An (almost) Tutorial on Reinforcement Learning

Satinder P. Singh

This is chapters 2, 3, and 4 from my Ph.D thesis titled "Learning to Solve Markovian Decision Tasks", 1993.

CHAPTER 2

LEARNING FRAMEWORK

This chapter presents a formal framework for formulating tasks involving embedded agents. Embedded agents are being studied in a number of different disciplines, such as AI, robotics, systems engineering, control engineering, and theoretical computer science. This dissertation is focussed on embedded agents that use repeated experience at solving a task to become more skilled at that task. Accordingly, the framework adopted here abstracts the task to that of learning a behavior that approximates optimization of a preset objective functional defined over the space of possible behaviors of the agent. The framework presented here closely follows the work of Barto *et al.* [12, 8] and Sutton [55, 56].

2.1 Controlling Dynamic Environments

In general, the environment in which an agent is embedded can be dynamic, that is, can undergo transformations over time. An assumption and idealization that is often made by dynamical system theorists as well as by AI researchers is that all the information that is of interest about the environment depends only on a finite set of variables that are functions of time $x_1(t), x_2(t), x_3(t), \ldots, x_n(t)$. These variables are often called *state variables* and form the components of the n-dimensional state vector $x(t)$. Mathematical models of the transformation process, i.e., models of the dynamics of the environment, relate the time course of changes in the environment to the state of the environment.

If the agent has no control over the dynamics of the environment, the fundamental problem of interest is that of *prediction*. Solving the prediction problem requires ascertaining an approximation to the sequence $\{x(t)\}$, or more generally an approximation of some given function of $\{x(t)\}$. A more interesting situation arises when the agent can influence the environment's transformation over time. In such a case, the fundamental problem becomes that of *prescription* or *control*, the solution to which prescribes the actions that the agent should execute in order to bring about the desired transformations in the environment. We will return to these two issues of prediction and control throughout this dissertation.

For several decades control engineers have been designing controllers that are able to transform a variety of dynamical systems to a desired goal state or that can track a desired state trajectory over time (e.g., Goodwin and Sin [26]). Such tasks are called regulation and tracking tasks respectively. Similarly researchers in AI have developed problem solving methods for finding sequences of operators that will transform an initial problem (system) state into a desired problem state, often called a 'goal' state.

2.2 Problem Solving and Control

Despite underlying commonalities, the separate development of the theory of problem solving in AI and the theory of regulation and tracking in control engineering led to differences in terminology. For example, the transformation to be applied to the environment is variously called an operator, an action, or a control. The external system to be controlled is called an environment, a process, or a plant. In addition, in control engineering the agent is called a controller. The terms agent, action, and environment will be used in this dissertation.

There are also differences in the algorithms developed by the two communities because of the differing characteristics of the class of environments chosen for study. Traditionally, AI has focussed almost exclusively on deterministic, discrete state and time problems, whereas control theorists have embraced stochasticity and have included continuous state and time problems in their earliest efforts. Consequently, the emphasis in AI has been on *search control* with the aim of reducing the average proportion of states that have to be searched before finding a goal state, while in control theory the emphasis has been on ensuring *stability* by dealing robustly with disturbances and stochasticity.

The focus on deterministic environments within AI has led to the development of planning and heuristic search techniques that develop *open-loop* solutions, or plans. An open-loop solution is a sequence of actions that is executed without reference to the ensuing states of the environment. Any uncertainty or model mismatch can cause plan failure, which is usually handled by replanning.[1] On the other hand, most control design procedures within control theory have been developed to explicitly handle stochasticity and consequently compute a *closed-loop* solution that prescribes actions as a function of the environment's state and possibly of time. Note that forming closed-loop solutions confers no advantage in purely deterministic tasks, because for every start state the sequence of actions executed under the closed-loop solution is an open-loop solution for that start state.

A common feature of most of the early research in AI and in control theory was their focus on off-line design of solutions using environment models. Later, control theorists developed *indirect* and *direct* adaptive control methods for dealing with problems in which a model was unavailable (e.g., Astrom and Wittenmark [2]). Indirect methods estimate a model of the environment incrementally and use the estimated model to design a solution. The same interleaving procedure of system identification and off-line design can be followed in AI problem solving. Direct methods on the other hand directly estimate the parameters of a single, known, parametrized control solution without explicitly estimating a model. As stated before, part of the motivation for RL researchers has been to develop direct methods for solving learning tasks involving embedded agents.

2.3 Learning and Optimal Control

Of more relevance to the theory of learning agents embedded in dynamic environments is a class of control problems studied by optimal control theorists in which the desired state trajectory is not known in advance but is part of the solution to be determined by the control design process.[2] The desired trajectory is one that extremizes some external performance criteria, or some objective function, defined over the space of possible solutions. Such control problems are called *optimal control*

[1] Some of the recent work on planning produces closed-loop plans by performing a cycle of sensing and open-loop planning (e.g., McDermott [37]).

[2] Regulation and tracking tasks can also be defined using the optimal control framework.

problems and have also been studied for several decades. The optimal control perspective provides a suitable framework for learning tasks in which an embedded agent repeatedly tries to solve a task, caching the partial solution and other information garnered in such attempts, and reuses such information in subsequent attempts to improve performance with respect to the performance criteria.

This perspective of optimal control as search is the important common link to the view of problem solving as search developed within the AI community. For some optimal control problems, gradient-based search techniques, such as calculus of variations (e.g., Kirk [35]), can be used for finding the extrema. For other optimal control problems, where non-linearities or stochasticity make gradient-based search difficult, dynamic programming (DP) is the only known general class of algorithms for finding an optimal solution.

The current focus on embedded agents in AI has fortunately come at a time when a confluence of ideas from artificial intelligence, machine learning, robotics, and control engineering is taking place (Werbos [62], Barto [5], Barto *et al.* [8], Sutton *et al.* [55, 56], Dean and Wellman [22]). Part of the motivation behind this current research is to combine the complementary strengths of research on planning and problem solving in AI and of research on DP in optimal control to get the best of both worlds (e.g., Sutton [55], Barto *et al.* [8], and Moore and Atkeson [39]). For example, techniques for dealing with uncertainty and stochasticity developed in control theory are now of interest to AI researchers developing architectures for agents embedded in real-world environments. At the same time, techniques for reducing search in AI problem solving can play a role in making optimal control algorithms more efficient in their exploration of the solution space (Moore [38]).

Another feature of AI problem solving algorithms, e.g., A^* (Hart *et al.* [29], Nilsson [41]), that should be incorporated into optimal control algorithms is that of determining solutions only in parts of the problem space that matter. Algorithms from optimal control theory, such as DP, find complete optimal solutions that prescribe optimal actions to every possible state of the environment. At least in theory, there is no need to find optimal actions for states that are not on the set of optimal paths from the set of possible start states (see Korf [36], and Barto *et al.* [8]).

Following Sutton *et al.* [56] and Barto *et al.* [12], in this dissertation the adaptive optimal control framework is used to formulate tasks faced by autonomous embedded agents. The next section presents the specific formulation of optimal control tasks that is commonly used in machine learning research on building learning control architectures for embedded agents.

2.4 Markovian Decision Tasks

Figure 2.1 shows a block diagram representation of a general class of tasks faced by embedded agents.[3] It shows an agent interacting with an external environment in a discrete time perception-action cycle. At each time step, the agent perceives its environment, executes an action and receives a payoff in return. Such tasks are called multi-stage decision tasks, or sequential decision tasks, in control theory and operations research. A simplifying assumption often made is that the task is *Markovian* which requires that at each time step the agent's immediate perception returns the *state* of the environment, i.e., provides all the information necessary to make the future perceptions and payoffs independent of the past perceptions. The action executed by the agent and the external disturbances determine the next state of the environment. Such multi-stage decision tasks are called Markovian decision tasks (MDTs).

MDTs are discrete time tasks in which at each of a finite, or countably infinite, number of time steps the agent can choose an action to apply to the environment. Let X be the finite set of environmental states and A be the finite set of actions available to the agent.[4] At time step t, the agent observes the environment's current state, denoted $x_t \in X$, and executes action $a_t \in A$. As a result the environment makes a transition to state $x_{t+1} \in X$ with probability $P^{a_t}(x_t, x_{t+1})$, and the agent receives

[3]Note that Figure 2.1 is just one possible block diagram representation; other researchers have used more complex block diagrams to capture some of the more subtle intricacies of tasks faced by embedded agents (e.g., Whitehead [63], Kaelbling [34]).

[4]For ease of exposition it is assumed that the same set of actions are available to the agent in each state of the environment. The extension to the case where different sets of actions are available in different states is straightforward.

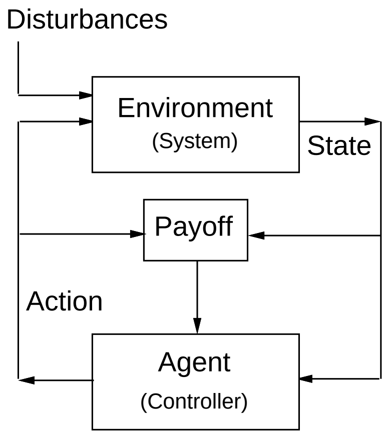

Figure 2.1 Markovian Decision Task. This figure shows a block diagram representation of an MDT. It shows an agent interacting with an external environment in a perception-action cycle. The agent perceives the state of the environment, executes an action, and gets a payoff in return. The action executed by the agent and the external disturbances change the state of the environment.

an expected payoff[5] $R^{a_t}(x_t) \in \mathbb{R}$. The process $\{x_t\}$ is called a Markovian decision process (MDP) in the operations research literature. The terms MDP and MDT will be used interchangeably throughout this dissertation.

The agent's *task* is to determine a policy for selecting actions that maximizes some cumulative measure of the payoffs received over time. Such a policy is called an *optimal policy*. The number of time steps over which the cumulative payoff is determined is called the *horizon* of the MDT. One commonly used measure for policies is the expected value of the discounted sum of payoffs over the time horizon of the agent as a function of the start state (see Barto *et al.* [12]). [6] This dissertation focuses on agents that have infinite life-times and therefore will have infinite horizons. Fortunately, infinite-horizon MDTs are simpler to solve than finite-horizon MDTs because with an infinite horizon there always exists a policy that is independent of time, called a *stationary* policy, that is optimal (see Ross [46]). Therefore, throughout

[5]All of the theory and the architectures developed in this dissertation extend to the formulation of MDTs in which payoffs are also a function of the next state, and are denoted $r^{a_t}(x_t, x_{t+1})$. In such a case, $R^{a_t}(x_t) \doteq E\{r^{a_t}(x_t, x_{t+1})\}$.

[6]The average payoff per time step received by an agent is another measure for policies that is used in the classical DP literature (Bertsekas [15]), and more recently in the RL literature (Schwartz [49], Singh [?]). This dissertation will only deal with the discounted measure for policies.

this dissertation one need only consider stationary policies $\pi : X \rightarrow A$ that assign an action to each state. Mathematically, the measure for policy π as a function of the start state x_0 is

$$V^{\pi}(x_0) \;=\; E\left[\sum_{t=0}^{\infty} \gamma^t R^{\pi(x_t)}(x_t)\right], \tag{2.1}$$

where E indicates expected value, and $\pi(x_t)$ is the action prescribed by policy π for state x_t. The discount factor γ, where $0 \leq \gamma < 1$, allows the payoffs distant in time to be weighted less than more immediate payoffs. The function $V^{\pi} : X \rightarrow \mathbb{R}$ is called the *value function* for policy π. The symbol V^{π} is used to denote both the value function and the vector of values of size $|X|$. An optimal control policy, denoted π^*, maximizes the value of every state.

For MDTs that have a horizon of one, called *single-stage* MDTs, the search for an optimal policy can be conducted independently for each state because an optimal action in any state is simply an action that leads to the highest immediate payoff, i.e.,

$$\pi^*(x) = \text{argmax}_{a \in A} R^a(x) \tag{2.2}$$

MDTs with a horizon greater than one, or multi-stage MDTs, face a difficult temporal credit assignment problem (Sutton [52]) because actions executed in the short-term can have long-term consequences on the payoffs received by the agent. Hence, to search for an optimal action in a state it may be necessary to examine the consequences of all action sequences of length equal to the horizon of the MDT.

Most physical environments have infinite state sets and are continuous time systems. However, tasks faced by agents embedded in such environments can frequently be modeled as MDTs by discretizing the state space and choosing actions at some fixed frequency. However, it is important to keep in mind that an MDT is only an abstraction of the physical task. Indeed, it may be possible to represent the same underlying physical task by several different MDTs, simply by varying the resolution of the state space and/or by varying the frequency of choosing actions. The choices made can impact the difficulty of solving the task. In general, the coarser the resolution in space and time, the easier it should be to find a solution. But at the same time better solutions may be found at finer resolutions. This tradeoff is a separate topic of research (e.g., Bertsekas [15]) and will only be partially addressed in this dissertation (see Chapter **??**).

The MDT framework for control tasks is a natural extension to stochastic environments of the AI state-space search framework for problem solving tasks. A rich variety of learning tasks from diverse fields such as AI, robotics, control engineering, and operations research can be formulated as MDTs. However, some care has to be taken in applying the MDT framework because it makes the strong assumption that the agent's perception returns the state of the environment, an assumption that may not be satisfied in some real-world tasks with embedded agents (Chapter **??** discusses this is greater detail).

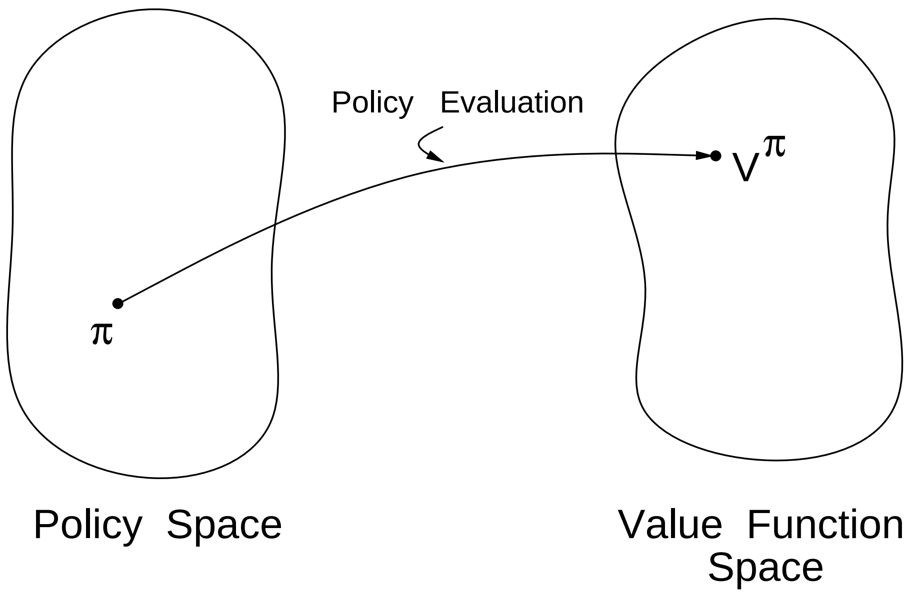

Figure 2.2 The Policy Evaluation Problem. This figure shows the two spaces of interest in solving MDTs: the policy space and the value function space. Evaluating a policy π maps it into a vector of real numbers V^π. Each component of V^π is the infinite-horizon discounted sum of payoffs received by an agent when it follows that policy and starts in the state associated with that component.

2.4.1 Prediction and Control

For infinite-horizon MDTs the two fundamental questions of prediction and control can be reduced to that of solving fixed-point equations.

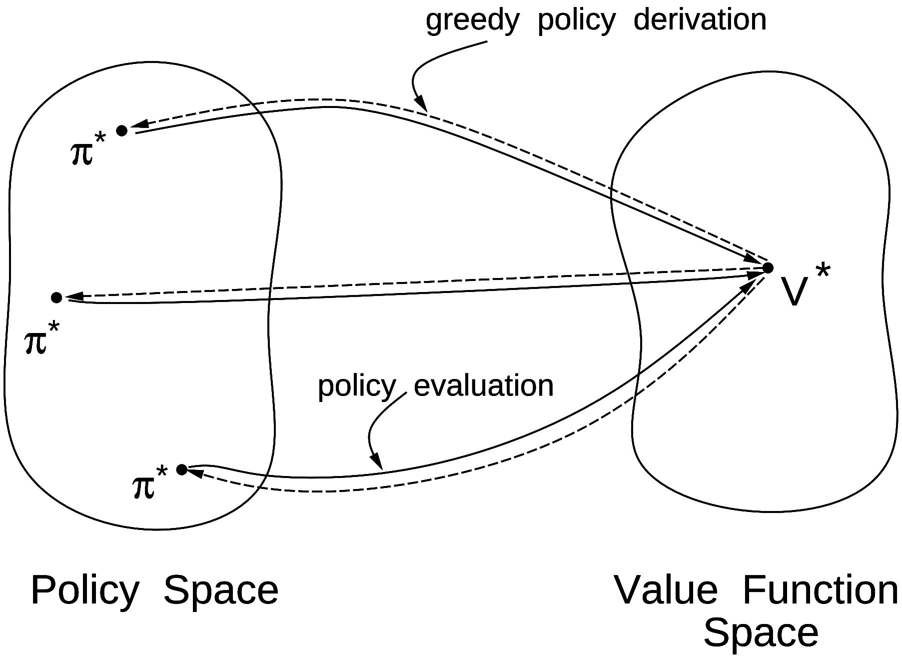

Figure 2.3 The Optimal Control Problem. Solving the optimal control problem requires finding a policy π^* that when evaluated maps onto a value function V^* that is componentwise larger than the value function of any other policy. The optimal policies are the only stationary policies that are greedy with respect to the unique optimal value function V^*.

- **Policy Evaluation:** The prediction problem for an MDT, shown in Figure 2.2, is called *policy evaluation* and requires computing the vector V^π for a fixed policy π. Let R^π be the vector of payoffs under policy π and let $[P]^\pi$ be the transition probability matrix under policy π. It can be shown that the following system of linear fixed-point equations of size $|X|$, written in vector form:

$$V \quad = \quad R^\pi + \gamma [P]^\pi V \tag{2.3}$$

  always has a unique solution, and that the solution is V^π, under the assumption that R^π is finite (Ross [46]).

- **Optimal Control:** The control problem for an MDT, shown in Figure 2.3, is that of finding an optimal control policy π^*. The search for an optimal policy has to be conducted only in the set of stationary policies, denoted $\mathcal{P}$, that is of size $|A|^{|X|}$. The value function for an optimal policy π^* is called the optimal value function and is denoted V^*. There may be more than one optimal policy, but the optimal value function is always unique (Ross [46]). It is known that the following system of nonlinear fixed-point equations, $\forall x \in X$:

$$V(x) \quad = \quad \max_{a \in A}(R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V(y)), \tag{2.4}$$

of dimension $|X|$ always has a unique solution, and that the solution is V^*, under the assumption that all the payoff vectors are finite. The set of recurrence relations, $\forall x \in X$,

$$V^*(x) \quad = \quad \max_{a \in A}(R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V^*(y)),$$

is known in the DP literature as the Bellman equation for infinite-horizon MDTs (Bellman [14]).

A policy π is *greedy* with respect to any finite value function V if it prescribes to each state an action that maximizes the sum of the immediate payoff and the discounted expected value of the next state as determined by the value function V. Formally, π is greedy with respect to V iff $\forall x \in X$, and $\forall a \in A$:

$$R^{\pi(x)}(x) + \gamma \sum_{y \in X} P^{\pi(x)}(x,y) \quad \geq \quad R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V(y).$$

Any policy that is greedy with respect to the optimal value function is optimal (see Figure 2.3). Therefore, once the optimal value function is known, an optimal policy for infinite-horizon MDTs can be determined by the following relatively straightforward computation:[7]

$$\pi^*(x) \quad = \quad \text{argmax}_{a \in A} \left[R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V^*(y) \right]. \tag{2.5}$$

In fact, solving the optimal control problem has come to mean solving for V^* with the implicit assumption that π^* is derived by using Equation 2.5.

---

[7]The problem of determining the optimal policy given V^* is a single-stage MDT with $(R^\pi + \gamma[P]^\pi V^*)$ playing the role of the immediate payoff function R^π (cf. Equation 2.2). If the size of the action set A is large, finding the best action in a state can itself become computationally expensive and is the subject of current research (e.g., Gullapalli [27]).

2.5 Conclusion

The MDT framework offers many attractions for formulating learning tasks faced by embedded agents. It deals naturally with the perception-action cycle of embedded agents, it requires very little prior knowledge about an optimal solution, it can be used for stochastic and non-linear environments, and most importantly it comes with a great deal of theoretical and empirical results developed in the fields of control theory and operations research. Therefore, the MDT framework incorporates many, but not all, of the concerns of AI researchers as their emphasis shifts towards studying complete agents in real-life environments. This dissertation will deal exclusively with learning tasks that can be formulated as MDTs. In particular, the next two chapters will present theoretical results about the application of DP and RL algorithms to abstract MDTs without reference to any real application. Subsequent chapters will use applications to test new RL architectures that address more practical concerns.

CHAPTER 3

SOLVING MARKOVIAN DECISION TASKS: DYNAMIC PROGRAMMING

This chapter serves multiple purposes: it presents a framework for describing algorithms that solve Markovian decision tasks (MDTs), it uses that framework to survey classical dynamic programming (DP) algorithms, it presents a new asynchronous DP algorithm that is based on policy iteration, and it presents a new family of DP algorithms that find solutions that are more robust than the solutions found by conventional DP algorithms. Convergence proofs are also presented for the new DP algorithms. The framework developed in this chapter is also used in the next chapter to describe reinforcement learning (RL) algorithms and serves as a vehicle for highlighting the similarities and the differences between DP and RL. Section 3.2 is solely a review while Sections 3.3 and 3.4 present new algorithms and results obtained by this author.

3.1 Iterative Relaxation Algorithms

In Chapter 2 it was shown that the prediction and control problems for embedded agents can be reduced to the problem of solving the following systems of fixed-point equations, $\forall x \in X$:

$$\begin{aligned} \text{Policy Evaluation } (\pi): \quad V(x) &= R^{\pi(x)}(x) + \gamma \sum_{y \in X} P^{\pi(x)}(x,y)V(y) && (3.1) \\ \text{Optimal Control:} \quad V(x) &= \max_{a \in A}(R^{a}(x) + \gamma \sum_{y \in X} P^{a}(x,y)V(y)). && (3.2) \end{aligned}$$

This chapter and the next focuses on algorithms that produce sequences of approximations to the solution value function — V^{π} for Equation 3.1 and V^{*} for Equation 3.2 — by iterating an "update" equation that takes the following general form:

$$\textit{new approximation} \;=\; \textit{old approximation} + \textit{rate parameter}\,(\ \textit{new estimate}$$

$$-\textit{old approximation}\ ), \tag{3.3}$$

where the *rate parameter* defines the proportion in which a *new estimate* of the solution value function and the *old approximation* are mixed together to produce a *new approximation.* This new approximation becomes the old approximation at the next iteration of the update equation. Such algorithms are called *iterative relaxation* algorithms.

The sequence of value functions produced by iterating Equation 3.3 is indexed by the iteration number and denoted $\{V_i\}$. Therefore the update equation can be written as follows:

$$V_{i+1} \quad = \quad V_i + \rho_i(U(V_i) - V_i) \tag{3.4}$$

where ρ_i is a relaxation parameter, V_i is the approximation of the solution value function after the $(i-1)^{st}$ iteration, and $U : V_i \rightarrow \mathcal{R}^{|X|}$ is an operator that produces a new estimate of the solution value function by using the approximate value function V_i.[1] In general, the value function is a vector, and at each iteration an arbitrary subset of its components can be updated. Therefore, it is useful to write down each component of the update equation as a function of the state associated with that component of the vector. Let the component of the operator U corresponding to state x be denoted $U_x : V_i \rightarrow \mathcal{R}$. The update equation for state x is

$$V_{i+1}(x) \quad = \quad V_i(x) + \rho_i(x)(U_x(V_i) - V_i(x)) \tag{3.5}$$

where the relaxation parameter is now a function of x.

In this chapter classical DP algorithms are derived as special cases of Equation 3.5. In the next chapter RL algorithms, such as Sutton's temporal differences (TD) and Watkins' Q-learning, will also be derived as special cases of Equation 3.5. The differences among the various iterative algorithms for solving MDTs are: 1) the definition of the operator U, and 2) the order in which the state-update equation is applied to the states of the MDT.

---

[1] More generally the operator U could itself be a function of the iteration number i. However, for the algorithms discussed in this dissertation, U is assumed fixed for all iterations.

Following Barto [6], it is convenient to represent MDTs, both conceptually and pictorially, as directed stochastic graphs. Figure 3.1 shows the outgoing transitions for an arbitrary state x from a stochastic graph representation of an MDT that in turn is an abstraction of some real-world environment. The nodes represent states and the transitions represent possible outcomes of executing actions in a state. A transition is directed from a *predecessor* state, e.g., x, to a *successor* state, e.g., y (Figure 3.1). Because the problem can be stochastic, a set of outgoing transitions from a state can have the same action label. Each transition has a payoff and a probability attached to it (not shown in Figure 3.1). For any state-action pair the probabilities across all possible transitions sum to one. Figure 3.1 shows that there can be more than one transition between any two states.

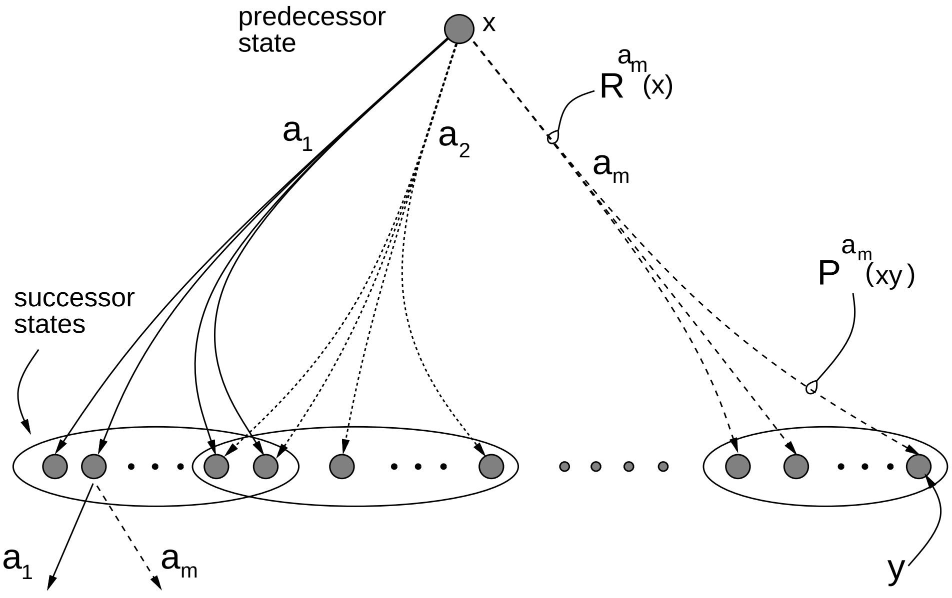

Figure 3.1 Directed Stochastic Graph Representation of an MDT. This figure shows a fragment of an MDT. The nodes represent states and the arcs represent transitions that are labeled with actions.

In this chapter and the next, stochastic graphs resembling Figure 3.1 will be used to help describe the update equations, and in particular to describe the computation performed by the operator U. A common feature of all the algorithms presented in this dissertation is that the operator U is *local*, in that it produces a new estimate of the value of a state x by accessing information only about states that can be reached from x in one transition.[2]

3.1.1 Terminology

This section presents terminology that will be used throughout this dissertation to describe iterative relaxation algorithms for solving MDTs. The state transition probabilities and the payoff function constitute a model of the environment.

- **Synchronous**: An algorithm is termed synchronous if in every $k|X|$ applications of the state-update equation the value of every state in set X is updated exactly k times. If the values of all the states are updated simultaneously, called Jacobi iteration, the algorithm given in Equation 3.5 can be written in the vector form of Equation 3.4. If the states are updated in some fixed order and the operator U always uses the most recent approximation to the value function, the algorithm is said to perform a Gauss-Sidel iteration.

- **Asynchronous:** Different researchers have used different models of asynchrony in iterative algorithms (e.g., Bertsekas and Tsitsiklis [16]). In this dissertation the term asynchronous is used for algorithms that place no constraints on the order in which the state-update equation is applied, except that in the limit the value of each state will be updated infinitely often. The set of states whose values are updated at iteration i is denoted S_i (as in Barto *et al.* [8]).

- **On-line** An on-line algorithm is one that not only learns a value function but also simultaneously controls a real environment. An on-line algorithm faces the *tradeoff between exploration and exploitation* because it has to choose between executing actions that allow it to improve its estimate of the value function and executing actions that return high payoffs.

---

[2]One can define both DP and RL algorithms that use operators that do more than a one-step search and access information about states that are not one-step neighbors, e.g., the multi-step Q-learning of Watkins [58]. Most of the theoretical results stated in this dissertation will also hold for algorithms with multi-step operators.

- **Off-line** The term off-line implies that the algorithm is using simulated experience with a model of the environment. Off-line algorithms do not face the exploration versus exploitation tradeoff because they design the control solution before applying it to the real environment.

- **Model-Based** A model-based algorithm uses a model of the environment to update the value function, either a model that is given a priori, or one that is estimated on-line using a system identification procedure. Note that the model does not have to be correct, or complete. A model-based algorithm can select states in the model in ways that need not be constrained by the dynamics of the environment, or the actions of the agent in the real environment. Algorithms that estimate a model on-line and do model-based control design on the estimated model are also called *indirect* algorithms (see, e.g., Astrom and Wittenmark [2], and Barto *et al.* [12]).

- **Model-Free** A model-free algorithm does not use a model of the environment and therefore does not have access to the state transition matrices or the payoff function for the different policies. A model-free algorithm is limited to applying the state-update equation to the state of the real environment. Model-free algorithms for learning control are also referred to as *direct* algorithms.

It is not possible to devise algorithms that satisfy an arbitrary selection of the above characteristics; the constraints listed in Table 3.1 apply. For on-line algorithms

Table 3.1 Constraints on Iterative Relaxation Algorithms

Algorithm Type		*Characteristics*
off-line	$\Rightarrow$	model-based
synchronous	$\Rightarrow$	off-line, and therefore model-based
on-line	$\Rightarrow$	asynchronous
model-free	$\Rightarrow$	on-line, and therefore asynchronous

that are model-free it may be difficult to satisfy the conditions required for convergence of asynchronous algorithms because it may be difficult, or even impossible, to ensure that every state is visited infinitely often. In practice, either restrictive assumptions are placed on the nature of the MDT, such as ergodicity, or appropriate constraints are imposed on the control policy followed while learning, such as using probabilistic policies (Sutton [54]). A model-based relaxation algorithm can be either synchronous or asynchronous. An algorithm that does not require a model of the environment can always be applied to a task in which a model is available simply by using the model to simulate the real environment. In general, an algorithm that needs knowledge of the transition probabilities cannot be applied without a model of the environment.

3.2 Dynamic Programming

DP is a collection of algorithms based on Bellman's [14] powerful principle of optimality which states that "an optimal policy has the property that whatever the initial state and action are, the remaining actions must constitute an optimal policy with regard to the state resulting from the first action." The optimal control equation 3.2 can be derived directly from Bellman's principle. Part of the motivation for this section is to develop a systematic "recipe-like" format for describing DP-based algorithms, and the reader will notice its repeated use throughout this dissertation to describe both old and novel algorithms.

3.2.1 Policy Evaluation

As shown by Equation 3.1, evaluating a fixed stationary policy π requires solving a linear system of equations. Define the successive approximation *backup* operator, B^π, in vector form as follows:

$$B^\pi(V) \quad = \quad R^\pi + \gamma[P]^\pi V. \tag{3.6}$$

From Equation 3.1, V^π is the unique solution to the following vector-equation

$$V = B^\pi(V). \tag{3.7}$$

The backup operator for state x is

$$B^\pi_x(V) \quad = \quad R^{\pi(x)}(x) + \gamma \sum_{y \in X} P^{\pi(x)}(x,y)V(y). \tag{3.8}$$

Operator B_x^π is called a backup operator because it "backs up" the value of the successor states (y's) to produce a new estimate of the value of the predecessor state x. Operator B^π requires a model because it requires knowledge of the state transition probabilities.

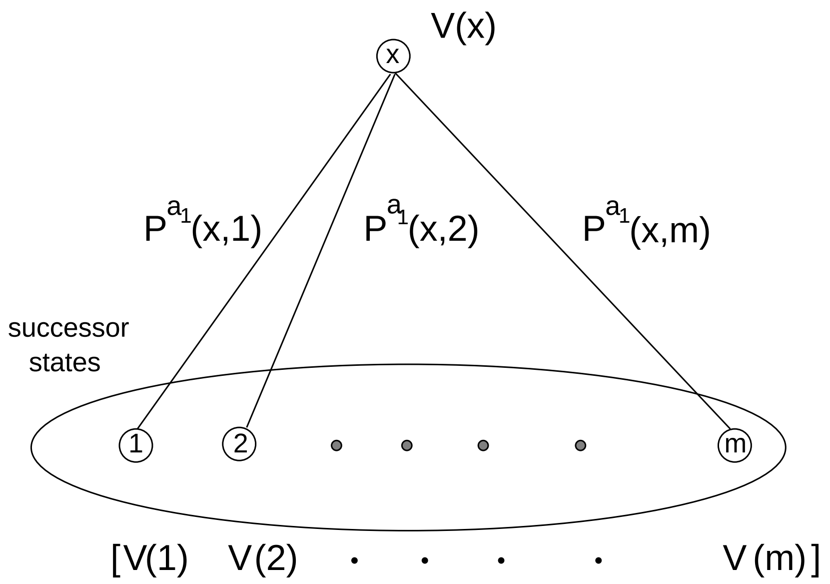

Figure 3.2 Full Policy Evaluation Backup. This figure shows the state transitions on executing action $\pi(x)$ in state x. Doing a full backup requires knowledge of the transition probabilities.

The computation involved in B_x^π can be explained with the help of Figure 3.2 which only shows the transitions for action $\pi(x) = a_1$ from state x. Operator B_x^π involves adding the immediate payoff to the expected value of the next state that can result from executing action $\pi(x)$ is state x. Operator B^π is a *full* backup operator because computing the expected value of the next state involves accessing information about *all of the possible next states* for state-action pair $(x, \pi(x))$. Operator B^π can be applied synchronously or asynchronously to yield the following algorithms:

(Jacobi) Synchronous successive approximation:

$$V_{i+1} = B^\pi(V_i), \qquad \text{and}$$

Asynchronous successive approximation:

$$V_{i+1}(x) = \begin{cases} B_x^\pi(V_i) & \forall x \in S_i. \\ V_i(x) & \forall x \in (X - S_i). \end{cases} \tag{3.9}$$

Equation 3.9 takes the form of the general iterative relaxation equation 3.5 with $\rho_x = 1$ and $U_x = B_x^\pi$.

Convergence: If $\gamma < 1$, the operator B^π is a contraction operator because $\forall V \in \mathcal{R}^{|X|}$, $||B^\pi(V) - V^\pi||_\infty \leq \gamma ||V - V^\pi||_\infty$, where $||.||_\infty$ is the l_∞ or max norm. Therefore, the synchronous successive approximation algorithm converges to V^π by the application of the contraction mapping theorem (see, e.g., Bertsekas and Tsitsiklis [16]). Convergence can be proven for asynchronous successive approximation by applying the asynchronous convergence theorem of Bertsekas and Tsitsiklis [16].

3.2.2 Optimal Control

Determining the optimal value function requires solving the nonlinear system of equations 3.2. Define the nonlinear value iteration *backup* operator, B, in vector-form as:

$$B(V) \quad = \quad \max_{\pi \in \mathcal{P}} (R^\pi + \gamma [P]^\pi V), \tag{3.10}$$

where throughout this dissertation the max over a set of vectors is defined to be the vector that results from a componentwise max. From Equation 3.2, the optimal value function V^* is the unique solution to the equation $V = B(V)$. The x-component of B is written as follows:

$$B_x(V) \quad = \quad \max_{a \in A} (R^a(x) + \gamma \sum_{y \in X} P^a(x,y) V(y)). \tag{3.11}$$

Operator B also requires a model because it assumes knowledge of the state transition probabilities.

The computation involved in operator B can be explained with the help of Figure 3.3 which shows state x and its one-step neighbors. Operator B_x involves computing the maximum value over all actions of the sum of the immediate payoff and the expected value of the next state for each action. Operator B_x is a *full* backup operator because it involves accessing all of the possible next states for all actions in state x. As in the policy evaluation case, the operator itself can be applied

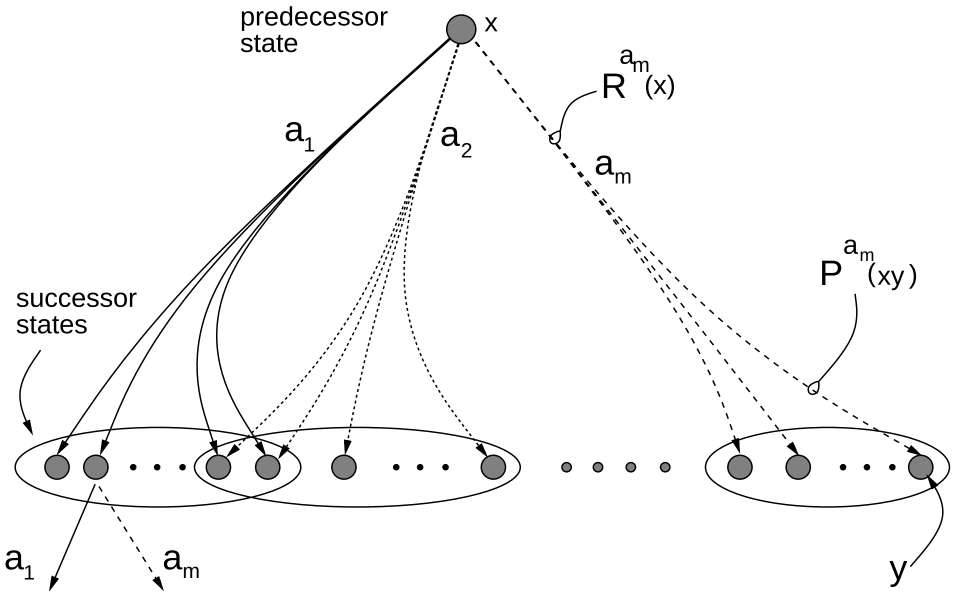

Figure 3.3 Full Value Iteration Backup. This figure shows all the actions in state x. Doing a full backup requires knowledge of the state transition probabilities.

synchronously or asynchronously to yield the following two algorithms:

(Jacobi) Synchronous Value Iteration:

$$V_{i+1} \quad = \quad B(V_i), \qquad\qquad \text{and}$$

Asynchronous Value Iteration:

$$V_{i+1}(x) = \begin{cases} B_x(V_i) & \forall x \in S_i, \\ V_i(x) & \forall x \in (X - S_i). \end{cases} \tag{3.12}$$

Equation 3.12 takes the form of the general iterative relaxation equation 3.5 with $\rho_x = 1$ and $U_x = B_x$. The asynchronous value iteration algorithm allows the agent to sample the state space by randomly selecting the state to which the update equation is applied.

Convergence: For $\gamma < 1$, B is a contraction operator because $\forall V \in \mathcal{R}^{|X|}$, $||B(V) - V^*||_\infty \leq \gamma ||V - V^*||_\infty$. Therefore, the synchronous value iteration algorithm can be proven to converge to V^* by the application of the contraction mapping theorem. Convergence can be proven for asynchronous value iteration with $\gamma < 1$ by applying the asynchronous convergence theorem of Bertsekas and Tsitsiklis [16]. The rate of convergence is governed by γ and the second largest eigenvalue, λ_2, of the transition probability matrix for the optimal policy. The smaller the value of γ and the smaller the value of λ_2, the faster the convergence.

Stopping Conditions: For Jacobi value iteration, $V_{i+1} = B(V_i)$, it is possible to define the following error bounds (Bertsekas [15]):

$$\begin{aligned} \underline{h_i} &= \frac{\gamma}{1-\gamma} \min_{x \in X} [B_x(V_i) - V_i(x)], \\ \overline{h_i} &= \frac{\gamma}{1-\gamma} \max_{x \in X} [B_x(V_i) - V_i(x)]. \end{aligned}$$

such that $\forall x \in X$, $B_x(V_i) + \underline{h_i} \leq V^*(x) \leq B_x(V_i) + \overline{h_i}$. The maximum and the minimum change in the value function at the i^{th} iteration bounds the max-norm distance between V_i and V^*. For asynchronous value iteration, these error bounds can be computed using the last visit to each state. Ensuring convergence to V^* may require an infinite number of iterations. In practice, value iteration can be terminated when $(\overline{h_i} - \underline{h_i})$ is small enough.

3.2.3 Discussion

This section presented a review of classical DP algorithms by casting them as iterative relaxation algorithms in a framework that allowed us to highlight the two aspects that differ across the various algorithms: the nature of the backup operator and the order in which it is applied to the states in the environment. The main difference between solving the policy evaluation and the optimal control problems is that in the first case a linear backup operator is employed while in the second a nonlinear backup operator is needed. In both problems the advantage of moving from synchronous to asynchronous is that the asynchronous algorithm can sample in predecessor-state space. The subject of the next section is a novel algorithm for solving the optimal control problem that is more finely asynchronous than the algorithms reviewed in this section because it can sample both in predecessor-state space and in action space.

CHAPTER 4

SOLVING MARKOVIAN DECISION TASKS: REINFORCEMENT LEARNING

The main contribution of this chapter is in establishing a hitherto unknown connection between stochastic approximation theory and reinforcement learning (RL). The stochastic approximation framework for RL provides a fairly complete theory of asymptotic convergence with lookup-table representations for some well known RL algorithms. A mixture of theory and empirical results are also provided to address partially the following question: which method, RL or dynamic programming (DP), should be applied to a particular problem if given a choice? The stochastic approximation framework leaves open several theoretical questions of great practical interest, and the second half of this chapter identifies and addresses some of them.

4.1 A Brief History of Reinforcement Learning

Early research in RL developed *non-associative* algorithms for solving single-stage Markovian decision tasks (MDTs) in environments with only one state, and had its roots in the work of psychologists working on mathematical learning theory (e.g., Bush and Mosteller [20]), and in the work of learning automata theorists (e.g., Narendra and Thatachar [40]). Later Barto *et al.* [7] developed an associative RL algorithm, they called the A_{RP} algorithm, that solves single-stage MDTs with multiple-state environments. Single-stage MDTs do not involve the temporal credit assignment problem. Therefore algorithms for solving single-stage MDTs are unrelated to DP algorithms (except in the trivial sense). In the early 1980s, in a landmark paper Barto *et al.* [11] described a technique for addressing the temporal credit assignment problem that culminated in Sutton's [53] paper on a class of techniques he called temporal difference (TD) methods (see also Sutton [52]).

In the late 1980s, Watkins [58] observed that the TD algorithm solves the linear policy evaluation problem for multi-stage MDTs (see, also Dayan [21], Barnard [3]).

Further, Watkins developed the Q-learning algorithm for solving MDTs and noted the approximate relationship between TD, Q-learning, and DP algorithms (see also Barto *et al.* [13, 8], and Werbos [61]). In this chapter, the connection between RL and MDTs is made precise. But first, a brief detour has to be taken to explain the Robbins-Monro [45] stochastic approximation method for solving systems of equations. Stochastic approximation theory forms the basis for connecting RL and DP.

4.2 Stochastic Approximation for Solving Systems of Equations

Consider the problem of solving a scalar equation $G(V) = 0$, where V is a scalar variable. The classical Newton-Raphson method for finding roots of equations iterates the following recursion:

$$V_{k+1} \quad = \quad V_k - G(V_k)/G'(V_k), \tag{4.1}$$

where V_1 is some initial guess at the root, V_k is the approximation after $k-1$ iterations, and $G'(V_k)$ is the first derivative of the function evaluated at V_k. It is known that the sequence $\{V_k\}$ converges under certain conditions.

Now suppose that the function G is unknown and therefore its derivative cannot be computed. Further suppose that for any V we can observe $Y(V) = G(V) + \epsilon$, where ϵ represents some random error with mean zero and variance $\sigma^2 > 0$. Robbins and Monro [45] suggested using the following recursion

$$V_{k+1} \quad = \quad V_k - \rho_k Y(V_k), \tag{4.2}$$

where $\{\rho_k\}$ are positive constants, such that $\sum \rho_k^2 < \infty$, and $\sum \rho_k = \infty$, and proved convergence in probability for Equation 4.2 to the root of G under the conditions that $G(V) > 0$ if $V > 0$, and $G(V) < 0$ if $V < 0$. The condition that ϵ have zero mean implies that $E\{Y(V_n)\} = G(V_n)$, i.e., $Y(V_n)$ is an unbiased sample of the function G at V_n. It is critical to note that $Y(V_n)$ is not an unbiased estimate of the root of $G(V)$, but only an unbiased estimate of the value of function G at V_n, the current estimate of the root. Equation 4.2 will play an essential role in establishing the connection between RL and DP in Section 4.3.

Following Robbin and Monro's work, several authors extended their results to multi-dimensional equations and derived convergence with probability one under weaker conditions (Blum [17], Dvoretzky [23], Schmetterer [48]). Appendix B presents a theorem by Dvoretzky [23] that is more complex but more closely related to the material presented in the following sections.

4.3 Reinforcement Learning Algorithms

This section uses Equation 4.2 to derive stochastic approximation algorithms to solve the policy evaluation and optimal control problems. In addition, the general framework of iterative relaxation algorithms developed in Chapter 3 is used to highlight the similarities and differences between RL and DP. As in Chapter 3 the aspects of the algorithms that are noteworthy are 1) the definition of the backup operator, and 2) the order in which the states are updated.

4.3.1 Policy Evaluation

The value function for policy π, V^π, is the unique solution to the following policy evaluation equation:

$$V \quad = \quad R^\pi + \gamma [P]^\pi V \tag{4.3}$$

Define the multi-dimensional function $G(V) \doteq V - (R^\pi + \gamma [P]^\pi V)$. The dimension corresponding to state x of function $G(V)$ can be written as follows:

$$G_x(V) \quad = \quad V(x) - (R^{\pi(x)}(x) + \gamma \sum_{y \in X} P^{\pi(x)}(x, y) V(y)).$$

The solution to the policy evaluation problem, V^π, is the unique root of $G(V)$.

Assume that the agent does not have access to a model of the environment and therefore cannot use the matrix $[P]^\pi$ in its calculations. Define $Y_x(V) = V(x) - (R^{\pi(x)}(x) + \gamma V(y))$, where the next state $y \in X$ occurs with probability $P^{\pi(x)}(x, y)$. Define the random matrix $[T]^\pi$ as an $|X| \times |X|$ matrix whose rows are unit vectors, with a 1 in column j of row i with probability $P^{\pi(i)}(i, j)$. Clearly, $E\{[T]^\pi\} = [P]^\pi$, and the vector form of $Y_x(V)$ can be written as follows: $Y(V) = V - (R^\pi + \gamma [T]^\pi V)$. Note that $E\{Y(V)\} = G(V)$.

A stochastic approximation algorithm to obtain the root of $G(V)$ can be derived from Equation 4.2 as follows:

(Jacobi) Synchronous Robbins-Monro Policy Evaluation:

$$\begin{aligned} V_{k+1} &= V_k - \rho_k Y(V_k) \\ &= V_k - \rho_k (V_k - (R^\pi + \gamma [T]^\pi V_k)), \end{aligned}$$

or for state x,

$$V_{k+1}(x) = V_k(x) - \rho_k(x)(V_k(x) - (R^{\pi(x)}(x) + \gamma V_k(y))) \tag{4.4}$$

where $\rho_k(x)$ is a relaxation parameter for state x. To compute $Y_x(V)$, the agent does not need to know the transition probabilities; as depicted in Figure 4.1 it can simply execute action $\pi(x)$ in the real environment and then observe the immediate payoff $R^{\pi(x)}(x)$ as well as the next state y. The value of the next state y can be retrieved from the data structure storing the V values.

The algorithm defined in Equation 4.4 is also an iterative relaxation algorithm of the form defined in Equation 3.5 with a backup operator $\mathcal{B}^\pi_x(V) = R^{\pi(x)}(x) + \gamma V(y)$. The full backup operator of the successive approximation (DP) algorithm: $B^\pi_x(V) = R^{\pi(x)}(x) + \gamma \sum_{y \in X} P^{\pi(x)}(x,y)V(y)$, computes the expected value of the next state to derive a new estimate of V^π. The operator $\mathcal{B}^\pi$, on the other hand, samples one next-state from the set of possible next states and uses the sampled state's value to compute a new estimate of V^π (shown in Figure 4.1). Therefore $\mathcal{B}^\pi$ is called the *sample* backup operator. Note that $\forall\, x \in X$, $E\{\mathcal{B}^\pi_x\}(V) = B^\pi_x(V)$. The RL algorithm represented in Equation 4.4 can be derived directly from the successive approximation (DP) algorithm (Chapter 3) by replacing the full backup operator, B^π, by a random, but unbiased, sample backup operator, $\mathcal{B}^\pi$.

Equation 4.4 is a Jacobi iteration, and its asynchronous version can be written as follows:

Asynchronous Robbins-Monro Policy Evaluation:

$$V_{k+1}(x) = V_k(x) - \rho_k(V_k(x) - (R^{\pi(x)}(x) + \gamma V_k(y))); \qquad \text{for } x \in S_k$$

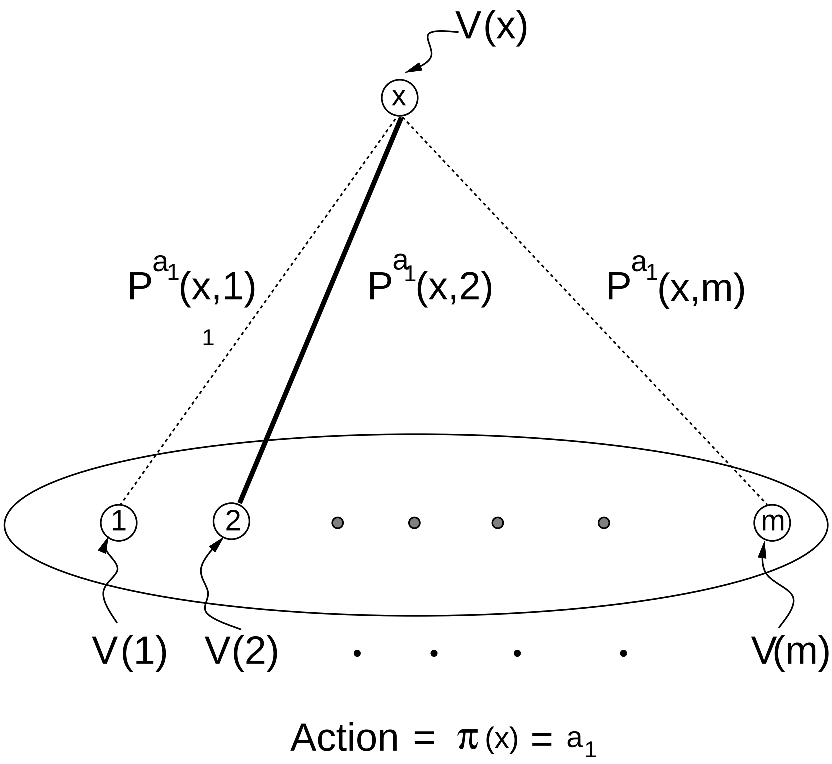

Figure 4.1 Policy Evaluation: Sample Backup. This figure shows the possible transitions for action $\pi(x)$ in state x. The transition marked with a solid line represents the actual transition when action $\pi(x)$ is executed once in state x. Therefore, the only information that a sample backup can consider is the payoff along that transition and the value of state 2.

$$V_{k+1}(z) \quad = \quad V_k(z); \qquad \text{for } z \in (X - S_k). \tag{4.5}$$

The on-line version of Equation 4.5, i.e., where the sets $S_t = \{x_t\}$ contain only the current state of the environment is identical to the TD(0) algorithm commonly used in RL applications (Sutton [53]). An asymptotic convergence proof for the synchronous algorithm (Equation 4.4) can be easily derived from Dvoretzky's theorem presented in Appendix B. However, more recently, Jaakkola, Jordan and Singh [31] have derived the following theorem for the more general asynchronous case (of which the synchronous algorithm is a special case) by extending Dvoretzky's stochastic approximation results:

Theorem 5: (Jaakkola, Jordan and Singh [31]) For finite MDPs, the algorithm defined by Equation 4.5 converges with probability one to V^π under the following conditions:

1. every state in set X is updated infinitely often,

2. V_0 is finite, and

3. $\forall x \in X$; $\sum_{i=0}^{\infty} \rho_i(x) = \infty$ and $\sum_{i=0}^{\infty} \rho_i^2(x) < \infty$.

Sutton [53] has defined a family of temporal difference algorithms called TD(λ), where $0 \leq \lambda \leq 1$ is a scalar parameter. The TD algorithm discussed above is just one algorithm from that family, specifically the TD(0) algorithm. See Jaakkola *et al.* [31] for a discussion of the connection between stochastic approximation and the general class of TD(λ) algorithms. Also, see Section **??** in Chapter **??** of this dissertation for further discussion about TD(λ) algorithms. Hereafter, the name TD will be reserved for the TD(0) algorithm.

4.3.2 Optimal Control

The optimal value function, V^*, is the unique solution of the following system of equations: $\forall x$,

$$V(x) \quad = \quad \max_{a \in A}(R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V(y)). \tag{4.6}$$

Define the component corresponding to state x of a multi-dimensional function $G(V)$ as follows:

$$G_x(V) \quad = \quad V(x) - \max_{a \in A}(R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V(y))$$

The solution to the optimal control problem, V^*, is the unique root of the nonlinear function $G(V)$.

Again, as for policy evaluation, assume that the agent does not have access to a model of the environment and therefore cannot use the transition probabilities in its calculations. The attempt to define a stochastic approximation algorithm for solving the optimal control problem by following the procedure used in Section 4.3.1 fails because the function $G(V)$ is nonlinear. Specifically, if $Y_x(V) \doteq V(x) - \max_{a \in A}(R^a(x) + \gamma V(y))$, then $E\{Y_x(V)\} \neq G_x(V)$, because *the expectation and the* max *operators do not commute*. The solution to this impasse lies in a clever trick employed by Watkins in his Q-learning algorithm.

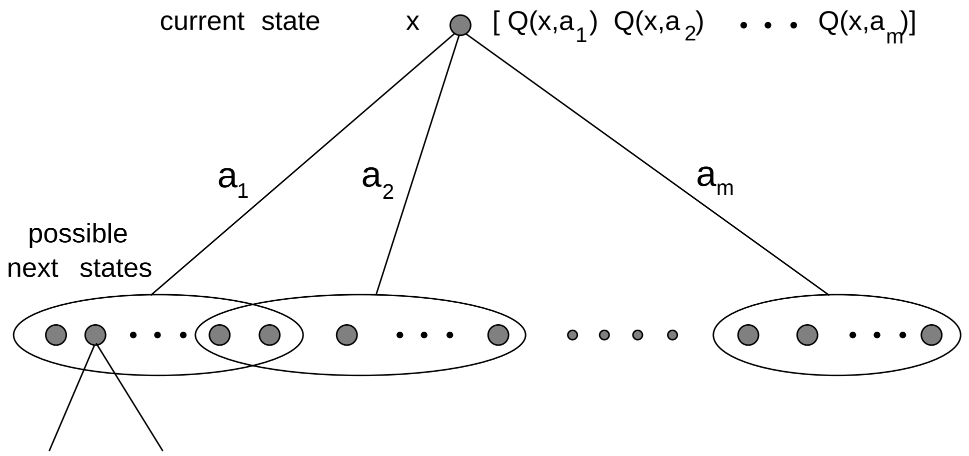

Figure 4.2 Q-values. This figure shows that instead of keeping just one value $V(x)$ for each state x, one keeps a set of Q-values, one for each action in a state.

Watkins proposed rewriting Equation 4.6 in the following expanded notation: instead of keeping one value, $V(x)$, for each state $x \in X$, the agent stores $|A|$ values for each state, one for each action possible in that state. Watkins proposed the notation $Q(x,a)$ as the Q-value for state-action pair (x,a). Figure 4.2 shows the expanded representation. The optimal control equations can be written in Q-notation as:

$$Q(x,a) \quad = \quad R^a(x) + \sum_{y \in X} P^a(x,y)(\max_{a' \in A} Q(y,a')); \qquad \forall (x,a) \in (X \times A). \quad (4.7)$$

The optimal Q-values, denoted Q^*, are the unique solution to Equation 4.7, and further $V^*(x) = \max_{a \in A} Q^*(x,a)$.

An iterative relaxation DP algorithm can be derived in terms of the Q-values by defining a new local full backup operator,

$$B_x^a(Q) \quad = \quad R^a(x) + \gamma \sum_{y \in X} P^a(x,y)(\max_{a' \in A} Q(y,a')). \qquad (4.8)$$

Operator $B_x^a(Q)$ requires a model of the environment because it involves computing the expected value of the maximum Q-values of all possible next states for state-action pair (x,a). Let $(X,A)_k \in (X \times A)$ be the set of state-action pairs updated at iteration k. An algorithm called Q-value iteration, by analogy to value iteration, can be defined by using the new operator $B(Q)$ as follows:

Asynchronous Q-value Iteration:

$$Q_{k+1}(x,a) \quad = \quad B_x^a(Q_k)$$

$$
\begin{aligned}
&= R^a(x) + \gamma \sum_{y \in X} P^a(x,y)(\max_{a' \in A} Q_k(y,a')); \qquad \text{for } (x,a) \in (X,A)_k \\
Q_{k+1}(z,b) &= Q_k(z,b); \qquad \text{for } (z,b) \in ((X \times A) - (X,A)_k). \qquad (4.9)
\end{aligned}
$$

One advantage of asynchronous Q-value iteration over asynchronous value iteration is that it allows the agent to select randomly both the state as well as the action to be updated (asynchronous value iteration only allows the agent to select the state randomly). A convergence proof for synchronous Q-value iteration takes the same form as a proof of convergence for synchronous value iteration because the operator $B(Q)$ is also a contraction (see Appendix ?? for proof). For the asynchronous case the proof requires the application of the asynchronous convergence theorem in Bertsekas and Tsitsiklis [16].

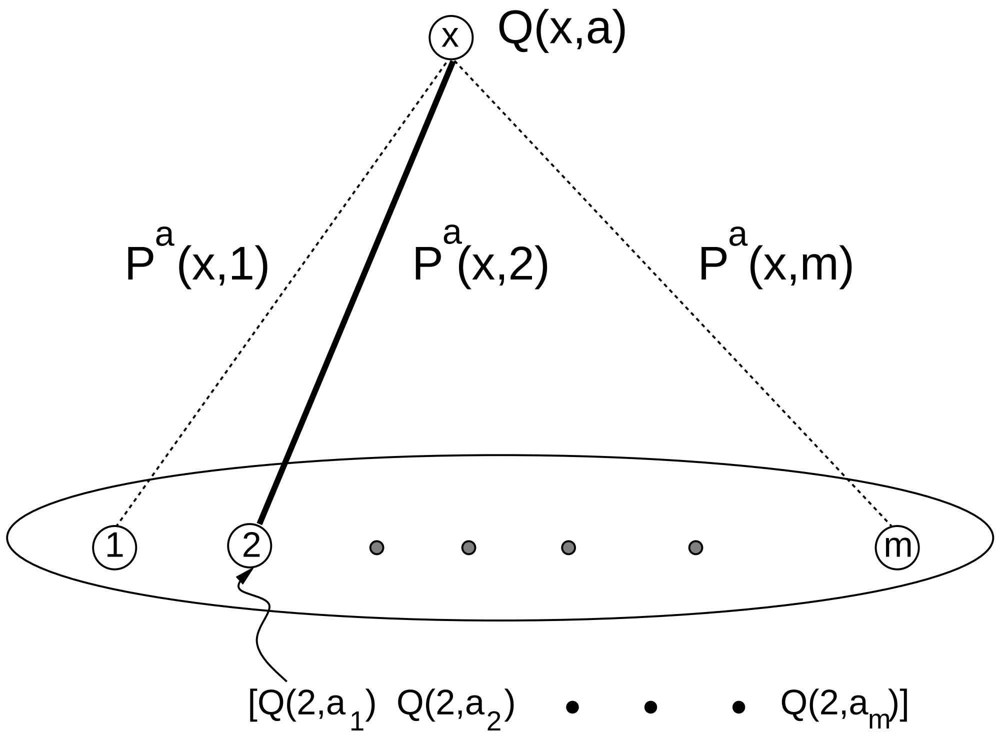

Figure 4.3 Sample Backup Operator. This figure shows the possible transitions for action a in state x. The transition marked in a solid line represents the actual transition that occurred on one occasion. The only information available for a sample backup then is the payoff along that transition and the Q-values stored for state 2.

Another advantage of using Q-values is that it becomes possible to do stochastic approximation by sampling not only the state-action pair but the next state as well.

Define $G_x^a(Q) = R^a(x) + \gamma \sum_{y \in X}(P^a(x,y) \max_{a' \in A} Q(y,a'))$. Further, define $Y_x^a(Q) = Q(x,a) - (R^a(x) + \gamma \max_{a' \in A} Q(y,a'))$, where the next state y occurs with probability $P^a(x,y)$. Note that $E\{Y_x^a(Q)\} = G_x^a(Q)$. Following the stochastic approximation recipe defined in Equation 4.2, the following algorithm can be defined:

Asynchronous Robbins-Monro Q-value Iteration:

$$
\begin{aligned}
Q_{k+1}(x,a) &= Q_k(x,a) - \rho_k(x,a) Y_x^a(Q_k); \qquad \text{for } (x,a) \in (X,A)_k) \\
&= Q_k(x,a) - \rho_k(x,a)(Q_k(x,a) \\
&\qquad -(R^a(x) + \gamma \max_{a' \in A} Q_k(y,a'))), \\
Q_{k+1}(z,b) &= Q_k(z,b); \qquad \text{for } (z,b) \in ((X \times A) - (X,A)_k). \qquad (4.10)
\end{aligned}
$$

The agent does not need a model to compute $Y_x^a(Q)$ because it can just execute action a in state x and compute the maximum of the set of Q-values stored for the resulting state y (Figure 4.3).

Equation 4.10 is also an iterative relaxation algorithm of the form specified in Equation 3.5 with a random sample backup operator $\mathcal{B}_x^a(Q) = R^a(x) + \gamma \max_{a' \in A} Q(y,a')$ that is unbiased with respect to $B_x^a(Q)$. The sample backup operator $\mathcal{B}_x^a(Q)$ is shown pictorially in Figure 4.3, where instead of computing the expected value of the maximum Q-values of all possible next states, it samples a next state with the probability distribution defined by the state-action pair (x,a), and computes its maximum Q-value. Note that $E\{\mathcal{B}(\mathcal{Q})\} = B(Q)$.

The on-line version of Equation 4.10, i.e., where $(X,A)_{t+1} = \{(x_t, a_t)\}$ is the Q-learning algorithm invented by Watkins. As in the case of policy evaluation, an asymptotic convergence proof for the synchronous version of Equation 4.10 can be derived in a straightforward manner from Dvoretzky's results. However, more recently, Jaakkola, Jordan, and Singh [31] have proved convergence for the asynchronous algorithm that subsumes the synchronous algorithm:

Theorem 6: (Jaakkola, Jordan and Singh [31]) For finite MDPs, the algorithm defined in Equation 4.10 converge with probability one to Q^* if the following conditions hold true:

1. every state-action pair in $(X \times A)$ is updated infinitely often,

2. Q_0 is finite, and

3. $\forall (x, a) \in (X \times A)$; $\sum_{i=0}^{\infty} \rho_i(x, a) = \infty$ and $\sum_{i=0}^{\infty} \rho_i^2(x, a) < \infty$.

4.3.3 Discussion

Convergence proofs for TD and Q-learning that do not use stochastic approximation theory already existed in the RL literature (Sutton [53] and Dayan [21] for TD, and Watkins [58] and Watkins and Dayan [59] for Q-learning). But these proofs, especially the one for Q-learning, are based on special mathematical constructions that obscure the underlying simplicity of the algorithms. The connection to stochastic approximation provides a uniform framework for proving convergence of all the different RL algorithms. It also provides a conceptual framework that emphasizes the main innovation of RL algorithms, that of using unbiased-sample backups, relative to previously developed algorithms for solving MDTs.

Figure 4.4 graphs the relationship between iterative DP and RL algorithms that solve the policy evaluation problem along the following two dimensions: synchronous versus asynchronous, and full backups versus sample backups. Figure 4.5 does the same for DP and RL algorithms that solve the optimal control problem. All of the algorithms developed in the DP literature lie on the upper corners of the squares in Figures 4.4 and 4.5. By adding the "sample versus full backup" dimension, RL researchers have added the lower two corners to these pictures. Only the lower left-hand corner (asynchronous and sample backup) contains model-free algorithms; all the other corners must contain model-based algorithms (marked M).

Figure 4.6 shows a "constraint-diagram" representation of DP and RL algorithms that makes explicit the increased generality of application of RL algorithms. It is a Venn-diagram that shows the constraints required for applicability of the different classes of algorithms presented in Chapters 3 and 4. It shows that on-line RL algorithms are the "weakest" in the sense that they are applicable whenever any of the other algorithms are applicable. The next level of off-line RL algorithms require a model and can be applied whenever asynchronous DP and synchronous DP are applicable. Of course, it is not clear that one would ever want to do off-line RL

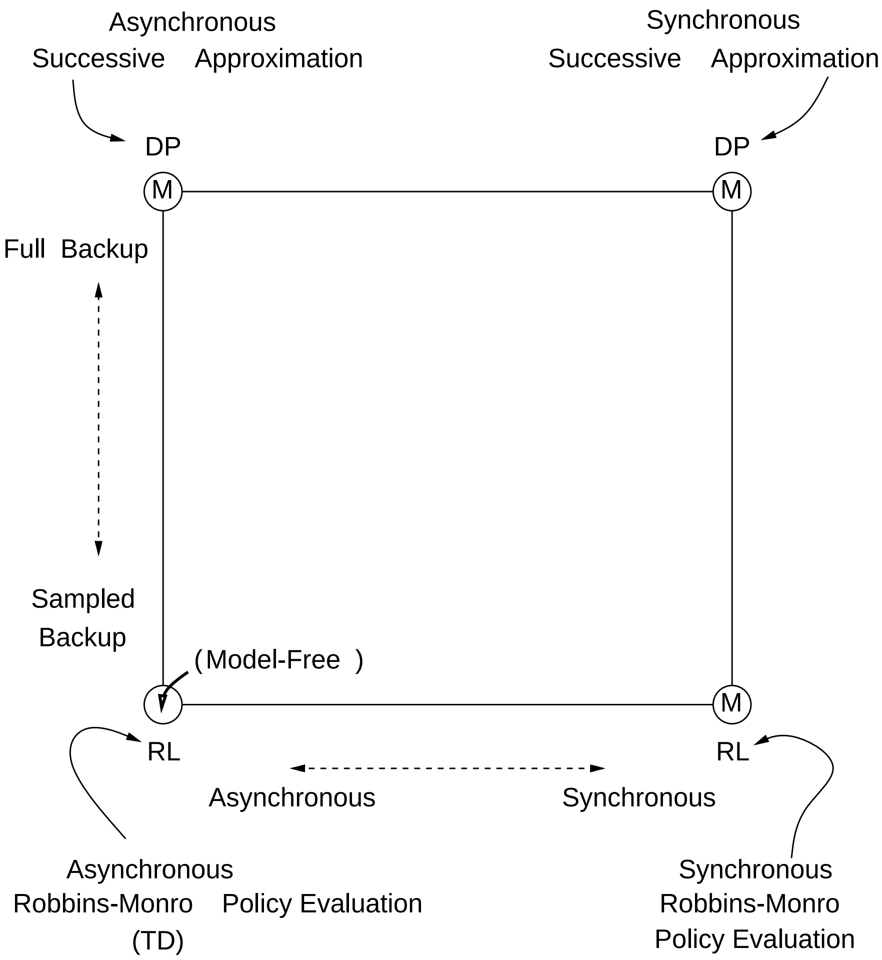

Figure 4.4 Iterative Algorithms for Policy Evaluation. This figure graphs the different DP and RL algorithms for solving the policy evaluation problem along two dimensions: full backup versus sample backup, and asynchronous versus synchronous.

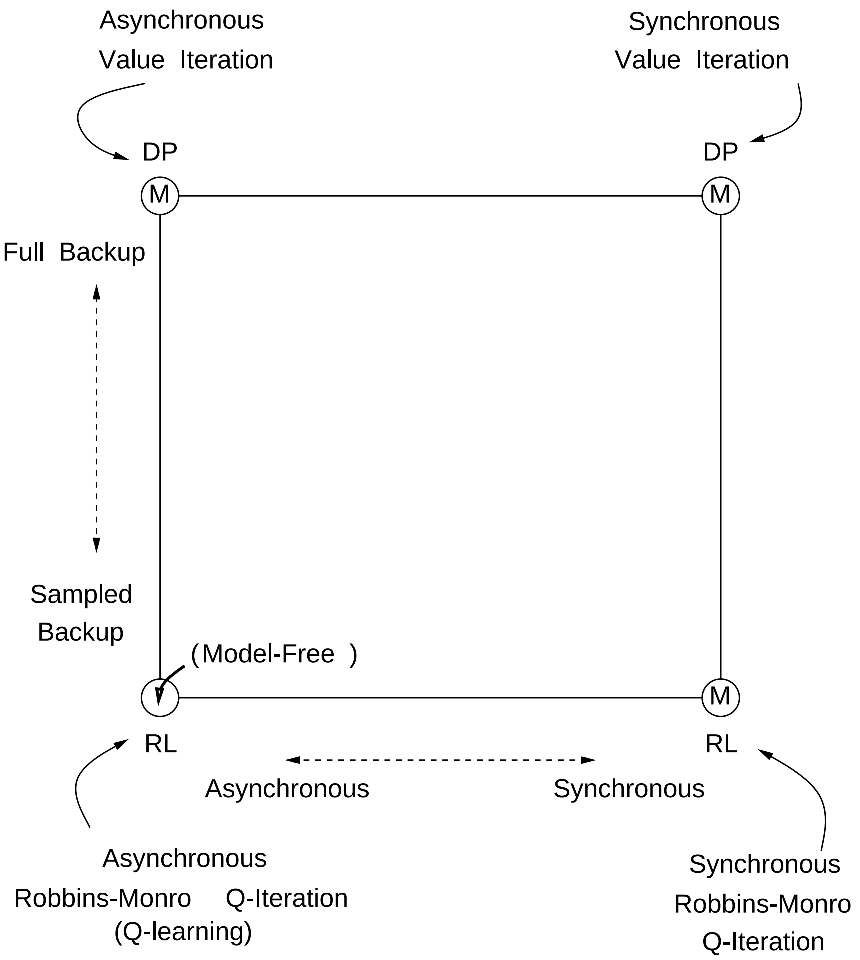

Figure 4.5 Iterative Algorithms for Solving the Optimal Control Problem. This figure graphs the different DP and RL algorithms for solving the optimal control problem along two dimensions: full backup versus sample backup, and asynchronous versus synchronous.

instead of asynchronous DP. Section 4.4 studies that question. Asynchronous DP, the next level, requires full backups and can be applied whenever synchronous DP algorithms can be applied. Synchronous DP is the most restrictive class of algorithms.

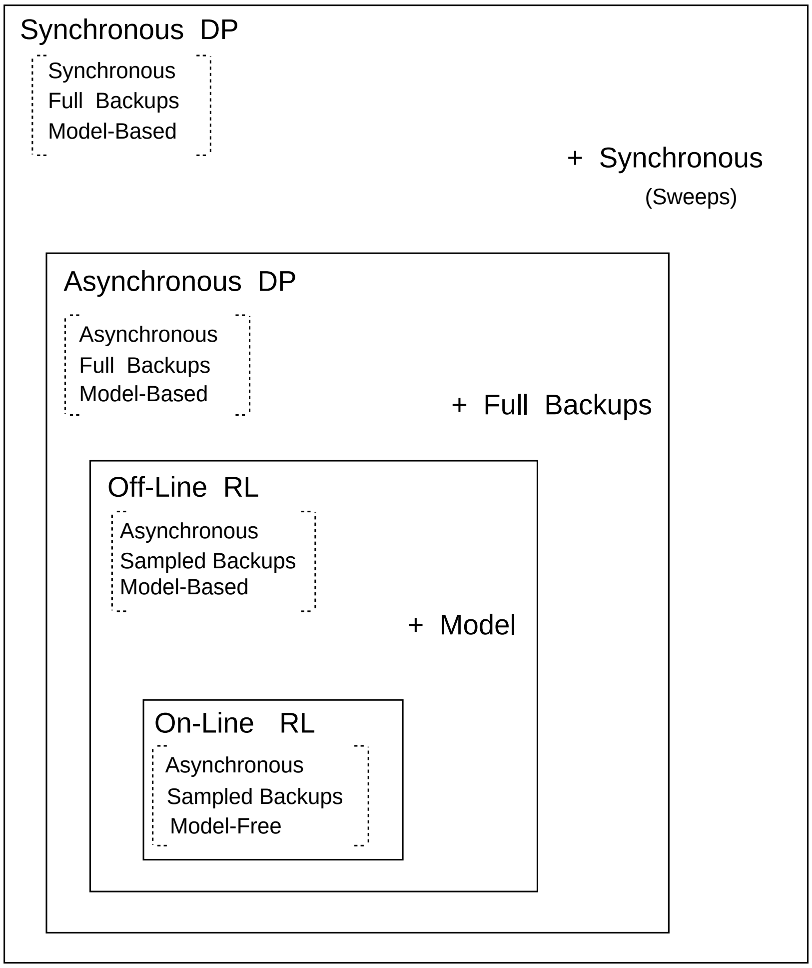

Figure 4.6 A Constraint-Diagram of Iterative Algorithms for Solving MDTs. This figure is a Venn-diagram like representation of the constraints required for all the different DP and RL algorithms reviewed in Chapters 3 and 4. Synchronous DP has the largest box because it places the most constraints.

4.4 When to Use Sample Backups?

A sample backup provides a noisy and therefore less informative estimate than a full backup. A natural question that arises then is: are there conditions under which it can be computationally advantageous to use an iterative algorithm based on sample backups over an algorithm based on full backups to solve the optimal control problem?

Table 4.1 Tradeoff Between Sample Backup and Full Backup.

Sample Backup	Full Backup
Cheap	Expensive
Noisy	Informative

To simplify the argument somewhat, the basis of comparison between algorithms is assumed to be the number of state transitions, whether real or simulated, required for convergence. The number of state transitions is a fair measure because both sample and full backups require one multiplication operation for each state transition (see Equations 3.12, 4.10, 3.9 and 4.5).

The number of state transitions involved in a full backup at state-action pair (x, a) is equal to the number of possible next states, or the *branching factor* for action a. A sample backup always involves a single transition. In general, the number of possible next states could be as high as $|X|$, and therefore a sample backup can be as much as $|X|$ times cheaper than a full backup. In summary, the increased information provided by a full backup comes at the cost of increased computation, raising the possibility that for some MDTs doing $|X|$ sample backups may be more informative than a single full backup (see Table 4.1). Note that the estimate derived from a single sample backup ($\mathcal{B}(V)$) is unbiased with respect to the estimate derived from a single full backup ($B(V)$).

If multiple sample backups are performed without changing the Q-value function Q (or V), it would reduce the variance of the resulting estimate, and thereby make it a closer approximation to the estimate returned by a single full backup. However, in general the value function changes after every application of a sample backup, and therefore multiple updates with a sample backup could lead to a better approximation to the optimal value function than one update with a full backup. Let us consider this possibility in the following two situations separately: one where the agent is given an accurate environment model to start with, and the other where the agent is not provided with an environment model.

4.4.1 Agent is Provided With an Accurate Model

The relative speed of convergence of an algorithm based on sample backups versus an algorithm based on full backups will in general depend on the order in which the backups are applied to the state-action space. To keep things as even as possible, this section presents results comparing synchronous off-line Q-learning that employs sample backups and synchronous Q-value iteration that employs full backups. Therefore the only difference between the two algorithms is in the nature of the backup operator applied to produce new estimates; both algorithms apply the operators in sweeps through the state-action space.

Clearly if the MDT is deterministic, sample backups and full backups are identical, and therefore Q-learning and Q-value iteration are identical. For stochastic MDTs the variance of the estimate based on a sample backup, say $\mathcal{B}_x^a(Q_k)$, about the estimate returned by a full backup, $B_x^a(Q_k)$, is a function of the variance in the product $P^a(x,y)V_k(y)$, where the next state y is chosen randomly with uniform probability. For the empirical studies presented in this section, it is assumed that the effect of the value function on the variance is negligible and therefore only the effect of the non-uniformity in the transition probabilities is important.

The relative performance of synchronous off-line Q-learning and synchronous Q-value iteration was studied on artificially constructed MDTs that are nearly deterministic, but whose branching factor is exactly $|X|$ for every state-action pair. The MDTs were constructed so that for each state-action pair the transition probabilities form a Gaussian on some permutation of the next-state set X. Because the problems are artificial, the amount of determinism, or inversely the variance of the Gaussian transition probability distribution, could be controlled to study the effect of decreasing determinism on the relative speeds of convergence of the following two algorithms:

1. `Algorithm 1 (Gauss-Sidel Q-value iteration):`
   ```
   for (i = 0; i < number-of-sweeps; i++)
       for each state-action pair
           do a full backup
   ```

2. `Algorithm 2 (Gauss-Sidel off-line Q-learning):`

```
for (i = 0; i < (number-of-sweeps × |X|); i++)
    for each state-action pair
        do a sample backup
```

Figures 4.7, 4.8, 4.9 and 4.10 show the relative performance of Algorithms 1 and 2 for MDTs with 50 states, 100 states, 150 states, and 200 states respectively. Each figure shows three graphs: the left-hand graph in the top panel shows the relative performance when the transition probability Gaussian has 95% of its probability mass concentrated on 3 states, the right-hand graph in the top panel shows the relative performance when the transition probability Gaussian has 95% of its mass concentrated on about 32% of the states, and the graph in the lower panel shows the relative performance when the variance of the transition probability Gaussian is designed so that 95% of the mass is concentrated on 65% of the states. For a fixed $|X|$, the increasing variance of the transition probability Gaussian is intended to reveal the decreasing advantage of Algorithm 2 over Algorithm 1 as the problem becomes less deterministic.

Each graph presents results averaged over 10 different runs with 10 different seeds for a random number generator. The x-axis shows the number of sweeps for Algorithm 1. For each sweep of Algorithm 1, $|X|$ sweeps of Algorithm 2 were performed. The performance of the two algorithms was computed as follows. After each sweep through the state space for Algorithm 1, and after every $|X|$ sweeps for Algorithm 2, the greedy policy was derived and fully evaluated. The error value after each sweep is the cumulative absolute difference between the value function of the greedy policy and the pre-computed optimal policy, averaged over the 10 different runs. The y-axis shows the error value.

Figures 4.7 ,4.8, 4.9 and 4.10 each show that Algorithm 2, which uses sample backups, clearly outperforms Algorithm 1, which uses full backups, by a big margin. Another expected result is that as the variance in the Gaussian transition probabilities is increased, the relative advantage of Algorithm 2 reduces in each case. For example, the ratio of the error value after the first sweep in Algorithm 1 to the error value after $|X|$ sweeps in Algorithm 2 in the 100-state MDT is about 60.0 for the most deterministic problem, about 8.0 in the middle problem, and about 5.0 in the least

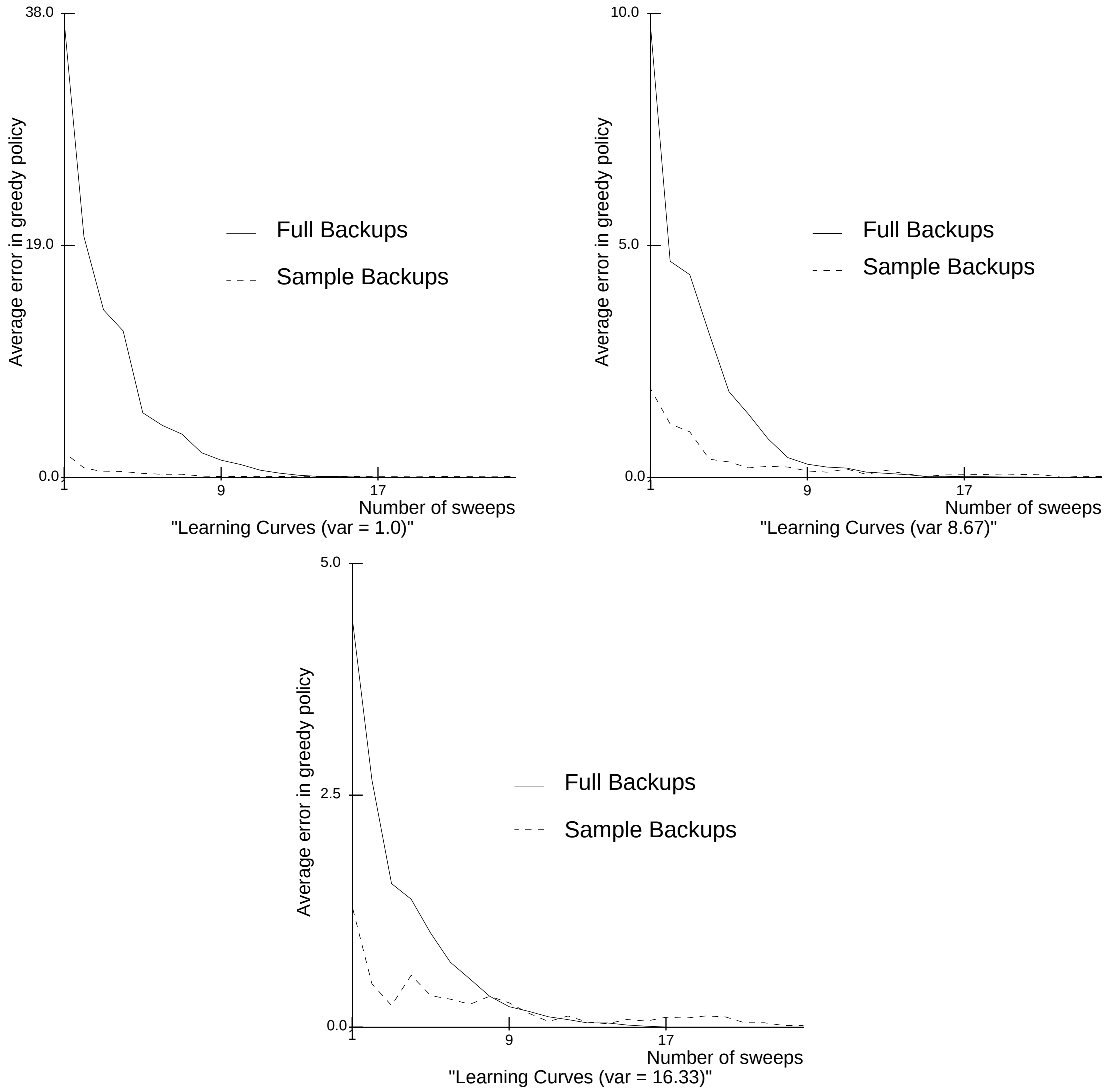

Figure 4.7 Full versus Sample Backups for 50 state MDTs. This figure shows three graphs: the upper left graph for MDTs that have 95% of the transition probability mass concentrated on 3 randomly chosen next states, the upper right graph for MDTs that have the transition probability mass concentrated on 32% of the states, and the lower graph for MDTs that have the transition probability mass concentrated on 65% of the states. The x-axis is the number of sweeps of Algorithm 1 (full backups). Note that for each sweep of Algorithm 1, 50 sweeps of Algorithm 2 (sample backups) are performed. The y-axis shows the average loss of the greedy policy. As expected the sample backup algorithm outperforms the full backup algorithm. Also as the amount of determinism is decreased the relative advantage of sample over full backups gets smaller.

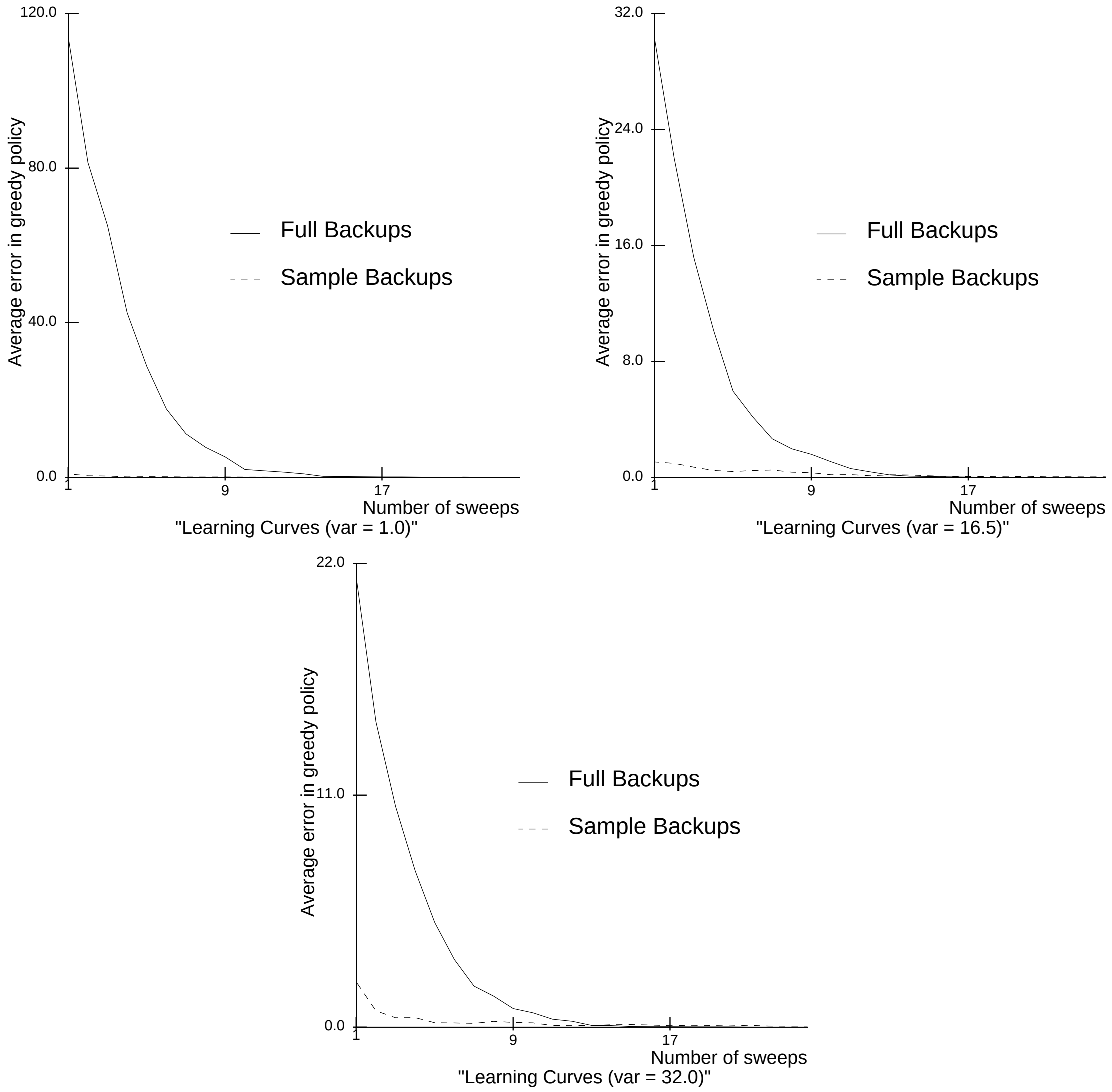

Figure 4.8 Full versus Sample Backups for 100 state MDTs. This figure shows three graphs: the upper left graph for MDTs that have 95% of the transition probability mass concentrated on 3 randomly chosen next states, the upper right graph for MDTs that have the transition probability mass concentrated on 32% of the states, and the lower graph for MDTs that have the transition probability mass concentrated on 65% of the states. The x-axis is the number of sweeps of Algorithm 1 (full backups). Note that for each sweep of Algorithm 1, 50 sweeps of Algorithm 2 (sample backups) are performed. The y-axis shows the average loss of the greedy policy. As expected the sample backup algorithm outperforms the full backup algorithm. Also as the amount of determinism is decreased the relative advantage of sample over full backups gets smaller.

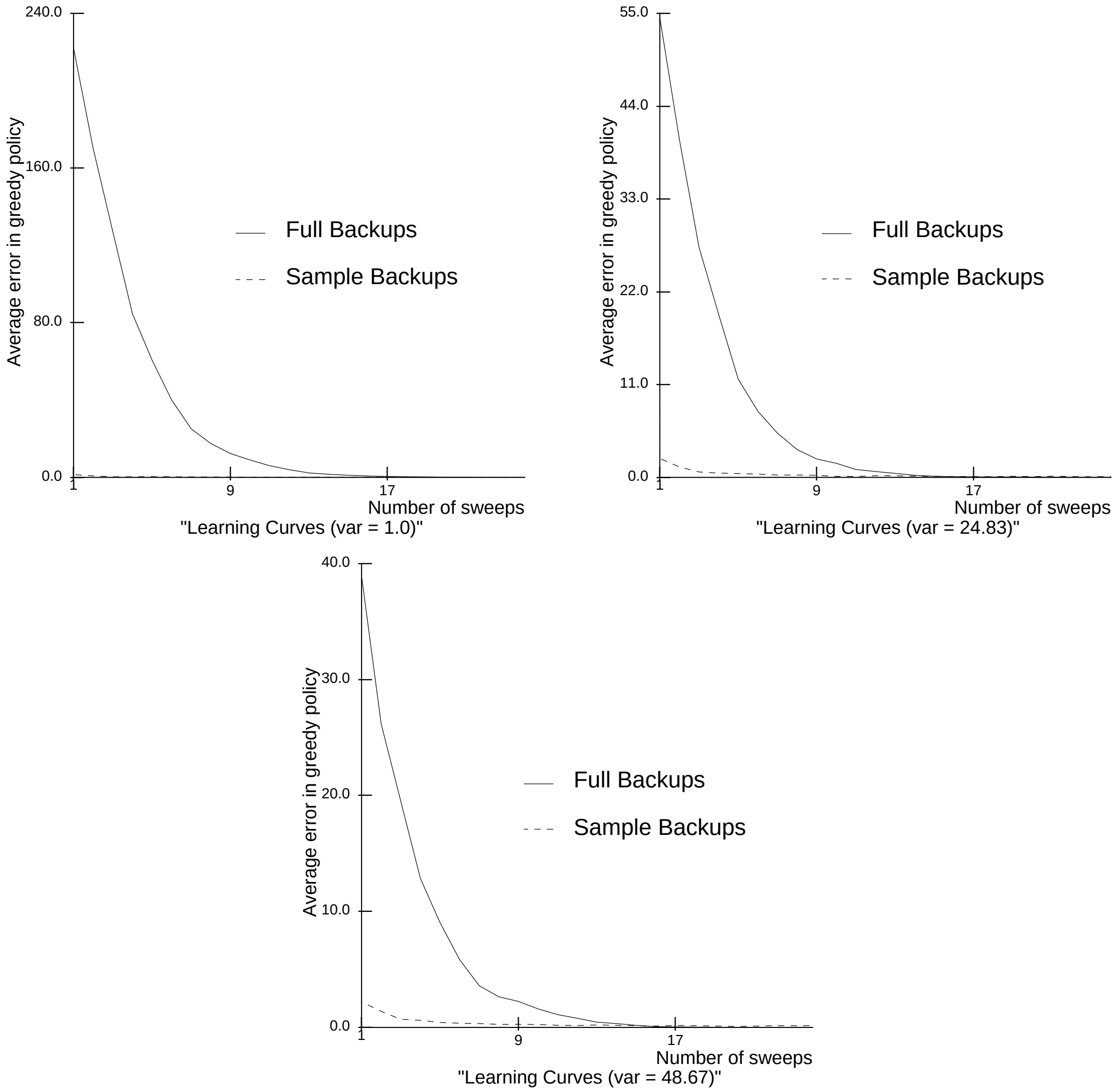

Figure 4.9 Full versus Sample Backups for 150 state MDTs. This figure shows three graphs: the upper left graph for MDTs that have 95% of the transition probability mass concentrated on 3 randomly chosen next states, the upper right graph for MDTs that have the transition probability mass concentrated on 32% of the states, and the lower graph for MDTs that have the transition probability mass concentrated on 65% of the states. The x-axis is the number of sweeps of Algorithm 1 (full backups). Note that for each sweep of Algorithm 1, 50 sweeps of Algorithm 2 (sample backups) are performed. The y-axis shows the average loss of the greedy policy. As expected the sample backup algorithm outperforms the full backup algorithm. Also as the amount of determinism is decreased the relative advantage of sample over full backups gets smaller.

deterministic problems. A similar decline is noticed in the other three sets of MDTs. Another effect to notice is that the relative advantage of Algorithm 2 over Algorithm 1 increases in ratio as the size of the MDT is increased, at least for the problem sizes studied here.

4.4.2 Agent is not Given a Model

If the agent is not given a model a priori, an algorithm that uses full backups would have to be adaptive, i.e., estimate a model on-line by using information about state transitions achieved in the real environment. In such a case, because an algorithm that uses full backups is still model-based, the relative advantages presented in the previous section in favor of algorithms that use sample backups will continue to hold. However, because the model is being estimated on-line, there are two additional reasons to prefer an algorithm that uses sample backups. First, an algorithm that uses sample backups can avoid the computational expense of building a model. Second, during the early stages of learning, the model will be highly inaccurate and can interfere with learning the value function by adding a "bias" to the full backup operator (see Barto and Singh [10, 9]).

Nevertheless, in general it is difficult to predict which of the two algorithms, one based on sample backups, and the other based on full backups on an estimated model, will outperform the other. The term adaptive full backup is used to denote a full backup performed on simulations from an estimated model. It is instructive to contrast both the sample backup algorithm and the adaptive full backup algorithm with a second non-adaptive full backup algorithm that is assumed to have access to the correct model, that is, to compare the following three backup operators:

$$\begin{aligned}
\text{Sample backup:} &= R^a(x) + \gamma \max_{a' \in A} Q(y, a') \\
\text{Adaptive full backup:} &= R^a(x) + \gamma \sum_{y \in X} [\hat{P}^a(x, y) \max_{a' \in A} Q(y, a')] \\
\text{Non-adaptive full backup:} &= R^a(x) + \gamma \sum_{y \in X} [P^a(x, y) \max_{a' \in A} Q(y, a')]
\end{aligned}$$

where $\hat{P}$ are the estimated transition probabilities that are learned on-line by the adaptive full backup algorithm.

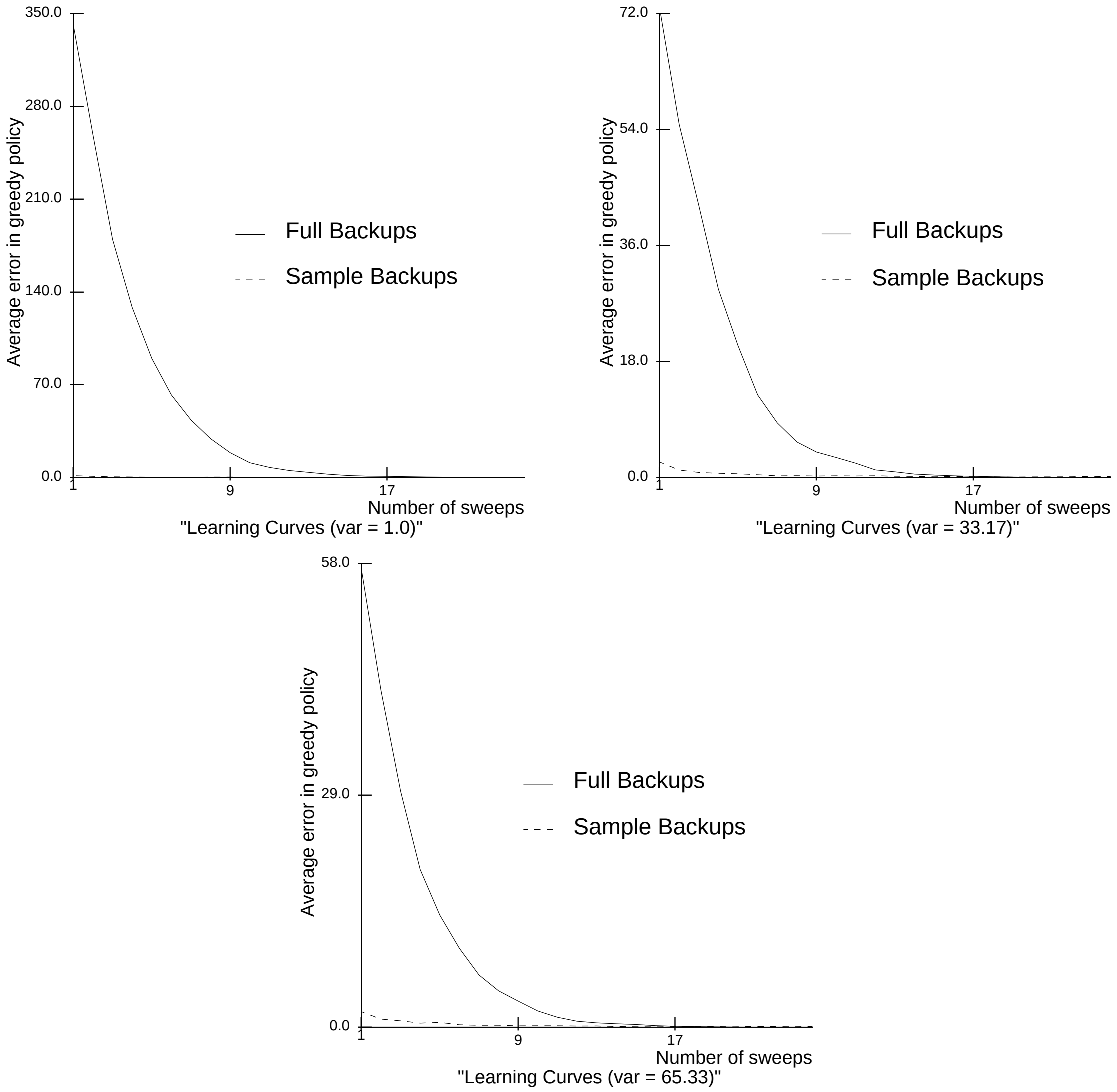

Figure 4.10 Full versus Sample Backups for 200 state MDTs. This figure shows three graphs: the upper left graph for MDTs that have 95% of the transition probability mass concentrated on 3 randomly chosen next states, the upper right graph for MDTs that have the transition probability mass concentrated on 32% of the states, and the lower graph for MDTs that have the transition probability mass concentrated on 65% of the states. The x-axis is the number of sweeps of Algorithm 1 (full backups). Note that for each sweep of Algorithm 1, 50 sweeps of Algorithm 2 (sample backups) are performed. The y-axis shows the average loss of the greedy policy. As expected the sample backup algorithm outperforms the full backup algorithm. Also as the amount of determinism is decreased the relative advantage of sample over full backups gets smaller.

Table 4.2 Bias-Variance Tradeoff in RL and Adaptive DP

Stage	*Sample backup*	*Adaptive full backup*
early	no bias low-high variance	high bias no variance
middle	no bias low-high variance	medium bias no variance
late	no bias low-high variance	low bias no variance

The following table summarizes the relative error in the estimates provided by the sample backup and the adaptive full backup with respect to a non-adaptive full backup as function of the stage of learning. Table 4.2 states that the sample backup is always unbiased with respect to the estimate provided by a full backup on a correct model. The variance in the sample backup can be low to high depending on the "skew" in the value function as well as in the transition probabilities. On the other hand, the adaptive full backup will have a high bias with respect to the non-adaptive case in the early stages of learning because of the large error in the estimated model, but as the estimate model becomes better over time, the bias will go away. Since the full backup operator computes expected values there is no variance in the estimate provided by the adaptive full backup operator.

The entries in Table 4.2 can lead to the conclusion that an algorithm that estimates a model, but uses sample backups during the early stages of learning, and then switches to doing full backups using simulated experience with the estimated model when its bias gets small enough, could get the best of both worlds. This conclusion is based on the fact that depending upon the bias and variance values it can be better to sample from a biased source with low variance than an unbiased source with high variance. However, it is unclear as to how the crossover point, i.e., the point at which to switch from sample backups to full backups, could be determined in practice, since the bias and the variances of the backup operators are not known. It may be possible to compute estimates of the bias and variance of the backup operators on-line to determine the crossover point, but in general it is unclear whether the potential savings will be more than the computational expense involved in doing so. In any case, the potential savings of such a hybrid method will be determined in large part by how early in the learning process the crossover point is reached.

4.4.3 Discussion

It was shown in this section that sample backup algorithms are likely to have significant advantage over full backup algorithms in MDTs that are nearly deterministic and yet have a large branching factor averaged over state-action pairs. While this does not conclude the ongoing debate over the relative merits of model-free versus model-based RL methods, it does provide additional evidence that there exist problems where model-free algorithms are more efficient than model-based ones.

4.5 Shortcomings of Asymptotic Convergence Results

A major limitation of most[1] current theoretical research on RL, including the research presented in this chapter, is its dependence on the following two assumptions:

1. that a lookup-table is used to represent the value (or Q-value) function,

2. that each state (or state-action pair) is updated infinitely often.

Both of these assumptions are unrealistic in practice.

Several researchers, including this author, have used function approximation methods other than lookup-tables, e.g., neural networks, to represent the value (or Q-value) function. With non lookup-table representations the following two factors could prevent convergence to V^* (or Q^*):

- V^* (or Q^*) may not be in the class of functions that the chosen function approximation architecture can represent. This is not possible to know in advance because V^* (and Q^*) is not known in advance. However, *constructive* function approximation approaches (e.g., Fahlman and Lebiere [24]) may be able to alleviate this problem.

---

[1]Sutton [53] and Dayan [21] have proved convergence for the TD algorithm when linear networks are used to represent the value functions. Bradtke [18] adapted Q-learning to solve linear quadratic regulation (LQR) problems and proved convergence under the assumption that a linear-in-the-parameters network is used to store the Q-value function.

- A more fundamental issue is that a non lookup-table function approximation method can generalize an update in such a way that the essential "contraction-based" convergence of DP-related algorithms may be thwarted.

This raises the following important question: if practical concerns dictate that value functions be approximated, how might performance be affected[2]? Is it possible that, despite some empirical evidence to the contrary (e.g., Barto *et al.* [11], Anderson [1], Tesauro [57]), small errors in approximations could result in arbitrarily bad performance in principle? If so, this could raise significant concerns about the use of function approximation in DP-based learning.

4.5.1 An Upper Bound on the Loss from Approximate Optimal-Value Functions

This section extends a result by Bertsekas [15] which guarantees that small errors in the approximation of a task's optimal value function cannot produce arbitrarily bad performance when actions are selected greedily. Specifically, the extension is in deriving an upper bound on performance loss which is slightly tighter than that in Bertsekas [15], and in deriving the corresponding upper bound for the newer Q-learning algorithm. These results also provide a theoretical justification for a practice that is common in RL. The material presented in this section was developed in collaboration with Yee and is reported in Singh and Yee [51].

Policies can be derived from value functions in a straightforward way. Given value function $\tilde{V}$, a *greedy policy* $\pi_{\tilde{V}}$ can be defined as

$$\pi_{\tilde{V}}(x) = \mathrm{argmax}_{a \in A} \left[R^a(x) + \gamma \sum_{y \in X} P^a(x, y)\, \tilde{V}(y) \right],$$

where ties for the maximum action are broken arbitrarily. Figure 4.11 illustrates the relationship between the evaluation of policies and the derivation of greedy policies. A greed policy $\pi_{\tilde{V}}$ gives rise to its own value function $V^{\pi_{\tilde{V}}}$, denoted simply as $V^{\tilde{V}}$. In

[2]Even with lookup-table representations, in practice it may be difficult to visit every state-action pair often enough to ensure convergence to the optimal value function. Thus the issue of how performance gets affected by using approximations to V^* is relevant even to lookup-tables.

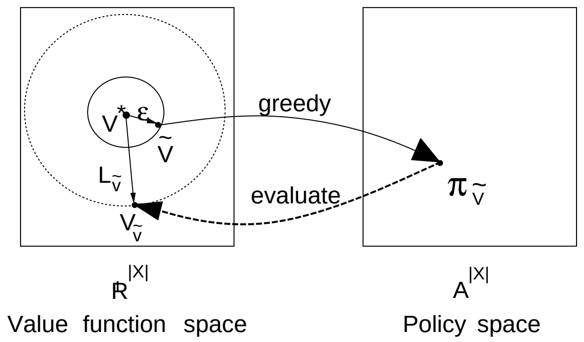

Figure 4.11 Loss From Approximating Optimal Value Function. Given $\tilde{V}$, an approximation within $\epsilon > 0$ of V^*, derive the corresponding greedy policy $\pi_{\tilde{V}}$. The resulting loss in value, $L_{\tilde{V}} = V^* - V^{\tilde{V}}$, is bounded above by $(2\gamma\epsilon)/(1-\gamma)$.

general, $\tilde{V} \neq V^{\tilde{V}}$, i.e., the value function used for deriving a greedy policy is different from the value function resulting from the policy's evaluation. Equality between $\tilde{V}$ and $V^{\tilde{V}}$ occurs if and only if $\tilde{V}$ is optimal, in which case any greedy policy will be optimal.

For a greedy policy $\pi_{\tilde{V}}$ derived from $\tilde{V}$, an approximation to V^*, define the *loss function* $L_{\tilde{V}}$ such that $\forall x \in X$,

$$L_{\tilde{V}}(x) = V^*(x) - V^{\tilde{V}}(x).$$

$L_{\tilde{V}}(x)$ is the expected loss in the value of state x resulting from the use of policy $\pi_{\tilde{V}}$ instead of an optimal policy. Note that $L_{\tilde{V}}(x) \geq 0$ because $V^*(x) \geq V_{\tilde{V}}(x)$. The following theorem gives an upper bound on the loss $L_{\tilde{V}}$.

Theorem 7: Let V^* be the optimal value function for a discrete-time MDT having finite states and actions and an infinite, geometrically discounted horizon, $\gamma \in [0, 1)$. If $\tilde{V}$ is a function such that $\forall x \in X$, $\left|V^*(x) - \tilde{V}(x)\right| \leq \epsilon$, and $\pi_{\tilde{V}}$ is a greedy policy for $\tilde{V}$, then $\forall x$,

$$L_{\tilde{V}}(x) \leq \frac{2\gamma\epsilon}{1-\gamma}.$$

The upper bound of Theorem 7 is tighter than the result in Bertsekas [15] by a factor of γ (Cf. [p. 236, #14(c)]). One interpretation of this result is that if the approximation to the optimal value function is off by no more than ϵ, then the average worst-case loss per time step cannot be more than $2\gamma\epsilon$, under a greedy policy.

Proof: There exists a state z that achieves the maximum loss: $\exists z \in X$, $\forall x \in X$, $L_{\tilde{V}}(z) \geq L_{\tilde{V}}(x)$. For state z consider an optimal action, $a = \pi^*(z)$, and the action of $\pi_{\tilde{V}}$, $b = \pi_{\tilde{V}}(z)$. Because $\pi_{\tilde{V}}$ is a greedy policy for $\tilde{V}$, b must appear at least as good as a:

$$\begin{aligned}
R^a(z) + \gamma \sum_{y \in X} P^a(z,y)\, \tilde{V}(y) &\leq R^b(z) + \gamma \sum_{y \in X} P^b(z,y)\, \tilde{V}(y) && (4.11)\\
R^a(z) - R^b(z) &\leq \gamma \sum_y \left[P^b(z,y)\ (V^*(y) + \epsilon) - P^a(z,y)\ (V^*(y) - \epsilon) \right] \\
R^a(z) - R^b(z) &\leq 2\gamma\epsilon + \gamma \sum_y \left[P^b(z,y)\, V^*(y) - P^a(z,y)\, V^*(y) \right]. && (4.12)
\end{aligned}$$

The maximal loss is

$$\begin{aligned}
L_{\tilde{V}}(z) &= V^*(z) - V^{\tilde{V}}(z) \\
&= R^a(z) - R^b(z) + \gamma \sum_y \left[P^a(z,y) V^*(y) - P^b(z,y) V^{\tilde{V}}(y) \right]. && (4.13)
\end{aligned}$$

Substituting from (4.12) gives

$$\begin{aligned}
L_{\tilde{V}}(z) &\leq 2\gamma\epsilon + \gamma \sum_y \Big[P^b(z,y)\, V^*(y) - P^a(z,y)\, V^*(y) \\
&\qquad + P^a(z,y)\, V^*(y) - P^b(z,y)\, V^{\tilde{V}}(y) \Big] \\
L_{\tilde{V}}(z) &\leq 2\gamma\epsilon + \gamma \sum_y P^b(z,y) \left[V^*(y) - V^{\tilde{V}}(y) \right] \\
L_{\tilde{V}}(z) &\leq 2\gamma\epsilon + \gamma \sum_y P^b(z,y)\, L_{\tilde{V}}(y)
\end{aligned}$$

Because, by assumption, $\forall y \in X,\ L_{\tilde{V}}(z) \geq L_{\tilde{V}}(y)$,

$$\begin{aligned}
L_{\tilde{V}}(z) &\leq 2\gamma\epsilon + \gamma \sum_y P^b(z,y)\, L_{\tilde{V}}(z) \\
L_{\tilde{V}}(z) &\leq \frac{2\gamma\epsilon}{1-\gamma}
\end{aligned}$$

Q.E.D.

This result extends to a number of related cases.

4.5.1.1 Approximate payoffs

Theorem 7 assumes that the expected payoffs are known exactly. If the true expected payoffs $R^a(x)$ are approximated by $\tilde{R}^a(x)$, the upper bound on the loss is as follows.

Corollary $7A$**:** If $\forall x \in X$, $\left|V^*(x) - \tilde{V}(x)\right| \leq \epsilon$ and $\forall a \in A$, $\left|R^a(x) - \tilde{R}^a(x)\right| \leq \alpha$, then $\forall x$,

$$L_{\tilde{V}}(x) \leq \frac{2\gamma\epsilon + 2\alpha}{1-\gamma},$$

where $\pi_{\tilde{V}}$ is the greedy policy for $\tilde{V}$.

Proof: Inequality (4.11) becomes

$$\tilde{R}^a(z) + \gamma \sum_{y \in X} P^a(z,y)\,\tilde{V}(y) \leq \tilde{R}^b(z) + \gamma \sum_{y \in X} P^b(z,y)\,\tilde{V}(y),$$

and (4.12) becomes

$$R^a(z) - R^b(z) \leq 2\gamma\epsilon + 2\alpha + \gamma \sum_y \left[P^b(z,y)\,V^*(y) - P^a(z,y)\,V^*(y)\right].$$

Substitution into (4.13) yields the bound. Q.E.D.

4.5.1.2 Q-learning

If neither the payoffs nor the state-transition probabilities are known, then the analogous bound for Q-learning is as follows. Evaluations are defined by

$$Q^\pi(x_t, a) = R^a(x_t) + \gamma E\left\{V_\pi(x_{t+1})\right\},$$

where $V_\pi(x) = \max_a Q^\pi(x,a)$. Given function $\tilde{Q}$, the greedy policy $\pi_{\tilde{Q}}$ is defined by

$$\pi_{\tilde{Q}}(x) = \operatorname{argmax}_{a \in A(x)} \tilde{Q}(x,a).$$

The loss is then expressed as

$$L_{\tilde{Q}}(x) = Q^*(x,\, \pi^*(x)) - \tilde{Q}(x,\, \pi_{\tilde{Q}}(x)).$$

Corollary $7B$**:** If $\forall x \in X$, $\forall a \in A(x)$, $\left|Q^*(x,a) - \tilde{Q}(x,a)\right| \leq \epsilon$, then $\forall x$,

$$L_{\tilde{Q}}(x) \leq \frac{2\epsilon}{1-\gamma}.$$

Proof: Inequality (4.11) becomes $\tilde{Q}(z,a) \leq \tilde{Q}(z,b)$, which gives

$$\begin{aligned}
Q^*(z,a) - \epsilon &\leq Q^*(z,b) + \epsilon \\
R^a(z) + \gamma \sum_y P^a(z,y)\,V^*(y) - \epsilon &\leq R^b(z) + \gamma \sum_y P^b(z,y)\,V^*(y) + \epsilon \\
R^a(z) - R^b(z) &\leq 2\epsilon + \gamma \sum_y \left[P^b(z,y)\,V^*(y) - P^a(z,y)\,V^*(y)\right].
\end{aligned}$$

Substitution into (4.13) yields the bound. Q.E.D.

4.5.2 Discussion

The bounds of Theorem 7 and its corollaries guarantee that the performance of DP-based learning approaches will not be far from optimal if (a) good approximations to optimal value functions are achieved, (b) a corresponding greedy policy is followed, and (c) the discount factor, γ, is not too close to 1.0. The analogous result holds for *indefinite* horizon, undiscounted MDTs (as defined in Barto *et al.*[8]). Although this result does not address convergence, it nevertheless helps to validate many practical approaches to DP-based learning that use approximations.

Theorem 7 does not directly address policy-derivation methods other than the indicated greedy one. For other methods, it is not clear, in general, what criteria to place on approximations of value functions because the criteria may depend upon the specifics of a derivation method. Greedy policy derivation allows one to specify an error-criterion on approximations, i.e., that they be within ϵ of optimal, under the max norm.

4.5.3 Stopping Criterion

Another unrealistic requirement to ensure convergence to V^* (or Q^*) is that each state (or state-action pair) be visited infinitely often. However, the real goal is not to converge to the optimal value function, but to derive an optimal policy. Indeed, as stated in the following theorem, for every finite MDT there is a spherical region (ball) around the optimal value function such that the greedy policy with respect to any value function in that ball is optimal.

Theorem 8: For any given finite action MDT, $\exists \epsilon > 0$, such that $\forall \tilde{V} \in (\Re^+)^{|X|}$, where $||\tilde{V} - V^*||_\infty < \epsilon$, any policy that is greedy with respect to $\tilde{V}$ is optimal.

Proof: For all $x \in X$, and $\forall a \in A$, define,

$$m(x,a) = (R^{\pi^*(x)}(x) + \gamma \sum_{y \in X} P^{\pi^*(x)}(x,y)V^*(y)) - (R^a(x) + \gamma \sum_{y \in X} P^a(x,y)V^*(y)),$$

$$= \quad Q^*(x, \pi^*(x)) - Q^*(x, a).$$

Note that $m(x, a) \geq 0$ by definition. Let $\Pi^*(x)$ be the set of optimal actions in state x. Define

$$2\gamma\epsilon \quad = \quad \min_{x \in X}\{ \min_{a \in (A - \Pi^*(x))} \{m(x, a)\}\}. \tag{4.14}$$

Clearly ϵ will only be zero *iff* all stationary policies are optimal, in which case ϵ can be set to any value greater than zero. Therefore by the definition of ϵ, $\forall x \in X$, and $\forall a \in (A - \Pi^*(x))$, $Q^*(x, \pi^*(x)) - Q^*(x, a) \geq 2\gamma\epsilon > 0$. Let $\tilde{Q}$ be the Q-value function derived from the approximation $\tilde{V}$ that is assumed to satisfy the following: $||\tilde{V} \ - \ V^*||_\infty < \epsilon$. Therefore, for $a \in (A - \Pi^*(x))$:

$$\begin{aligned}
\tilde{Q}(x, a) &= R^a(x) + \gamma \sum_{y \in X} P^a(x, y)\tilde{V}(y) \\
&< R^a(x) + \gamma \sum_{y \in X} P^a(x, y)(V^*(y) + \epsilon) \\
&< [R^a(x) + \gamma \sum_{y \in X} P^a(x, y)V^*(y)] + \gamma\epsilon \\
&< Q^*(x, a) + \gamma\epsilon \\
&< Q^*(x, \pi^*(x)) - \gamma\epsilon \\
&< R^{\pi^*(x)}(x) + \gamma \sum_{y \in X} P^a(x, y)V^*(y) - \gamma\epsilon \\
&< R^{\pi^*(x)}(x) + \gamma \sum_{y \in X} P^a(x, y)(\tilde{V}(y) + \epsilon) - \gamma\epsilon \\
&< \tilde{Q}(x, \pi^*(x)),
\end{aligned}$$

which implies that the greedy actions with respect to $\tilde{V}$ with ϵ defined in Equation 4.14 will be elements of the set $\Pi^*(x)$.

Q.E.D.

It is possible that long before infinite visits to every state, the policy greedy with respect to the estimate of the optimal value function becomes optimal, e.g., when the estimated value function is the ball specified in Theorem 8. Unfortunately, in RL algorithms there is no general way to detect the iteration at which the greedy policy becomes optimal. It would be very beneficial to discover "stopping conditions" for RL algorithms which if met would indicate that with high probability the greedy policy

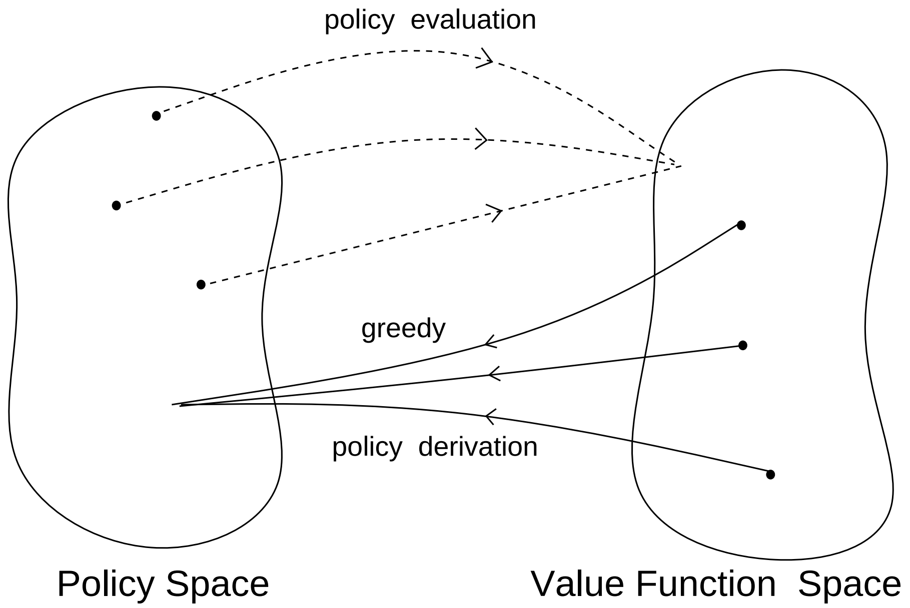

Figure 4.12 Mappings between Policy and Value Function Spaces. The greedy policy derivation mapping from value function space to policy space is many-to-one, nonlinear, and highly discontinuous. The policy evaluation mapping from policy space to value function space is many-to-one but linear. This makes it very difficult to discover stopping conditions.

derived from the current estimate of the value function would be optimal or close to optimal.

As seen in Section 4.5.1, the process of characterizing the possible loss from an approximation in value function space requires mapping the approximation into policy space and back into value function space (cf. Figures 4.11 and 4.12). This makes the discovery of stopping conditions hard because the mapping from the space of value functions to the space of stationary policies is many-to-one (Figure 4.12), and because it is non-linear and full of discontinuities. While the policy evaluation mapping from policy space to value function space is linear, it is also many-to-one. Another factor that complicates matters is that RL algorithms use sample backups, which are noisy and therefore only a probabilistic argument can be made in the first place.

4.6 Conclusion

This chapter establishes an important connection between stochastic approximation theory and RL and shows how RL researchers have added the sample backup versus full backup dimension to the class of iterative algorithms for solving MDTs. It provides empirical evidence that at least for certain MDTs, specifically those that are nearly deterministic and yet have a high branching factor, doing sample backups may be more efficient than doing full backups. Some preliminary theoretical work is presented on two of the pressing questions left unanswered by asymptotic convergence results: how do approximations affect convergence, and what are suitable stopping conditions. It is shown that minimizing max-norm distance to V^* is a good error criteria for function approximation, both because getting close to V^* will mean that the resulting greedy policy cannot be arbitrarily bad, and because there is a ball around V^* within which all greedy policies are optimal. The rest of this dissertation is focused on developing new RL architectures that address more practical concerns: learning multiple RL tasks efficiently, and ensuring safe performance while learning on-line.

APPENDIX B

DVORETZKY'S STOCHASTIC APPROXIMATION THEORY

After the initial Robbins-Monro paper on stochastic approximation algorithms that proved convergence in probability, several researchers derived stronger results of convergence with probability one and mean-square convergence under conditions that are more general than the ones proposed in the original Robbins-Monro paper. For the purposes of Chapter 4 a result derived by Dvoretzky is the most relevant. Dvoretzky [23] studied a problem more general than finding roots of equations. He studied convergent deterministic iterative processes of the form $V_{n+1} = T_n(V_n)$, where T_n is a deterministic operator. He derived conditions under which the stochastic process $V_{n+1} = T_n(V_n) + D_n$, where D_n is mean-zero noise, would converge to the fixed point of the original deterministic process.

The Robbins-Monro stochastic approximation iteration

$$V_{n+1} = V_n - \rho_n Y(V_n) \tag{B.1}$$

where $Y(V_n) = G(V_n) + \epsilon_n$, where $G(V)$ is deterministic and ϵ_n is mean-zero noise, can be rewritten in the form studied by Dvoretzky as follows:

$$V_{n+1} = (V_n - \rho_n G(V_n)) + \rho_n (G(V_n) - Y(V_n)), \tag{B.2}$$

where $T_n(V_n) = V_n - \rho_n G(V_n)$, and $D_n = G(V_n) - Y(V_n)$.

Let V_n be measurable random variables assuming values in a (finite or infinite dimensional) (real) Hilbert space $\mathcal{H}$. Let T_n be function for $\mathcal{H}$ to $\mathcal{H}$. Let $|.|$ denote the norm in $\mathcal{H}$.

Consider the basic recurrence scheme:

$$V_{n+1} = T_n(V_n) + D_n. \tag{B.3}$$

Theorem:. Let α_n, β_n and ϕ_n be non-negative real numbers satisfying

$$\lim_{n=\infty} \alpha_n = 0$$

$$\sum_{i=1}^{\infty} \beta_i < \infty$$
$$\sum_{i=1}^{\infty} \phi_i = \infty, \tag{B.4}$$

Let V^* be a point of $\mathcal{H}$ and T_n satisfy the following equation

$$|T_n(V_n) - V^*| \leq \max\left[\alpha_n, (1 + \beta_n - \phi_n)|V_n - V^*|\right] \tag{B.5}$$

be satisfied for all $V_n \in \mathcal{H}$.

Then the iteration B.3 together with the conditions

$$E\{D_n|V_n\} = 0 \tag{B.6}$$

$\forall n$, and

$$\sum_{i=1}^{\infty} E\{D_n^2\} < \infty \tag{B.7}$$

imply

$$P\{\lim_{n=\infty} V_n = V^*\} = 1. \tag{B.8}$$

If moreover, $E\{V_1^2\} < \infty$ then also

$$\lim_{n=\infty} E(V_n - V^*)^2 = 0 \tag{B.9}$$

BIBLIOGRAPHY

[1] C.W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Amherst, MA, 1986.

[2] K.J. Astrom and B. Wittenmark. *Adaptive Control.* Addison-Wesley, 1989.

[3] E. Barnard. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):357–365, 1993.

[4] A.G. Barto. personal communication.

[5] A.G. Barto. Connectionist learning for control: An overview. In T. Miller, R.S. Sutton, and P.J. Werbos, editors, *Neural Networks for Control.* MIT Press, Cambridge, MA. To appear.

[6] A.G. Barto. Learning to act: A perspective from control theory, July 1992. AAAI invited talk.

[7] A.G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[8] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. Technical Report 93-02, University of Massachusetts, Amherst, MA, 1993. Submitted to: AI Journal.

[9] A.G. Barto and S.P. Singh. Reinforcement learning and dynamic programming. In *Sixth Yale Workshop on Adaptive and Learning Systems*, pages 83–88, New Haven, CT, 1990.

[10] A.G. Barto and S.P. Singh. On the computational economics of reinforcement learning. In *Proceedings of the 1990 Connectionist Models Summer School*, San Mateo, CA, Nov. 1990. Morgan Kaufmann.

[11] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE SMC*, 13:835–846, 1983.

[12] A.G. Barto, R.S. Sutton, and C. Watkins. Learning and sequential decision making. In M. Gabriel and J.W. Moore, editors, *Learning and Computational Neuroscience.* MIT Press, Cambridge, MA, 1990.

[13] A.G. Barto, R.S. Sutton, and C. Watkins. Sequential decision problems and neural networks. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 686–693, San Mateo, CA, 1990. Morgan Kaufmann.

[14] R.E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[15] D.P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[16] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[17] J.R. Blum. Multidimensional stochastic approximation method. *Ann. of Math. Stat.*, 25, 1954.

[18] S.J. Bradtke. Reinforcement learning applied to linear quadratic regulation. In *Advances in Neural Information Processing Systems 5*, pages 295–302, San Mateo, CA, 1993. Morgan Kaufmann.

[19] J. Bridle. *Probablistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition.* Springer-Verlag, New York, 1989.

[20] R.R. Bush and F. Mosteller. *Stochastic Models for Learning.* Wiley, New York, 1955.

[21] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8(3/4):341–362, May 1992.

[22] T.L. Dean and M.P. Wellman. *Planning and Control.* Morgan Kauffman, 1991.

[23] A. Dvoretzky. On stochastic approximation. In *Proceedings of the third Berkeley symposium on Mathematical Statisitics and Probability*, pages 39–55, 1956.

[24] S.E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532, San Mateo, CA, 1990. IEEE, Morgan Kaufmann.

[25] W. Fun and M.I. Jordan. The moving basin: Effective action-search in adaptive control, 1992. submitted to Neural Computation.

[26] G.C. Goodwin and K.W. Sin. *Adaptive Filtering Prediction and Control.* Englewood Cliffs, 1984.

[27] V. Gullapalli. *Reinforcement Learning and its application to control.* PhD thesis, University of Massachusetts, Amherst, MA 01003, 1992.

[28] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities.* University Press, Cambridge, England, 2 edition, 1952.

[29] P.E. Hart, N.J. Nilsson, and B. Rapahel. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. Cybernet.*, SSC-4(2):100–107, 1968.

[30] R. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, 1960.

[31] T. Jaakkolla, M.I. Jordan, and S.P. Singh. Convergence of stochastic iterative dynamic programming algorithms. In J.D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems 6*, pages 703–710. Morgan Kaufmann, 1994.

[32] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1), 1991.

[33] M.I. Jordan and R.A. Jacobs. Learning to control an unstable system with forward modeling. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufmann.

[34] L.P. Kaelbling. *Learning in Embedded Systems.* PhD thesis, Stanford University, Department of Computer Science, Stanford, CA, 1990. Technical Report TR-90-04.

[35] D.E. Kirk. *Optimal control theory: an introduction.* Englewood Cliffs, 1970.

[36] R.E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[37] D.V. McDermott. Planning and acting. *Cognitive Science*, 2, 1978.

[38] A.W. Moore. personal communication.

[39] A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13(1), October 1993.

[40] K.S. Narendra and M.A.L. Thathachar. *Learning Automata: An Introduction.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[41] N.J. Nilsson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[42] M.L. Puterman and S.L. Brumelle. The analytic theory of policy iteration. In *Dynamic Programming and its Applications*, New York, 1978. Academic Press.

[43] M.L. Puterman and M.C. Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11), July 1978.

[44] R.L. Rivest. Game tree searching by min/max approximation. *Artificial Intelligence*, 34:77–96, 1988.

[45] H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Stat.*, 22:400–407, 1951.

[46] S. Ross. *Introduction to Stochastic Dynamic Programming.* Academic Press, New York, 1983.

[47] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations.* Bradford Books/MIT Press, Cambridge, MA, 1986.

[48] L. Schmetterer. Stochastic approximation. In *Proceedings of the fourth Berkeley Symposium on Mathematics and Probability*, pages 587–609, 1960.

[49] A. Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth Machine Learning Conference*, 1993.

[50] S.P. Singh. Soft dynamic programming algorithms: Convergence proofs, 1993. Poster at CLNL93.

[51] S.P. Singh and R.C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16:227–233, 1994.

[52] R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1984.

[53] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[54] R.S. Sutton. Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. of the Seventh International Conference on Machine Learning*, pages 216–224, San Mateo, CA, 1990. Morgan Kaufmann.

[55] R.S. Sutton. Planning by incremental dynamic programming. In L. Birnbaum and G. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop*, pages 353–357, San Mateo, CA, 1991. Morgan Kaufmann.

[56] R.S. Sutton, A.G. Barto, and R.J. Williams. Reinforcement learning is direct adaptive optimal control. In *Proceedings of the American Control Conference*, pages 2143–2146, Boston, MA, 1991.

[57] G.J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4):257–277, May 1992.

[58] C.J.C.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge Univ., Cambridge, England, 1989.

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

[60] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[61] P.J. Werbos. Consistency of HDP applied to a simple reinforcement learning problem. *Neural Networks*, 3(2):179–189, 1990.

[62] P.J. Werbos. Approximate dynamic programming for real-time control and neural modelling. In D.A. White and D.A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, pages 493–525. Van Nostrand Reinhold, 1992.

[63] S.D. Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action.* PhD thesis, University of Rochester, 1992.

[64] R.J. Williams and L.C. Baird. A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. In *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems*, pages 96–101, 1990.