* for π. The *optimal* state-value function gives the value of a state under an optimal policy:

$$\begin{aligned} V^*(s) &= \max_\pi V^\pi(s) && (2.3) \\ &= \max_{a \in \mathcal{A}_s} E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \;\middle|\; s_t = s, a_t = a\right\} \end{aligned}$$

$$= \max_{a \in \mathcal{A}_s} \left[r_s^a + \gamma \sum_{s'} p_{ss'}^a V^*(s') \right]. \tag{2.4}$$

Any policy that achieves the maximum in (2.3) is by definition an optimal policy. Thus, given V^*, an optimal policy is easily formed by choosing in each state s any action that achieves the maximum in (2.4). Planning in reinforcement learning refers to the use of models of the environment to compute value functions and thereby to optimize or improve policies. Particularly useful in this regard are Bellman equations, such as (2.2) and (2.4), which recursively relate value functions to themselves. If we treat the values, $V^\pi(s)$ or $V^*(s)$, as unknowns, then a set of Bellman equations, for all $s \in \mathcal{S}$, forms a system of equations whose unique solution is in fact V^π or V^* as given by (2.1) or (2.3). This fact is key to the way in which all temporal-difference and dynamic programming methods estimate value functions.

For learning methods, of particular importance is a parallel set of value functions and Bellman equations for state–action pairs rather than for states. The value of taking action a in state s under policy π, denoted $Q^\pi(s,a)$, is the expected discounted future reward starting in s, taking a, and henceforth following π:

$$\begin{aligned} Q^\pi(s,a) &= E\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \;\middle|\; s_t = s, a_t = a, \pi \right\} \\ &= r_s^a + \gamma \sum_{s'} p_{ss'}^a V^\pi(s') \\ &= r_s^a + \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s,a') Q^\pi(s',a'). \end{aligned}$$

This is known as the *action-value function* for policy π. The *optimal* action-value function is

$$\begin{aligned} Q^*(s,a) &= \max_\pi Q^\pi(s,a) \\ &= r_s^a + \gamma \sum_{s'} p_{ss'}^a \max_{a'} Q^*(s',a'). \end{aligned}$$

Finally, many tasks are episodic in nature, involving repeated trials, or *episodes*, each ending with a reset to a standard state or state distribution. In these *episodic tasks*, we include a single special *terminal state*, arrival in which terminates the current episode. The set of regular states plus the terminal state (if there is one) is denoted $\mathcal{S}^+$. Thus, the s' in $p^a_{ss'}$ in general ranges over the set $\mathcal{S}^+$ rather than just $\mathcal{S}$ as stated earlier. In an episodic task, values are defined by the expected cumulative reward up until termination rather than over the infinite future (or, equivalently, we can consider the terminal state to transition to itself forever with a reward of zero).

2.4 Prior Work on Temporal Abstraction in Reinforcement Learning

MDPs have emerged as useful models for stochastic planning and control problems. The ability to reason at the level of temporally abstract actions is key to speeding up the learning and planning techniques for MDPs.

Singh (1992a; 1992b) introduced hierarchies of "abstract actions", which achieve different tasks, as well as a hierarchy of models, with variable temporal resolution. This research is one of the sources of inspiration for the work presented in this dissertation. Singh uses a special purpose gating architecture to switch between abstract actions, and specialized learning algorithms for this architecture. We provide a very general framework, which allows abstract actions to be treated similarly to primitive actions for the purposes of learning and planning in an MDP.

Another source of inspiration for this research is the work of Kaelbling (1993b; 1993a), who proposed the idea of using subgoals both in order to learn sub-policies, and in order to collapse the state space. The initial approach was to learn paths to all the possible goals in an environment. Moore, Baird & Kaelbling (1998) extend this work by describing an efficient way to determine useful subgoals. The idea of learning paths to many subgoals in parallel from the same real experience is also used in the intra-option algorithms described in chapter 5, but our algorithm is more general, and we prove convergence properties for it.

The starting point of this research is the work on modeling MDPs at multiple temporal scales (Sutton & Pinette, 1985; Sutton, 1995). Sutton (1995) introduced multi-time models for modeling a policy at several different time scales. But this work was limited to learning models for just one policy, while behaving according to that policy. In this dissertation we develop algorithms that learn models for many different options, even if the data comes from an unrelated behavior. We also deal with the complete control problem, in which we do not only evaluate models for options, but we also use them in making decisions about which options should be executed.

Our work is also related to the recent work of Parr (1998; Parr & Russell, 1998) on hierarchical abstract machines and Dietterich (1998) on MAXQ learning. These are two frameworks for learning about temporally extended actions that are very related to options. The common idea of all these frameworks is that a temporally extended action can be defined as a way of behaving, together with a termination condition. The frameworks differ in the details of how a way of behaving is represented. We take this idea very literally, and we represent an extended action (option) as a policy, together with a (stochastic) termination condition. This is a very simple and general formulation, which allows us to develop algorithms for improving options, and for learning about many options in parallel. Parr uses finite-state machines to provide constraints on the internal policy of an extended action. Dietterich uses value functions to represent the internal policies of extended actions. Each value function depends only on a subset of the variables used to describe the state of the MDP, and all value functions are learned at the same time. In chapter 3 we show how options can be used to represent extended actions expressed as HAMs or MAXQ hierarchies.

All of these recent frameworks recognize that temporally extended actions together with an MDP define a higher-level *Semi-Markov Decision Process (SMDP)*. SMDPs have been used as a standard formalism for the study and control of discrete-event dynamical systems (Cassandras, 1993; Puterman, 1994). The relationship between our work and SMDPs is explored in detail in chapter 4.

A related line of research connects temporal abstraction and state abstraction, by constructing abstract representations along both dimensions at the same time. Feudal RL (Dayan & Hinton, 1993) is a strictly hierarchical technique, which recursively partitions the state space and the time scale from one level to the next. Several approaches use temporal information to determine state representations that facilitate learning. The successor representation (Dayan, 1993) represents the state of a reinforcement learning system by the states that it can reach. McCallum (1995) uses utile suffix memory to represent the state space in environments with hidden information. Wiering and Schmidhuber (1998) introduced HQ-learning, a method for temporal and state abstraction in Partially Observable MDPs (POMDPs).

Several other learning algorithms have been developed for special classes of MDPs and value function representations. For instance, Wixson (1991) uses a variation of Q-learning with a modular agent architecture, which allows switching among actions that achieve different subtasks. The approach is illustrated in an active vision task.

Ring (1994) uses two different methods for constructing "behavior hierarchies". The first method constructs behaviors as units with randomly chosen weights in a neural network. The second approach, the temporal transition hierarchy, constructs a neural network representing the behaviors at the lowest level, and the policy choosing among them at the higher level.

Karlsson (1997) uses policies operating in subsets of the state space for solving subtasks in a reinforcement learning system, but these policies are not executed serially. Instead, choice of primitive action to take is base on weighting the recommendations of several policies.

Meuleau et al. (1998) introduce a form of MDP decomposition in which loosely coupled MDPs are first decomposed into several value functions, corresponding to different tasks, and then the decision is made based on the state in each separate MDP. Hauskrecht et. al (1998) demonstrate a method through which an MDP can be modified, by reducing the number of states, using the framework of options.

Recent research is also targeted towards finding temporally extended actions automatically. Drummond (1998) proposes a system that combines RL and case-based reasoning, by re-using previously learned pieces to compute the value function for new tasks. The most interesting aspect of this work is the method for determining the pieces of the value function that can be transferred to new tasks. A computer vision technique called a "snake" is used to detect big jumps in the value function. These jumps are used to define the boundaries between different pieces.

Thrun and Schwartz (1995) describe a method that attempts to learn temporally extended actions that are re-usable when the goal changes. The method constructs simultaneously the policy that chooses among the extended actions, the policy inside each of these actions and the region of the state space in which each action is applicable. McGovern (1998), Digney (1996) and Andre (1998) also propose methods for automatically acquiring macro-actions.

Hierarchical approaches to RL have been successfully integrated with behavior-based robotics (Brooks, 1986) in several large scale applications (see e.g. Mataric, 1997; Kalmar, Szepeszvari & Lorincz, 1997; Stone & Veloso, 1998; Asada et. al, 1996;; Uchibe et.al, 1996; Digney, 1996). Mahadevan and Connell (1992) demonstrated the success of a subsumption-style architecture in which simple behaviors are acquired using RL and then are combined (by a pre-wired coordination scheme) to solve a complex task. Lin (1993) also used the decomposition of a complex task into smaller subtasks, each having its own limited subdomain in the state space and its own reward function. A robot can learn a behavior for solving each subtask, and then use RL at the higher level as well in order to determine the best combination of sub-behaviors.

Dorigo and Colombetti (1994) used a learning classifier system, implemented using a genetic algorithm, and RL to determine the internal policies of behaviors that achieve subtasks. Switching among subtasks is determined by the internal (control) state of the agent.

Huber & Grupen (1997) use RL and a hybrid discrete event dynamical system to learn walking gaits for a robot. At the low level, the robot uses a set of pre-existing controllers that can generate collision-free motion and optimize forces and posture. At the higher level, reinforcement learning is used to determine which of the controllers should be applied, depending on a set of discrete variables describing the state of the system. A similar approach is used by Coelho et.al (1998) to learn how to control an agent in a non-Markovian environment.

All these robotic applications use controllers in order to learn efficiently in very complex domains. These controllers can be viewed as temporally extended actions. In this dissertation we develop the theory for learning and planning with such controllers.

CHAPTER 3

OPTIONS

In this chapter we introduce options, the main element of our framework for temporal abstraction. The term "options" denotes our generalization of primitive actions to include temporally extended courses of action. An option is a way of behaving – a closed-loop policy for taking action. Options are initiated, make decisions regarding which actions to take for some period of time, and then terminate. When an option terminates, the agent selects another option to be executed. Examples of options include complex courses of action, such as picking up an object or traveling to a distant city, as well as primitive actions, such as joint torques and muscle activations.

In this chapter we formalize these intuitive ideas, by defining what an option is and showing how an option can be specified. We focus on two kinds of representations for options: a flat representation, in which an option chooses among actions, and a hierarchical representation, in which an option chooses among other options.

The representation of an option tells the agent how the option is executed, but it has no information about how good an option is. An agent needs information about the long-term reward it can expect from each option in order to decide which options to choose. Therefore, we define value functions for options and policies over options, analogous to the usual value functions for actions and policies over actions. In the following chapters we will focus on algorithms for efficiently computing such value functions.

3.1 Policies and Options

In reinforcement learning, a way of behaving is represented formally by a policy (as defined in Section 2.3). Usually a policy specifies a distribution over primitive action choices based only on the current state of the environment. Such policies are called *Markov*. In general, policies can be *non-stationary*, which means that they make decisions based on all the states, actions and rewards observed since the beginning of time. In order to handle temporally extended courses of action, we use a special case of non-stationary policies that has two specific characteristics.

First, we allow a policy to specify actions based on all the events since it was initiated. These events are contained in the partial history of the agent. A *partial history* $h_{t,T}$ is the sequence of all states, actions and rewards from time t up to time $T \geq t$:

$$h_{t,T} \stackrel{\text{def}}{=} \langle s_t, a_t, r_{t+1}, s_{t+1}, \ldots s_T \rangle. \tag{3.1}$$

For ease of notation, we denote $h_{t,t} = \langle s_t \rangle$ by s_t. We denote by $\mathcal{H}$ the set of all possible partial histories, and by $\mathcal{H}_t$ the set of all possible partial histories from time t on.

Second, we introduce a *reset action* τ, which represents the decision to terminate a policy. With these extensions, we can define an option as follows:

Definition 1 (Option) *An* option *o is a mapping from partial histories and actions to probabilities of taking each action after each partial history:*

$$o : \mathcal{H} \times (\mathcal{A} \cup \{\tau\}) \rightarrow [0, 1], \tag{3.2}$$

where

$$o(\langle s_{t_0}, \ldots s_t \rangle, a) = Pr\{a_t = a \mid o \textit{is initiated at } t_0, \langle s_{t_0}, \ldots s_t \rangle\}$$

For partial histories containing only one state, the probability of the reset action τ has to be either 1 or 0: $o(s_t, \tau) = 1$ or $o(s_t, \tau) = 0, \forall s_t \in \mathcal{S}$.

Because an option specifies a probability distribution over actions after each partial history, the probabilities of all actions must sum to one:

$$\sum_{a \in \mathcal{A} \cup \{\tau\}} o(h_{t,T}, a) = 1, \, \forall h_{t,T} \in \mathcal{H}$$

In episodic tasks, the termination of an episode also terminates any option that could be executed at that time: $o(h_{t,T}, \tau) = 1$ for all partial histories $h_{t,T}$ for which s_T is a terminal state of the MDP. We will refer to the representation of an option as a probability distribution over actions (3.2) as an *explicit representation* of the option.

For now, we assume that the execution of an option is *call-and-return*. An option o can be initiated in any state $s \in \mathcal{S}$. If o is initiated in s at time t, then the agent picks an action a_t according to the probability distribution $o(s, \cdot)$. If $a_t = \tau$, the option o terminates immediately, on the same time step. If $a_t \in \mathcal{A}$, the agent receives one time step later a reward r_{t+1} and transitions to a new state s_{t+1}. At time $t+1$, the agent picks an action a_{t+1} according to the probability distribution $o(h_{t,t+1}, \cdot)$. If $a_{t+1} = \tau$, the option o terminates at $t+1$. If $a_{t+1} \in \mathcal{A}$, the agent receives one time step later a reward r_{t+2} and transitions to a new state s_{t+2}, and so on. In later chapters we will investigate other models of execution as well.

Primitive actions $a \in \mathcal{A}$ are a special case of options. Each action a corresponds to an option that selects a everywhere a is available, and that always lasts exactly one step:

$$\begin{aligned} o(s,a) &= 1 \text{ and } o(\langle s,a,r,s' \rangle, \tau) = 1, \forall s,r,s' \text{ such that } a \in A_s \text{ and } p^a_{ss'} > 0 \\ o(s,\tau) &= 1, \forall s \text{ such that } a \notin A_s \end{aligned}$$

Because the primitive actions are options, the agent's choice at each decision point is entirely among options, some of which persist for a single time step, others which are temporally extended.

Another important special case of options is Markov options (Sutton et al., 1998a; Sutton et al., 1999a).

Definition 2 (Markov option) *An option is said to be* Markov *if*

$$o(h, a) = o(s, a),$$

for all states $s \in \mathcal{S}$, for all actions $a \in \mathcal{A} \cup \{\tau\}$ and all partial histories $h \in \mathcal{H}$ that end in state s.

Markov options include the usual Markov policies used in MDPs. An option corresponding to a Markov policy never terminates: $o(s, \tau) = 0, \forall s \in \mathcal{S}$.

Timeout options are options that terminate after a fixed number of time steps. These options are not Markov because the decision to take the reset action τ depends on the length of the partial history since the option was initiated, not on the state of the system. Such options are very useful for real-time systems in which execution of a controller has to be aborted after some time has elapsed even if a target state has not been reached.

Given a set of options $\mathcal{O}$, one can define *policies over options* in a way similar to the definition of conventional policies over actions. When initiated in a state, s_t, the Markov policy over options $\mu : \mathcal{S} \times \mathcal{O} \to [0, 1]$ selects an option $o \in \mathcal{O}$ according to probability distribution $\mu(s_t, \cdot)$. The option o is then initiated in s_t, determining actions until it terminates in s_{t+k}, at which point a new option is selected, according to $\mu(s_{t+k}, \cdot)$, and so on. In this way a policy over options, μ, determines a conventional policy over actions, or *flat policy*, $\pi = f(\mu)$. Henceforth the unqualified term *policy* is used for policies over options, which include flat policies as a special case.

Note that even if a policy is Markov and all of the options it selects are Markov, the corresponding flat policy is unlikely to be Markov if any of the options are temporally extended. The action selected by the flat policy in state s_t depends not just on s_t

but on the option being followed at that time, and this depends stochastically on the partial history $h_{t_0,t}$ since the policy was initiated at time t_0.

3.2 Specifying Options

Explicitly specifying the entire probability distribution (3.2) for an option can be cumbersome and confusing, especially in large state and action spaces. In this section we introduce two alternative ways of specifying an option. The first way is by specifying a pre-condition, a way of picking actions, and a post-condition. In this case, the specification is still flat, because the three elements are given in terms of states and actions. The second way is to define a multi-level hierarchy. We show that no special-purpose methods are needed to handle the fully hierarchical case. These two ways of specifying options are potentially more clear and concise than specifying the option explicitly as a probability distribution over partial histories and actions.

3.2.1 Flat Options

A *flat option representation* is a triple $\langle \mathcal{I}, \pi, \beta \rangle$, where:

- $\mathcal{I} \subseteq \mathcal{S}$ is the *initiation set*, containing the states in which the option may be initiated
- $\pi : \mathcal{H} \times \mathcal{A} \to [0, 1]$ is the *internal policy*, which determines the way in which the option picks primitive actions
- $\beta : \mathcal{H} \to [0, 1]$, is the *termination condition*, which gives the probability that the option terminates after each history.

In our previous work (Sutton et al., 1998a; Sutton et al., 1999a) we have used the term *semi-Markov option* to denote options that are represented in this way. If π and β make decisions based only on the current state, the option is a *flat Markov option.*

A flat option $\langle \mathcal{I}, \pi, \beta \rangle$ can be initiated in state s if and only if $s \in \mathcal{I}$. If the option is initiated, then actions are selected according to π until the option terminates stochastically according to β. In episodic tasks, termination of an episode also terminates the current option (i.e., β maps the terminal state to 1 in all options). More precisely, if the option is initiated at time t, then the agent selects action a_t according to the probability distribution $\pi(s_t, \cdot)$. Then the agent receives a reward r_{t+1} and transitions to a new state s_{t+1}. At time $t+1$, the option terminates with probability $\beta(h_{t,t+1})$ or continues, with probability $1-\beta(h_{t,t+1})$. If the option continues, then a new primitive action a_{t+1} is chosen according to the probability distribution $\pi(h_{t,t+1}, \cdot)$, and so on. Note that options specified by a flat representation always take at least one time step before terminating. The general definition of options (3.2) does not impose this restriction.

As an illustration for the flat representation of options, consider an option for a mobile robot to dock with its battery charger. The option might be defined only for states $\mathcal{I}$ in which the battery charger is within sight. A hand-crafted controller π could be initiated in those states and direct the robot during the operation. The termination condition β would be 1 outside of $\mathcal{I}$ and when the robot is successfully docked.

The initiation set and termination condition of an option together restrict its range of application in a potentially useful way. In particular, they limit the range over which the option's policy has to be defined. For Markov options, for instance, it is natural to assume that all states where an option might continue are also states where the option might be started (i.e., that $\{s|\beta(s) < 1\} \subseteq \mathcal{I}$). In this case, π needs to be defined only over $\mathcal{I}$ rather than over all of $\mathcal{S}$.

Limiting the initiation set can also limit the number of options that need to be considered in every state in which the agent has to make a choice. Given a set of options, their initiation sets implicitly define a set of available options $\mathcal{O}_s$ for each state $s \in \mathcal{S}$. Having a small number of options in the set $\mathcal{O}_s, \forall s$, is a way of avoiding the increase in deliberation cost due to the use of options.

The flat representation of options has the same expressive power as the explicit representation, as shown in the following theorem:

Theorem 1 *For any flat representation $\langle \mathcal{I}, \pi, \beta \rangle$, there exists a unique option o whose execution from all states $s \in \mathcal{S}$ produces the same probability distribution over partial histories for any MDP. Conversely, for any option o, there exists a unique flat representation $\langle \mathcal{I}, \pi, \beta \rangle$ whose execution for all states $s \in \mathcal{S}$ produces the same probability distribution over partial histories for any MDP.*

Proof: For the first part, we consider that $\langle \mathcal{I}, \pi, \beta \rangle$ is given and construct a mapping $o : \mathcal{H} \times \mathcal{A} \cup \{\tau\} \to [0, 1]$ that produces equivalent choices of action. Let $h \in \mathcal{H}$ be the current history since option o started. Then we can define o as follows:

$$\begin{aligned} o(s, \tau) &= 1, \forall s \notin \mathcal{I} \\ o(h, \tau) &= \beta(h) \\ o(h, a_j) &= (1 - \beta(h))\pi(h, a_j), \forall a_j \in \mathcal{A} \end{aligned}$$

We show that o and $\langle \mathcal{I}, \pi, \beta \rangle$ produce the same distribution over partial histories by induction over the history length k. For $k = 0$, both representations will stop immediately for states $s \notin \mathcal{I}$ and make the same choices of action by construction for $s \in \mathcal{I}$.

Assume that o and $\langle \mathcal{I}, \pi, \beta \rangle$ produce the same distribution for all partial histories up to length k. Given a partial history h of length k, ending in s_k, by construction o and $\langle \mathcal{I}, \pi, \beta \rangle$ will make the same choices of action. Then the probability of the next partial history is:

$$\Pr\{\langle h, a_j, r_s^{a_j}, s' \rangle \mid h\} = o(h, a_j)p_{ss'}^{a_j} = (1 - \beta(h))\pi(h, a_j)p_{ss'}^{a_j}$$

$$\Pr\{s_k \mid h\} = o(h, \tau) = \beta(h)$$

Conversely, assume that the explicit mapping $o : \mathcal{H} \times \mathcal{A} \cup \{\tau\} \to [0, 1]$ is given. Then the corresponding flat representation can be constructed as follows:

$$\begin{aligned}
\mathcal{I} &= \{s | o(s, \tau) < 1\} \\
\beta(h) &= o(h, \tau), \forall h \in \mathcal{H} \\
\pi(h, a_j) &= \frac{o(h, a_j)}{1 - o(h, \tau)}, \forall h \in \mathcal{H} \text{ such that } o(h, \tau) \neq 1
\end{aligned}$$

Again, in order to prove that o and $\langle \mathcal{I}, \pi, \beta \rangle$ induce the same distribution over partial histories, we use induction over the length of the history k. For $k = 0$, by definition the two representations stop immediately in the same states. Assume that o and $\langle \mathcal{I}, \pi, \beta \rangle$ produce the same distribution for all partial histories up to length k. Given a partial history h of length k, ending in s_k, the probability of the next partial history is:

$$\Pr\{\langle h, a_j, r_s^{a_j}, s' \rangle \mid h\} = (1 - \beta(h))\pi(h, a_j)p_{ss'}^{a_j} = (1 - o(h, \tau))\frac{o(h, a_j)}{1 - o(h, \tau)}p_{ss'}^{a_j} = o(h, a_j)p_{ss'}^{a_j}$$

$$\Pr\{s_k \mid h\} = \beta(h) = o(h, \tau)$$

This proves the induction step and concludes the theorem. $\diamond$

3.2.2 Hierarchical Options

The explicit and flat representations of options provide one level of abstraction on top of the primitive actions. So far, we have considered that an option makes choices among the primitive actions and the reset action. In this section we define hierarchical options, which make their choices among other options. This extension is natural, given the uniform treatment of primitive actions and options in our framework. As we show in this section, no special-purpose methods are needed to handle hierarchies with multiple levels of options.

Let $\mathcal{O}$ be a set of options. A *history over options* $\chi_{t,T}$ is a sequence of states, options and rewards:

$$\chi_{t,T} = \langle s_t o_t r_{t+k} s_{t+k} \ldots s_T o_T \rangle,$$

where o_T is optional and denotes the option chosen at s_T (the most recent choice point) and which is currently executing. Note that, given a partial history $h_{t,T}$, there can be many different histories over options $\chi_{t,T}$ that could have generated $h_{t,T}$. Conversely, a history over options can lead to many corresponding partial histories, if the options or the environment are stochastic. Given a set of options $\mathcal{O}$, we denote by $\mathcal{H}_{\mathcal{O}}$ the set of all possible histories over options.

A *hierarchical option representation* over a set of options $\mathcal{O}$ is a triple $o = \langle \mathcal{I}, \mu, \beta \rangle$ where $\mathcal{I} \subseteq \mathcal{S}$ is an initiation set, $\mu : \mathcal{H}_{\mathcal{O}} \times \mathcal{O} \to [0, 1]$ is an internal policy over options, and $\beta : \mathcal{H}_{\mathcal{O}} \to [0, 1]$ is a termination condition.

For any hierarchical representation of an option, there exists an equivalent explicit representation, i.e., a representation which generates the same distribution over partial histories. We show this formally later in this section.

The definition of hierarchical options mimics exactly the definition of flat options from the previous section. The main difference is that the internal policy chooses among general options instead of only among primitive actions. We can also define *hierarchical Markov options*, for which the policy and the termination condition take into account only the current state, rather than the whole history since the initiation of the option. Later in this chapter we will show that the hierarchical representation of options has the same power as the explicit representation.

The hierarchical option representation is extremely general, but somewhat impractical, because it keeps track of complete history information, which is extremely expensive. For practical purposes, histories are either limited to some small duration (as in the case of Markov options), or they are memorized in some parsimonious way (e.g. through sufficient statistics).

Several other hierarchical representations for temporally extended actions have been proposed in AI and RL. Now we will show how the action representations they propose can be mapped into hierarchical options:

1. *Macro-operators* have been extensively studied in classical AI (see e.g. Fikes, Hart & Nilsson, 1972; Korf, 1985; Laird, Rosenbloom & Newell, 1986; Knoblock, 1990). A macro-operator is an open-loop sequence of other macro-operators. The initiation of a macro operator depends on the state of the system, but the action decisions during the execution of the macro operator only depend on the macro operators already executed. In our framework, a macro-operator can be viewed as a hierarchical option with an open-loop policy for selecting among the available sub-options. The history information contains only the options executed, without any details about the intermediate states or the rewards received.

2. *Hierarchical abstract machines (HAMs)* (Parr & Russell, 1998; Parr, 1998) are hierarchical representations in which the action choice at each level is constrained by a finite state machine (or a "program"). The machines on one level can call programs from a lower level. If a machine completely specifies the choices of action, it can be viewed as an option, for which the internal policy is specified by the corresponding finite state machine. Instead of an explicit complete history, each machine keeps track of several internal state variables, in addition to the environment variables. These variables represent efficiently the part of the history that is relevant for the decision-making process. But HAMs allow a user to specify just constraints over the policies, without specifying the precise action choices. In this case, just the policy that the HAM finds after learning can be represented as an option.

3. *MAXQ* (Dietterich, 1998) is an alternative framework for specifying a hierarchy of options. Each state is represented by a vector of state variables, and each option takes into account only the variables that are relevant for its level. It is the task of the system's designer to specify which variables should be taken into account at each level. Options at each level are Markov and they are acquired using standard reinforcement learning techniques, such as Q-learning.

Hierarchical options make their choices based on a history over options. It is not immediately clear that such options do not need to be treated using special computational methods. We will now show that for any hierarchical option, there is an explicit representation that generates the same distribution over histories. This property enables us to treat hierarchical options in the same way as flat options, without designing special-purpose methods. Executing hierarchical options involves two basic operations: sequencing (composition) of two or more options, and stochastic choice from the given set of options. We now show that for each of these basic operations, there is an explicit representation that generates the same history distribution.

Lemma 1 (Sequencing) *For any two options o_1 and o_2, and for any MDP, there exists an option o whose execution from any state $s \in \mathcal{S}$ produces the same distribution of histories as the execution of o_1 in s, followed by the execution of o_2.*

Proof: Let $h_{t,T} \in \mathcal{H}$ be a history obtained by executing o_1 followed by o_2. If no additional information is available about the option that is executing, then in general, two situations are possible:

1. option o_1 might still be executing, in which case it is still picking primitive actions;

2. option o_1 terminated at some intermediate time step k and o_2 has taken over from there.

Using this observation, option o can be specified as follows:

$$
\begin{aligned}
o(h_{t,T}, a_j) &= o_1(h_{t,T}, a_j) + \sum_{k=t}^{T} o_1(h_{t,k}, \tau) o_2(h_{k,T}, a_j), \forall a_j \in \mathcal{A} \\
o(h_{t,T}, \tau) &= \sum_{k=t}^{T} o_1(h_{t,k}, \tau) o_2(h_{k,T}, \tau)
\end{aligned}
$$

By this construction, the distribution of actions after any given history is the same for o as it is for the execution of o_1 followed by o_2. If two options generate the same action choices after each history, then they will generate the same distribution over histories, by the same induction argument as the one used to prove theorem 1. $\diamond$

Corollary 1 *For any sequence of options $o_1 \dots o_n$ and for any MDP, there exists an option o whose execution from any state $s \in \mathcal{S}$ produces the same distribution of histories as the execution of the given sequence starting in s.*

Proof: The proof is immediate by induction over n. The base case $n = 2$ is proven in lemma 1. For the induction step, consider the set of options $o_1, \dots o_n, o_{n+1}$. From the induction hypothesis, the sequence $o_1 \dots o_n$ can be represented explicitly using an option o'. Then from theorem 1, the sequence of o', o_{n+1} can be represented explicitly by some option o, which proves the corollary. $\diamond$

Lemma 2 (Stochastic choice) *Let $\mathcal{O} = \{o_1 \dots o_n\}$ be a finite set of options and let $\mu : \mathcal{S} \times \mathcal{O} \to [0, 1]$ be a Markov policy that chooses from the options in $\mathcal{O}$. Then there exists an option o such that, for any MDP, the execution of o starting from any state $s \in \mathcal{S}$ generates the same distribution over histories as a single choice of option performed in s according to distribution $\mu(s, \cdot)$.*

Proof: Let $h \in \mathcal{H}$ be the history over primitive actions since μ was initiated. Then o should take into account the likelihood of each of the possible options being active, given the observed history, and average the suggested choices of action from each option. Formally, o can be defined as follows:

$$\begin{aligned} o(h,\tau) &= \sum_{o_i \in \mathcal{O}} \Pr\{o_i \mid h\} o_i(h,\tau) \\ o(h,a_j) &= \sum_{o_i \in \mathcal{O}} \Pr\{a_j \mid o_i, h\} = \sum_{o_i \in \mathcal{O}} \Pr\{o_i \mid h\} o_i(h,a_j) \end{aligned}$$

Because h has been observed, using Bayes rule, we have:

$$\Pr\{o_i \mid h\} = \mu(s,o_i)\Pr\{h \mid o_i\},$$

where s is the first state of h. The second factor can be computed immediately from o_i and from the dynamics of the environment.

By this construction, o makes the same action choices for any history as applying μ for one step. By induction on the length of the history, analogous to theorem 1, o and μ will generate the same distribution over histories. $\diamond$

Based on these results, we can show that any hierarchical option has an explicit representation:

Theorem 2 *Let $\langle \mathcal{I}, \mu, \beta \rangle$ be a hierarchical option that chooses from the set $\mathcal{O} = \{o_1 \ldots o_n\}$. Then there exists an explicit representation o such that, for any MDP, the execution of o from any state $s \in \mathcal{S}$ produces the same distribution of histories as the execution of $\langle \mathcal{I}, \pi, \beta \rangle$ starting in s..*

Proof: The proof is based on the results and proof techniques used in lemmas 1 and 2. Consider a history $h_{t,T}$. We have to determine all the possible histories over options that could have generated this real history in the environment. Such histories can have from one option to at most $T - t$ options in them, and we have to consider all possible breakdown points. So the choice of o for each history can be written as a sum of probabilities, taking into account all these possible breakdown points:

$$
\begin{aligned}
o(h_{t,T},\tau) &= \sum_{k=0}^{T-t} P_k(h_{t,T},\tau)\\
o(h_{t,T},a_j) &= \sum_{k=0}^{T-t} P_k(h_{t,T},a_j),
\end{aligned}
$$

where $P_k(h_{t,T},\tau)$ and $P_k(h_{t,T},a_j)$ denote the probabilities of taking τ and a_j respectively after $h_{t,T}$, assuming that k reset points have occurred between t and T.

Let us consider first the situation in which no breakpoint occurred yet (the first option picked by o is still executing). This is actually the stochastic choice case presented in lemma 2, so P_0 can be expressed as follows:

$$
\begin{aligned}
P_0(h_{t,T},a) &= \sum_{o_i\in\mathcal{O}} \Pr\{o_i \mid h_{t,T},\mu\} o_i(h_{t,T},a), \forall a\in\mathcal{A}\\
P_0(h_{t,T},\tau) &= \sum_{o_i\in\mathcal{O}} \Pr\{o_i \mid h_{t,T},\mu\} o_i(h_{t,T},\tau)\beta(\langle s_t o_i r_{t,T} s_T\rangle),
\end{aligned}
$$

where $r_{t,T}$ is the total discounted reward observed during the period from t to T: $r_{t,T} = r_t + \ldots + \gamma^{T-t} r_T$. Based on the proof of theorem 1,

$$
\Pr\{o_i \mid h_{t,T},\mu\} = \mu(s_t,o_i)\Pr\{h_{t,T} \mid o_i\}.
$$

Now consider the case in which one termination occurred between t and T. In this case, the probabilities P_1 can be determined analogously to lemma 1:

$$
\begin{aligned}
P_1(h_{t,T},a) &= \sum_{k=t+1}^{T} \sum_{o_i,o_j\in\mathcal{O}} \Pr\{\langle s_t o_i r_{t,k} s_k o_j\rangle \mid h_{t,T},\mu\} o_j(h_{k,T},a)\\
P_1(h_{t,T},\tau) &= \sum_{k=t+1}^{T} \sum_{o_i,o_j\in\mathcal{O}} \Pr\{\langle s_t o_i r_{t,k} s_k o_j\rangle \mid h_{t,T},\mu\} o_j(h_{k,T},\tau)\beta(\langle s_t o_i r_{t,k} s_k o_j r_{k,T} s_T\rangle)
\end{aligned}
$$

The conditional probability of the history over options can be decomposed as follows:

$$\begin{aligned}
\Pr\{\langle s_t o_i r_{t,k} s_k o_j\rangle \mid h_{t,T}, \mu\} &= \Pr\{o_i \mid h_{t,k}, \mu\} o_i(h_{t,k}, \tau) \Pr\{o_j \mid \langle s_t o_i r_{t,k} s_k\rangle, h_{k,T}, \mu\} \\
&= \Pr\{o_i \mid h_{t,k}, \mu\} o_i(h_{t,k}, \tau) \mu(\langle s_t o_i r_{t,k} s_k\rangle, o_j) \Pr\{h_{k,T} \mid o_j\}
\end{aligned}$$

The same unrolling technique can be use for computing any term P_k.

By this construction, and by lemmas 1 and 2, o produces the same distribution of histories as $\langle \mathcal{I}\mu\beta\rangle$. $\diamond$

Corollary 2 *Any hierarchical option o choosing from a given set of hierarchical options $\mathcal{O} = \{o_1 \ldots o_n\}$ can be represented explicitly by a mapping $o : \mathcal{H} \times \mathcal{A} \cup \{\tau\} \to [0, 1]$.*

Proof: Based on theorem 2, we can successively "flatten" each level of hierarchical options, providing the respective explicit representation. The corollary is immediate by induction on the number of levels in the hierarchy. $\diamond$

This theoretical result shows that hierarchies of arbitrary depth can be flattened, and therefore treated in the same way as flat options. Therefore, from now on we will use the general term "options" regardless whether the representation is flat or hierarchical. We will present our theoretical results for the explicit representation of options, and indicate any specializations for flat or hierarchical representations.

3.3 Value Functions for Options

So far we have described how options can be specified and executed. But knowing how an option is executed is not enough for an agent that is trying to make decisions about which options it should choose. In order to make decisions, the agent needs long-term predictions about the consequences of behaving according to different policies over options. Now we generalize the usual state value and action value functions to apply to options and policies over options.

Let $\mathcal{E}(\pi, s, t)$ denote the event of a flat policy π being initiated in s at time t. It is also useful to define $\mathcal{E}(o, h, t)$, the event of an option o *continuing* from history h at time t, where h is a history ending in s_t. In continuing, actions are selected as if the history had preceded s_t. That is, a_t is selected according to $o(h, \cdot)$. If the action chosen is not the reset action, then the agent receives a reward r_{t+1} and transitions to a state s_{t+1}. On the next time step, the action choice will be made according to $o(\langle h a_t r_{t+1} s_{t+1} \rangle, \cdot)$.

Definition 3 *The* value of a state $s \in \mathcal{S}$ under a flat semi-Markov policy π *is the expected return if the policy is initiated in* s*:*

$$V^{\pi}(s) \stackrel{\text{def}}{=} E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \,\middle|\, \mathcal{E}(\pi, s, t)\right\}. \tag{3.3}$$

Definition 4 *The* value of a state under a policy over options μ *is the value of the state under the corresponding flat policy:*

$$V^{\mu}(s) \stackrel{\text{def}}{=} V^{f(\mu)}(s), \forall s \in \mathcal{S}. \tag{3.4}$$

It is natural to generalize action-value functions to *option*-value functions. Given an option o and a policy over options μ, let $o\mu$ denote the policy that first follows o until it terminates and then starts choosing according to μ in the resulting state.

Definition 5 *The* value of taking option o in state $s \in \mathcal{I}$ under policy μ *is:*

$$Q^{\mu}(s, o) \stackrel{\text{def}}{=} E\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \,\middle|\, \mathcal{E}(o\mu, s, t)\right\} \tag{3.5}$$

Finally, there are generalizations of *optimal* value functions to options and to policies over options. Of course the conventional optimal value functions V^* and Q^* are not affected by the introduction of options; one can ultimately do just as well with primitive actions as one can with options. Nevertheless, it is interesting to know how well one can do with a restricted set of options that does not include all the actions. For example, in planning one might first consider only high-level options in order to find an approximate plan quickly. Let $\mathcal{O}$ denote a restricted set of options and let $\Pi(\mathcal{O})$ be the set of all policies selecting only from options in $\mathcal{O}$.

Definition 6 *Given a set of options $\mathcal{O}$, the* optimal value function for $\mathcal{O}$ *is:*

$$V^*_{\mathcal{O}}(s) \stackrel{\text{def}}{=} \max_{\mu \in \Pi(\mathcal{O})} V^{\mu}(s) \tag{3.6}$$

Definition 7 *Given a set of options $\mathcal{O}$, the* optimal option-value function for $\mathcal{O}$ *is:*

$$Q^*_{\mathcal{O}}(s, o) \stackrel{\text{def}}{=} \max_{\mu \in \Pi(\mathcal{O})} Q^{\mu}(s, o) \tag{3.7}$$

Definition 8 *Given a set of options, $\mathcal{O}$, a corresponding* optimal policy*, denoted $\mu^*_{\mathcal{O}}$, is any policy that achieves $V^*_{\mathcal{O}}$, i.e., for which $V^{\mu^*_{\mathcal{O}}}(s) = V^*_{\mathcal{O}}(s)$ in all states $s \in \mathcal{S}$.*

3.4 Conclusions

In this chapter we defined several important components of our framework for temporal abstraction in reinforcement learning. We defined options, which are essentially closed-loop, stochastic ways of behaving that terminate. Options can be temporally extended, and their choices of action can depend on the whole history of events since the option was initiated.

We defined three ways of specifying options. The explicit representation and the flat representation describe the options in terms of their choices of primitive actions.

In the hierarchical representation, options choose from a set of other options. The hierarchical representation of options subsumes other representations for temporally extended actions that have been introduced in AI and RL, such as macro-operators, hierarchies of abstract machines and MAXQ hierarchies. We showed that any hierarchy of options can be mapped into a flat option. This result enables us to treat both representations in a uniform way, both for theoretical and for practical purposes.

Finally, we defined value functions for options and policies over options. These value functions are analogous to the state and action value functions used in RL. Especially important are the optimal value functions for a set of options; knowing these value functions allows an agent to behave optimally while choosing from the available options. In the next two chapters, we focus on planning and learning algorithms that allow an agent to compute efficiently such optimal value functions.

CHAPTER 4

SMDP METHODS FOR LEARNING AND PLANNING WITH OPTIONS

The options framework presented in the previous chapter allows us to specify temporally extended courses of action that an agent can execute. But knowing how an option is executed is not enough in order to reason about it, or to use it effectively. We need information about the consequences of options, and we need algorithms that allow us to plan ways of behaving using options.

In this chapter we address the issue of learning and planning with options. In particular, we present a class of learning and planning methods called SMDP methods. These methods are adapted from the theory of Semi-Markov Decision Processes (SMDPs), which are used to model continuous-time, discrete-event systems. Our key observation is that by introducing options in an MDP we obtain an SMDP. Then we can define models of options, analogous to the models of SMDP actions, and we can use all the learning and planning methods available in SMDPs. In this chapter we illustrate how using SMDP methods and options results in faster learning and planning compared to the usual MDP methods.

4.1 SMDPs and Options

How can we compute optimal value functions and optimal policies for different sets of options? In order to answer this question, note that options are closely related to the actions in a special kind of decision problem known as a *semi-Markov decision process*, or *SMDP* (e.g., see Puterman, 1994). SMDPs are related to MDPs, but more appropriate for modeling discrete-event systems. Formally, an SMDP is a tuple

$\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{Q} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{R}$ is the reward function, and $\mathcal{Q}$ is the joint distribution of the next state and the transit time. More specifically, if the system is in state s and chooses action a at the current decision epoch, then $\mathcal{Q}(t, s'|s, a)$ denotes the probability that the next decision epoch occurs within time t, and that the system will be in state s' at that time.

An SMDP is usually viewed as a decision process overlaid on top of a *natural process*. The underlying system can change state between decision epochs, but these changes do not provide any relevant information to the agent. From this perspective, *any MDP together with a fixed set of options is an SMDP*. The MDP is the natural process, and the options are the actions in the SMDP process. Each decision epoch occurs at the moment of a reset action, and at that time, the system can choose a new option to execute. The reward ($\mathcal{R}$) and transition ($\mathcal{Q}$) distributions are well defined for each state and option by the underlying MDP and by the option itself. These distributions are well defined because MDPs are Markov and every option makes its choices based only on events that occurred since the option was initiated. Therefore, the next state, reward and transition time depend only on the option and the state in which it was initiated. The transit times of options are always discrete, but this is simply a special case of the arbitrary real intervals permitted in SMDPs. This result can be stated more formally as follows:

Theorem 3 (MDP + Options = SMDP) *For any MDP, and any fixed set of options defined on that MDP, the decision process that selects among those options, executing each to termination, is an SMDP.*

This relationship among MDPs, options, and SMDPs provides a basis for the theory of planning and learning methods with options. In particular, all the SMDP planning and learning methods can be applied immediately to the case in which temporally extended options are used in an MDP. The remainder of this chapter discusses SMDP planning and learning methods. Although our formalism is slightly different, these results are in essence taken or adapted from prior work in SMDP theory (see, e.g. Puterman, 1994), and reinforcement learning (Bradtke & Duff, 1995; Parr & Russell, 1998; Singh, 1992a; Singh, 1992b; Sutton, 1995; Precup & Sutton, 1997; Precup et al., 1997; Precup & Sutton, 1998; Precup et al., 1998; McGovern et al., 1997). A result very similar to theorem 3 was proved in detail by Parr (1998).

4.2 SMDP Planning

Planning with options requires a model of their consequences. Fortunately, the appropriate form of model for options is known from existing SMDP theory. For each state in which an option may be initiated, this kind of model predicts the state in which the option will terminate and the total reward received along the way. These quantities are discounted in a particular way. For any option o, let $\mathcal{E}(o,s,t)$ denote the event of o being initiated in state s at time t. Then the reward part of the model of o for any state $s \in \mathcal{S}$ is

$$r_s^o = E\left\{ r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{k-1} r_{t+k} \;\middle|\; \mathcal{E}(o,s,t) \right\}, \tag{4.1}$$

where $t+k$ is the random time at which o terminates. The state-prediction part of the model of o for state s is:

$$p_{ss'}^o = \sum_{k=1}^{\infty} p(s',k)\,\gamma^k, \forall s' \in \mathcal{S}, \tag{4.2}$$

where $p(s',k)$ is the probability that the option terminates in s' after k steps. Thus, $p_{ss'}^o$ is a combination of the likelihood that s' is the state in which o terminates together with a measure of how delayed that outcome is relative to γ. This kind of model is called a *multi-time model* (Precup & Sutton, 1997; Precup & Sutton, 1998) because it describes the outcome of an option not at a single time but at potentially many different times, appropriately combined. The definition of $p_{ss'}^o$ differs slightly from that given previously for primitive actions. Under the new definition, the model of transition from state s to state s' for an action a is not simply the corresponding transition probability, but the transition probability *times* γ.

Multi-time models can be generalized to handle the case in which options continue after a given history. If $\mathcal{E}(h, o, t)$ denotes the event of o continuing after history h which ends in state s_t, then equations (4.1) and (4.2) hold when replacing s with h.

Multi-time models can be used to write Bellman equations for general policies and options. For any Markov policy over options μ, the state-value function can be written as:

$$V^{\mu}(s) = E\left\{r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k}) \;\middle|\; \mathcal{E}(\mu, s, t)\right\},$$

where k is the duration of the first option selected by μ. By using the model definition, this equation can be rewritten as:

$$V^{\mu}(s) = \sum_{o \in \mathcal{O}_s} \mu(s, o) \left[r_s^o + \sum_{s'} p_{ss'}^o V^{\mu}(s') \right], \tag{4.3}$$

which is the *Bellman equation for the state-value function*. The corresponding Bellman equation for the value of an option o in state $s \in \mathcal{I}$ is:

$$\begin{aligned} Q^{\mu}(s, o) &= E\left\{r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^{\mu}(s_{t+k}) \;\middle|\; \mathcal{E}(o, s, t)\right\} \\ &= E\left\{r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k \sum_{o' \in \mathcal{O}_s} \mu(s_{t+k}, o') Q^{\mu}(s_{t+k}, o') \;\middle|\; \mathcal{E}(o, s, t)\right\} \\ &= r_s^o + \sum_{s'} p_{ss'}^o \sum_{o' \in \mathcal{O}_s} \mu(s', o') Q^{\mu}(s', o'). \end{aligned} \tag{4.4}$$

Note that all these equations specialize to the usual Bellman equations used for solving MDPs, in the case in which μ is a conventional policy and o is a conventional primitive action. Also note that $Q^{\mu}(s, o) = V^{o\mu}(s)$.

Similarly, the *optimal Bellman equations* can be written for general policies and options. For instance, the optimal value function given that options are chosen only from the set $\mathcal{O}$can be expressed as:

$$\begin{aligned} V^*_{\mathcal{O}}(s) &= \max_{o \in \mathcal{O}_s} E\left\{ r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^*_{\mathcal{O}}(s_{t+k}) \;\middle|\; \mathcal{E}(o, s, t) \right\}, && (4.5) \\ &= \max_{o \in \mathcal{O}_s} \left[r^o_s + \sum_{s'} p^o_{ss'} V^*_{\mathcal{O}}(s') \right]. && (4.6) \end{aligned}$$

The Bellman equations for the optimal option-value function are:

$$\begin{aligned} Q^*_{\mathcal{O}}(s, o) &= E\left\{ r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k V^*_{\mathcal{O}}(s_{t+k}) \;\middle|\; \mathcal{E}(o, s, t) \right\}, \\ &= E\left\{ r_{t+1} + \cdots + \gamma^{k-1} r_{t+k} + \gamma^k \max_{o' \in \mathcal{O}_{s_{t+k}}} Q^*_{\mathcal{O}}(s_{t+k}, o') \;\middle|\; \mathcal{E}(o, s, t) \right\}, \\ &= r^o_s + \sum_{s'} p^o_{ss'} \max_{o' \in \mathcal{O}_{s'}} Q^*_{\mathcal{O}}(s', o'). && (4.7) \end{aligned}$$

If $V^*_{\mathcal{O}}$ and models of the options are known, then optimal policies can be formed by choosing in any proportion among the maximizing options in (4.5) or (4.6). Or, if $Q^*_{\mathcal{O}}$ is known, then optimal policies can be found without a model by choosing in each state s in any proportion among the options o for which $Q^*_{\mathcal{O}}(s, o) = \max_{o'} Q^*_{\mathcal{O}}(s, o')$. In this way, computing approximations to $V^*_{\mathcal{O}}$ or $Q^*_{\mathcal{O}}$ become key goals of planning and learning methods with options.

Each of the Bellman equations for options, (4.3), (4.4), (4.5), and (4.7), defines a system of equations whose unique solution is the corresponding value function. These Bellman equations can be used as update rules in dynamic-programming-like planning methods for finding the value functions. Typically, solution methods for this problem maintain an approximation of $V^*_{\mathcal{O}}(s)$ or $Q^*_{\mathcal{O}}(s, o)$ for all states $s \in \mathcal{S}$ and all options $o \in \mathcal{O}_s$. For example, *synchronous value iteration* (SVI) with options starts with an arbitrary approximation V_0 to $V^*_{\mathcal{O}}$ and then computes a sequence of new approximations $\{V_k\}$ by

$$V_k(s) = \max_{o \in \mathcal{O}_s} \left[r_s^o + \sum_{s' \in \mathcal{S}} p_{ss'}^o V_{k-1}(s') \right], \forall s \in \mathcal{S}. \tag{4.8}$$

The action-value form of SVI starts with an arbitrary approximation Q_0 to $Q^*_{\mathcal{O}}$ and then computes a sequence of new approximations $\{Q_k\}$ by:

$$Q_k(s, o) = r_s^o + \sum_{s' \in \mathcal{S}} p_{ss'}^o \max_{o' \in \mathcal{O}_{s'}} Q_{k-1}(s', o'), \forall s \in \mathcal{S}, o \in \mathcal{O}_s. \tag{4.9}$$

Note that these algorithms reduce to the conventional value iteration algorithms in the special case that $\mathcal{O} = \mathcal{A}$. Standard results from SMDP theory guarantee that these processes converge for general options: $\lim_{k \to \infty} V_k = V^*_{\mathcal{O}}$ and $\lim_{k \to \infty} Q_k = Q^*_{\mathcal{O}}$ for all sets of options $\mathcal{O}$.

The plans (policies) found using temporally extended options are approximate in the sense that they achieve only $V^*_{\mathcal{O}}$, which is less than the maximum possible, V^*. Parr (1998) provides some bounds on the sub-optimality of any given set of options. On the other hand, if the models used to find them are correct, then they are guaranteed to achieve $V^*_{\mathcal{O}}$. This is the *value achievement* property of planning with options. This contrasts with planning methods that abstract over state space, which generally cannot be guaranteed to achieve their planned values even if their models are correct (e.g., Dean and Lin, 1995).

4.3 Illustration: Rooms Example

As a simple illustration of planning with options, consider the *rooms example*, a gridworld environment of four rooms shown in Figure 4.1. The cells of the grid correspond to the states of the environment. From any state the agent can perform one of four actions, `up`, `down`, `left` or `right`, which have a stochastic effect. With probability 2/3, the actions cause the agent to move one cell in the corresponding direction, and with probability 1/3, the agent moves instead in one of the other three directions, each with probability 1/9. In either case, if the movement would take the

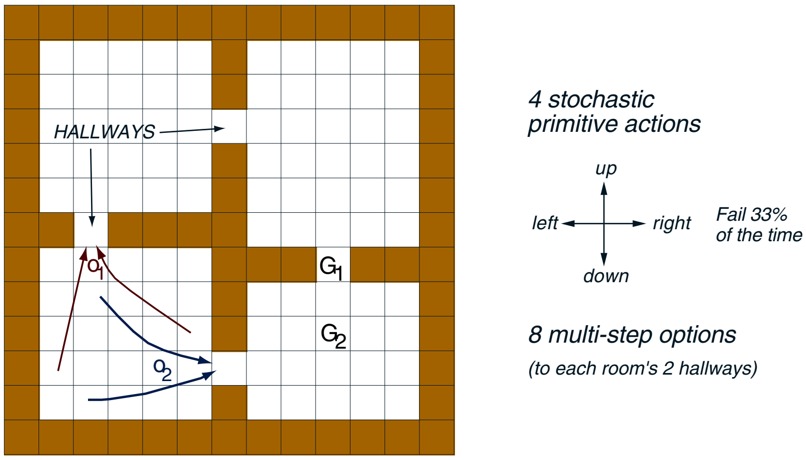

Figure 4.1. The rooms example is a gridworld environment with stochastic cell-to-cell actions and room-to-room hallway options. Two of the hallway options are suggested by the arrows labeled o_1 and o_2. The labels G_1 and G_2 indicate two locations used as goals in experiments described below.

agent into a wall, then the agent remains in the same cell. The rewards are zero on all state transitions.

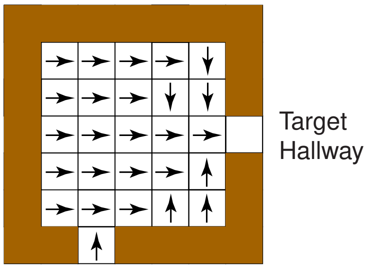

Figure 4.2. The policy underlying one of the eight hallway options.

Two built-in *hallway options* are provided in each of the four rooms. These options are designed to take the agent from anywhere within the room to one of the two hallway cells leading out of the room. A hallway option's policy π follows a shortest path within the room to its target hallway while minimizing the chance of stumbling into the other hallway. For example, the policy for one hallway option is shown in Figure 4.2. The termination condition for each hallway option is zero for states within the room and 1 for states outside the room, including the hallway states. Each option is also restricted to last at least one time step before terminating. The initiation set

$\mathcal{I}$ comprises the states within the room plus the non-target hallway state leading into the room. Note that these options are deterministic and Markov, and that an option's policy is not defined outside of its initiation set. The set of eight hallway options is denoted by $\mathcal{H}$. For each option $o \in \mathcal{H}$, its accurate model r_s^o and $p_{ss'}^o$, is also provided a priori, for all $s \in \mathcal{I}$ and $s' \in \mathcal{S}$ (assuming there is no goal state). Note that although the transition models $p_{ss'}^o$ are nominally large (order $|\mathcal{I}| \times |\mathcal{S}|$), in fact they are sparse, and relatively little memory (order $|\mathcal{I}| \times 2$) is actually needed to hold the nonzero transitions from each state to the two adjacent hallway states. The off-target hallway states are exceptions in that they have three possible outcomes: the target hallway, themselves, and the neighboring state in the off-target room.

Now consider a sequence of planning tasks for navigating within the grid to a designated goal state, in particular, to the hallway state labeled G_1 in Figure 4.1. Formally, the goal state is a state from which all actions lead to the terminal state with a reward of +1. The discount factor is $\gamma = 0.9$.

As a planning method, we used SVI as given by (4.8), with various sets of options $\mathcal{O}$. The initial value function V_0 was 0 everywhere except the goal state, which was initialized to its correct value, $V_0(G_1) = 1$, as shown in the leftmost panels of Figure 4.3. This figure contrasts planning with the original actions ($\mathcal{O} = \mathcal{A}$) and planning with the hallway options and not the original actions ($\mathcal{O} = \mathcal{H}$). The upper part of the figure shows the value function after the first two iterations of SVI using just primitive actions. The region of accurately valued states moved out by one cell on each iteration, but after two iterations most states still had their initial arbitrary value of zero. The lower part of the figure shows the corresponding value functions for SVI with the hallway options. In the first iteration all states in the rooms adjacent to the goal state became accurately valued, and in the second iteration all the states become accurately valued. Although the values continued to change by small amounts over subsequent iterations, a complete and optimal policy was known by this time. Rather

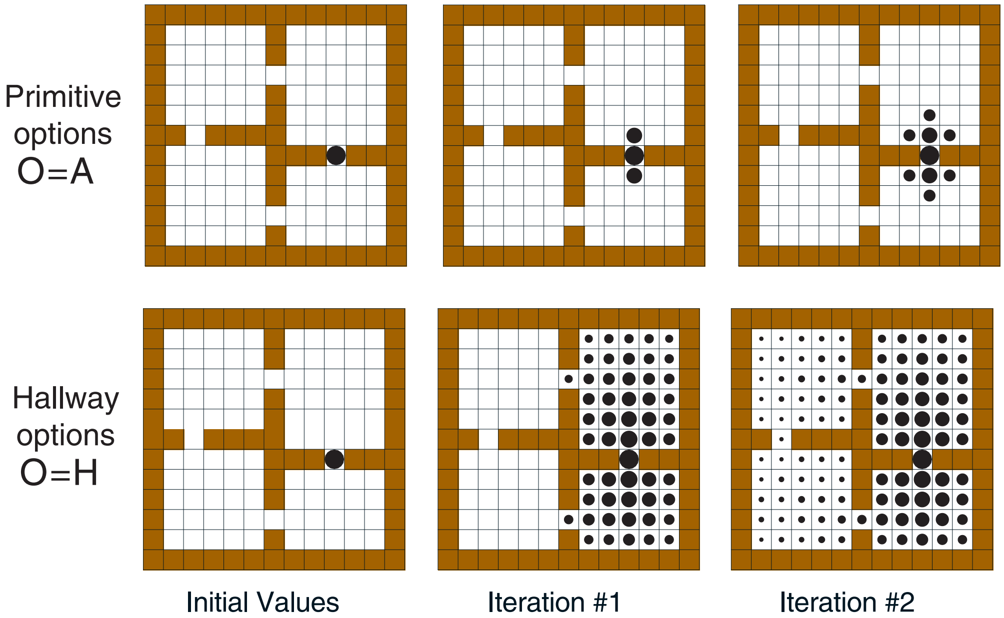

Figure 4.3. Value functions formed over iterations of planning by synchronous value iteration with primitive actions and with hallway options. The hallway options enabled planning to proceed room-by-room rather than cell-by-cell. The area of the disk in each cell is proportional to the estimated value of the state, where a disk that just fills a cell represents a value of 1.0.

than planning step-by-step, the hallway options enabled the planning to proceed at a higher level, room-by-room, and thus be much faster.

This example is a particularly favorable case for the use of multi-step options because the goal state is a hallway, the target state of some of the options. Next, we consider a case in which there is no such coincidence, in which the goal lies in the middle of a room, in the state labeled G_2 in Figure 4.1. The hallway options and their models were just as in the previous experiment. In this case, planning with the hallway options alone could never completely solve the task, because these take the agent only to hallways and thus never to the goal state. Figure 4.4 shows the value functions found over five iterations of SVI using *both* the hallway options and options corresponding to the primitive actions (i.e., using $\mathcal{O} = \mathcal{A} \cup \mathcal{H}$). In the first two iterations, accurate values were propagated from G_2 by one cell per iteration by the models corresponding to the primitive actions. After two iterations, however, the first hallway state was reached, and subsequently room-to-room planning using the temporally extended hallway options dominated. Note how the state in the lower right corner was given a nonzero value during iteration three. This value corresponds to the plan of first going to the hallway state above and then down to the goal; it was overwritten by a larger value corresponding to a more direct route to the goal in the next iteration. Because of the hallway options, a close approximation to the correct value function was found everywhere by the fourth iteration; without these options, only the states within three steps of the goal would have been given non-zero values by this time.

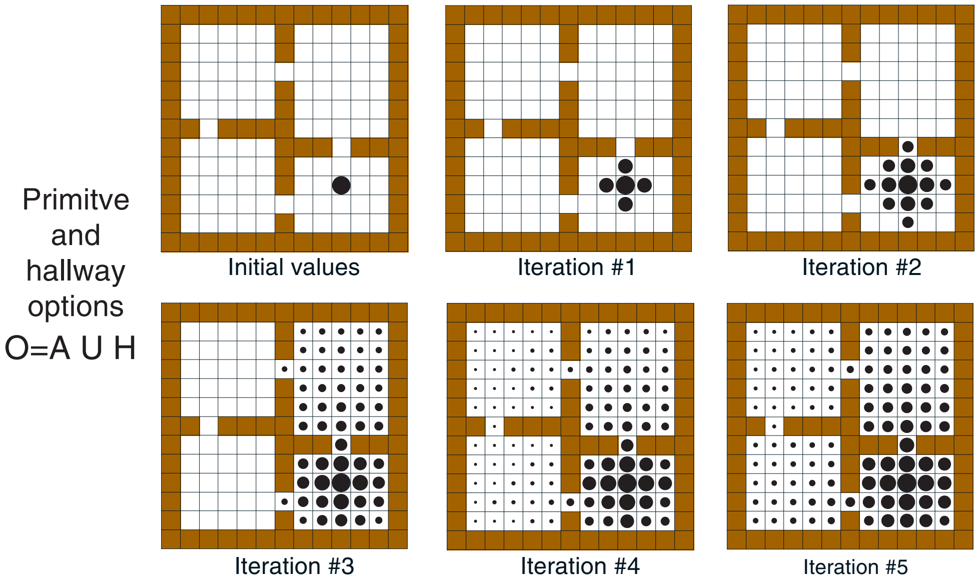

Figure 4.4. An example in which the goal is different from the subgoal of the hallway options. Planning here was done by SVI with options $\mathcal{O} = \mathcal{A} \cup \mathcal{H}$. Initial progress was due to the models of the primitive actions, but by the third iteration room-to-room planning dominated and greatly accelerated planning.

4.4 Illustration: Random Options

In the rooms example, the options are designed to capture an important feature of the environment: hallway states are important for navigation, because they are funneling states between different regions of the state space. But options can help to speed up planning even if they are not so carefully designed (Precup et al., 1998).

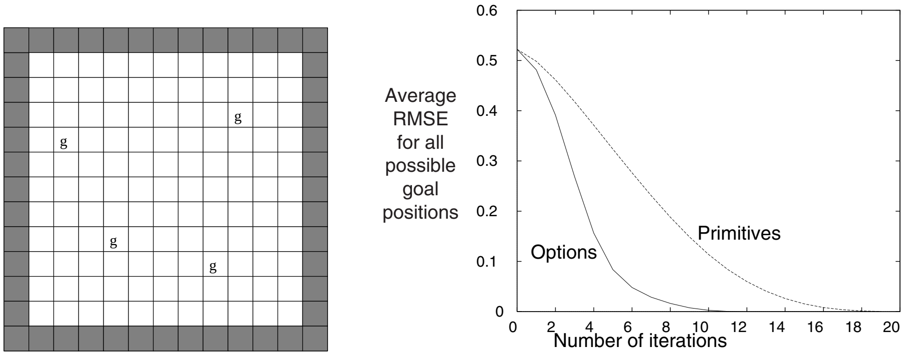

Figure 4.5. Empty gridworld task. Options allow the error in the value function estimation to decrease more quickly

In order to illustrate this point, consider the task depicted on the left panel of figure 4.5. The dynamics of the environment is the same as in the previous example. In addition to the four primitive actions, the agent can use four additional higher-level options, to travel to each of the marked locations. These locations have been chosen randomly inside the environment. Accurate multi-time models for all the options are also available.

The agent is repeatedly given new goal positions and it needs to compute optimal paths to these positions as quickly as possible. In this experiment, we considered all possible goal positions. In each case, the value of the goal state is 1, there are no rewards along the way, and the discounting factor is $\gamma = 0.9$. We performed planning according to the SVI method, where the starting values are $V_0(s) = 0$ for all the states except the goal state, for which $V_0(goal) = 1$. In the first experiment, the agent was only allowed to use primitive actions, while in the second case, it used both the primitive actions and the higher-level options.

The right panel in figure 4.5 shows the average root mean squared error in the estimate of the optimal value function over the whole environment. The average is computed over all possible positions of the goal state. The use of higher-level options introduces a significant speedup in convergence, even though the options have been chosen arbitrarily. Note that an iteration using all the options is slightly more expensive than an iteration using only primitive actions. This aspect can be improved by using more sophisticated methods of ordering the options before doing the update.

SVI was used in these examples because it is a particularly simple planning method that makes the potential advantage of temporally extended options particularly clear. In large problems, SVI is impractical because the number of states is too large to complete many iterations, often not even one. In practice it is often necessary to be very selective about the states updated, the options considered, and even the next states considered. These issues are not resolved by using temporally extended options, but they are not greatly aggravated either. Options provide a tool for dealing with them more flexibly. Planning with options need not be more complex than planning with actions. In the SVI experiments above there were four primitive options and eight hallway options, but in each state only two hallway options needed to be considered. In addition, the models of the primitive actions generate four possible successors with non-zero probability whereas the multi-step options generate only two. Thus, planning with the temporally extended options was actually computationally cheaper than conventional SVI in this case. In the second experiment this was not the case, but the use of multi-step options still did not greatly increase the computational costs. In general, of course, there is no guarantee that multi-step options will reduce the overall expense of planning. For example, Hauskrecht et al. (1998) showed that adding such options may actually slow SVI if the initial value function is optimistic. Research with deterministic macro-operators has identified a related "utility problem" (an increase in the cost of deliberation) when too many macros are used (e.g., see Etzioni, 1990; Minton, 1990; Tambe, Newell, and Rosenbloom, 1990; Greiner and Jurisica, 1992; Gratch and DeJong, 1996). Temporal abstraction provides the flexibility to reduce greatly computational complexity, but can also have the opposite effect if used indiscriminately. Pruning efficiently the space of options to consider during planning and learning will be the topic of future work.

4.5 SMDP Value Learning

The problem of finding an optimal policy over a set of options $\mathcal{O}$ can also be addressed by SMDP learning methods, as developed by Bradtke and Duff (1995), Parr and Russell (1998), Mahadevan et al. (1997), or McGovern, Sutton & Fagg (1997). Like in the planning method discussed above, each option is viewed as an indivisible, opaque unit. In a given state s, the learning agent can pick an option o and executes it until the option terminates, in some state s'. Based on the experience accumulated between s and s', an approximate option-value function $Q(s,o)$ is updated. For example, the SMDP version of one-step Q-learning (Bradtke & Duff, 1995), which we call *SMDP Q-learning*, updates after each option termination by:

$$Q(s,o) \leftarrow Q(s,o) + \alpha \left[r + \gamma^k \max_{o' \in \mathcal{O}_{s'}} Q(s',o') - Q(s,o) \right], \tag{4.10}$$

where k denotes the number of time steps elapsing between s and s', r denotes the cumulative discounted reward over this time, and it is implicit that the step-size parameter α may depend arbitrarily on the states, option, and time steps. The estimate $Q(s,o)$ converges to $Q^*_{\mathcal{O}}(s,o)$ for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$ under conditions similar to those for conventional Q-learning (Parr, 1998).

As an illustration, we applied SMDP Q-learning to the rooms example (Figure 4.1) with the goal at G_1 and at G_2 respectively. As in the case of planning, three different sets of options were used: $\mathcal{A}$ (only the primitive actions), $\mathcal{H}$ (only the hallway options), and $\mathcal{A} \cup \mathcal{H}$ (both primitive and hallway options). In all cases, options were selected from the set according to an ϵ-*greedy* method dependent on the current option-value estimates. That is, given the current estimates $Q(s,o)$, let $o^* = \arg\max_{o \in \mathcal{O}_s} Q(s,o)$

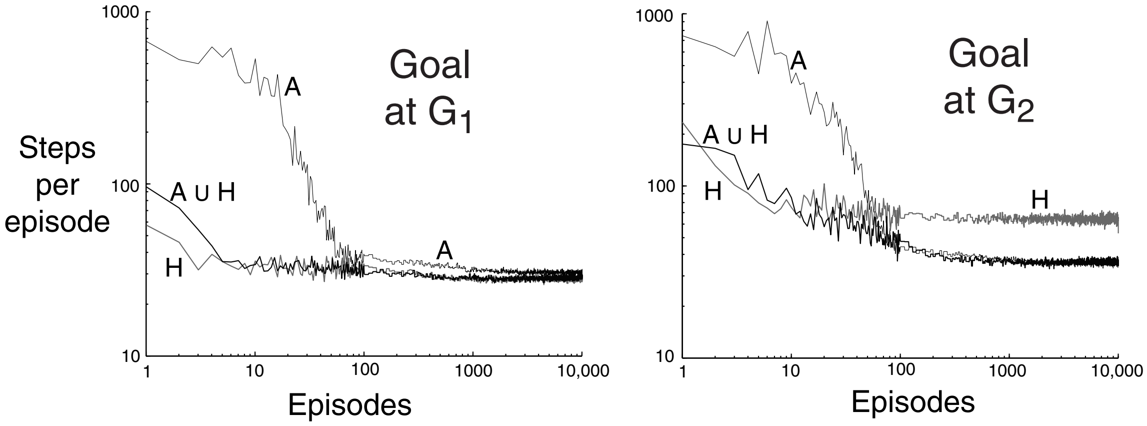

Figure 4.6. Performance of SMDP Q-learning in the rooms example with various goals and sets of options. After 100 episodes, the data points are averages over groups of 10 episodes to make the trends clearer. The step size parameter was optimized to the nearest power of 2 for each goal and set of options. The results shown used $\alpha = \frac{1}{8}$ in all cases except that with $\mathcal{O} = \mathcal{H}$ and G_1 ($\alpha = \frac{1}{16}$) and that with $\mathcal{O} = \mathcal{A} \cup \mathcal{H}$ and G_2 ($\alpha = \frac{1}{4}$).

denote the best valued action (with ties broken randomly). Then the policy used to select options was:

$$\mu(s, o) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{O}_s|} & \text{if } o = o^* \\ \frac{\epsilon}{|\mathcal{O}_s|} & \text{otherwise,} \end{cases}$$

for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$. The probability of a random action, ϵ, was 0.1 in all the experiments. The initial state of each trial was in the upper-left corner. Figure 4.6 shows learning curves for both goals and all sets of options. In all cases, temporally extended options caused the goal to be reached much more quickly, even on the very first trial. With the goal at G_1, these methods maintained an advantage over conventional Q-learning throughout the experiment, presumably because they did less exploration. The results were similar with the goal at G_2, except that the $\mathcal{H}$ method performed worse than the other methods in the long term. This is because the best solution requires several steps of primitive actions (the hallway options alone find the best solution running between hallways and sometimes stumbling upon G_2).

For the same reason, the advantages of the $\mathcal{A} \cup \mathcal{H}$ method over the $\mathcal{A}$ method were also reduced.

4.6 Conclusions

In this chapter we developed the link between options and SMDPs. We showed that an MDP with a set of options is an SMDP. This enables us to define multi-time models of options, similar to the models of actions in an SMDP. We also showed how SMDP learning and planning methods could be applied to find a solution faster than standard MDP methods.

SMDP methods work at the level of options only, without using any of the information about the underlying MDP, or the internal structure of the options. The model of execution of the options is also restricted to be call-and-return. In subsequent chapters we will present new learning and planning algorithms, which take advantage of the underlying MDP structure to obtain better solutions in a more efficient way.

CHAPTER 5

INTRA-OPTION LEARNING

SMDP methods apply to options, but only when they are treated as opaque indivisible units. Once an option has been selected, such methods require that its policy be followed until the option terminates. More interesting and potentially more powerful methods are possible by looking inside options and by altering their internal structure. In the rest of the dissertation we focus on methods that exploit the MDP structure underlying the SMDP defined by the options.

In this chapter we propose an alternative to SMDP learning methods. One drawback to SMDP learning methods is that they need to execute an option to termination before they can learn about it. Therefore, they cannot be used for non-terminating options and can only be applied to one option at a time – the option that is executing at that time. However, if we are willing to look at the structure *inside* the options, then we can use special temporal-difference methods to learn usefully about an option before the option terminates. We call these *intra-option learning* methods because they learn about an option from a fragment of experience *within* the option.

Intra-option methods are examples of *off-policy* learning methods (Sutton & Barto, 1998). Off-policy learning methods learn about the consequences of one policy from data generated by following another, potentially different policy. In general, off-policy learning can greatly multiply learning because many policies can be learned about at the same time, while only one can be followed. This is true also in the case of intra-option methods, which can be used to learn simultaneously about many different options from the same experience. Moreover, they can learn about the values of executing options *without ever executing* those options, as long as some action selections are made that are consistent with the option. Therefore, they are more flexible than SMDP learning, and they make more efficient use of the real experience.

Intra-option methods were introduced by Sutton (1995) for a prediction problem with a single unchanging policy. In this chapter we present general intra-option learning algorithms for learning models of options and value functions over options (Sutton, Precup & Singh, 1998b). These are temporal-difference one-step learning algorithms that converge to correct values under standard assumptions. Versions of these algorithms using eligibility traces are considered in subsequent chapters.

5.1 Intra-Option Model Learning

In this section we introduce a new method for learning to approximate the model r_s^o and $p_{ss'}^o$ of an option o, given experience and knowledge of o. The most straightforward approach to learning the model of an option is to execute the option to termination many times in each state s, recording the resultant next states s', cumulative discounted rewards r, and elapsed times k. These outcomes can then be averaged to approximate the expected values for r_s^o and $p_{ss'}^o$ given by (4.1) and (4.2). For example, an incremental learning rule for this could update its estimates $\hat{r}_s^o$ and $\hat{p}_{sx}^o$, for all $x \in \mathcal{S}$, after each execution of o initiated in state s and terminated in state s', by

$$\hat{r}_s^o = \hat{r}_s^o + \alpha[r - \hat{r}_s^o], \quad \text{and} \tag{5.1}$$

$$\hat{p}_{sx}^o = \hat{p}_{sx}^o + \alpha[\gamma^k \delta_{xs'} - \hat{p}_{sx}^o], \forall x \in \mathcal{S}^+, \tag{5.2}$$

$\delta_{s'x} = 1$ if $s' = x$ and is 0 else, and where the step-size parameter, α, may be constant or may depend on the state, option, and time. For example, if α is 1 divided by the number of times that o has been experienced in s, then these updates maintain the estimates as sample averages of the experienced outcomes. However the averaging is done, we call these *SMDP model-learning methods* because they are based on jumping from initiation to termination of each option, ignoring what happens along the way. In the special case in which o is a primitive action, the SMDP model-learning methods reduce exactly to those used to learn conventional one-step models of actions.

Let us consider first the case in which o is a deterministic Markov option. Based on an execution of o from t to $t+k$, SMDP methods extract a single training example for r^o_s and $p^o_{ss'}$. But because o is Markov, it is, in a sense, also initiated at each of the steps between t and $t+k$. The jumps from each intermediate s_{t+i} to s_{t+k} are also valid experiences with o, experiences that can be used to improve estimates of $r^o_{s_{t+i}}$ and $p^o_{s_{t+i}s'}$. Or consider an option that is very similar to o and which would have selected the same actions, but which would have terminated one step later, at $t+k+1$ rather than at $t+k$. Formally this is a different option, and formally it was not executed, yet all this experience could be used for learning relevant to it. In fact, an agent can often learn about an option from experience that is only slightly related (occasionally selecting the same actions) to what would be generated by executing the option. This is the idea of off-policy training — to make full use of whatever experience occurs in order to learn as much possible about all options, irrespective of their role in generating the experience.

Just as there are Bellman equations for value functions, there are also Bellman equations for models of options. Consider the intra-option learning of the model of a Markov option $o = \langle \mathcal{I}, \pi, \beta \rangle$. The correct model of o is related to itself by

$$r^o_s = \sum_{a \in \mathcal{A}_s} \pi(s,a) E\Big\{ r + \gamma(1-\beta(s')) r^o_{s'} \Big\} \tag{5.3}$$

$$= \sum_{a \in \mathcal{A}_s} \pi(s,a) \left[r^a_s + \sum_{s' \in \mathcal{S}} p^a_{ss'} (1-\beta(s')) r^o_{s'} \right] \tag{5.4}$$

where r and s' are the reward and next state given that action a is taken in state s, and

$$
\begin{aligned}
p^o_{sx} &= \sum_{a \in \mathcal{A}_s} \pi(s,a) \gamma E\Big\{(1-\beta(s'))p^o_{s'x} + \beta(s')\delta_{s'x}\Big\} \\
&= \sum_{a \in \mathcal{A}_s} \pi(s,a) \sum_{s' \in \mathcal{S}} (p^a_{ss'}(1-\beta(s'))p^o_{s'x} + \beta(s')\delta_{s'x})
\end{aligned}
$$

for all $s, x \in \mathcal{S}$. How can we turn these Bellman equations into update rules for learning the model? First consider that action a_t is taken in s_t and that the way it was selected is consistent with $o = \langle \mathcal{I}, \pi, \beta \rangle$, that is, that a_t was selected with the distribution $\pi(s_t, \cdot)$. Then the Bellman equations above suggest the following temporal-difference update rules:

$$
\hat{r}^o_{s_t} \leftarrow \hat{r}^o_{s_t} + \alpha \left[r_{t+1} + \gamma(1-\beta(s_{t+1}))\hat{r}^o_{s_{t+1}} - \hat{r}^o_{s_t} \right] \tag{5.5}
$$

and

$$
\hat{p}^o_{s_t x} \leftarrow \hat{p}^o_{s_t x} + \alpha[\gamma(1-\beta(s_{t+1}))\hat{p}^o_{s_{t+1} x} + \gamma\beta(s_{t+1})\delta_{s_{t+1} x} - \hat{p}^o_{s_t x}], \tag{5.6}
$$

where $\hat{p}^o_{ss'}$ and $\hat{r}^o_s$ are the estimates of $p^o_{ss'}$ and r^o_s, respectively, and α is a positive step-size parameter. The method we call *one-step intra-option model learning* applies these updates to every option consistent with every action taken.

Of course, this is just the simplest intra-option model-learning method. How can we lift the assumptions that o, the option learned about, is Markov, deterministic and represented by a flat representation ? We consider first learning the models of stochastic options. Equations (5.5) and (5.6) apply in this case only if choices of action are made according to the distribution $\pi(s_t, \cdot)$. This is not the case in general if different options are executing, and we want to update the models of all options on every time step. It is not obvious how to keep track of the actual action distribution at state s_t.

The solution we adopt here is a standard trick used in off-policy learning methods: we increase the size of the models by the number of actions. Therefore, instead of keeping track of r^o_s and $p^o_{ss'}$ we will have one component for each primitive action, r^{ao}_s

and $p^{ao}_{ss'}$. The reward component r^{ao}_s is the expected cumulative discounted reward for starting in state s, performing action a and acting according to o afterwards. Similarly, $p^{ao}_{ss'}$ is the expected probability of transition from s to s' when performing action a and then acting according to o, discounted appropriately by the duration of the sequence ao. The multi-time model of the option can be computed as an average of these more explicit models:

$$\begin{aligned} r^o_s &= \sum_a \pi(s,a) r^{ao}_s \\ p^o_{ss'} &= \sum_a \pi(s,a) p^{ao}_{ss'} \end{aligned}$$

With this change in the models, equations (5.5) and (5.6) can be used to learn each component of the models. If action a_t is taken in state s_t, the models are updated by:

$$\hat{r}^{a_t o}_{s_t} \leftarrow \hat{r}^{a_t o}_{s_t} + \alpha \left[r_{t+1} + \gamma(1-\beta(s_{t+1}))\hat{r}^o_{s_{t+1}} - \hat{r}^{a_t o}_{s_t} \right] \tag{5.7}$$

and

$$\hat{p}^{a_t o}_{s_t x} \leftarrow \hat{p}^{a_t o}_{s_t x} + \alpha[\gamma(1-\beta(s_{t+1}))\hat{p}^o_{s_{t+1}x} + \gamma\beta(s_{t+1})\delta_{s_{t+1}x} - \hat{p}^{a_t o}_{s_t x}], \tag{5.8}$$

In the general case of options that depend on the partial history since they were initiated, there is no direct way for applying intra-option learning to acquire state models, because the Bellman equations do not hold inside such options. Of course, one can always execute these options to termination and use SMDP model learning. The other alternative to consider is to expand the models of the options even further, by introducing separate components r^{ao}_h and $p^{ao}_{hs'}$ for each history. With these changes, equations (5.7) and (5.8) hold for these detailed models. Of course, it is impractical to keep general models that depend on whole histories. This approach could be practical in special cases in which the history can be summarized efficiently, or represented using function approximation. More efficient algorithms may be possible in the case of hierarchical Markov options. This is a topic for further research.

5.2 Illustration of Intra-Option Model Learning

As an illustration, consider the application of SMDP and intra-option model-learning methods to the rooms gridworld environment shown in Figure 4.1. The only difference from the previous setup is that there are small negative rewards for each action, with means uniformly distributed between 0 and -1. The rewards are also perturbed by Gaussian noise with standard deviation 0.1. For this experiment, there is no goal state.

The eight hallway options are given as before, but their models are not given and must be learned. Experience is generated by selecting randomly in each state among the two possible options and four possible primitive actions, with no goal state. In the SMDP model-learning method, equations (5.1) and (5.2) were applied whenever an option was selected, whereas, in the intra-option model-learning method, equations (5.5) and (5.6) were applied on every step to all options that were consistent with the action taken on that step. In this example, all options are deterministic, so consistency with the action selected means simply that the option would have selected that action.

For the SMDP method, the step-size parameter was varied so that the model estimates were sample averages, which should give fastest learning. The results of this method are labeled "SMDP 1/t" on the graphs. We also looked at results using a fixed learning rate. In this case and for the intra-option method we tried step-size values of $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, and $\frac{1}{16}$, and picked the best value for each method.

Figure 5.1 shows the learning curves for all three methods, using the best α values, when a fixed alpha was used. The left panel shows the average and maximum absolute error in the reward predictions, and the right panel shows the average absolute error and the maximum absolute error in the transition predictions, averaged over the eight options and over 30 independent runs. The intra-option method approached the correct values more rapidly than the SMDP methods.

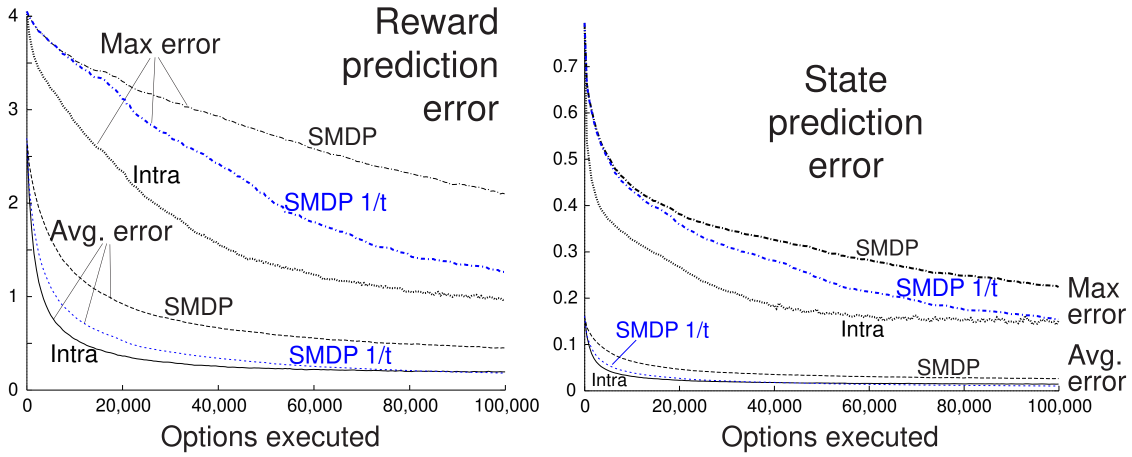

Figure 5.1. Learning curves for model learning by SMDP and intra-option methods.

5.3 Intra-Option Value Learning

We turn now to the intra-option learning of option values, and thus of optimal policies over options. Again, in the case of options that depend on the partial history since initiation, the SMDP methods described in the previous chapter may be the only feasible solution. But in special cases, such as that of Markov options, we can do better if we are willing to look inside the options and consider intra-option methods. As in model learning, intra-option methods for value learning are potentially more efficient than SMDP methods because they extract more training examples from the same experience.

It is convenient to introduce new notation for the value of a state–option pair given that the option is Markov and executing upon *arrival* in the state:

$$U^*_{\mathcal{O}}(s,o) = (1-\beta(s))Q^*_{\mathcal{O}}(s,o) + \beta(s)\max_{o'\in\mathcal{O}} Q^*_{\mathcal{O}}(s,o'),$$

Then we can write Bellman-like equations that relate $Q^*_{\mathcal{O}}(s,o)$ to expected values of $U^*_{\mathcal{O}}(s',o)$, where s' is the immediate successor to s after initiating Markov option $o = \langle \mathcal{I}, \pi, \beta \rangle$ in s:

$$
\begin{aligned}
Q^*_{\mathcal{O}}(s,o) &= \sum_{a\in\mathcal{A}_s} \pi(s,a) E\left\{ r + \gamma U^*_{\mathcal{O}}(s',o) \,\Big|\, s,a \right\} \\
&= \sum_{a\in\mathcal{A}_s} \pi(s,a) \left[r^a_s + \sum_{s'} p^a_{ss'} U^*_{\mathcal{O}}(s',o) \right],
\end{aligned}
$$

where r is the immediate reward upon arrival in s'. Now consider learning methods based on this Bellman equation. Suppose action a_t is taken in state s_t to produce next state s_{t+1} and reward r_{t+1}, and that a_t was selected in a way consistent with the Markov policy π of an option $o = \langle \mathcal{I}, \pi, \beta \rangle$. That is, suppose that a_t was selected according to the distribution $\pi(s_t, \cdot)$. Then the Bellman equation above suggests applying the off-policy one-step temporal-difference update:

$$
Q(s_t,o) \leftarrow Q(s_t,o) + \alpha\Big[(r_{t+1} + \gamma U(s_{t+1},o)) - Q(s_t,o) \Big], \tag{5.9}
$$

where

$$
U(s,o) = (1-\beta(s))Q(s,o) + \beta(s) \max_{o'\in\mathcal{O}} Q(s,o')
$$

The method we call *one-step intra-option Q-learning* applies this update rule to every option o consistent with every action taken a_t.

Theorem 4 (Convergence of one-step intra-option Q-learning) *For any set of deterministic Markov options $\mathcal{O}$, one-step intra-option Q-learning converges with probability 1 to the optimal Q-values, $Q^*_{\mathcal{O}}$, for every option, regardless of what options are executed during learning, provided every primitive action gets executed in every state infinitely often.*

Proof: On experiencing $\langle s, a, r, s' \rangle$, for every option o that picks action a in state s, intra-option Q-learning performs the following update:

$$
Q(s,o) \leftarrow Q(s,o) + \alpha(s,o)[r + \gamma U(s',o) - Q(s,o)].
$$

Let a be the action selection by deterministic Markov option $o = \langle \mathcal{I}, \pi, \beta \rangle$. Our result follows directly from Theorem 1 of Jaakkola et al. (1994) and the observation that the expected value of the update operator $r + \gamma U(s', o)$ yields a contraction, as shown below:

$$
\begin{aligned}
|E\{r + \gamma U(s', o)\} - Q^*_{\mathcal{O}}(s, o)| &= |r^a_s + \sum_{s'} p^a_{ss'} U(s', o) - Q^*_{\mathcal{O}}(s, o)| \\
&= |r^a_s + \sum_{s'} p^a_{ss'} U(s', o) - r^a_s - \sum_{s'} p^a_{ss'} U^*_{\mathcal{O}}(s', o)| \\
&\leq |\sum_{s'} p^a_{ss'} \Big[(1 - \beta(s'))(Q(s', o) - Q^*_{\mathcal{O}}(s', o)) \\
&+ \beta(s')(\max_{o' \in \mathcal{O}} Q(s', o') - \max_{o' \in \mathcal{O}} Q^*_{\mathcal{O}}(s', o')) \Big]| \\
&\leq \sum_{s'} p^a_{ss'} \max_{s'', o''} |Q(s'', o'') - Q^*_{\mathcal{O}}(s'', o'')| \\
&\leq \gamma \max_{s'', o''} |Q(s'', o'') - Q^*_{\mathcal{O}}(s'', o'')|
\end{aligned}
$$

$\diamond$

5.4 Illustration of Intra-Option Value Learning

Again, we use the rooms gridworld environment presented in Figure 4.1, this time with a goal state positioned at G1. At the beginning of each episode, the agent starts in a random cell inside the rooms. When the agent enters the goal state, it gets a reward of 1 and the episode ends. In all the experiments the discount parameter was $\gamma = 0.9$ and all the initial value estimates were 0.

In each of the four rooms we provide, as before, two built-in hallway options designed to take the agent from anywhere within the room to one of the two hallway cells leading out of the room. The policies underlying the options follow the shortest expected path to the hallway.

For the first experiment, we applied the intra-option method without selecting the hallway options. In each episode, the agent started at a random cell and thereafter selected primitive actions randomly, with equal probability. On every transition, the update (5.9) was applied first to the primitive action taken, then to any of the hallway options that were consistent with it. The hallway options were updated in clockwise order, starting from any hallways that faced up from the current state. The value of the step-size parameter was $\alpha = 0.01$.

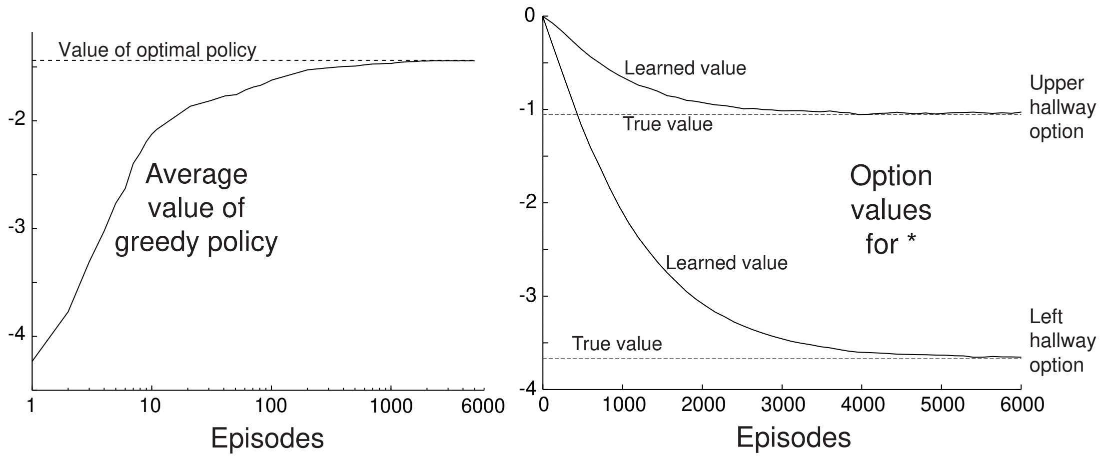

Figure 5.2. The learning of option values by intra-option methods without ever selecting the options. The value of the greedy policy goes to the optimal value (left panel) as the learned values approach the correct values (as shown for one state, in the right panel).

This is a case in which SMDP methods would not be able to learn anything about the hallway options, because these options are never executed. However, the intra-option method learned the values of these actions effectively, as shown in Figure 5.2. The left panel shows the value of the greedy policy learned by the intra-option method, averaged over $\mathcal{S}$ and over 30 repetitions of the whole experiment. The right panel shows the correct and learned values for the two hallway options that apply in state $G2$ from Figure 4.1. Similar convergence to the true values was observed for all the other states and options.

So far we have illustrated the effectiveness of intra-option learning in a context in which SMDP methods do not apply. How do intra-option methods compare to SMDP methods when both are applicable? In order to investigate this question, we used the same environment, but now we allowed the agent to choose among the hallway options as well as the primitive actions, which were treated as one-step options. In this case, SMDP methods can be applied, because all the options are actually executed. We experimented with two SMDP methods: one-step SMDP Q-learning (Bradtke and Duff, 1995) and a hierarchical form of Q-learning called *macro Q-learning* (McGovern, Sutton and Fagg, 1997). The difference between the two methods is that, when taking a multi-step option, SMDP Q-learning only updates the value of that option, whereas macro Q-learning also updates the values of the one-step options (actions) that were taken along the way.

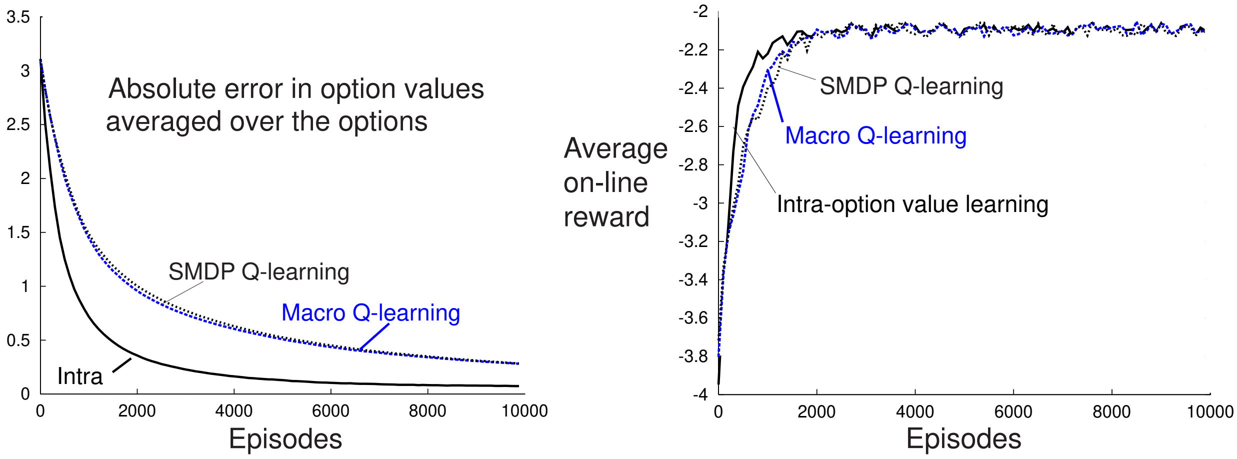

Figure 5.3. Comparison of SMDP, intra-option and macro Q-learning. Intra-option methods converge faster to the correct values.

In this experiment, options were selected not at random, but in an ϵ-greedy way dependent on the current option-value estimates. That is, given the current estimates $Q(s,o)$, let $o^* = \arg\max_{o \in \mathcal{O}_s} Q(s,o)$ denote the best valued action (with ties broken randomly). Then the policy used to select options was

$$\mu(s,o) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{O}_s|} & \text{if } o = o^* \\ \frac{\epsilon}{|\mathcal{O}_s|} & \text{otherwise,} \end{cases}$$

for all $s \in \mathcal{S}$ and $o \in \mathcal{O}$. The probability of a random action, ϵ, was set at 0.1 in all cases. For each algorithm, we tried step-size values of $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, and $\frac{1}{16}$ and then picked the best one.

Figure 5.3 shows two measures of the performance of the learning algorithms. The left panel shows the average absolute error in the estimates of $Q^*_{\mathcal{O}}$ for the hallway options, averaged over the input sets $\mathcal{I}$, the eight hallway options, and 30 repetitions of the whole experiment. The intra-option method showed significantly faster learning than any of the SMDP methods. The right panel shows the quality of the policy executed by each method, measured as the average reward over the state space. The intra-option method was also the fastest to learn by this measure.

5.5 Conclusions

In this chapter we introduced a new class for learning methods for learning the models and values of options. Intra-options methods are designed to make very efficient use of the data generated from an agent's experience in the environment, regardless of the behavior that generated the data. Intra-option learning allows an agent to learn about many different options from the same data. As illustrated in our experiments, these methods are both more flexible and more efficient than the SMDP methods presented in chapter 4. The speed gain of intra-options methods increases if there is a significant amount of overlap between options in their choices of primitive actions.

Intra-options learning methods can provide significant speed improvements over SMDP methods in the case in which real experience is expensive to generate, but computation can be performed cheaply. This is the typical situation for RL agents. When using intra-option learning, on should keep in mind that these methods increase the amount of computation per time step. On every time step, intra-option learning algorithms could potentially update the information regarding all options. If the agent has many options available, then the amount of computation can become an issue. Of course, in such situations, the agent can always choose to update information only for part of the options.

All the intra-option learning methods presented so far were one-step methods. It would be expected that their speed could be improved even further by using eligibility traces. However, the topic of eligibility traces for off-policy methods has not been studied in depth yet. The next two chapters are dedicated to this subject.

CHAPTER 6

OFF-POLICY LEARNING: MONTE CARLO METHODS

The intra-option learning methods introduced in the previous chapter have brought into focus the larger class of off-policy learning methods. Off-policy learning is key to learning efficiently if an agent uses many options, because it allows learning about many options in parallel, from the same real experience. This can generate big speedups compared to on-policy learning methods, such as SMDP learning, which can only learn about the option being followed. The ability to learn about many options in parallel alleviates the utility problem generated by increasing the action space of the agent.

In RL, off-policy learning has been studied mostly for control problems, in which the agent learns about the optimal policy for acting in an MDP while following a different, stochastic policy. The most well-known off-policy learning algorithm is probably Q-learning (Watkins, 1989). Convergence results have been established for one-step off-policy learning methods. However, the theory of multi-step and eligibility trace methods for off-policy learning has not been developed very much.

The intra-option learning algorithms that we presented so far were all one-step algorithms. Our goal now is to extend these algorithms to include eligibility traces. Eligibility traces have been shown to speed up temporal-difference learning algorithms in general (Sutton & Barto, 1998). Using eligibility traces for intra-option methods should have the same effect.

In order to explore off-policy learning in depth, we are going to consider the more general case of model-free policy evaluation. It is convenient to consider this case because it is simpler than the option model learning and option value learning considered in the previous chapter. This allows us to focus on the issue of performing multi-step off-policy updates. All the ideas that we will develop for policy evaluation can be easily applied to intra-option learning, with the benefit of even faster learning.

6.1 Policy Evaluation

In this and the next chapter we consider the problem of estimating the state-action value function Q^π, for a given *target policy* π, from experience generated by following a different *behavior policy* b. Both π and b are fully specified by the probability distribution of the actions in each state. The two policies are completely arbitrary, except that the behavior policy must be *soft*, meaning that it must have a non-zero probability of selecting every action in each state. We will restrict the discussion to the episodic framework, in which the agent interacts with the environment in an infinite sequence of episodes, numbered $m = 1, 2, 3, \ldots$, each of which consists of a finite number of time steps, $t = 0, 1, 2, \ldots, T_m$.

The value of a state-action pair is the expected value of the total reward received starting from that state, taking that action, and following the target policy afterwards:

$$Q^\pi(s,a) = E\Big\{r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-1} r_T \mid s_t = s, a_t = a, \pi\Big\}, \tag{6.1}$$

where T is the duration of an episode. In this chapter we focus on *Monte Carlo* learning, in which we have samples of such total rewards, but distributed according to the behavior policy b instead of the target policy π.

A standard technique for learning expected values from samples when there is a mismatch between distributions is importance sampling; we apply two variations of it here. We also introduce a new algorithm that uses importance sampling corrections while taking advantage of the special structure of MDPs. In the next chapter we focus on temporal-difference algorithms for solving the same problem.

6.2 Importance Sampling

Importance Sampling (Rubinstein, 1981) is a standard technique for estimating the expected value of a random variable x with distribution d from samples, when the samples are drawn from another distribution d'. For instance, the target distribution d could be normal, while the sampling distribution d' is uniform (see Figure 6.1).

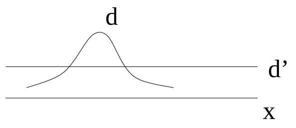

Figure 6.1. Different target and sampling distributions

In its classical form, importance sampling computes the expected value $E\{x \mid d\}$ based on a simple observation:

$$E_d\{x\} = \int_x x\, d(x)\, dx = \int_x x \frac{d(x)}{d'(x)} d'(x) dx = E_{d'}\left\{ x \frac{d(x)}{d'(x)} \right\},$$

which leads to the importance sampling estimator:

$$IS = \frac{1}{n} \sum_{i=1}^{n} x_i \frac{d(x_i)}{d'(x_i)}, \tag{6.2}$$

where x_i are samples selected according to d'. This estimator computes the average of the sample values, where each sample is weighted differently based on the ratio of its likelihood of occurring under the two distributions. This weighting gives more importance to samples that occur rarely under the sampling distribution d' but occur frequently under the target distribution d. If d and d' are the same, then all the samples have a weight of 1, and the estimator becomes the usual arithmetic average of the samples. The importance sampling estimator (6.2) is *consistent*, meaning that it converges with probability 1 to $E_d\{x\}$ as the number of samples goes to infinity, and *unbiased*, meaning its expected value after any number of samples is also $E_d\{x\}$ (Rubinstein, 1981).

A less known variant of this technique is *weighted importance sampling*, which performs a weighted average of the samples, with weights $\frac{d(x_i)}{d'(x_i)}$. The weighted importance sampling estimator is:

$$ISW = \frac{\sum_{i=1}^{n} x_i \frac{d(x_i)}{d'(x_i)}}{\sum_{i=1}^{n} \frac{d(x_i)}{d'(x_i)}} \tag{6.3}$$

The weighted importance sampling estimator (6.3) is a consistent but biased estimator of $E\{x \mid d\}$ (Rubinstein, 1981). Nevertheless, his estimator is often faster and more stable in practice than (6.2). Intuitively, this property is due to the fact that, if an unlikely event occurs, its weight will be very large, and will cause a large variation in the classical estimator. In the weighted estimator, the large weight appears in the denominator as well, and therefore smoothes the variation.

6.3 Applying Importance Sampling to MDPs

In the case of MDPs, the samples come in the form of episodes, which are complete sequences of states, actions and rewards, ending in a terminal state:

$$s_0 a_0 r_1 s_1 a_1 r_2 \ldots s_{T_{m-1}} a_{T_{m-1}} r_{T_m} s_{T_m}.$$

The goal is to estimate the state-action value function $Q^\pi(s,a)$ for a given state s and action a. Let M be the number of episodes containing state-action pair (s,a) and t_m be the first time t when $(s_t, a_t) = (s,a)$ in the mth of these episodes. Then we define the first-visit importance sampling estimate for $Q^\pi(s,a)$ as

$$Q^{IS}(s,a) \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^{M} R_m w_m, \tag{6.4}$$

where R_m is the return following (s,a) in episode e,

$$R_m \stackrel{\text{def}}{=} r_{t_m+1} + \gamma r_{t_m+2} + \ldots + \gamma^{T_m - t_m - 1} r_{T_m}$$

and w_m is the importance sampling weight assigned to episode m:

$$w_m \stackrel{\text{def}}{=} \frac{\pi_{t_m+1}}{b_{t_m+1}} \frac{\pi_{t_m+2}}{b_{t_m+2}} \cdots \frac{\pi_{T_m-1}}{b_{T_m-1}}.$$

Here, and in the following sections, we denote by $\pi_t = \pi(s_t, a_t)$ and similarly $b_t = b(s_t, a_t)$. Similar estimators can be computed for every-visit Monte Carlo as well. First-visit estimators have the advantage of being unbiased (Singh & Sutton, 1996), and therefore we will use such estimators in this dissertation.

Similarly, we define the *weighted importance sampling estimator* (Sutton & Barto, 1998) as

$$Q^{ISW}(s, a) \stackrel{\text{def}}{=} \frac{\sum_{m=1}^{M} R_m w_m}{\sum_{m=1}^{M} w_m}. \tag{6.5}$$

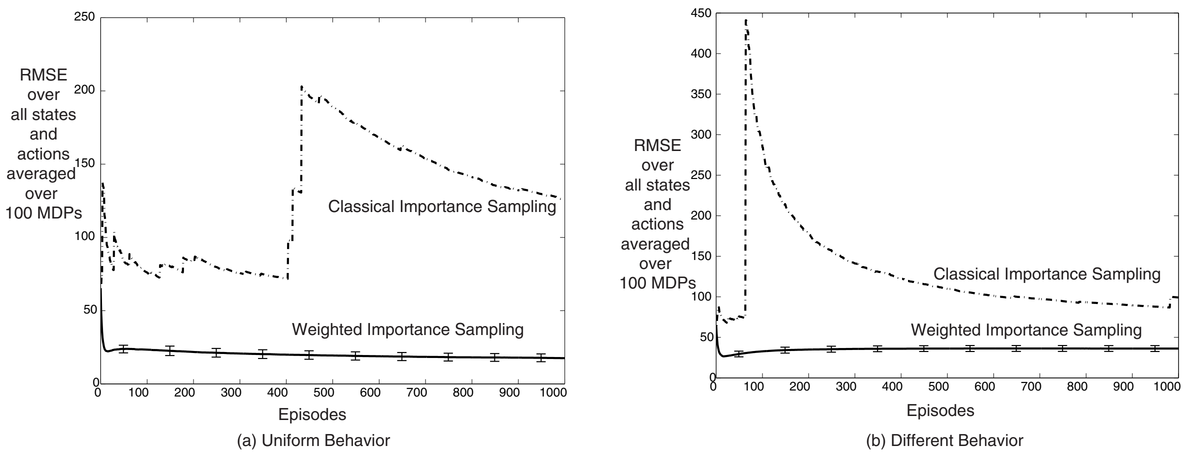

Figure 6.2. Comparison of classical and weighted importance sampling on 100 randomly generated MDPs. On the left, the behavior policy chose 50-50 from the two actions. On the right, the behavior policy chose with 20-80 probabilities, exactly opposite to the target policy. In both cases, the weighted algorithm is faster and more stable.

Figure 6.3 presents an empirical comparison of the classical and weighted importance sampling estimators. The comparison was performed using 100 different randomly generated MDPs. Each MDP has 100 states, one of which is terminal. Two actions were available in each nonterminal state, and each action branched to four next states, with random probabilities (the partition of unity was selected by picking three random split points uniformly randomly from $[0, 1]$). The immediate rewards for each state-action pair were chosen uniformly randomly, between 0 and 1. The target policy was to select the first action with 80% probability and the second action with 20% probability. We used two different behavior policies. In the *uniform behavior* case (left panel) both actions were equally likely, whereas in the *different behavior* case, the first action was selected with 20% probability and the second action with 80% probability, resulting in a policy very different from the target policy. The initial state of each episode was chosen uniformly randomly from the nonterminal states. All the MDPs terminated with probability 1, so we used $\gamma = 1$.

Figure 6.3 shows, for each estimator, the root of the total mean squared error between the estimator and the true action values for the 200 state-action pairs, averaged over the 100 MDPs. this measure is computed at the beginning of learning, and after each of the first 1000 episodes. For the weighted importance sampling algorithm, the graph also includes error bars equal to one standard deviation. For the classical importance sampling, the maximum standard deviation is on the order of 3000, therefore we omitted the error bars. This result confirms the fact that the classical importance sampling algorithm has very high variance, which recommends against its use in practice. Also, as shown in the figure, the weighted version of the algorithm is faster and more stable than the classical version. This result was consistent across all MDPs we experimented with.

6.4 Per-Decision Importance Sampling

Both importance sampling algorithms presented so far require known Markov behavior policies. They are also inherently Monte Carlo algorithms, because they put a weight on the total return R_m obtained during an episode. There is no easy way of implementing either algorithm in an incremental fashion, for instance by performing TD-like updates after every step of the execution. In order to be able to perform such updates, an algorithm should perform a weighting of each reward r_t obtained along the trajectory followed during the episode.

In this section we present a new algorithm that performs importance sampling weightings for each decision step along the way. Such a weighting can be computed if, instead of treating each return as one indivisible sample, we take into account the fact that the returns come from an underlying MDP. We will focus here on the Monte Carlo version of the estimator. In the next chapter we present a natural TD implementation.

In order to justify the estimator, let us examine the term $R_m w_m$ from equations (6.4) and (6.5): The terms of the sum can be naturally separated into two parts, one containing the $\frac{\pi}{b}$ ratios from t_{m+1} to $i-1$, and one containing the ratios from i to T_{m-1}. Intuitively, the weight on reward r_i should not depend on the future after time i, only on the history to that point. This is the idea behind the *per-decision importance sampling estimator*:

$$Q^{PD}(s,a) \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^{M} \sum_{k=1}^{T_m - t_m} \gamma^{k-1} r_{t_m+k} \prod_{i=t_m+1}^{t_m+k-1} \frac{\pi_i}{b_i}. \tag{6.6}$$

The estimator weights each reward along a trajectory according to the likelihood of the trajectory up to that point, under the target and the behavior policy. If the target and the behavior policy are the same, the estimator is simply the average of the total returns from each episode. We now show that this estimate is indeed correct:

Theorem 5 *The per-decision importance sampling estimator Q^{PD} given by (6.6) is a consistent unbiased estimator of $Q^\pi(s,a)$.*

Proof: We know that the classical importance sampling estimator Q^{IS} is consistent and unbiased:

$$E\left\{ \left(\sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k} \right) \prod_{i=t+1}^{T-1} \frac{\pi_i}{b_i} \mid s_t = s, a_t = a, b \right\} = Q^\pi(s,a).$$

We will show that the per-decision importance sampling estimator Q^{PD} has the same expected value as Q^{IS}. Let us move the importance sampling correction inside the sum, and examine the expectation for the k-th term:

$$
\begin{aligned}
&E\left\{\gamma^{k-1} r_{t+k} \prod_{i=t+1}^{T-1} \frac{\pi_i}{b_i} \mid s_t = s, a_t = a, b\right\} = \\
&\quad = E\left\{\gamma^{k-1} r_{t+k} \frac{\pi_{t+1}}{b_{t+1}} \cdots \frac{\pi_{t+k-1}}{b_{t+k-1}} \mid s_t, a_t, \ldots s_{t+k-1}, a_{t+k-1}\right\} \\
&\quad \cdot E\left\{\frac{\pi_{t+k}}{b_{t+k}} \cdots \frac{\pi_{T-1}}{b_{T-1}} \mid s_t, a_t, \ldots s_{t+k}, a_{t+k}, b\right\}.
\end{aligned}
$$

Since the underlying environment is an MDP, the second factor can be re-written as:

$$
E\left\{\frac{\pi_{t+k}}{b_{t+k}} \cdots \frac{\pi_{T-1}}{b_{T-1}} \mid s_{t+k}, a_{t+k}, b\right\}.
$$

The expected value of this term is 1. Therefore,

$$
\begin{aligned}
&E\left\{\left(\sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k}\right) \prod_{i=t+1}^{T-1} \frac{\pi_i}{b_i} \mid s_t = s, a_t = a, b\right\} = \\
&\quad E\left\{\sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k} \prod_{i=t+1}^{t+k-1} \frac{\pi_i}{b_i} \mid s_t = s, a_t = a, b\right\},
\end{aligned}
$$

which concludes the proof. $\diamond$

We can also devise a weighted version of the per-decision importance sampling algorithm. The reason for such a version is to smooth out large variations in the updates, if unlikely events happen. The idea is simply to divide the estimator by the sum of the weights during each episode:

$$
Q^{PDW}(s,a) \stackrel{\text{def}}{=} \frac{\sum_{m=1}^{M} \sum_{k=1}^{T_m - t_m} \gamma^{k-1} r_{t_m+k} \prod_{i=t_m+1}^{t_m+k-1} \frac{\pi_i}{b_i}}{\sum_{m=1}^{M} \sum_{k=1}^{T_m - t_m} \gamma^{k-1} \prod_{i=t_m+1}^{t_m+k-1} \frac{\pi_i}{b_i}}. \qquad (6.7)
$$

This *weighted per-decision importance sampling estimator* is consistent but biased, just like the weighted importance sampling estimator Q^{ISW}.

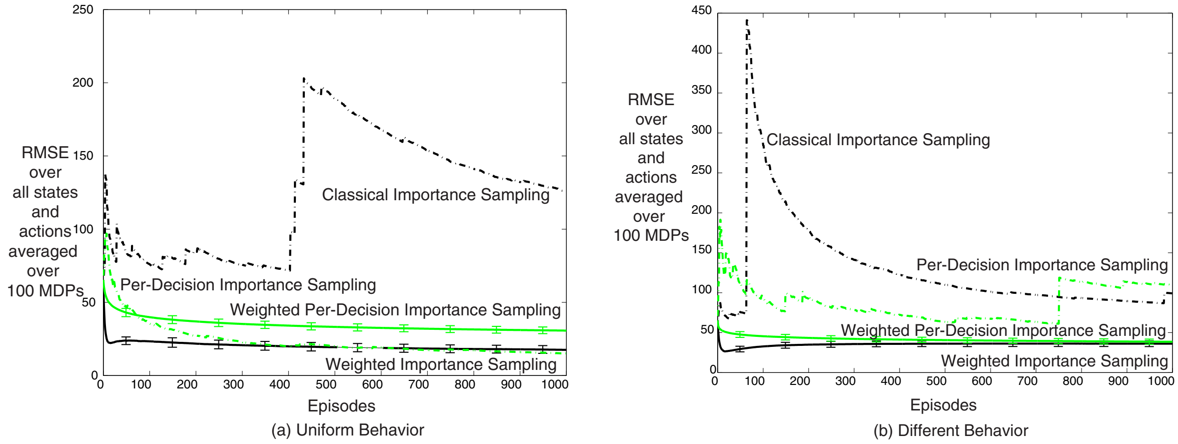

Figure 6.3. Comparison of Classical (Per-Return) and Per-Decision Monte Carlo Importance Sampling Algorithms

Figure 6.4 presents an empirical comparison of the per-decision algorithms with the classical (per-return) version, on the same testbed of 100 randomly generated MDPs (described in detail in section 6.3). The error measure is again the root of the total mean squared error for all the state-action pairs, averaged over the 100 MDPs. For the weighted per-decision algorithm, we also show error bars equal to one standard deviation. The standard deviation for the unweighted per-decision was on the order of 100 in the uniform behavior case (left panel) and on the order of 500 in the different behavior case (right panel). Since the weighted per-decision estimator has significantly smaller variance and more stable behavior, we recommend its use instead of the unweighted version, even though it is not consistently faster (as seen in the left panel).

6.5 Conclusions

In this chapter we presented Monte Carlo algorithms for policy evaluation, based on importance sampling corrections. One of the algorithms is a straightforward application of importance sampling. The other algorithm, per-decision importance sampling, is a new method, which takes into account the fact that the reward samples come from an MDP. We have shown that per-decision importance sampling algorithm converges to correct Q-values, like classical importance sampling, and the empirical results suggest that it is comparable in terms of speed.

Both importance sampling algorithms presented so far build Q-value estimates from samples of total reward. Since our ultimate goal is to use these ideas in on-line algorithms for options, we would like to have more on-line, incremental implementations of these algorithms. This will be the focus of the next chapter.

CHAPTER 7

OFF-POLICY LEARNING: TEMPORAL-DIFFERENCE METHODS

Temporal-Difference (TD) learning is one of the central ideas of reinforcement learning algorithms (Sutton & Barto, 1998). One advantage of TD methods over Monte Carlo methods is their natural implementation in an on-line, incremental manner. This allows them to be applicable in infinite-horizon as well as trial-based tasks. TD methods using eligibility traces are also faster than Monte Carlo methods. In the case of off-policy learning for policy evaluation, there exists an immediate one-step TD algorithm. However, there are no known eligibility trace TD algorithms. In this chapter we present two such algorithms. One is a TD implementation of the per-decision importance sampling presented in the previous chapter. The second algorithm, called tree backup, extends the one-step algorithm, and has the advantage of converging to correct values even when the behavior policy is non-Markov or unknown.

7.1 One-Step TD Learning

The one-step TD learning algorithm is based on the Bellman equation for state-action values:

$$Q^{\pi}(s,a) = r_s^a + \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a') Q^{\pi}(s',a') \tag{7.1}$$

This equation can be turned into the following update rule:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha(r_{t+1} + \gamma \sum_{a} \pi(s_{t+1},a) Q(s_{t+1},a)) \tag{7.2}$$

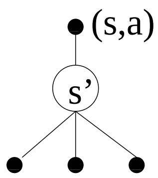

Figure 7.1. Backup tree for one-step TD

The backup tree (Sutton & Barto, 1998) for this algorithm is shown in Figure 7.1. The hollow circles represent the states, and the filled circles represent actions. At each step along a trajectory, there are several possible choices of action according to the target policy. The one-step target for the TD update combines the value estimates for for these actions according to their probabilities of being taken under the target policy. Then the update moves the value function estimate towards the target.

The one-step algorithm converges with probability 1 to the correct value function Q^π if the learning rate α is decreased appropriately over time. This follows simply from the general convergence theorem of Jaakkola, Jordan & Singh (1994).

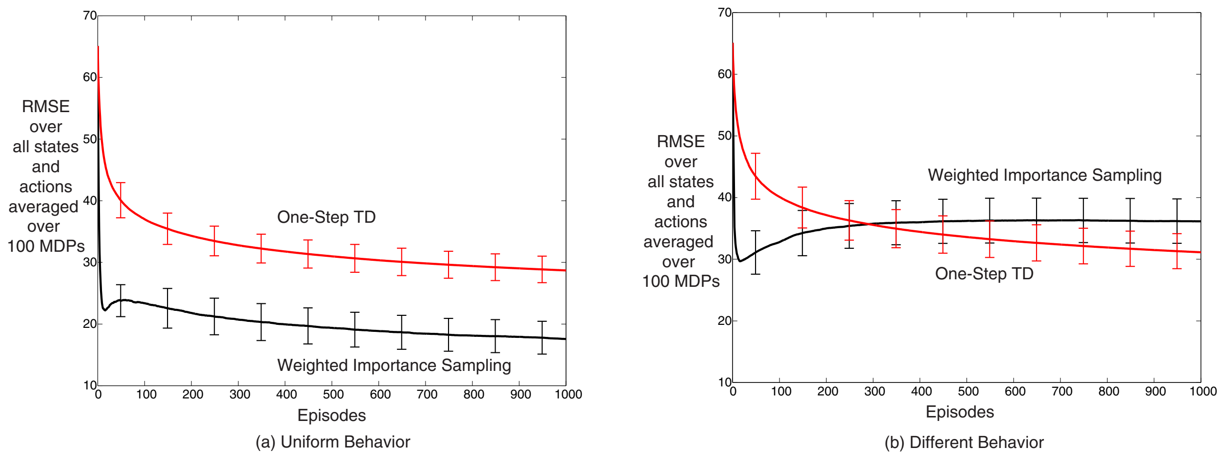

Figure 7.2. Comparison of One-Step TD and Monte Carlo Importance Sampling

Figure 7.1 contains a simple comparison between the one-step and the weighted importance sampling algorithms. The comparison was performed on the same set of 100 randomly generated MDPs used in the experiments reported in the previous chapter. For the one-step TD algorithm, the step size was $\alpha = 1/n(s,a)$, where $n(s,a)$

is the number of first visits to state-action pair (s, a). All the updates were done off-line, at the end of an episode. This setting eliminates the need to take into account the effect of the step-size parameter and the difference in speed due to on-line updating (which is known to be faster in practice). The left panel compares the algorithms for a uniformly random behavior policy. In this case, the one-step algorithm is significantly slower than the Monte Carlo algorithm, because it does not take advantage of the fact that long portions of trajectories are likely to occur under both the target and the behavior policy. The right panel contains the same comparison for the case of very different behavior and target policies (80%-20% vs. 20%-80%). In this case, the importance sampling algorithm has a very small advantage in the beginning of learning in terms of speed of reducing error, but in the long run the one-step algorithm is marginally faster.

7.2 Temporal-Difference Per-Decision Importance Sampling

The per-decision importance sampling estimator presented in the previous chapter can be implemented in a temporal-difference manner.

Algorithm 1 Online, Eligibility-Trace Version of Per-Decision Importance Sampling

1. Update the eligibility traces for all states:

$$\begin{aligned} e_t(s,a) &= e_{t-1}(s,a)\,\gamma\,\lambda\frac{\pi(s_t,a_t)}{b(s_t,a_t)}, \qquad \forall s,a \\ e_t(s,a) &= 1, \text{iff } t = t_m(s,a), \end{aligned}$$

   where $\lambda \in [0,1]$ is an eligibility trace decay factor.

2. Compute the TD error:

$$\delta_t = r_{t+1} + \gamma\frac{\pi(s_{t+1},a_{t+1})}{b(s_{t+1},a_{t+1})}Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$

3. Update the action-value function:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha\, e_t(s,a)\,\delta_t, \qquad \forall s,a$$

An eligibility-trace version of per-decision importance sampling is given by Algorithm 1. The algorithm maintains eligibility traces for each state-action pair in the usual manner of temporal-difference (TD) algorithms. The only difference is that here the eligibility trace is multiplied on each step not just by a decay-rate λ, but also by an importance sampling factor $\frac{\pi(s_t,a_t)}{b(s_t,a_t)}$. This factor corrects for the effect of the behavior policy. The algorithm shown uses *online updating*, meaning that the value estimates are updated on every time step. The *offline* version would make the same increments and decrements as shown, but only at the end of each episode. The changes are accumulated "on the side" until the end of the episode, the value estimates remaining unchanged until then. Under offline updating the algorithm can be made to exactly implement the per-decision importance sampling estimator Q^{PD} by choosing $\lambda = 1$ and $\alpha(s,a) = 1/n(s,a)$, where $n(s,a)$ is the number of times state-action pair s, a has previously been updated. Another choice for α causes the algorithm to exactly implement the corresponding weighted estimator, Q^{PDW}. The algorithm remains consistent under general λ and general decreasing α:

Theorem 6 *For any soft, stationary behavior policy b, and any $\lambda \in [0,1]$ that does not depend on the action a_t, Algorithm 1 with offline updating converges w.p.1 to Q^π, under the usual step-size conditions on α.*

Proof: The proof is structured in two stages. First, we consider the corrected truncated return corresponding to Q^{PD}. The corrected truncated return sums the rewards obtained from the environment for only n steps, then uses the current estimate of the value function to approximate the remaining value:

$$R_t^{(n)} = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} \prod_{l=t+1}^{t+k-1} \frac{\pi_l}{b_l} + \gamma^n Q(s_{t+n}, a_{t+n}) \prod_{l=t+1}^{t+n-1} \frac{\pi_l}{b_l}$$

We need to show that $R_t^{(n)} - Q^\pi$ is a contraction mapping in the max norm. If this is true for any n, then by applying the general convergence theorem, the n-step return converges to Q^π. Then any convex combination will also converge to Q^π. For example, any combination using a λ parameter in the style of eligibility traces will converge to Q^π.

Let $\Omega(s,a,k)$ denote the set of all possible trajectories of k state-action pairs starting with (s,a):

$$\Omega(s,a,k) = \{\langle s_0, a_0, s_1, a_1, \ldots s_{k-1}, a_{k-1}\rangle | s_0 = s, a_0 = a\},$$

and let ω denote any such trajectory: $\omega = \langle s_0, a_0, s_1, a_1, \ldots s_{k-1}, a_{k-1}\rangle$. Then the expected value of the corrected truncated return for state-action (s,a) can be expressed as follows:

$$\begin{aligned}
&E\Big\{R_t^{(n)} \mid s_t = s, a_t = a, b\Big\} = \\
&\quad \sum_{k=1}^{n} \sum_{\omega \in \Omega(s,a,k)} \Pr\{\omega \mid s_0 = s, a_0 = a, b\} \gamma^{k-1} r_k \prod_{l=1}^{k-1} \frac{\pi_l}{b_l} \\
&\quad + \sum_{\omega \in \Omega(s,a,n)} \Pr\{\omega \mid s_0 = s, a_0 = a, b\} \gamma^{n} Q(s_n, a_n) \prod_{l=1}^{n-1} \frac{\pi_l}{b_l} \\
&\quad = \sum_{k=1}^{n} \sum_{\omega \in \Omega(s,a,k)} \left(\prod_{l=1}^{k-1} p_{s_{l-1}s_l}^{a_{l-1}} b(s_l, a_l) \right) \gamma^{k-1} r_k \prod_{l=1}^{k-1} \frac{\pi_l}{b_l} \\
&\quad + \sum_{\omega \in \Omega(s,a,n)} \left(\prod_{l=1}^{n-1} p_{s_{l-1}s_l}^{a_{l-1}} b(s_l, a_l) \right) \gamma^{n} Q(s_n, a_n) \prod_{l=1}^{n-1} \frac{\pi_l}{b_l} \\
&\quad = \sum_{k=1}^{n} \gamma^{k-1} \sum_{\omega \in \Omega(s,a,k)} r_{s_{k-1}}^{a_{k-1}} \prod_{l=1}^{k-1} p_{s_{l-1}s_l}^{a_{l-1}} \pi(s_l, a_l) + \gamma^n \sum_{\omega \in \Omega(s,a,k)} Q(s_n, a_n) \prod_{l=1}^{k-1} p_{s_{l-1}s_l}^{a_{l-1}} \pi(s_l, a_l)
\end{aligned}$$

By applying the Bellman equation for Q^π iteratively n times, we obtain:

$$\begin{aligned}
Q^\pi(s,a) = &\sum_{k=1}^{n} \sum_{\omega \in \Omega(s,a,k)} \gamma^{k-1} r_{s_{k-1}}^{a_{k-1}} \prod_{l=1}^{k-1} p_{s_{l-1}s_l}^{a_{l-1}} \pi(s_l, a_l) \\
&+ \gamma^n \sum_{\omega \in \Omega(s,a,k)} Q^\pi(s_{t+n}, a_{t+n}) \prod_{l=1}^{k} p_{s_{l-1}s_l}^{a_{l-1}} \pi(s_l, a_l).
\end{aligned}$$

Therefore,

$$\max_{(s,a)} |E\Big\{R_t^{(n)} \mid b\Big\} - Q^\pi(s,a)| \leq \gamma^n \max_{(s,a)} |Q(s,a) - Q^\pi(s,a)|.$$

This means that any n-step return is a contraction in the max norm, and therefore, by applying theorem 1 of Jaakkola, Jordan and Singh (1994), it converges to Q^π.

In the second stage, we show that by applying the updates of Algorithm 1 for n successive steps, we perform the same update as by using the n-step return $R_t^{(n)}$. The eligibility trace for state-action pair (s,a) can be re-written as:

$$e_t(s,a) = \gamma^{t-t_m} \prod_{l=t_m+1}^{t} \frac{\pi_l}{b_l}.$$

We have:

$$\begin{aligned}
&\sum_{k=1}^{n} e_{t+k-1}(s,a)\delta_{t+k-1}(s,a) = \\
&\quad \sum_{k=1}^{n} \gamma^{k-1} \left(\prod_{l=t+1}^{t+k-1} \frac{\pi_l}{b_l}\right) \left(r_{t+k} + \gamma \frac{\pi(s_{t+k}, a_{t+k})}{b(s_{t+k}, a_{t+k})} Q(s_{t+k}, a_{t+k}) - Q(s_{t+k-1}, a_{t+k-1})\right) \\
&\quad = \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} \prod_{l=t+1}^{t+k-1} \frac{\pi_l}{b_l} + \gamma^n Q(s_{t+n}, a_{t+n}) \prod_{l=t+1}^{t+n-1} \frac{\pi_l}{b_l} - Q(s_t, a_t) \\
&\quad = R_t^{(n)} - Q(s_t, a_t).
\end{aligned}$$

Since our algorithm is equivalent to applying a convex mixture of n-step updates, and each update converges to correct Q-values, algorithm 1 converges to correct Q-values as well. $\diamond$

7.3 Tree Backup Algorithm

The importance sampling methods that we have discussed so far all use knowledge of the probabilities of different actions under the behavior policy in their updates. Therefore, they require that the behavior policy be known, Markov (purely a function

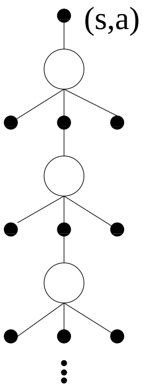

Figure 7.3. Backup diagram for the tree backup algorithm

of the current state) and explicitly represented as action probabilities. However, for agents that generate behavior using options, none of these may be true. The one-step TD-learning algorithm, on the other hand, estimates correct Q-values regardless of the behavior policy being followed. The behavior policy can be non-stationary, non-Markov and completely unknown, yet the computation is still correct, because the behavior policy is not used in the updates. the only requirement on the behavior policy is that it should be non-starving, i.e., that it never reaches a time after which a state-action pair is never visited again. In this section we introduce a new algorithm, which combines this advantage with the speed advantages of eligibility traces.

The main idea of the algorithm is illustrated in figure 7.3. At each step along a trajectory, there are several possible choices of action according to the target policy. As described in section 7.1, the one-step target combines the Q-value estimates for these actions, according to their probabilities of being taken under the target policy. At each step, the behavior policy chooses one of the actions, and for that action, one time step later, there is a new estimate of its Q-value, based on the reward received and the value of the next state. The tree backup algorithm then forms a new target, using the old Q-value estimates for the actions that were not taken, and the new estimate, for the value that was actually taken. This process can be iterated over many steps. If we iterate it over n steps, we obtain the *n-step tree-backup estimator*:

$$
\begin{aligned}
Q_n^{TB}(s,a) \stackrel{\text{def}}{=} \frac{1}{M} \sum_{m=1}^{M} \gamma^n Q(s_{t_m+n}, a_{t_m+n}) \prod_{i=t_m+1}^{t_m+n} \pi_i \\
+ \sum_{k=t_m+1}^{t_m+n} \gamma^{k-t_m+1} \prod_{i=t_m+1}^{k-1} \pi_i \left(r_k + \gamma \sum_{a \neq a_k} \pi(s_k, a) Q(s_k, a) \right)
\end{aligned} \tag{7.3}
$$

For $n = 1$, the tree backup estimator reduces to the familiar one-step TD estimator, TD(0).

Algorithm 2 Online, Eligibility-Traces Version of Tree Backup

1. Update the eligibility traces for all states:

$$
\begin{aligned}
e_t(s,a) &= e_{t-1}(s,a)\, \gamma\, \lambda\, \pi(s_t, a_t), \qquad \forall s, a \\
e_t(s,a) &= 1 \text{ iff } t = t_m(s,a)
\end{aligned}
$$

where $\lambda \in [0,1]$ is an eligibility trace decay parameter.

2. Compute the TD error:

$$
\delta_t = r_{t+1} + \gamma \sum_{a \in \mathcal{A}} \pi(s_{t+1}, a) Q(s_{t+1}, a) - Q(s_t, a_t)
$$

3. Update the action-value function:

$$
Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha\, e_t(s,a)\, \delta_t, \qquad \forall s, a
$$

The tree backup estimator also has a simple incremental implementation using eligibility traces. An online version of this implementation is given by Algorithm 7.3. In general, λ can be chosen as a function of the state s_t, but cannot depend on the action a_t. A choice of λ that is dependent on the state can have empirical advantages. For example, in the experiments reported in the next section, the eligibility traces were divided at each step by $\max_a \pi(s_t, a)$. This prevents the traces from decaying too quickly.

Theorem 7 *For any non-starving behavior policy, for any choice of $\lambda \in [0,1]$ that does not depend on the actions chosen at each state, the offline version of Algorithm 2 converges w.p.1 to Q^π, under the usual conditions on α.*

Proof: The proof is again in two stages. First we show that $E\Big\{Q_n^{TB}(s,a) \mid b\Big\} - Q^\pi$ is a contraction, in order to apply again theorem 1 of Jaakkola, Jordan and Singh (1994). We use a proof by induction.

Let Q be the current estimate of the value function. For $n = 1$:

$$\begin{aligned}
&\max_{(s,a)} |E\Big\{Q_1^{TB}(s,a) \mid b\Big\} - Q^\pi(s,a)| = \\
&\quad = \max_{(s,a)} |r_s^a + \gamma \sum_{s',a'} p_{ss'}^a \pi(s',a') Q(s',a') - r_s^a - \gamma \sum_{s',a'} p_{ss'}^a \pi(s',a') Q^\pi(s',a')| \\
&\quad \leq \gamma \max_{(s,a)} |Q(s,a) - Q^\pi(s,a)|.
\end{aligned}$$

For the induction step, we assume that

$$\max_{(s,a)} |E\Big\{Q_n^{TB}(s,a) \mid b\Big\} - Q^\pi(s,a)| \leq \gamma \max_{(s,a)} |Q_n^{TB}(s,a) - Q^\pi(s,a)|,$$

and we show the same holds for $Q_{n+1}^{TB}(s,a)$. We can re-write $Q_{n+1}^{TB}(s,a)$ as follows:

$$Q_{n+1}^{TB}(s,a) = r_{t+1} + \gamma \sum_{a' \in \mathcal{A}} \pi(s_{t+1},a') Q(s_{t+1},a')(1 - \mathcal{I}_{a'a_{t+1}}) + \mathcal{I}_{a'a_{t+1}} Q_n^{TB}(s_{t+1},a'),$$

where $\mathcal{I}_{a'a_{t+1}}$ is an indicator variable equal to 1 if $a' = a_{t+1}$ and 0 otherwise. Then we have:

$$\begin{aligned}
&\max_{(s,a)} |E\Big\{Q_{n+1}^{TB}(s,a) \mid b\Big\} - Q^\pi(s,a)| = \\
&\quad \max_{(s,a)} |r_s^a + \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a') E\Big\{(1 - \mathcal{I}_{a'a_{t+1}}) Q(s',a') + \mathcal{I}_{a'a_{t+1}} Q_n^{TB}(s',a') \mid b\Big\} \\
&\quad - r_s^a - \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a') Q^\pi(s',a')| \\
&\quad = \gamma \max_{(s,a)} | \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a') \\
&\qquad E\left\{(1 - \mathcal{I}_{a'a_{t+1}})(Q(s',a') - Q^\pi(s',a')) + \mathcal{I}_{a'a_{t+1}}(Q_n^{TB}(s',a') - Q^\pi(s',a')) | b\right\} |
\end{aligned}$$

$$\leq \gamma \max_{(s,a)} |Q(s,a) - Q^\pi(s,a)|.$$

By applying now theorem 1 of Jaakkola, Jordan and Singh (1994), we can conclude that any n-step return converges to the correct action value.

Since all the n-step returns converge to Q^π, any convex linear combination of n-step returns also converges to Q^π. In particular, we can use a fixed λ parameter, as is usually done in TD(λ), or even a λ parameter dependent on the state.

For the second part of the proof, we show that applying Algorithm 2 (with $\lambda = 1$) for n steps is equivalent to using $Q_n^{TB}(s,a)$. The eligibility trace for state-action pair (s,a) can be re-written as:

$$e_{t+k}(s,a) = \gamma^k \prod_{l=t+1}^{t+k} \pi(s_l, a_l)$$

By adding and subtracting the weighted action value, $\pi(s_{t+k}, a_{t+k})Q(s_{t+k}, a_{t+k})$ for the action taken on each step from the return, and regrouping, we have:

$$\begin{aligned} Q(s_t,a_t) \ &+ \sum_{k=1}^{n} \gamma^{k-1} \prod_{l=t+1}^{t+k-1} \pi(s_l,a_l) \left(r_{t+k} + \gamma \sum_{a \in \mathcal{A}} \pi(s_{t+k},a) Q(s_{t+k},a) - Q(s_{t+k-1}, a_{t+k-1}) \right) \\ &= Q(s_t,a_t) + \sum_{k=1}^{n} e_{t+k}(s_t,a_t)\delta_{t+k}, \end{aligned}$$

which concludes the proof. $\diamond$

7.4 Empirical Comparison

We compared all the algorithms discussed so far on the same benchmark of 100 different random MDPs discussed in chapter 6. The results are presented in figure 7.4. The left panel presents results using a uniformly random behavior policy. The right panel contains results for a behavior policy that is very different from the target policy (see chapter 6 for details of the experimental setup). The results for the one-step TD algorithm and the importance sampling algorithms have already been discussed in

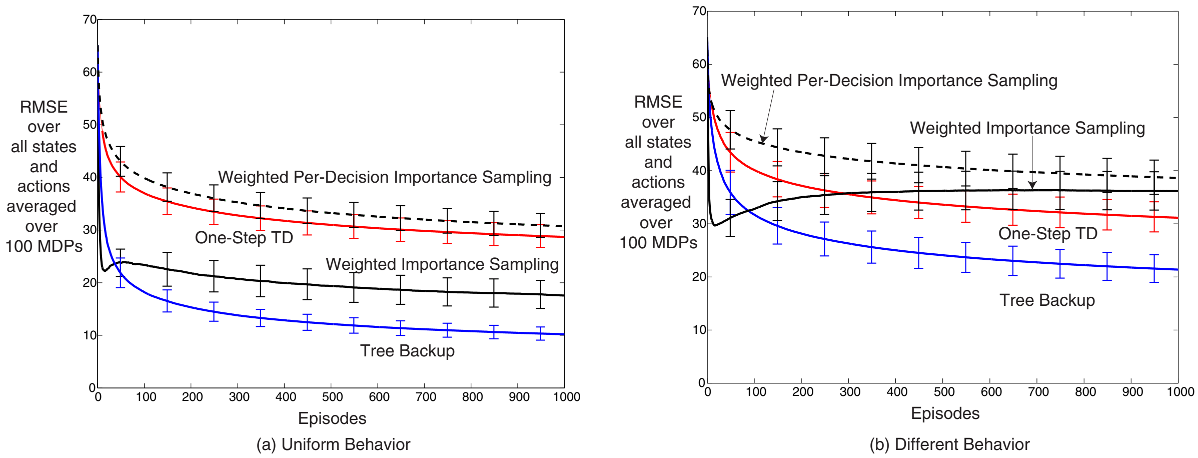

Figure 7.4. Comparison of all the off-policy learning algorithms on a testbed of 100 random MDPs.

the previous sections. The new results on these graphs are the root mean square error curves and standard deviations for the tree backup algorithm.

The tree backup algorithm is slower than weighted importance sampling in the beginning of learning, but it is significantly faster in the long run, as shown in the graphs. In the second case, the one-step TD method is also faster than the importance sampling methods in the long run. This is due to the fact that the trials are quite different from the ones one would obtain under the target policy. Therefore, a lot of the data is discarded by importance sampling, while the one-step and tree backup use it more efficiently. The weighted per-decision importance sampling algorithm comparable or worse to the other algorithms.

The tree backup estimator was uniformly the most efficient of all methods in the medium and long term, beaten only by weighted importance sampling by small amounts for small numbers of episodes. In summary, our results strongly favor the tree backup algorithm, because of its superior overall performance and because of its weaker requirements of the behavior policy.

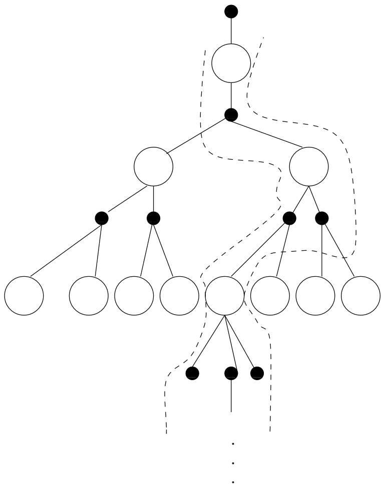

Figure 7.5. Full trajectory tree for an MDP

7.5 A Unified View of the Two Multi-Step TD Algorithms

In order to understand better the two multi-step TD algorithms presented so far, let us consider the full trajectory tree presented in Figure 7.5. The root of the tree is a state-action pair, and the tree contains all the possible states and actions at each point. States are represented with hollow circles, and actions with filled circles. One trajectory through the tree can be obtained by sampling the states at each branching according to the environment's transition probabilities, and by sampling actions according to the behavior policy.

Both the multi-step algorithms do backups along such trajectories. The per-decision importance sampling algorithm uses the actual rewards obtained during the trajectory. Because the sampling at each action ramification is done according to the behavior policy probabilities, the importance sampling correction is necessary to ensure correct estimates. The tree backup algorithm considers all possible actions at each step, not just the one taken. It backs up values according to a cut like the one represented with the dotted line in figure 7.5. Because all action choices are considered in the backup, the convergence is guaranteed for any behavior policy that is non-starving (i.e. takes every action in every state infinitely often).

This interpretation suggests that the two algorithms can be combined, without losing the convergence guarantees. If at a given state, the behavior policy is Markov and it is known, we can use per-decision importance sampling. If the behavior is unknown and/or non-Markov, we can use the tree backup algorithm. The mixture could ensure faster and more stable error reduction than either algorithm alone. We have not yet explored this idea empirically.

The two multi-step TD algorithms presented here also have an interesting relationship to multi-step Q-learning. In the control case, both algorithms cut the eligibility trace whenever an exploratory action is taken. Their updates are equivalent to Watkins's Q(λ) algorithm. Of course, no convergence results have been established for eligibility trace algorithms in the control case. Our hope is that we will be able to extend the convergence results presented in the previous sections to the control case.

7.6 Applying Multi-Step Off-Policy Learning to Intra-Option Learning

In chapter 5, we presented intra-option learning algorithms, which are a form of off-policy learning for acquiring the values and models of options. In this section we present an application of the ideas presented above to learning about the models of options.

The eligibility trace update used by the tree backup algorithm can be applied immediately in this case. The only change is that the eligibility traces have to take into account the probability of the option terminating in each state (as in Sutton, 1995). Algorithm 3 presents an intra-option mode learning algorithm that uses eligibility traces.

In order to illustrate the algorithm, we use the same gridworld used in the experiments with one-step intra-option learning (see Figure 4.1). Experience is generated by selecting randomly in each state among the two possible options and four possible actions, with no goal state.

Algorithm 3 Eligibility Traces Intra-Option Model Learning

1. Initialize $r^o_s(a)$ and $p^o_{ss'}(a)$ to arbitrary values.
2. Initialize traces for all options: $e_o(s,a) = 0, \forall s, a$.
3. On every transition s_t, a_t, r_t, s_{t+1}:
   (a) Update the eligibility traces for all options and all state-action pairs:
   $$\begin{aligned} e_o(s,a) &\leftarrow e_o(s,a)\gamma\lambda(1-\beta_o(s_t))\pi_o(s_t,a_t), \forall s, a \\ e_o(s_t,a_t) &\leftarrow e_o(s_t,a_t) + 1 \end{aligned}$$
   (b) Update the models, for all s, a, o:
   $$\begin{aligned} r^o_s(a) &\leftarrow r^o_s(a) + \alpha e_o(s,a)(r_t + (1-\beta(s_{t+1}))\gamma r^o_{s_{t+1}} - r^o_{s_t}(a_t)) \\ p^o_{ss'}(a) &\leftarrow p^o_{ss'}(a) + \alpha e_o(s,a)((1-\beta(s_{t+1}))\gamma p^o_{s_{t+1}} + \beta(s_{t+1})\gamma\delta_{s's_{t+1}} - p^o_{s_ts'}(a_t)), \end{aligned}$$
   where $\delta_{s's_{t+1}}$ is an indicator function that has value 1 if $s' = s_{t+1}$ and 0 otherwise.

We compared the results of the algorithm presented above with the SMDP algorithm given by equations (5.1) and (5.2). For the SMDP method, the step-size parameter was varied so that the model estimates were sample averages, which should give fastest learning. The results of this method are labeled "SMDP optimized" on the graphs. We also looked at results using a fixed learning rate. In this case and for the intra-option methods we tried step-size values of $\alpha = \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$, and $\frac{1}{16}$, and picked the best value for each method. For the intra-option updates, we used two values of λ: $\lambda = 0$ (which gives the algorithm we studied in chapter 5) and $\lambda = 1$. Figure 7.6 shows the learning curves for all three methods, using the best α values, when a fixed alpha was used.

All intra-option algorithms were faster than SMDP methods. The algorithm with $\lambda = 1$ is the fastest to converge, but asymptotes to a higher error. Note that in the case of $\lambda = 1$, the intra-option algorithm is performing similar updates to the SMDP algorithm, but at the same time it is adjusting more options. In our illustration, at

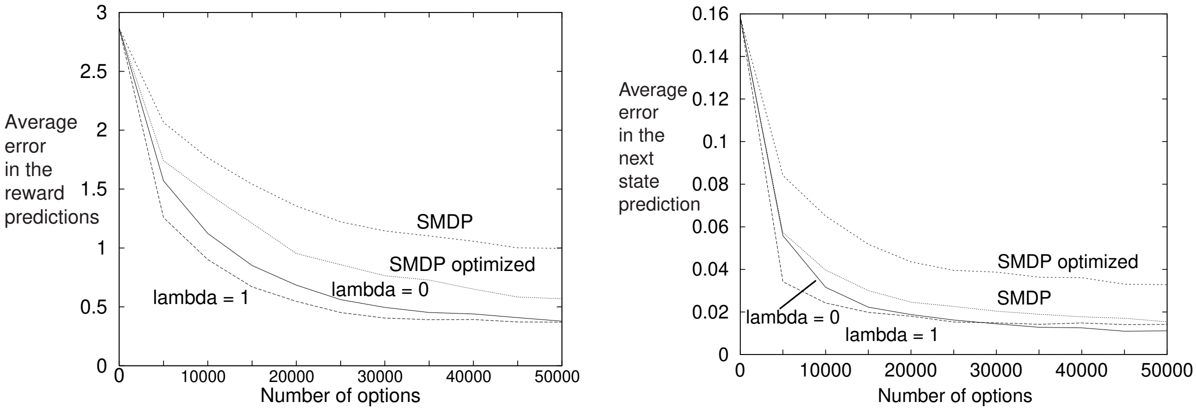

Figure 7.6. Error curves for the reward predictions (left panel) and next-state predictions (right panel) for the SMDP, one-step intra-option learning, and intra-option learning with tree backup eligibility traces.

most two options can be updated on every time step, but we anticipate that the effect would be more pronounced if more options were available for parallel updates.

7.7 Conclusions

In this chapter we presented two novel temporal-difference algorithms for performing off-policy learning with eligibility traces. We proved that these algorithms converge to correct action values under offline updating in the tabular case. These are the first convergence results for off-policy learning with eligibility traces. The tree backup algorithm converges correctly for non-stationary and non-Markov behavior policies, as long as they are non-starving. this algorithm also performed best in our experiments. We have shown how the tree backup algorithm can be applied for learning the models of options in an intra-option manner. Using this idea produces faster learning that one-step intra-option methods.

In the future, it would be interesting to study the theoretical properties of these algorithms, such as the bias-variance trade-offs, and the relationship of these algorithms to control learning methods, such as Q(λ). More empirical experience with these algorithms would also be useful in understanding their properties.

CHAPTER 8

ADAPTING OPTIONS

So far we have assumed that the options are given, and we have focused on methods for learning and planning with options. Given a set of options, one can use such methods to find the optimal value function of the SMDP defined by the options. This value function is typically lower than the optimal value function of the original MDP problem.

The next step we consider now is to improve the existing options. For example, if the option is `open-the-door`, then it is natural to adapt its policy over time to make it more effective and efficient in opening the door, which may make it more generally useful. In this chapter we address the issue of adapting a given set of options to provide a better solution for the task at hand. We present methods for changing the termination conditions and the policies of the options.

8.1 Interruption

Suppose we have determined the option-value function $Q^\mu(s, o)$ for some policy μ and for all state–options pairs s, o that could be encountered while following μ. This function tells us how well we do while following μ committing irrevocably to each option, but it can also be used to re-evaluate our commitment on each step. Suppose at time t we are in the midst of executing option o. If o is Markov in s, then we can compare the value of continuing with o, which is $Q^\mu(s_t, o)$, to the value of terminating o and selecting a new option according to μ, which is $V^\mu(s) = \sum_{o'} \mu(s, o')Q^\mu(s, o')$. If the latter is more highly valued, then why not terminate o and allow the switch?

This new way of behaving is indeed better, as shown below. But this is a change in the termination condition of o and thus requires stepping outside the existing set of options, outside the SMDP.

We can characterize the new way of behaving as following a policy μ' that is the same as the original one, but over new options, i.e. $\mu'(s, o') = \mu(s, o)$, for all $s \in \mathcal{S}$. Each new option o' is the same as the corresponding old option o except that it terminates whenever termination seems better than continuing according to Q^μ. We call such a μ' an *interrupted policy*. We will now state a general theorem, which extends the case described above, in that options do not have to be Markov, and interruption is optional at each state where it could be done. This lifts the requirement that Q^μ be completely known.

Theorem 8 (Interruption) *For any MDP, any set of options $\mathcal{O}$, and any Markov policy $\mu : \mathcal{S} \times \mathcal{O} \mapsto [0, 1]$, define a new set of options, $\mathcal{O}'$, with a one-to-one mapping between the two option sets as follows: for every $o \in \mathcal{O}$ we define a corresponding $o' \in \mathcal{O}'$, where $o'(h, \cdot) = o(h, \cdot)$ except that for any history h in which $Q^\mu(h, o) < V^\mu(s)$, where s is the final state of h, we may choose to set $o'(h, \tau) = 1$. Any histories whose termination conditions are changed in this way are called* interrupted histories*. Let μ' be the policy over o' corresponding to μ: $\mu'(s, o') = \mu(s, o)$, where o is the option in $\mathcal{O}$ corresponding to o', for all $s \in \mathcal{S}$. Then:*

1. *$V^{\mu'}(s) \geq V^\mu(s)$ for all $s \in \mathcal{S}$.*

2. *If from state $s \in \mathcal{S}$ there is a non-zero probability of encountering an interrupted history upon initiating μ' in s, then $V^{\mu'}(s) > V^\mu(s)$.*

Proof: The idea is to show that, for an arbitrary start state s, executing the option given by the interrupted policy μ' and then following policy μ thereafter is no worse than always following policy μ. In other words, we show that the following inequality holds:

$$\sum_{o'} \mu'(s,o')[r_s^{o'} + \sum_{s'} p_{ss'}^{o'} V^\mu(s')] \geq V^\mu(s) = \sum_o \mu(s,o)[r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')]. \tag{8.1}$$

If this is true, then we can use it to expand the left-hand side, repeatedly replacing every occurrence of $V^\mu(x)$ on the left by the corresponding $\sum_{o'} \mu'(x,o')[r_x^{o'} + \sum_{x'} p_{xx'}^{o'} V^\mu(x')]$. In the limit, the left-hand side becomes $V^{\mu'}$, proving that $V^{\mu'} \geq V^\mu$. Because $\mu'(s,o') = \mu(s,o) \forall s \in \mathcal{S}$, we need to show that

$$r_s^{o'} + \sum_{s'} p_{ss'}^{o'} V^\mu(s') \geq r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s'). \tag{8.2}$$

Let Γ denote the set of all interrupted histories: $\Gamma = \{h \in \mathcal{H} : o(h,\tau) \neq o'(h,\tau)\}$. Then, the left hand side of (8.2) can be re-written as:

$$E\left\{r + \gamma^k V^\mu(s') \;\middle|\; \mathcal{E}(o',s), h_{ss'} \notin \Gamma\right\} + E\left\{r + \gamma^k V^\mu(s') \;\middle|\; \mathcal{E}(o',s), h_{ss'} \in \Gamma\right\}, \tag{8.3}$$

where s', r, and k are the next state, cumulative reward, and number of elapsed steps following option o from s ($h_{ss'}$ is the history from s to s'). Trajectories that end because of encountering a history $h_{ss'} \notin \Gamma$ never encounter a history in Γ, and therefore also occur with the same probability and expected reward upon executing option o in state s. Therefore, we can re-write the second term of (8.3) as:

$$E\left\{o(h_{ss'},\tau)[r + \gamma^k V^\mu(s')] + (1 - o(h_{ss'},\tau))[r + \gamma^k Q^\mu(h_{ss'},o)] \;\middle|\; \mathcal{E}(o',s), h_{ss'} \in \Gamma\right\}.$$

This proves (8.1) because for all $h_{ss'} \in \Gamma$, $Q^\mu_{\mathcal{O}}(h_{ss'},o) \leq V^\mu(s')$. Note that strict inequality holds in (8.2) if $Q^\mu_{\mathcal{O}}(h_{ss'},o) < V^\mu(s')$ for at least one history $h_{ss'} \in \Gamma$ that ends a trajectory generated by o' with non-zero probability. $\diamond$

As one application of this result, consider the case in which μ is an optimal policy for a given set of Markov options $\mathcal{O}$. By planning or learning we can determine the SMDP optimal value function $V^*_{\mathcal{O}}$ and the optimal policy $\mu^*_{\mathcal{O}}$ that achieves it. This is indeed the best that can be done without changing $\mathcal{O}$, that is, in the SMDP defined by $\mathcal{O}$, but less than the best possible achievable in the MDP, which is V^*. But of course we typically do not wish to work directly in the primitive options because of the computational expense. The interruption theorem gives us a way of improving over $\mu^*_{\mathcal{O}}$ with very little additional computational expense, by stepping outside $\mathcal{O}$. The only additional expense is the cost of checking (on each time step) if a better option exists, which is negligible compared to the combinatorial process of computing $Q^*_{\mathcal{O}}$. Kaelbling (1993b) and Dietterich (1998) demonstrated such a performance improvement by interrupting temporally extended actions based on a value function found at a higher level, in different settings.

8.2 Examples of Interruption

Figure 8.1 shows a simple example of applying the interruption theorem. The task is to navigate from a start location to a goal location within a continuous two-dimensional state space. The actions are movements of length 0.01 in any direction from the current state. Rather than work with these low-level actions, infinite in number, we introduce seven landmark locations in the space. For each landmark we define a controller that takes us to the landmark in a direct path. Each controller is only applicable within a limited range of states, in this case within a certain distance of the corresponding landmark. Each controller then defines a flat option: the circular region around the controller's landmark is the option's initiation set, the controller itself is the policy, and the arrival at the target landmark is the termination condition. We denote the set of seven landmark options by $\mathcal{O}$. Any action within 0.01 of the

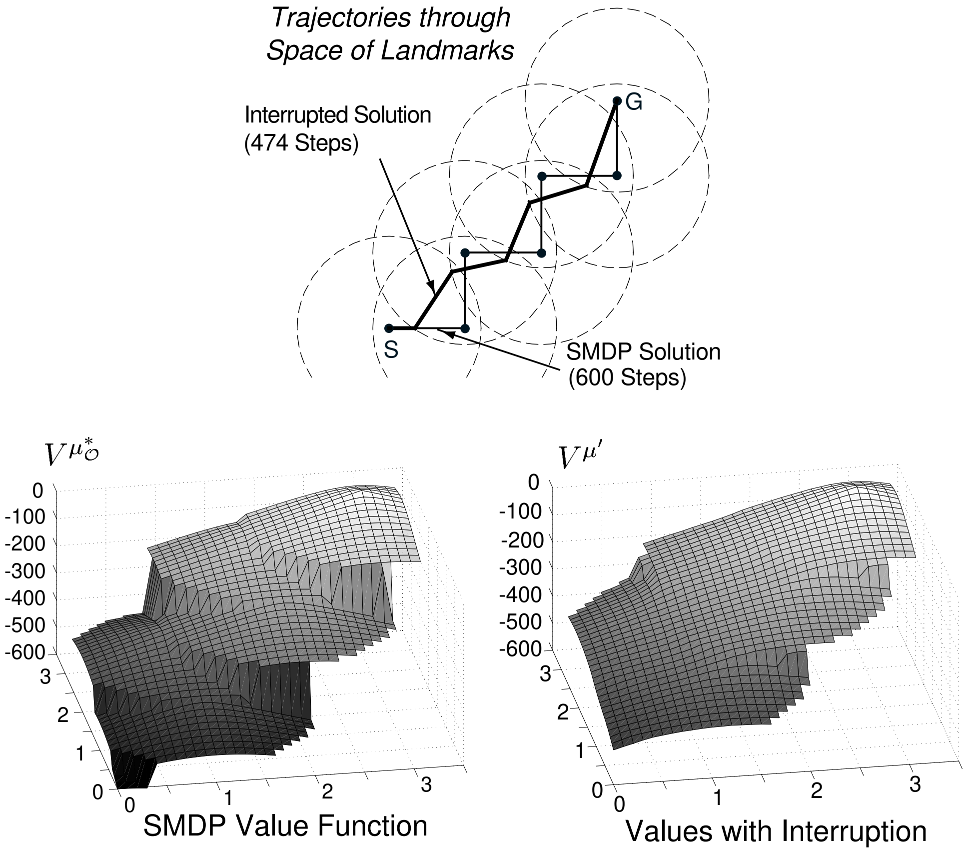

Figure 8.1. Interruption in navigating with landmark-directed controllers. The task (up) is to navigate from S to G in minimum time using options based on controllers that run each to one of seven landmarks (the black dots). The circles show the region around each landmark within which the controllers operate. The thin line shows the optimal behavior that uses only these controllers run to termination, and the thick line shows the corresponding interrupted behavior, which cuts the corners. The lower panels show the state-value functions for the SMDP-optimal and interrupted policies. Note that the latter is greater.

goal location transitions to the terminal state, $\gamma = 1$, and the reward is -1 on all transitions, which makes this a minimum-time task.

One of the landmarks coincides with the goal, so it is possible to reach the goal while picking only from $\mathcal{O}$. The optimal policy within $\Pi(\mathcal{O})$ runs from landmark to landmark, as shown by the thin line in Figure 8.1. This is the optimal solution to the SMDP defined by $\mathcal{O}$ and is indeed the best that one can do while picking only from these options. But of course one can do better if the options are not followed all the way to each landmark. The trajectory shown by the thick line in Figure 8.1 cuts the corners and is shorter. This is the interrupted policy with respect to the SMDP-optimal policy. The interrupted policy takes 474 steps from start to goal which, while not as good as the optimal policy in primitive actions (425 steps), is much better than the SMDP-optimal policy, which takes 600 steps. The state-value functions, $V^{\mu^*_\mathcal{O}}$ and $V^{\mu'}$ for the two policies are also shown in Figure 8.1.

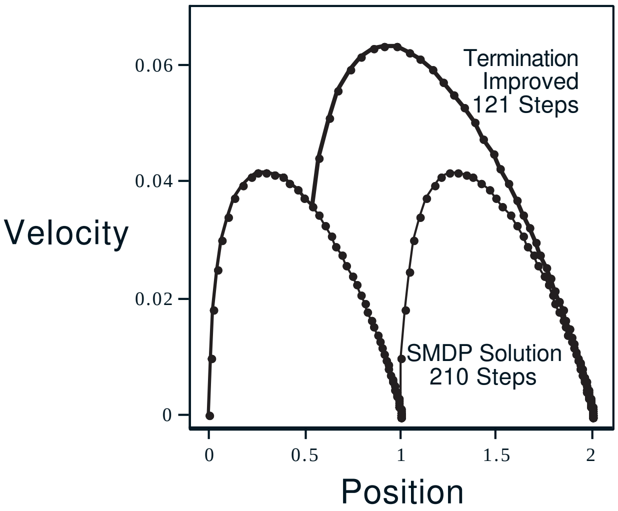

Figure 8.2. Phase-space plot of the SMDP and interrupted policies in a simple dynamical task. The system is a mass moving in one dimension: $x_{t+1} = x_t + \dot{x}_{t+1}$, $\dot{x}_{t+1} = \dot{x}_t + a_t - 0.175\dot{x}_t$ where x_t is the position, $\dot{x}_t$ the velocity, 0.175 a coefficient of friction, and the action a_t an applied force. This continuous system is controlled at a discrete time scale of 0.001. Two controllers are provided as options, one that drives the position to $x^* = 1$ and the other to $x^* = 2$. Whichever option is being followed at time t, its target position x^* determines the action taken, according to $a_t = 0.01(x^* - x_t)$.

Figure 8.2 shows results for an example using controllers/options with dynamics. The task here is to move a mass along one dimension from rest at position 0 to at rest at position 2, again in minimum time. There is no option that takes the system all the way from 0 to 2, but we do have an option that takes it from 0 to 1, both at rest, and another option that takes it from any position greater than 0.5 to 2. Both options control the system precisely to its target position and to zero velocity, terminating only when both of these are correct to within $\epsilon = 0.0001$. Using just these options, the best that can be done is to first move precisely to 1 at rest using the first option, then re-accelerate and move to 2 at rest using the second option. This SMDP-optimal solution is much slower than the corresponding interrupted policy, as shown in Figure 8.2. Because of the need to slow down to near-zero velocity at 1, it takes over 200 time steps, whereas the improved policy takes only 121 time steps.

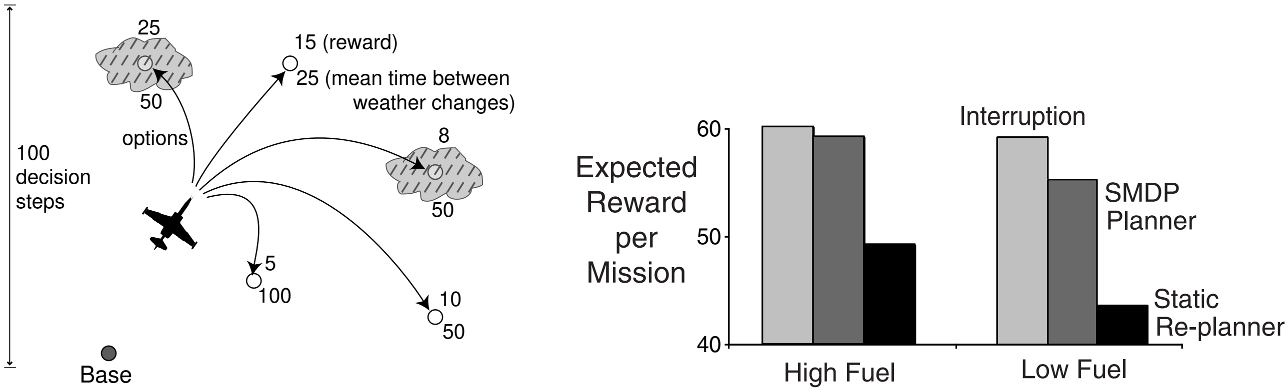

Figure 8.3. The mission planning task and the performance of policies constructed by SMDP methods, interruption of the SMDP policy, and an optimal static re-planner that does not take into account possible changes in weather conditions.

Figure 8.3 presents a more complex, mission planning task (Sutton, Singh, Precup & Ravindran, 1999). A mission is a flight from base to observe as many sites as possible, from a given set of sites, and return to base without running out of fuel. The local weather at each site flips from cloudy to clear according to independent Poisson processes. If the sky at a given site is cloudy when the plane gets there, no observation is made and the reward is 0. If the sky is clear, the plane gets a reward, according to the importance of the site. The positions, rewards, and mean time between two weather changes for each site are given in the left panel of Figure 8.3. The plane has a limited amount of fuel, and it consumes one unit of fuel during each time tick. If the fuel runs out before reaching the base, the plane crashes and receives a reward of -100.

The primitive actions are tiny movements in any direction (there is no inertia). The state of the system is described by several variables: the current position of the plane, the fuel level, the sites that have been observed so far, and the current weather at each of the remaining sites. The state-action space has approximately 24.3 billion elements (assuming 100 discretization levels of the continuous variables) and is intractable by normal dynamic programming methods. We introduced options that can take the plane to each of the sites (including the base), from any position in the input space. The resulting SMDP has only 874,800 elements and it is feasible to exactly determine $V^*_{\mathcal{O}}(s')$ for all sites s'. From this solution and the model of the options, we can determine $Q^*_{\mathcal{O}}(s,o) = r^o_s + \sum_{s'} p^o_{ss'} V^*_{\mathcal{O}}(s')$ for any option o and any state s in the whole space.

We performed asynchronous value iteration using the options in order to compute the optimal option-value function, and then used the interruption approach based on the values computed. The policies obtained by both approaches were compared to the results of a static planner, which exhaustively searches for the best tour assuming the weather does not change, and then re-plans whenever the weather does change. The graph in Figure 8.3 shows the reward obtained by each of these methods, averaged over 100 independent simulated missions. The policy obtained by interruption performs significantly better than the SMDP policy, which in turn is significantly better than the static planner.

8.3 Termination Iteration

Interruption is a first step in adapting the options for the task at hand. It is computationally inexpensive and easy to apply while acting, without necessarily making a permanent change of the options. In this section we present a new algorithm for altering the termination conditions of the options.

The planning version of this algorithm, which we call *termination iteration*, is presented below. The main idea is to alternate interruption of options with *merging*, which eliminates reset actions whenever the same option would be chosen again. The goal of merging is to prevent the options from becoming too short, due to interruption.

Algorithm 4 Termination Iteration

1. **Initialization Step:** The initial set of options, $\mathcal{O}_0$, is evaluated by computing the models of the options and then doing SMDP planning. This yields the initial optimal Q-value function, $Q_{\mathcal{O}_0}$, and the initial optimal policy $\mu^*_{\mathcal{O}_0}$.

2. **Iterate** the following three steps on the current set of options $\mathcal{O}_i$:

   (a) Interruption: all the options are interrupted after histories after which it is not optimal to continue executing them:

   $$o(h, \tau) \leftarrow 1, \forall o \text{ such that } Q^*_{\mathcal{O}_i}(h, o) < V^*_{\mathcal{O}_i}(s),$$

   where s is the last state of h. This yields a new set of options $\mathcal{O}'_i$.

   (b) Evaluation: the new termination conditions and associated options $\mathcal{O}'_i$ are evaluated by recomputing models and SMDP planning, yielding $Q^*_{\mathcal{O}'_i}$ and $\mu^*_{\mathcal{O}'_i}$.

   (c) Merging: the options are modified to obtain the "longest" duration options that do not change the behavior of the current optimal policy $\mu^*_{\mathcal{O}'_i}$:

   $$o(h, \tau) \leftarrow 0, \forall o \text{ such that } Q^*_{\mathcal{O}'_i}(h, o) = V^*_{\mathcal{O}'_i}(s).$$

   This yields a new set of options $\mathcal{O}_{i+1}$ whose Q-values are exactly $Q^*_{\mathcal{O}'_i}$.

3. **Stopping Condition:** $V^*_{\mathcal{O}_{i+1}} = V^*_{\mathcal{O}_i}$.

For any set of options $\mathcal{O}$, there exists at least one setting of termination probabilities, which we call the *optimal termination probabilities w.r.t* $\mathcal{O}$, such that the associated optimal value for every state is at least as large as the optimal value for that state under any setting of termination probabilities. An option set derived from $\mathcal{O}$ that has optimal termination probabilities is denoted $\mathcal{O}^*$, and the associated optimal value function $V^*_{\mathcal{O}^*}$ and optimal policy $\mu^*_{\mathcal{O}^*}$.

Finding some set of optimal termination probabilities is trivial. For instance, we can define the options such that they always terminate after the first primitive action. In this case, the value function obtained is the value function of the underlying MDP, and is the best one can do. However, the computational advantage of options comes from the options whose expected duration is longer than that of the usual primitive options. Therefore, our goal is to find such a set of optimal termination probabilities with the longest expected durations. The termination iteration algorithm fulfills this goal, as we show below.

Theorem 9 (Convergence of Termination Iteration) *Let $\{\mathcal{O}_i\}$ be the sequence of option sets produced by the algorithm termination iteration algorithm 4. If for some i, $V^*_{\mathcal{O}_{i+1}} = V^*_{\mathcal{O}_i}$, then $V^*_{\mathcal{O}_{i+1}} = V^*_{\mathcal{A}_{\mathcal{O}_0}}$, i.e., the option set $\mathcal{O}_{i+1}$ has optimal terminal probabilities.*

Proof: We prove the theorem by contradiction. If the algorithm has reached the stopping condition, then for all histories h and all options $o \in \mathcal{O}_{i+1}$, either $Q^*_{\mathcal{O}_{i+1}}(h, o) = V^*_{\mathcal{O}_{i+1}}(s)$ or $o(h, \tau) = 1$, where h ends in s. This is because the termination conditions are changed based on the value function and, and the value function does not change from option set $\mathcal{O}_i$ to option set $\mathcal{O}_{i+1}$.

By the interruption theorem 8, the value function of a set of options is improved every time when a change in termination probabilities is due to interruption. The merging step of the algorithm does not change the value function at all. In order to see this, consider the value function $V^*_{\mathcal{O}'_i}$ and $V^*_{\mathcal{O}_{i+1}}$. By construction, the termination probabilities of $\mathcal{O}_{i+1}$ are less than or equal to those of $\mathcal{O}'_i$. Therefore, $V^*_{\mathcal{O}'_i} \geq V^*_{\mathcal{O}_{i+1}}$, because shorter duration policies can always be pieced together to do at least as well as longer policies. On the other hand, let $\mu^*_{\mathcal{O}'_i}$ be the optimal policy choosing from options in $\mathcal{O}'_i$, and $\mu_{\mathcal{O}_{i+1}}$ a policy defined over options in $\mathcal{O}_{i+1}$, such that $\mu_{\mathcal{O}_{i+1}}(s, o') = \mu^*_{\mathcal{O}'_i}(s, o)$, where o' is the option corresponding to o. Given the way in which $\mathcal{O}_{i+1}$ was constructed, $V^*_{\mathcal{O}'_i} = V^{\mu_{\mathcal{O}_{i+1}}} \leq V^*_{\mathcal{O}_{i+1}}$. Therefore, $V^*_{\mathcal{O}'_i} = V^*_{\mathcal{O}_{i+1}}$. This proves that the value function is improved after every iteration of the algorithm.

Because only interruption improves the value function, we have to show that the stopping condition implies that no further interruption will yield a better optimal value function. Let us assume that we increase $o(h)$ to 1 for some option $o \in \mathcal{O}_{i+1}$ and history h, obtaining a new option o'. Assume that this operation increases the optimal value for some state s (which is the start state of h). Let $\mu^*_{\mathcal{O}_{i+1}}$ be the optimal policy obtained just before making this last change, and $V^*_{\mathcal{O}_{i+1}}$ be the corresponding optimal value function. By executing o' in state s and thereafter following $\mu^*_{\mathcal{O}_{i+1}}$, the value of state s increases:

$$E\left\{r_s^{o'} + \gamma^k V^*_{\mathcal{O}_{i+1}}(s') \;\middle|\; \mathcal{E}(o', s)\right\} > V^*_{\mathcal{O}_{i+1}}(s) \geq Q^*_{\mathcal{O}_{i+1}}(s, o).$$

Let us consider an arbitrary state x and divide the trajectories obtained by executing o' from state s into those that terminate at x and those that continue after x. Then we have:

$$\begin{aligned}
&E\left\{r_s^{o'} + \gamma^k V^*_{\mathcal{O}_{i+1}}(s') \;\middle|\; \mathcal{E}(o', s), s' \neq x\right\} + E\left\{r_s^{o'} + \gamma^k V^*_{\mathcal{O}_{i+1}}(s') \;\middle|\; \mathcal{E}(o', s), s' = x\right\} \\
&\quad > Q^*_{\mathcal{O}_{i+1}}(s, o) = E\left\{r_s^{o} + \gamma^k V^*_{\mathcal{O}_{i+1}}(s') \;\middle|\; \mathcal{E}(o, s), s' \neq x\right\} + \\
&\qquad E\left\{r_s^{o} + \gamma^k [o(h_{sx}, \tau) V^*_{\mathcal{O}_{i+1}}(s') + (1 - o(h_{sx}, \tau)) Q^*_{\mathcal{O}_{i+1}}(h_{sx}, o) \;\middle|\; \mathcal{E}(o, s), s' = x\right\}
\end{aligned}$$

This must mean that both $V^*_{\mathcal{O}_{i+1}}(h_{sx}) > Q^*_{\mathcal{O}_{i+1}}(h_{sx}, o)$ and $o(h_{sx}, \tau) < 1$, which contradicts the stopping condition. $\diamond$

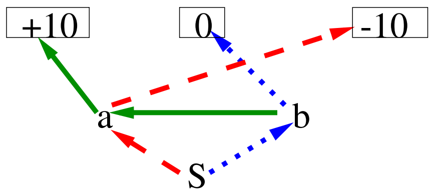

Figure 8.4. Simple MDP in which adding more options can decrease the quality of the interrupted policy.

This theorem shows that termination iteration will give a solution at least as good as the optimal SMDP solution for the given set of options. However, there is no guarantee regarding the amount of improvement obtained in this way. For instance, consider the example in Figure 8.3. The starting state is S and there are three deterministic Markov options, marked respectively with solid line, dashed line and dotted line in Figure 8.3. The reward in -1 per time step, there is no discounting, and the values of the terminal states are as marked in the figure. Let us consider the set $\mathcal{O}_1$ formed by the solid line and dashed line options, as well as the set $\mathcal{O}_2$ formed by the dashed line and dotted line options. The first set has a higher SMDP value function for all the states. However, when termination iteration is performed for this set, the number of steps necessary to get from state s to the final $+10$ reward is 3, so the value of s is 7. When doing termination iteration on $\mathcal{O}_2$, only two steps are necessary to get to the $+10$ reward. So even though $V^*_{\mathcal{O}_1} > V^*_{\mathcal{O}_2}$, for the termination improved sets, this inequality is reversed. This phenomenon is due to the existence, in the set $\mathcal{O}_2$, of an option which has a good beginning part but a catastrophic finish. Termination iteration helps to retain only the good part, which helps to obtain a better value function in the end.

8.4 Illustration of Termination Iteration

As an illustration, we applied the termination iteration algorithm 4 to the rooms environment presented in Figure 4.1. The goal state is at $G2$. We use only hallway options, defined as in our previous experiments. The algorithm converged after three iterations.

Figure 8.5 shows the termination conditions obtained by applying termination iteration to the options in the south-east room. Because the goal is at position $G2$, the optimal solution with the original options is to run back and forth between the two hallways $G1$ and $G3$, and occasionally stumble upon the goal. By applying termination iteration, the termination conditions of the options are changed as shown by the small squares in Figure 8.5. If we consider the trajectories that the option going to $G1$ generates‘, we can see that these trajectories would be terminated when they start getting farther from the goal state. A similar change happens to the trajectories generated by the option for going to $G3$. The change of the termination conditions

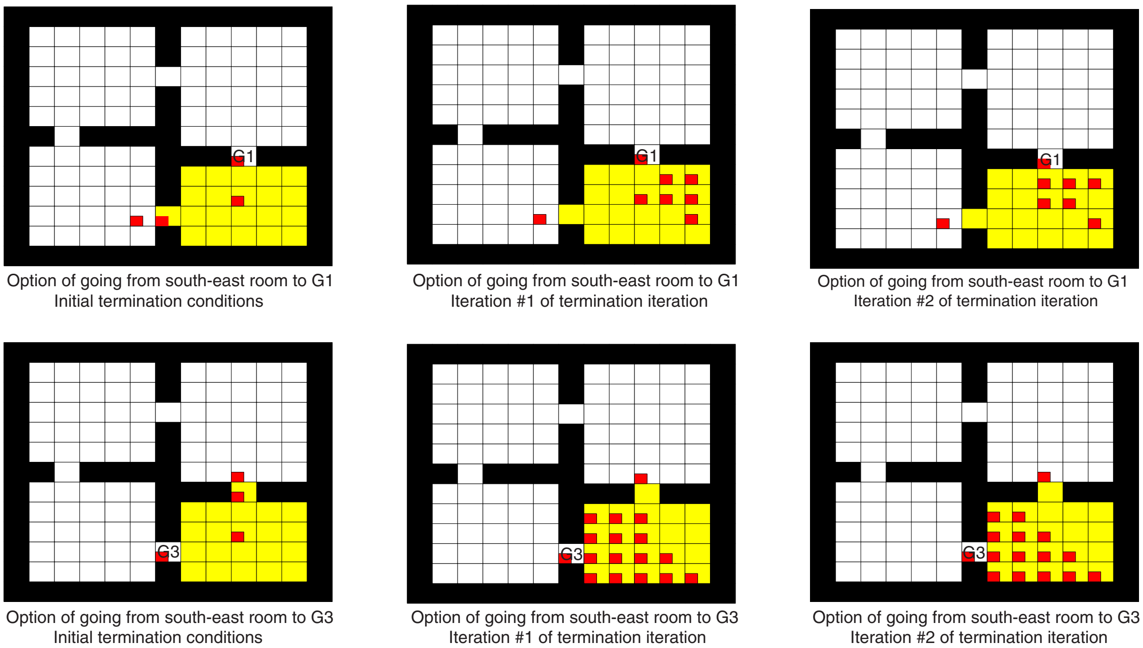

Figure 8.5. The result of termination iteration in the rooms environment, in the room containing the goal. The states in which the options terminate immediately are marked by small squares. The initiation sets of the options are shaded. The two options available in the room are terminated immediately if the option would take the agent away from the goal state.

decreases the amplitude of the agent's oscillations between the two hallways, and the goal state is reached significantly faster when planning only with the hallway options.

Figure 8.6 shows the termination conditions obtained in the north-west room. Both options are terminated in all states in which they are not optimal. In this case, the option to go to state $G5$ is suboptimal in most states, because it leads away from the goal. Therefore, this option is basically cut into primitive actions. The option to go to the hallway $G4$, which is part of the optimal solution for most states, is preserved almost intact.

Of course, this is just the most straightforward way to apply termination iteration. In practice, the algorithm can be applied more incrementally. For instance, instead of doing full planning every time to evaluate the set of options, one could apply SMDP

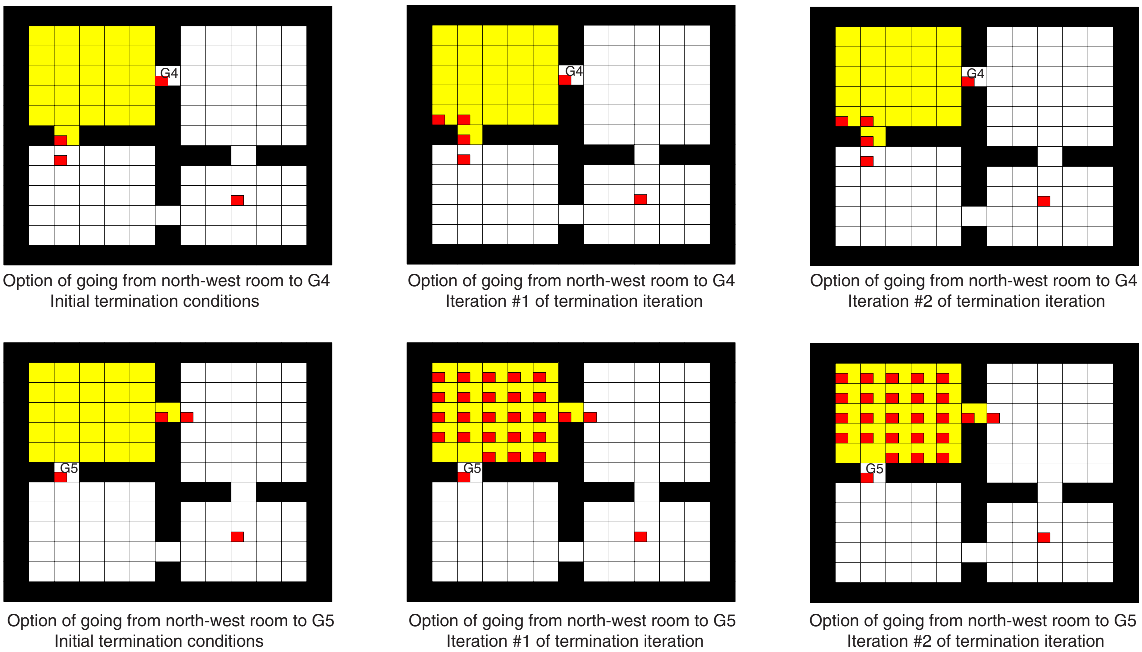

Figure 8.6. The result of termination iteration in the rooms environment, in the north-west room. One of the options gets very fragmented.

Q-learning, or intra-option learning, and improve the termination conditions as the agent goes along, even if the value function is not perfect.

8.5 Policy Iteration for Options

Changing the termination conditions of options is a special case of a more general algorithm, which adapts the policies of the options to whatever task there is at hand. The planning version of this algorithm, *policy iteration for options*, is presented below, for the case of Markov options. The main idea of the algorithm is to change the internal policy of the options such that it picks the best primitive action with respect to the current value function.

If the algorithm is iterated until the stopping criterion is met, then it recovers the optimal policy of the underlying MDP, as we prove below. As a first step, we prove a more general result, showing that the improvement step leads to a better value function.

Algorithm 5 Policy Iteration for Options

1. **Initialization:** Start with a given set of Markov options $\mathcal{O}_0$.
2. **Iterate** the following steps:
   (a) **Evaluation:** Compute the models of the options r_s^{ao} and $p_{ss'}^{ao}$ and the optimal value function for the current set of options, $Q^*_{\mathcal{O}_i}$.
   (b) **Improvement:** If for some state, $r_s^{ao} + \sum_{s'} p_{ss'}^{ao} V^*_{\mathcal{O}_i}(s') > Q^*_{\mathcal{O}_i}(s, o)$, then change option o to take a in s:
   $$o(s,a) \leftarrow 1 \text{ and } o(s,a') \leftarrow 0 \forall a' \neq a$$
3. **Stopping Condition:** $V^*_{\mathcal{O}_i} = V^*_{\mathcal{O}_{i+1}}$.

Theorem 10 (Policy Improvement) *For any MDP, any set of Markov options $\mathcal{O}$ and any Markov policy $\mu : \mathcal{S} \times \mathcal{O} \mapsto [0,1]$, define a new set of options $\mathcal{O}'$ with a one-to-one mapping between the two option sets as follows: for every $o \in \mathcal{O}$ we define a corresponding $o' \in \mathcal{O}'$ where $o'(s,\cdot) = o(s,\cdot)$ except for every state in which $Q^\mu(s,o) < r_s^{ao} + \sum_{s'} p_{ss'}^{ao} V^\mu(s')$, we may choose to set $o(s,a) = 1$. Let μ' be the policy over options corresponding to μ: $\mu'(s,o') = \mu(s,o)$, where o is the option in $\mathcal{O}$ corresponding to o', for all $s \in \mathcal{S}$. Then $V^{\mu'}(s) \geq V^\mu(s), \forall s$, and there exists at least one state for which the inequality is strict.*

Proof: The proof is very similar to the proof for the interruption theorem. The idea is to show that, for an arbitrary state s, executing the option given by the policy μ' and then following policy μ thereafter is no worse than always following policy μ. More formally, we have to show that:

$$\sum_{o'} \mu'(s,o')[r_s^{o'} + \sum_{s'} p_{ss'}^{o'} V^\mu(s')] \geq V^\mu(s) = \sum_o \mu(s,o)[r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')].$$

If this is true, then we can use this inequality to expand the left side, by repeatedly replacing every occurrence of $V^\mu(x)$ with $\sum_{o'} \mu'(x,o')[r_x^{o'} + \sum_{x'} p_{xx'}^{o'} V^\mu(x')]$. In the limit, the left side becomes $V^{\mu'}$, proving that $V^{\mu'} \geq V^\mu$. If no changes were made to option o in state s, then $r_s^{o'} + \sum_{s'} p_{ss'}^{o'} V^\mu(s') = r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')$. If option o was changed, then from the hypothesis and the fact that o is Markov, we have:

$$r_s^{o'} + \sum_{s'} p_{ss'}^{o'} V^\mu(s') = r_s^{ao} + \sum_{s'} p_{ss'}^{ao} V^\mu(s') > Q^\mu(s,o) = r_s^o + \sum_{s'} p_{ss'}^o V^\mu(s')$$

Because $\mu'(s,o') = \mu(s,o)$, this proves our theorem. $\diamond$

Note that if we include the termination action τ into the set we consider, and if we define the model of this action by: $r_s^\tau = V^\mu(s)$ and $p_{ss'}^\tau = 0, \forall s' \in \mathcal{S}$, then the interruption theorem 8, when applied for Markov options, is a special case of the policy improvement theorem 10.

Theorem 11 (Convergence of Policy Iteration for Options) *For any MDP and any initial set of options $\mathcal{O}$, the policy iteration algorithm 5 converges to the optimal value function V^* for the MDP.*

Proof: By theorem 10, every iteration of the algorithm will improve on the previous value function. In order to prove that when the stopping criterion is met, $V_{\mathcal{O}_i}^* = V^*$, we use a proof by contradiction. Suppose that when the stopping criterion is met, there is some state s for which $a^* \neq a$, where a^* is the action suggested by the optimal policy, and a is the primitive action suggested by the best option in s. Then we have:

$$\begin{aligned} r_s^{a^*o} + \sum_{s'} p_{ss'}^{a^*o} V_{\mathcal{O}_i}^*(s') &= r_s^{a^*} + \gamma \sum_{s''} p_{ss''}^{a^*} (r_{s''}^o + \sum_{s'} p_{s''s'}^o V_{\mathcal{O}_i}^*(s') \\ &= r_s^{a^*} + \gamma \sum_{s''} p_{ss''}^{a^*} V_{\mathcal{O}_i}^*(s'') \\ &\geq Q_{\mathcal{O}_i}^*(s,o). \end{aligned}$$

which means that the algorithm would have continued for one more iteration. $\diamond$

Of course, in practice the complete planning version of the algorithm is too expensive to apply. However, this is an important theoretical result, because it guarantees that any set of options can be improved Enoch to eventually reach the optimal solution of the MDP, if that is desired. In practice, one could interleave steps of learning the models with improving the policies of the options.

8.6 Conclusions

In this chapter we presented methods for changing a given set of options to be more adequate for the task at hand. These methods are based on looking at the internal structure of the options and changing it by algorithms inspired by policy iteration. By applying these changes iteratively, one can improve the options in such a way as to obtain the best solution possible for the MDP. Changing the termination conditions of options can be done with minimal additional computation.

CHAPTER 9

CREATING NEW OPTIONS FROM SUBGOALS

It is natural to think of options as achieving subgoals of some kind. Then each option's policy could be learned to achieve the corresponding subgoals as well as possible. Given subgoals for options, it is relatively straightforward to design off-policy intra-option learning methods to learn the policies that achieve those subgoals. For example, it may be possible to simply apply Q-learning to learn independently about each subgoal and option (as in Singh, 1992; Lin, 1993; Dorigo & Colombetti, 1994; Thrun & Schwartz, 1995).

On the other hand, it is not clear which is the best way to formulate subgoals to associate with options, or even what the basis for evaluation should be. One of the important considerations is the extent to which models of options constructed to achieve one subgoal can be transferred to aid in planning to achieve another. We would like a long-lived learning agent to face a continuing series of subtasks that result in its being more and more capable.

In this chapter we present an approach to the creation of options, based on associating options with subgoals of achievement. The formalization of subgoals we present here suffices to illustrate some of the possibilities and problems that arise. A larger issue that we do not address is the source of the subgoals. We assume that the subgoals are given and focus on how options can be learned and tuned to achieve them, and on how learning toward different subgoals can aid each other. The issue of formulating useful subgoals is investigated currently by several researchers (McGovern, 1998; Moore et al., 1998; Andre, 1998; Baum, 1998)

9.1 Subgoals of Achievement

A simple way to formulate a subgoal is by assigning a *subgoal value*, $g(s)$, to each state s in a subset of states $\mathcal{G} \subseteq \mathcal{S}$. These values indicate how desirable it is to terminate in each state in $\mathcal{G}$. For example, to learn a hallway option in the rooms task, the target hallway might be assigned a subgoal value of $+1$ while the other hallway and all states outside the room might be assigned a subgoal value of 0. Let $\mathcal{O}_g$ denote the set of options that terminate only and always in the states $\mathcal{G}$ in which g is defined. Given a subgoal-value function $g : \mathcal{G} \rightarrow \Re$, one can define a new state-value function, denoted $V_g^o(s)$, for options $o \in \mathcal{O}_g$, as the expected value of the cumulative reward if option o is initiated in state s, plus the subgoal value $g(s')$ of the state s' in which it terminates (discounted appropriately). Similarly, we can define a new action-value function $Q_g^o(s,a) = V_g^{ao}(s)$ for actions $a \in \mathcal{A}_s$ and options $o \in \mathcal{O}_g$.

Finally, we can define *optimal* value functions for any subgoal g:

$$V_g^*(s) = \max_{o \in \mathcal{O}_g} V_g^o(s)$$

and

$$Q_g^*(s,a) = \max_{o \in \mathcal{O}_g} Q_g^o(s,a)$$

. Finding an option that achieves these maximums (an *optimal option* for the subgoal) is then a well-defined subtask. For Markov options, this subtask has Bellman equations and methods for learning and planning just as in the original task. For example, the one-step tabular Q-learning method for updating an estimate $Q_g(s_t, a_t)$ of $Q_g^*(s_t, a_t)$ is

$$Q_g(s_t,a_t) \leftarrow Q_g(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q_g(s_{t+1},a_{t+1}) - Q_g(s_t,a_t)\right], \text{ if } s_{t+1} \notin \mathcal{G},$$

and

$$Q_g(s_t, a_t) \leftarrow Q_g(s_t, a_t) + \alpha \left[r_{t+1} + \gamma g(s_{t+1}) - Q_g(s_t, a_t) \right], \text{ if } s_{t+1} \in \mathcal{G}.$$

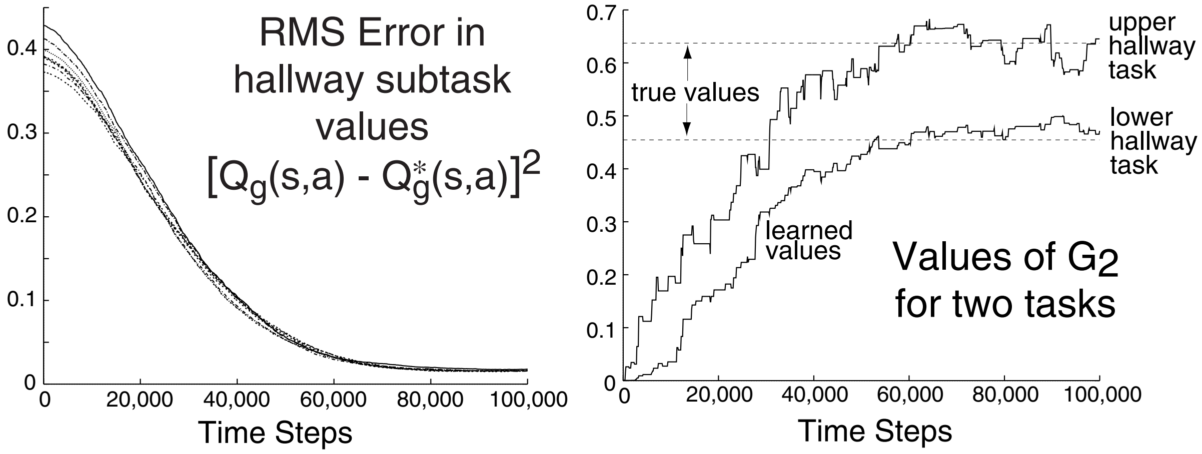

Figure 9.1. Learning subgoal-achieving hallway options under random behavior. Shown on the left is the error between $Q_g(s,a)$ and $Q^*_g(s,a)$ averaged over $s \in \mathcal{I}$, $a \in \mathcal{A}$, and 30 repetitions. The right panel shows the learned values for two options at one state (maximum over action values) approaching their correct values.

As a simple example, we applied this method to learn the policies of the eight hallway options in the rooms example. Each option was assigned subgoal values of $+1$ for the target hallway and 0 for all states outside the option's room, including the off-target hallway. The initial state was that in the upper left corner, actions were selected randomly with equal probability, and there was no goal state. The parameters were $\gamma = 0.9$ and $\alpha = 0.1$. All rewards were zero. Figure 9.1 shows the learned action values $Q_g(s,a)$ for each of the eight subgoals/options reliably approaching their ideal values, $Q^*_g(s,a)$.

It is interesting to note that, in general, the policies learned to achieve subgoals depends in detail on the precise values assigned by g to the subgoal states. For example, suppose nonzero expected rewards were introduced into the rooms task in all states, distributed uniformly between 0 and -1. Then a subgoal value of $+10$ (at the target hallway) results in an optimal policy that goes directly to the target hallway and away from the other hallway, as shown on the left in Figure 9.2, whereas

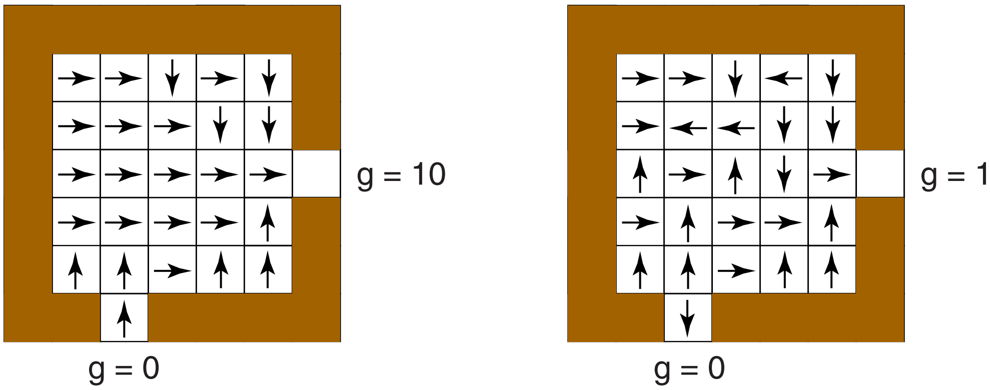

Figure 9.2. Two different optimal policies for options given two different subgoal values at the target hallway. A subgoal value of +10 (left) results in a more direct policy than a subgoal of +1.

a subgoal value of +1 may result in an optimal policy that goes only indirectly to the target hallway, as shown on the right in Figure 9.2. A roundabout path may be preferable in the latter case to avoid unusually large penalties. In the extreme it may even be optimal to head for the off-target hallway, or even to spend an infinite amount of time running into a corner and never reach any subgoal state. This is not a problem, but merely illustrates the flexibility of this subgoal formulation. For example, we may want to have two options for `open-the-door`, one of which opens the door only if it is easy to do so, for example, if is unlocked, and one which opens the door no matter what, for example, by breaking it down if need be. If we had only the first option, then we would not be able to break down the door if necessary. If we had only the second then we would not be able to choose to open the door without committing to breaking it down if it was locked, which would greatly diminish the option's usefulness. The ability to learn and represent options for different intensities of subgoals, or different balances of outcome values, is an important flexibility.

9.2 Subgoals and Transfer

Subgoals, options, and models of options enable interesting new possibilities for reinforcement learning agents. For example, we could present the agent with a series of tasks as subgoals, perhaps graded in difficulty. For each, the agent would be directed to find an option that achieves the subgoal and to learn a model of the option. Although the option and model are constructed based on the task, note that they can be transferred to any other task. The option just says what to do; if behaving that way is a useful substep on another task, then it will help on that task. Similarly, the model just predicts the consequences of behaving that way; if that way of behaving is a useful substep on another task, then the model will help in planning to use that substep. As long as the model is accurate for its option it may be useful in planning the solution to another task. Singh (1992a,b,c) and Lin (1993) provide some simple examples of learning solutions to subtasks and then transferring them to help solve a new task.

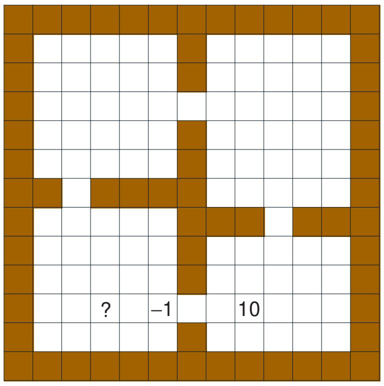

Figure 9.3. A subgoal to which a hallway option does *not* transfer. The option for passing from the lower-left room through to the state with subgoal value 10 no longer works because of the state with subgoal value -1. The original model of this option is overpromising with respect to the subgoal.

On the other hand, assuring that the models of options remain accurate across changes in tasks or subgoals is far from immediate. The most severe problem arises when the new subgoal prevents the successful completion of an option whose model has previously been learned. Figure 9.3 illustrates the problem in a rooms example. Here we assume the options and models have already been learned, then a new subgoal is considered that assigns a high value, 10 to a state in the lower-right room but a low value, -1, to a state that must be passed through to enter that room from the lower-left room. The -1 subgoal state makes it impossible to pass between the two rooms if we consider only options that terminate when they reach a subgoal state. Yet the prior model indicates that it is still possible to travel from the lower-left room "through" the -1 state to the hallway state and thereby to the 10-valued state. Thus, planning with this model will lead inevitably to a highly-valued but poor policy. Such problems can arise whenever the new subgoal involves states that may be passed through when an option is executed.

A simple idea is to alter the subgoal formulation such that subgoal states *can* be passed through: stopping in them and collecting the subgoal value is optional rather than required. In this case, an option is obtained as the solution to an *optimal stopping task* (see, e.g. Puterman, 1994). An optimal stopping task is defined as an MDP in which there is a value for stopping specified in some states. If the agent decides to stop, it collects this terminal value. If it continues to act, it collects the usual rewards. Such a formulation prevents the problem illustrated in Figure 9.3.

Finally, note that in order to plan successfully, the models of the options, do not necessarily have to be accurate, just *non-overpromising* (Precup & Sutton, 1998). In other words, they do not have to predict the correct outcome, just an outcome whose expected value is less than or equal to the value of the correct outcome. This finesse may enable important special cases to be handled simply. For example, any new subgoal involving states $\mathcal{G}$ that all have the same subgoal value, can probably be safely used for transfer. The sort of problem shown in Figure 9.3 can never occur in such cases either.

CHAPTER 10

CONCLUSIONS AND FUTURE WORK

10.1 Contributions

In this thesis we presented a general framework for describing temporally extended actions in the context of reinforcement learning and MDPs. This framework applies to stochastic environments, and it enables us to express temporally extended actions in the form of closed-loop policies and goal-oriented behavior, in a more general way than what has been possible in classical AI approaches to temporal abstraction.

Options and models of options are a form of knowledge representation that is expressive, clear and suitable for learning and planning. The models of options are computed from observations of the environment and therefore can be interpreted and changed by the agent without the need for human intervention. SMDP theory provides the foundation of the theory of options. In this dissertation we showed how a set of options in an MDP defines a related SMDP. This connection to SMDP theory provides a rich set of planning and learning methods, convergence results, and a natural and general way of predicting the effects of options.

The most interesting feature of the options framework is that it allows an agent to work simultaneously with high-level and low-level temporal representations. In addition to reasoning and learning about options, the agent can examine their internal structure in order to make predictions, learn and plan at multiple temporal scales. In this dissertation we presented several novel algorithms that take advantage of the internal structure of options. For instance, intra-option learning algorithms use this structure in order to learn efficiently about many options from the same real experience. Intra-option learning is an example of off-policy learning, a general approach for learning about the effects of one way of behaving while behaving in a different way. Off-policy learning is key to the efficiency of learning in a system that can potentially consider many options. We described off-policy learning algorithms that have strong convergence guarantees and are also very efficient in practice.

We also presented new algorithms for improving options by changing their termination conditions and their internal policies. These algorithms enable an agent to start with a generic set of options and then tune it to improve its performance on any given task.

Our research suggests that options can enable faster learning and planning in complex environments, as well as in the case in which an agent is faced with a whole family of tasks that it has to solve. Options are a way of representing the actions available to the agent at an appropriate time scale. In many environments, a solution to a task can contain a long series of actions, but a much shorter sequence of options. This means that a solution in terms of options can be computed much faster by a planning algorithm, since there are fewer decisions to consider. Options also have the side effect of compressing the state space, because we only need value predictions for states in which options terminate.

Options also provide an easy and principled way of providing prior knowledge for a learning agent. In many domains of interest, people know useful controllers or heuristics for acting, or they can guess useful subgoals that can represent steps towards a solution. Options are a natural way to specify such knowledge.

Options are very useful in the case in which an agent is not faced with just one task, but with a series of tasks that are related. In this case, options can represent solutions to subtasks that can be saved and reused over time. For instance, a robot might need to navigate to different destinations in a building. Having options such as following a corridor or going up and down stairs would be useful in this case for going to any destination.

The options framework allows us to go beyond the usual view in which an agent learns a single policy to solve a particular task. The framework enables learning and prediction about many ways of behaving, which can be useful in different situations and at different stages of learning. This research can be viewed as a first step towards a general architecture for constructing representations that facilitate learning.

10.2 Future Research Directions

There are many research areas that offer a lot of promise for future work:

Finding better state representations: Action and perception are intimately related. Objects can be described by the opportunities that they afford for action. For instance, the most salient feature of a chair is the fact that it can be used to sit on. Doors can be opened and closed, and they are used to pass between rooms. Models of options capture knowledge about the temporally extended effects of options. Such models can form the basis for representing concepts, in a way that can be understood by the system, and validated by acting in the world. For instance, consider a robot learning to recognize its battery charger. The most useful concept for it is the set of states from which it can successfully dock, and this is exactly what would be produced by the model of a docking option. Such action-oriented concepts can be learned and tested by the robot without external supervision.

Parametric representations for options and their models: There are many reasons for which one would want a parametric representation of options. First, people usually find it natural to specify controllers or behaviors depending on parameters. Having parametric representations of options would enable them to do this in a more natural way. Second, parametric representations of options would facilitate parametric representations of their models. Models are the primary way of encoding predictive knowledge about the effects of options.

They enable reasoning about options and using options for solving tasks. In this thesis we assumed that models are constructed specifically for each option. The only generalization that can occur is over states. This leads to an explosion in the number of models if an agent has a lot of options, which in turn slows down planning. One way to alleviate this utility problem is to represent options and their models parametrically. In this case, a model is a function from states and some parametric representation of options, to predictions about the intermediate reward and the transition after execution. Finding good ways of representing options and models, and determining how to do learning and planning with such models could be a key step in solving the utility problem.

Special hierarchical learning methods: In this thesis we have shown that hierarchies of options with multiple levels can be flattened and treated as one-level hierarchies. However, more efficient learning algorithms can be obtained by designing special methods for hierarchies with multiple levels of options. For instance, in the case of intra-option learning algorithms for hierarchical Markov options, one can prioritize the order in which the options are updated, from bottom up, such that the computation is more efficient than just flattening the hierarchy at every point. Other special-purpose computational tricks can probably be used for other multiple-level hierarchies of options.

Using function approximation and state abstraction with options: Most of the illustrations in this thesis used finite discrete MDPs. But the options framework allows the use of state abstraction and function approximation to deal with exponentially large and continuous state spaces. The simplest way in which this can be done is to use standard methods in order to represent value functions and models of options. More empirical experience using such methods with options would be very useful.

Evaluating sets of options: As we have shown in some of the illustrative examples, options can speed up planning significantly, but with a loss in the quality of the solution obtained. Of course, the quality of the solution can be improved by methods such as termination and policy iteration. But in this case, the final solution varies in quality as well, as seen in chapter 8. Moreover, sets of options have different utilities at different stages of learning. For instance, in the beginning of learning, options that help the agent explore its state space can be very useful, even if they do not provide a lot of reward in the long run (and therefore would not be part of the optimal policy). It would be very interesting to formulate mathematical criteria describing the utility of different sets of options depending on the stage of the learning process and the learning or planning algorithms being used. This would enable comparisons between different sets of options, and facilitate finding good sets of options.

Creating options and finding subgoals: In chapter 9 we illustrated one way of creating new options, by formulating an optimal stopping task over the state space. But the terminal states and their values are hand-picked by the designer of the system. In general, we would like a system to find such states and values automatically. There are heuristics that can be used in special cases, such as using funneling states as subgoals (McGovern, 1998). The issue of finding good subgoals is intimately related with the problem of evaluating different sets of options. There are many interesting questions regarding the utility of subgoals and the automatic creation of subgoals. We have not addressed such questions in this dissertation, but we believe that the options framework provides a good language for formulating these questions in a clear and precise way.

Solving POMDPs by using options: Consider an agent inside a long featureless corridor. If there are no salient features to distinguish different positions, the agent will suffer from a hidden state problem. But if it uses an option that takes it all the way to the end of the corridor, where the state can be observed exactly, then there is no problem with being inside the corridor. As long as there is some way of behaving that takes the agent between fully observable states, the problem can be treated as an SMDP. In general, options and models of options may help avoid partial observability.

BIBLIOGRAPHY

Andre, D. (1998). Learning hierarchical behaviors. *NIPS'98 Workshop on Abstraction and Hierarchy in Reinforcement Learning.*

Asada, M., Noda, S., Tawaratsumida, S., & Hosada, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, *23*, 279–303.

Baum, E. (1998). Manifesto for an evolutionary economics of intelligence. In C.M.Bishop (Ed.), *Neural networks and machine learning.* Springer-Verlag.

Bellman, R. E. (1957). A Markov Decision Process. *Journal of Mathematical Mechanics*, *6*, 679–684.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Brackbill, J. U., & Cohen, B. I. (Eds.). (1985). *Multiple time scales.* Academic Press.

Bradtke, S. J., & Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov Decision Problems. *Advances in Neural Information Processing Systems 7* (pp. 393–400). MIT Press.

Brafman, R. I., & Tennenholtz, M. (1997). Modeling agents as qualitative decision makers. *Aritifcial Intelligence*, *94*, 217–268.

Brockett, R. W. (1993). Hybrid models for motion control systems. In H. Trentelman and J. Willems (Eds.), *Perspectives in control,*, 29–54. Birkhauser.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 14–23.

Cassandras, C. G. (1993). *Discrete event systems. Modelling and performance analysis.* Irwin and Aksen.

Crites, R. H., & Barto, A. G. (1996). Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference* (pp. 1017–1023). MIT Press.

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, *5*, 613–624.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. *Advances in Neural Information Processing Systems 5* (pp. 271–278). Morgan Kaufmann.

Dean, T., & Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. *Proceedings of IJCAI'95.*

DeJong, G. F. (1994). Learning to plan in continuous domains. *Artificial Intelligence*, *65*, 71–141.

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning.* Morgan Kaufmann.

Digney, B. L. (1996). Emergent hierarchical control structures: Learning hierarchical/reactive relationships in reinforcement learning environments. *From Animals to Animats 4: SAB'96* (pp. 363–373). MIT Press.

Dorigo, M., & Colombetti, M. (1994). Robot shaping: Developing autonomous agents through shaping. *Artificial Intelligence*, *71*, 321–370.

Drescher, G. L. (1991). *Made-up minds. A constructivist approach to artificial intelligence.* MIT PRess.

Drummond, C. (1998). Composing functions to speed up reinforcement learning in a changing world. *Machine Learning: ECML98. 10th European Conference on Machine Learning, Chemnitz, Germany, April 1998. Proceedings* (pp. 370–381). Springer.

Etzioni, O. (1990). Why PRODIGY/EBL works. *Proceedings of the Eighth National Confernce on Artificial Intelligence, AAAI* (pp. 916–922).

Fikes, R., P.E.Hart, & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*, 251–288.

Godbole, D. N., Lygeros, J., & Sastry, S. (1995). Hierarchical hybrid control: A case study. *Hybrid Control II* (pp. 166–190). Springer-Verlag.

Gratch, J., & DeJong, G. (1996). A statistical approach to adaptive problem solving. *Artificial Intelligence.*

Greiner, R., & Jurisica, I. (1992). A statistical approach to solving the EBL utility problem. *Proceedings of the Tenth National conference on Artificial Intelligence* (pp. 241–248).

Grossman, R. L., Nerode, A., Ravn, A. P., & Rischel, H. (Eds.). (1993). *Hybrid systems*, vol. 736 of *Lecture Notes in Computer Science.* Springer-Verlag.

Hauskrecht, M., Meuleau, N., Boutilier, C., Kaelbling, L. P., & Dean, T. (1998). Hierarchical solution for Markov Decision Processes using macro-actions. *Proceedings of the Fourteenth International Conference on Uncertainty In Artificial Intelligence.*

Howard, R. (1960). *Dynammic programming and Markov Decision Processes.* MIT Press.

Huber, M., & Grupen, R. A. (1997). A feedback control structure for on-line learning tasks. *Robotics and Autonomous Systems*, *22*, 303–315.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*, 285–317.

Jaakkola, T., Jordan, M., & Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, *6*, 1185–1201.

J.A.Coelho, J., Araujo, E. G., Huber, M., & Grupen, R. A. (1998). Dynamical categories and control policy selection. *Proceedings of the ISIC/CIRA/ISAS'98 Conference.* IEEE.

Kaelbling, L. P. (1993a). Hierarchical learning in stochastic domains: Preliminary results. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 167–173). Morgan Kaufmann.

Kaelbling, L. P. (1993b). Learning to achieve goals. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1094–1098). Morgan Kaufmann.

Kalmár, Z., Szepesvári, C., & Lörincz, A. (1997). Module based reinforcement learning for a real robot. *Proceedings of the Sixth European Workshop on Learning Robots* (pp. 22–32).

Karlsson, J. (1997). *Learning to solve multiple goals.* Doctoral dissertation, University of Rochester.

Kleer, J. D., & Brown, J. S. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, *24*, 7–83.

Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI'90.*

Kokotovic, P., Khalil, H. K., & O'Reilly, J. (1986). *Singular perturbation methods in control: Analysis and design.* Academic Press.

Korf, R. E. (1985). *Learning to solve problems by searching for macro-operators.* Pitman Publishing Ltd.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, *33*, 65–88.

Kuipers, B. J. (1979). Commonsense knowledge of space: Learning from experience. *Proceedings of IJCAI-79* (pp. 499–501).

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Levinson, R., & Fuchs, G. (1994). *A patter-weight formulation of search knowledge* (Technical Report UCSC-CRL-94-10). University of California, Santa Cruz.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293–321.

Lin, L.-J. (1993). *Reinforcement learning for robots using neural networks.* Doctoral dissertation, Carnegie Mellon University.

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

Mahadevan, S., Marchallek, N., Das, T. K., & Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 202–210). Morgan Kaufmann.

Mataric, M. (1997). Reinforcement learning in the multi-robot domain. *Autonomous Robots*, *4*, 73–83.

McCallum, A. K. (1995). *Reinforcement learning with selective perception and hidden state.* Doctoral dissertation, University of Rochester.

McCormick, S. F. (1989). *Multilevel adaptive methods for partial differential equations.* Society for Industrial and Applied Mathematics.

McGovern, A. (1998). acQuire-macros: An algorithm for automatically learning macro-actions. *NIPS'98 Workshop on Abstraction and Hierarchy in Reinforcement Learning.*

McGovern, A., Sutton, R. S., & Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. *Grace Hopper Celebration of Women in Computing* (pp. 13–17).

Meuleau, N., Hauskrecht, M., Kim, K.-E., Peshkin, L., Kaelbling, L. P., Dean, T., & Boutilier, C. (1998). Solving very large weakly coupled Markov Decision Processes. *Proceedings of the Fifteenth National Conference on Artificial Intelligence.*

Minton, S. (1988). *Learning search control knowledge. An explanation-based approach.* Kluwer Academic Publishers.

Moore, A. W., Baird, L., & Kaelbling, L. P. (1998). Multi-Value-Functions: Efficient automatic action hierarchies for multiple goal MDPs. *NIPS'98 Workshop on Abstraction and Hierarchy in Reinforcement Learning.*

M.Uchibe, M.Asada, & K.Hosada (1996). Behavior coordination for a mobile robot using reinforcement learning. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1329–1336).

Naidu, D. S. (1988). *Singular perturbation methods in control systems.* London, UK: Peter Peregrinus Ltd.

Naidu, D. S., & Rao, A. K. (1985). *Singular perturbation analysis of discrete control systems.* No. 1154 in Lecture Notes in Mathematics. Springer-Verlag.

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Prentice-Hall.

Nilsson, N. J. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research, 1*, 139–158.

Oates, T., & Cohen, P. R. (1996). Searching for planning operators with context-dependent and probabilistic effects. *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 863–868).

Parr, R. (1998). *Hierarchical control and learning for Markov Decision Processes.* Doctoral dissertation, Computer Science Division, University of California, Berkeley, USA.

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems 10.* MIT Press.

Pierce, D., & Kuipers, B. (1994). Learning to explore and build maps. *Proceedings of the Twelfth National Conference on Artificial Intelligence.*

Precup, D., & Sutton, R. S. (1997). Multi-Time models for reinforcement learning. *ICML'97 Workshop: The Role of Models in Reinforcement Learning.*

Precup, D., & Sutton, R. S. (1998). Multi-time models for temporally abstract planning. *Advances in Neural Information Processing Systems 10 (Proceedings of NIPS'97)* (pp. 1050–1056). MIT Press.

Precup, D., Sutton, R. S., & Singh, S. (1997). Planning with closed-loop macro actions. *Working Notes of the AAAI Fall Symposium '97 on Model-directed Autonomous Systems* (pp. 70–76).

Precup, D., Sutton, R. S., & Singh, S. (1998). Theoretical results on reinforcement learning with temporally abstract options. *Machine Learning: ECML98. 10th European Conference on Machine Learning, Chemnitz, Germany, April 1998. Proceedings* (pp. 382–393). Springer.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete stochastic dynamic programming.* Wiley.

Ring, M. (1994). *Continual learning in reinforcement environments.* Doctoral dissertation, University of Texas at Austin.

Rosenstein, M. T., & Cohen, P. R. (1998). Concepts from time series. *Proceedings of the Fifteenth National Conference on Artificial Intelligence.*

Rubinstein, R. (1981). *Simulation and the Monte Carlo method.* NewYork: Wiley.

Ryan, M., & Pendrith, M. D. (1998). RL-TOPS: An architecture for modularity and reuse in reinforcement learning. *Proceedings of the Fifteenth Intermnational Conference no Machine Learning.* Morgan Kaufman.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, *5*, 115–135.

Sacerdoti, E. D. (1977). *A structure for plans and behavior.* Elsevier North-Holland.

Say, A. C. C., & Kuru, S. (1996). Qualitative system identification: Deriving structure from behavior. *Artificial Intelligence*, *83*, 75–141.

Singh, S., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning.*

Singh, S. P. (1992a). Reinforcement learning with a hierarchy of abstract models. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 202–207). MIT/AAAI Press.

Singh, S. P. (1992b). Scaling reinforcement learning by learning variable temporal resolution models. *Proceedings of the Ninth International Conference on Machine Learning* (pp. 406–415). Morgan Kaufmann.

Stone, P., & Veloso, M. (1999). Team-partitioned, opaque-transition reinforcement learning. *RoboCup-98: Robot Soccer World Cup II.* Springer Verlag.

Stone, P., & Veloso, M. (in press). A layered approach to learning client behaviors in the RoboCup soccer server. *Applied Artificial Intelligence.*

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1995). TD models: Modeling the world as a mixture of time scales. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 531–539). Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* MIT Press.

Sutton, R. S., & Pinette, B. (1985). The learning of world models by connectionist networks. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society* (pp. 54–64).

Sutton, R. S., Precup, D., & Singh, S. (1998a). *Between MDPs and Semi-MDPs: learning, planning, and representing knowledge at multiple temporal scales* (Technical Report 98-74). University of Massachusetts, Amherst, MA 01003.

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

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

Sutton, R. S., Singh, S., Precup, D., & Ravindran, B. (1999b). Improved switching among temporally abstract actions. *Advances in Neural Information Processing Systems 11 (Proceedings of NIPS'98* (pp. 1066–1072). MIT Press.

Tambe, M., Newell, A., & Rosenbloom, P. (1990). The problem of expensive chunks and its solution by restricting expensiveness. *Machine Learning*.

Tesauro, G. J. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*, 58–68.

Thrun, S., & Schwartz, A. (1995). Finding structure in reinforcement learning. *Advances in Neural Information Processing Systems 7* (pp. 385–392). MIT Press.

Watkins, C. J. C. H. (1989). *Learning with delayed rewards*. Doctoral dissertation, Psychology Department, Cambridge University, Cambridge, UK.

Wiering, M., & Schimdhuber, J. (1998). HQ-Learning. *Adaptive Behavior*, *6*, 219–246.

Wixson, L. E. (1991). Scaling reinforcement learning techniques via modularity. *Proceedings of the Eighth International Conference on Machine Learning* (pp. 368–372). Morgan Kaufmann.

Zhang, W., & Dietterich, T. G. (1995). A reinforcement learning approach to job-shop scheduling. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1114–1120).