International Journal of Research Studies in Science, Engineering and Technology
Volume 6, Issue 11, 2019, PP 53-73
ISSN 2349-476X
DOI: https://doi.org/10.22259/2349-476X.0611006

A Survey on Algorithms for Reinforcement Learning

Motumarri Jahnavi[1], Palla Swathi[1], Dr. Bheemalingaiah[2]

[1]*CSE Department,Malineni Lakshmaiah Women"s Engineering College, Guntur, AP*

[2]*Associate Professor, CSE Department, Malineni Lakshmaiah Women"s Engineering College, Guntur, AP*

ABSTRACT

Reinforcement learning is a learning paradigm that focuses on teaching a system how to govern itself in order to optimise a numerical performance measure that represents a long-term goal. Reinforcement learning differs from supervised learning in that the learner receives only partial feedback on his or her predictions. Furthermore, through in uencin gth efutur estat eo fth econtrolle dsystem, the forecasts may have long-term eects. The goal in reinforcement learningis to develop cient learning algorithms, as well as to understand the algorithms' merits and limitations. Reinforcement learning is of great interest because of the large number of practical applications that it can be used to address, ranging from problems in articial intelligence to operations research or control engineering. In this book, we focus on those algorithms of reinforcement learning that build on the powerful theory of dynamic programming. We give a fairly comprehensive catalog of learning problems

Keywords: *reinforcement learning; Markov Decision Processes; temporal difference learning; stochastic approximation; two-timescale stochastic approximation; Monte-Carlo meth- ods; simulation optimization; function approximation; stochastic gradient methods; least- squares methods; overfitting; bias-variance tradeoff; online learning; active learning; plan- ning; simulation; PAC-learning; Q-learning; actor-critic methods; policy gradient; natural gradient.*

Overview

Reinforcement learning (RL) is a learning issue as well as a machine learning subfield. It refers to learning to operate a system in order to maximise some numerical value that indicates a long-term goal as a learning problem. Figure 1 depicts a typical situation in which reinforcement learning takes place: A controller is given the state of the controlled system as well as a reward for the most recent state shift. It then calculates and sends back an action to the system. The system responds by transitioning to a new state, and the cycle begins again. The challenge is to figure out how to regulate the system in such a way that the total reward is maximised. The details of how the data is obtained and how performance is judged differ between the learning challenges.

We assume in this book that the system we want to govern is stochastic. Furthermore, we assume that the measurements of the system's state are sufficiently detailed for the controller to avoid having to reason about how to acquire information about the system state. Markovian Decision Processes are the ideal framework for describing problems with these properties (MDPs). Dynamic programming, which turns the challenge of finding a good controller into the problem of finding a good value function, is the typical method for'solving' MDPs. However, except in the simplest circumstances when the MDP has a small number of states and actions, dynamic programming is not recommended.

The challenge of finding a good controller is transformed into the problem of finding a good value function. Dynamic programming, with the exception of the simplest scenarios when the MDP has a small number of states and actions, is not possible. The RL algorithms we'll talk about here are a technique of turning infeasible dynamic programming methods into workable algorithms that may be used to solve large-scale issues.

The ability of RL algorithms to achieve this goal is based on two essential concepts. The first approach is to use samples to express the dynamics of the control problem in a concise manner. This is significant for two reasons: first, For starters, it enables dealing with learning circumstances in which the dynamics are uncertain. Second, even if the dynamics are favourable, The use of sophisticated function approximation methods to compactly describe value functions is the second important notion behind RL algorithms. This is significant because it enables for the manipulation of huge, high-dimensional state and action spaces. Furthermore, the two concepts are complementary: Samples may be concentrated on a tiny subset of the spaces to which they belong, which creative function approximation techniques could exploit. The heart of creating, assessing, and using RL algorithms is an understanding of the interplay between dynamic programming, sampling, and function approximation.

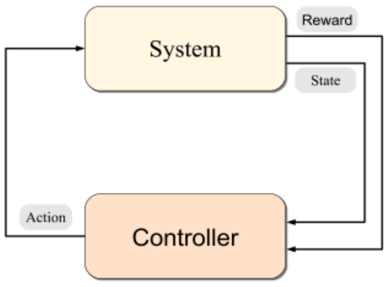


Figure 1: The basic reinforcement learning scenario

The goal of this book is to provide the reader with a glimpse into this lovely field. We are, however, far from the first to set out to achieve this goal. In The book Bertsekas and Tsitsiklis (1996), which explored the theoretical basis, was published after that. Sutton and Barto, the RL'fathers,' published their book a few years later, in which they expressed their ideas on RL in a very clear and approachable manner (Sutton and Barto, 1998). Bertsekas' two-volume book (2007a,b), which devotes one chapter to RL approaches, provides a more modern and complete review of the tools and techniques of dynamic programming/optimal control. 1 When a field is fast evolving, books might become out of date very quickly. In fact, Bertsekas maintains an online version of Chapter 6 of Volume II of his book to keep up with the rising body of new findings. Chang et al. (2008) concentrates on adaptive sampling (i.e., simulation-based performance optimization), whereas Busoniu et al. (2010) recently published a book on function approximation. As a result, RL researchers have access to a substantial body of literature. What appears to be lacking, however, is a self-contained and yet relatively brief summary that can assist newcomers to the field in developing a good sense of the state of the art, as well as existing researchers in broadening their field overview, an article similar to Kaelbling et al. (1996), but with updated contents. This small book's sole objective is to fill this void. We had to make a few, hopefully not too troubling compromises in order to keep the content short. We originally made a compromise by simply presenting data for the total expected discounted reward criterion. This decision is driven by a number of factors. The background on MDPs and dynamic programming is kept ultra-compact as the next compromise (although an appendix is added that explains these basic results). Apart from that, the book strives to cover a little bit of everything related to RL, to the point that the reader should be able to comprehend the whats and hows, as well as apply the algorithms provided. Naturally, we had to be picky about what we showed. The choice was made to concentrate on the fundamental algorithms, ideas, and theories accessible at the time. The user's choices, as well as the compromises that come with them, were described with special care. We tried to be as objective as possible, but some personal issues came up.

Advanced undergraduate and graduate students, as well as academics and practitioners who want a brief review of the state of the art in RL, are the intended audience. Researchers who are currently working on RL may find it interesting to read about aspects of the literature that they are unfamiliar with, in order to widen their RL perspective. It is assumed that the reader is familiar with the fundamentals of linear algebra, calculus, and probability theory. We presume that the reader is familiar with random variables, conditional expectancies, and Markov chains, among other ideas. The reader should be familiar with statistical learning theory, although it is not required, as the main concepts will be explained as needed. Knowledge of machine learning regression techniques will be relevant in several portions of the book.

This book is divided into three sections. We offer the essential background in the first section, Section 2. The notation is introduced here, followed by a brief explanation of Markov Decision Process theory and a discussion of the fundamental dynamic programming techniques. Readers who are familiar with MDPs and dynamic programming should scan through this section to get a feel for the notation. Readers who are unfamiliar with the subject. When it comes to MDPs, it's important to spend enough time here before going on because the rest of the book is mainly based on the findings and ideas offered here. The remaining two components are divided into two pieces, one for each of the two basic RL issues (see Figure 2). The difficulty of learning to predict values linked with states is examined in Section 3. We begin by outlining the fundamental concepts for the so-called tabular scenario, in which the MDP is small enough that one value per state may be stored in an array created in the main memory of a computer. The first algorithm discussed is TD(, which can be seen of as a learning equivalent to dynamic programming's value iteration. Following that, we analyse the more difficult issue in which there are more states than can be stored in a computer's memory. Clearly, the table holding the values must be compressed in this scenario. In a broad sense, this Following that, three new gradient-based approaches (GTD2 and TDC) are described, which can be thought of as enhanced versions of TD() in that they avoid some of the convergence issues that TD() has. The least-squares approaches (namely, LSTD() and -LSPE) are next discussed and compared to the incremental methods outlined above. Finally, we discuss the various options for implementing function approximation as well as the tradeoffs that these options entail.

The second section (Section 4) is devoted to control learning algorithms. First, we'll go through some strategies for improving online performance. We present the "optimism in the face of uncertainty" principle in particular, as well as ways for exploring their environment based on it. Algorithms that are cutting-edge are The remainder of this section is focused to strategies aimed at developing methods for large-scale applications. Because learning in large-scale MDPs is more challenging than learning in small-scale MDPs, the learning aim is lowered to learning a good enough policy in the limit. To begin, direct approaches are explored, which try to directly estimate the optimal action-values. These can be thought of as the learning analogue of dynamic programming's value iteration. The description of actor-critic approaches follows, which can be considered of as the dynamic programming counterpart of the policy iteration process. Both direct policy improvement and policy gradient (i.e., using parametric policy classes) strategies are discussed.

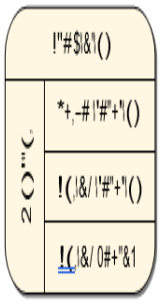


Figure 2: Types of reinforcement learning problems and approaches.

MARKOV DECISION PROCESSES

The objective of this section is to introduce the notation that will be used in the following sections, as well as the most important facts from the theory of Markov Decision Processes (MDPs) that we will need throughout the rest of the book. Readers who are already familiar with MDPs should scan through this part to get a feel for the notation. Readers who are unfamiliar with MDPs should devote enough time to this section to grasp the details. Appendix A contains proofs of the majority of the results (with some simplifications). If you want to understand more about MDPs, you should read one of the many good works on the subject, such as Bertsekas and Shreve (1978), Puterman (1994), or the two-volume book by Bertsekas and Shreve.

Preliminaries

The set of natural numbers is denoted by N: N = 0, 1, 2, etc., whereas the set of real numbers is denoted by R. A column vector is denoted by the symbol v (unless it is transposed, in which case it is denoted by vT). (u, v) = d uivi is the inner product of two finite-dimensional vectors, u, v Rd. $||u||^2$ =(u, u). $||u||_a$ = maxi=1,...,d |ui| defines the maximal norm for vectors. while f = supxX |f (x)| is the definition of a function f: X R. Lipschitz with modulus L R is a mapping T across the metric spaces (M1, d1), (M2, d2) if for each a, b M1, d2(T (a), T (b)) Ld1 (a, b). T is Lipschitz if the modulus L is less than one is referred to as

Markov Decision Processes

We limit our discussion to countable MDPs and the discounted total anticipated reward requirement for clarity. The results, however, also apply to continuous state-action MDPs under certain technical conditions. This is also true of the findings reported in the book's later chapters. A countable MDP is defined as M = (X, A, P0), where X denotes the countable non-empty set of states and A is the countable non-empty set of actions. P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x The following is P0's semantics: P0(U|x, a) yields the chance that the next state and the next state and the next state and the next state and the next state and the next state and the next state and the next state and the next state and the next state and the next state and the next state and the next state. For clarity, we will only address countable MDPs and the discounted total projected reward need. However, under specific technical conditions, the results also apply to continuous state-action MDPs. This holds true for the conclusions presented in the book's subsequent chapters.

M = (X, A, P0) is a countable MDP, with X being the countable non-empty set of states and A denoting the countable non-empty collection of actions. |x, a) P0(|x, a) P0(|x, a) P |x, a) P0(|x, a) P0(|x, a) P |x, a) P0(|x, a) P0(|x, a) P |x, a) P0(|x, a) P0(|x, a) P |x, a) P0(|x, a) P0(|x, a) P P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a) P0(|x, a P0's semantics are as follows: P0(U|x, a) gives the probability that the next state and the next state will occur.

$P(x, a, y) = P_0(\{y\} \times \mathrm{R} \mid x,$
$a)$.

In addition to P, P_0 also gives rise to *the immediate reward function* $r : X \times A \rightarrow \mathrm{R}$, whichgivestheexpectedimmediaterewardrec eivedwhenaction*a*ischoseninstate*x*:If $(Y_{(x,a)},R_{(x,a)}) \sim P_0(\cdot|x,a)$,then

$$r(x, a) = \mathrm{E} \overset{\Sigma}{} R_{(x,a)} \overset{\Sigma}{} .$$

We'll assume that the rewards are bounded by some number R > 0 in the following: for any (x, a) X A, |R(x,a)| R virtually certainly. 3 It follows that if the random rewards are constrained by R, then r = sup(x,a)X A |r(x, a)| R also holds. If both X and A are finite, the MDP is called finite.

Markov Decision Processes are a modelling method for sequential decision-making situations in which a decision maker interacts with a system in a sequential manner. This interaction occurs when an MDP M is used: Let t N be the current time (or stage), and Xt X be the current state.

The random state of the system and the action taken by the decision maker at time t are denoted by and At A, respectively. Once the action has been chosen, it is submitted to the system, which performs the following transition:

$(X_{t+1},R_{t+1}) \sim P_0(\cdot|X_t,A_t)$. (1)

Xt+1 is a random number, and P (Xt+1 = y|Xt = x, At = a) = P(x, a, y) holds for any x, y X, an A. In addition, E [Rt+1|Xt, At] = r (Xt, At). After that, the decision maker observes the next state Xt+1 and rewards Rt+1, selects a new action At+1 A, and repeats the process. The decision maker's purpose is to devise a method of selecting actions that maximises the projected total discounted reward.

Based on the observed past, the decision maker can choose its actions at any time. A behaviour is a rule that describes how the actions are chosen. A random state-action-reward sequence is defined by the decision maker's behaviour and some beginning random state X0. and $A_t \in A$ denote the random state of the system and the action chosen by the decision maker at time *t*, respectively. Once the action is selected, it is sent to the system, which makes a transition:

$$(X_{t+1},R_{t+1}) \sim P_0(\cdot|X_t,A_t). \quad (1)$$

In particular, X_{t+1} is random and P $(X_{t+1}= y|X_t= x, A_t= a) = P(x, a, y)$ holds for any $x, y \in X$, $a \in A$. Further, E $[R_{t+1}|X_t, A_t] = r(X_t, A_t)$. The decision maker then observes the next state X_{t+1} and reward R_{t+1}, chooses a new action $A_{t+1} \in A$ and the process is repeated. The goal of the decision maker is to come up with a way of choosing the actions so as to maximize the expected total discounted reward.

Based on the observed past, the decision maker can choose its actions at any time. A behaviour is a rule that describes how the actions are chosen. A random state-action-reward sequence ((Xt, At, Rt+1); t 0) is defined by a decision maker's behaviour and an initial random state X0, where (Xt+1, Rt+1) is connected to (Xt, At) by (1) and At At time step t, the payment is calculated as follows: The cost of purchasing At items is calculated as KIAt>0 + cAt. As a result, there is a fixed entry fee K. proportionality factor h > 0 of the inventory Finally, when z units are sold, the manager gets paid the sum of p z, where p > 0. We need p > h to make the challenge interesting; otherwise, there will be no motivation to order new products.

As an MDP, this problem can be expressed as follows: Allow the size of the inventory in the evening of day t 0 to be the state Xt. As a result, X = 0, 1,..., M, with M N being the maximum inventory size. The action At is the action prescribed by the behaviour based on the history X0, A0, R1,..., Xt1, At1, Rt, Xt. 4 The whole discounted sum of the benefits incurred is defined as the return underlying a behaviour:

$$R= \sum_{t=0}^{\infty} \gamma^t R_{t+1}.$$

As a result, if 1, incentives received in the future will be valued exponentially less than those obtained at the beginning. A discounted reward MDP is one in which the return is defined by this formula. The MDP is said to be undiscounted when it equals 1.

Regardless of how the process begins, the decision-purpose maker's is to adopt a behaviour that maximises the expected return. It is argued that such maximising behaviour is optimal. Example 1 (lost sales and inventory control): Consider the difficulty of maintaining day-to-day inventory control in the face of fluctuating demand: Every evening, the decision maker must determine the number of goods to be ordered for the following day. The ordered quantity arrives in the morning, and the inventory is replenished. Some stochastic demand is realised during the day, where the demands are independent and have a similar fixed distribution, as shown in Figure 3. The inventory manager's purpose is to manage the inventory in such a way that the present monetary worth of the predicted total future income is maximised.

returns the number of products ordered on day t's evening. As a result, we can choose A = 0, 1,..., M because orders larger than the inventory do not need to be considered.

$$X_{t+1} = ((X_t + A_t) \wedge M - D_{t+1})^+, \quad (2)$$

where $a \wedge b$ *is a shorthand notation for the minimum of the numbers a, b,* $(a)^+ = a \vee 0 =$ max$(a, 0)$ *is the positive part of a, and* $D_{t+1} \in \mathbb{N}$ *is the demand on the* $(t + 1)^{th}$*day. By assumption,* $(D_t;\ t > 0)$ *is a sequence of independent and identically distributed* (i.i.d.*) integer-valued random variables. The revenue made on day t*

$$R_{t+1} = -K\mathbb{I}_{\{A_t>0\}} - c((X_t + A_t) \wedge M - X_t)^+ - hX_t + p((X_t + A_t) \wedge M - X_{t+1})^+ \qquad (3)$$

Equations (2)–(3) can be written in the compact form

$$(X_{t+1}, R_{t+1}) = f(X_t, A_t, D_{t+1}), \qquad (4)$$

with an appropriately chosen function f. Then, P_0 is given by

$$P_0(U \mid x, a) = \mathbb{P}(f(x, a, D) \in U) = \sum_{d=0}^{\infty} \mathbb{I}_{\{f(x,a,d)\in U\}}\, p_D(d).$$

Here $p_D(\cdot)$ is the probability mass function of the random demands and $D \sim p_D(\cdot)$. This finishes the definition of the MDP underlying the inventory optimization problem.

One of the many operations research challenges that leads to an MDP is inventory control. Other issues include transportation system optimization, timetable optimization, and production optimization. Many engineering optimal control issues, such as the optimal control of chemical, electrical, or mechanical systems, naturally involve MDPs (the latter class includes the problem of controlling robots). MDPs can be used to illustrate a variety of information theory challenges (e.g., optimal coding, optimising channel allocation, or sensor networks). Finance is another significant source of issues. Optimal portfolio management and option pricing are examples of these.

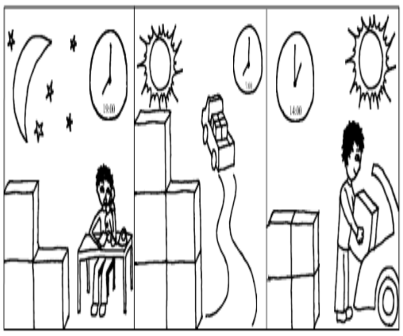


Figure 3: Illustration of the inventory management problem

The MDP was readily specified by a transition function f (cf., (4)) in the instance of the inventory control problem. In fact, transition functions are as powerful as transition kernels: any MDP gives rise to some transition function f, and any MDP gives rise to some transition function f.

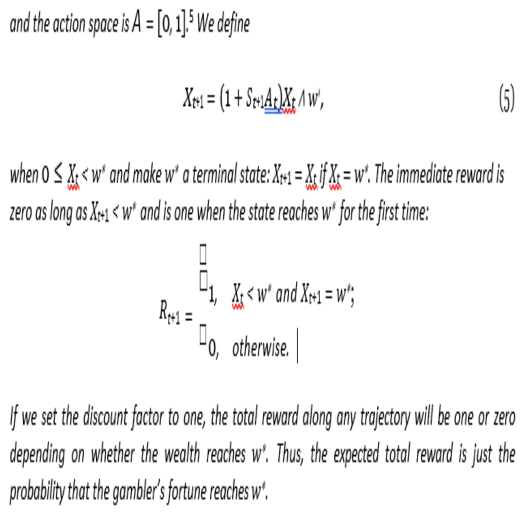
Not all acts are important in all states in some problems. Ordering more things than one has place for in the inventory, for example, is counterproductive. Such meaningless (or forbidden) acts, on the other hand, can always be remapped to other actions, as seen above. This is unnatural in some circumstances and leads to a tangled web of dynamics. Then it could be a good idea to add an extra mapping that assigns the set. Some statuses are impossible to depart in some MDPs: If x is such a state, Xt+s = x virtually certainly holds for any s 1 as long as Xt = x, regardless of what actions are chosen after time t. By convention, we'll suppose that in such terminal or absorbing situations, no reward is incurred. Episodic MDPs are those that have such states. The (usually random) time period from the beginning of time until a terminal condition is attained is referred to as an episode. We frequently consider undiscounted incentives in an episodic MDP, i.e. when.

Exercising 2 (Gambling): A gambler participates in a game in which she can bet any fraction of a dollar. Xt 0 at [0, 1] of her present fortune She reclaims her stake and doubles it.

$X_{t+1} = (1 + S_{t+1}A_t)X_t$.

Here $(S_t;\ t \geq 1)$ is a sequence of independent random variables taking values in $\{-1, +1\}$ with$P(S_{t+1}=1)=p$.Thegoalofthegambleristomaximizetheprobabilitythatherwealth reachesanapriorigivenvalue$w^*>0$.Itisassumedthattheinitialwealthisin$[0,w^*]$.

This problem can be represented as an episodic MDP, where the state space is $X = [0, w^*]$



and the action space is $A = [0, 1]$.[5] We define

$$X_{t+1} = (1 + S_{t+1}A_t)X_t \wedge w', \qquad (5)$$

when $0 \leq X_t < w^*$ and make w^* a terminal state: $X_{t+1} = X_t$ if $X_t = w^*$. The immediate reward is zero as long as $X_{t+1} < w^*$ and is one when the state reaches w^* for the first time:

$$R_{t+1} = \begin{cases} 1, & X_t < w^* \text{ and } X_{t+1} = w^*; \\ 0, & \text{otherwise.} \end{cases}$$

If we set the discount factor to one, the total reward along any trajectory will be one or zero depending on whether the wealth reaches w^*. Thus, the expected total reward is just the probability that the gambler's fortune reaches w^*.

The reader inexperienced with MDPs could believe that all MDPs come with useful finite, one-dimensional state- and action-spaces based on two examples shown so far. If only it were so! In practise, state- and action-spaces are frequently vast, multidimensional spaces in practical applications. The dimensionality of the state space in a robot control application, for example, can be 3—6 times the number of joints the robot has. The state space of an industrial robot may easily be 12—20 dimensions, whereas the state space of a humanoid robot could easily be 100 dimensions. Items would have numerous types in a real-world inventory control application, and prices and costs would change based on the state of the "market," whose condition would therefore form part of the inventory control application.

Value Functions

The most obvious technique to determine an optimal behaviour in an MDP is to list all possible behaviours and then pick the ones that yield the best value for each beginning state. This approach isn't feasible because there are too many behaviours in general. Calculating value functions is a superior technique. In this method, one first computes the so-called optimal value function, which then allows for the very simple determination of an ideal behaviour. When the process is initiated from state x, the optimal value, V(x), of state x X delivers the highest feasible expected return. The optimal value function V: X R is named after it.

Function a behaviour is optimal if it reaches the ideal values in all states. Deterministic stationary policies are a unique type of behaviour that, as we'll see shortly, play a crucial part in MDP theory. They are defined by a mapping that connects states to actions (i.e., X A). The action At is selected using the following at any time t 0.

$A_t=(X_t)$.(6)

More generally, a stochastic stationary policy (or just stationary policy) π maps states to distributions over the action space. When referring to such a policy π, we shall use $\pi(a|x)$ to denote the probability of action a being selected by πinstatex. Note that if a stationary policy is followed in an MDP, i.e.,if

$A_t \sim \pi(\cdot|X_t), \qquad t \in \mathbb{N},$

A (time-homogeneous) Markov chain will be used to represent the state process (Xt; t 0). The set of all stationary policies shall be referred to as stat. For the sake of brevity, we'll refer to "policy" rather than "stationary policy" throughout the following, in the hopes that this won't cause any confusion.

A Markov reward process (MRP) is induced by a stationary policy and an MDP: M = (X, P0), where P0 now assigns a probability measure across X R to each state, determines an MRP. The stochastic process ((Xt, Rt+1); t 0) is generated by an MRP M, where (Xt+1, Rt+1) P0(| Xt). (Note that (Zt; t 0), Zt = (Xt, Rt) is a time-homogeneous Markov process, where R0 is an arbitrary random variable, and ((Xt, Rt+1) is a time-homogeneous Markov process, where R0 is an arbitrary random variable.)

$\sum_{a \in A} \pi(a|x) P_0(\cdot \mid x, a)$. An MRP is called finite if its state space is finite.

Let us now define value functions underlying stationary policies.[b] For this, let us fix some policy $\pi \in \Pi_{stat}$. The *value function*, $V^{\pi} : X \to \mathbb{R}$, underlying π is defined by

$$V^{\pi}(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\Big|\, X_0 = x\right], \quad x \in X, \qquad (7)$$

with the understanding that I the process (Rt; t 1) obtained while following policy is the "reward-part" of the process ((Xt, At, Rt+1); t 0), and (ii) X0 is chosen at random such that P (X0 = x) > 0 holds for all states x. For each state, this second condition ensures that the conditional expectation in (7) is well-defined. If the initial state distribution meets this criterion, the definition of values is unaffected.

The value function underlying an MRP is defined the same way and is denoted by V:

$$V(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\Big|\, X_0 = x\right], \quad x \in X.$$

In an MDP, it will also be useful to define the action-value function, Q: X A R, that underpins a policy stat: Assume that the first action A0 is chosen at random, with P (A0 = a) > 0 for all an A, and that the actions in later phases of the decision process are chosen according to policy. The resulting stochastic process is ((Xt, At, Rt+1); t 0), where X0 is the same as in the definition of V. Then X is optimal for all states at the same time. It's worth noting that in order for (8) to hold, (|x) must be focused on the set of actions that maximise Q(x,). In general, an action that maximises Q(x,) for some state x is called greedy with respect to Q in state x, given some action-value function, Q: X A R. Greedy w.r.t. Q is a policy that picks greedy actions only with respect to Q in all states.

As a result, a greedy policy with respect to Q is optimal, i.e., knowing Q is enough to establish an optimal policy. Knowing V, r, and P is also sufficient to act. The next question is how to find *V*∗or*Q*∗. Let us start with the simple rquestion of how to find the value function of a policy:

$$Q^{\pi}(x, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\Big|\, X_0 = x, A_0 = a\right], \quad x \in X, a \in A.$$

Similarly to $V^*(x)$, the *optimal action-value* $Q^*(x, a)$ at the state-action pair (x, a) is defined as the maximum of the expected return under the constraints that the process starts at state x, and the first action chosen is a. The underlying function $Q^* : X \times A \to \mathbb{R}$ is called the *optimal action-value function.*

The optimal value- and action-value functions are connected by the following equations:

$$\begin{aligned} V^*(x) &= \sup_{a \in A} Q^*(x, a), && x \in X, \\ Q^*(x, a) &= r(x, a) + \gamma \sum_{y \in X} P(x, a, y) V^*(y), && x \in X, a \in A. \end{aligned}$$

In the class of MDPs considered here, an optimal stationary policy always exists:

$$V^*(x) = \sup_{\pi \in \Pi_{stat}} V^{\pi}(x), \quad x \in X.$$

In fact, any policy $\pi \in \Pi_{stat}$ which satisfies the equality

$$\sum_{a \in A} \pi(a|x) Q^*(x, a) = V^*(x) \qquad (8)$$

Fact 1 (Bellman Equations for Deterministic Policies): *Fix an MDP $M = (X, A, P_0)$, a discount factor γ and deterministic policy $\pi \in \Pi_{stat}$. Let r be the immediate reward function of M. Then V^{π} satisfies*

$$V^{\pi}(x) = r(x, \pi(x)) + \gamma \sum_{y \in X} P(x, \pi(x), y) V^{\pi}(y), \qquad x \in X. \tag{9}$$

This system of equations is called the Bellman equation *for V^{π}. Define the* Bellman operator *underlying π, $T^{\pi} : \mathrm{R}^X \to \mathrm{R}^X$, by*

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

With the help of T^{π}, Equation (9) can be written in the compact form

$$T^{\pi}V^{\pi} = V^{\pi}. \tag{10}$$

Note that this is a linear system of equations in V^{π} and T^{π} is an affine linear operator. If $0 < \gamma < 1$ then T^{π} is a maximum-norm contraction and the fixed-point equation $T^{\pi}V = V$ has a unique solution.

When the state space X is finite, say, it has D states, R^X can be identified with the D-dimensional Euclidean space and $V \in \mathrm{R}^X$ can be thought of as a D-dimensional vector: $V \in \mathrm{R}^D$. With this identification, $T^{\pi}V$ can also be written as $r^{\pi} + \gamma P^{\pi}V$ with an appropriately defined vector $r^{\pi} \in \mathrm{R}^D$ and matrix $P^{\pi} \in \mathrm{R}^{D \times D}$. In this case, (10) can be written in the form

$$r^{\pi} + \gamma P^{\pi} V^{\pi} = V^{\pi}. \tag{11}$$

The above facts also hold true in MRPs, where the Bellman operator $T : \mathrm{R}^X \to \mathrm{R}^X$ is defined by

$$(TV)(x) = r(x) + \gamma \sum_{y \in X} P(x, y) V(y), \quad x \in X.$$

The optimal value function is known to satisfy a certain fixed-point equation:

Fact 2 (Bellman Optimality Equations): *The optimal value function satisfies thefixed-point equation*

$$V^{*}(x) = \sup_{a \in A} \left\{ r(x, a) + \gamma \sum_{y \in X} P(x, a, y) V^{*}(y) \right\}, \qquad x \in X. \tag{12}$$

Note that this is a nonlinear operator due to the presence of sup. *With the help of T^{*}, Equation (12) can be written compactly as*

$$T^{*}V^{*} = V^{*}.$$

If $0 < \gamma < 1$, then T^{*} is a maximum-norm contraction, and the fixed-point equation $T^{*}V = V$ has a unique solution.

In order to minimize clutter, in what follows we will write expressions like $(T^{\pi}V)(x)$ as $T^{\pi}V(x)$, with the understanding that the application of operator T^{π} takes precedence to the applycation of the point evaluation operator, "$\cdot\,(x)$".

The action-value functions underlying a policy (or an MRP) and the optimal action-value function also satisfy some fixed-point equations similar to the previous ones:

Fact 3 (Bellman Operators and Fixed-point Equations for Action-value Functions): *With a slight abuse of notation, define $T^{\pi} : \mathrm{R}^{X \times A} \to \mathrm{R}^{X \times A}$ and $T^{*} : \mathrm{R}^{X \times A} \to \mathrm{R}^{X \times A}$ as follows:*

$$T^{\pi}Q(x, a) = r(x, a) + \gamma \sum_{y \in X} P(x, a, y) Q(y, \pi(y)), \qquad (x, a) \in X \times A, \tag{14}$$

$$T^{*}Q(x, a) = r(x, a) + \gamma \sum_{y \in X} P(x, a, y) \sup_{a' \in A} Q(y, a'), \qquad (x, a) \in X \times A. \tag{15}$$

Note that T^{π} is again affine linear, while T^{*} is nonlinear. The operators T^{π} and T^{*} are maximum-norm contractions. Further, the action-value function of π, Q^{π}, satisfies $T^{\pi}Q^{\pi} = Q^{\pi}$ and Q^{π} is the unique solution to this fixed-point equation. Similarly, the optimal action- value function, Q^{*}, satisfies $T^{*}Q^{*} = Q^{*}$ and Q^{*} is the unique solution to this fixed-point equation.

Dynamic Programming Algorithms for Solving MDPs

The above facts provide the basis for the *value*- and *policy-iteration* algorithms. *Value iteration* generates a sequence of value functions.

$V_{k+1}=T^{*}V_k, \qquad k\geq 0,$

whereV_0isarbitrary.ThankstoBanach'sfixed-pointtheorem,(V_k;$k\geq 0$)convergestoV^{*} at a geometricrate.

Valueiterationcanalsobeusedinconjunctionwithaction-valuefunctions; inwhichcase, it takes the form

$Q_{k+1}=T^{*}Q_k, \qquad k\geq 0,$

whichagainconvergestoQ^{*}atageometricrate.TheideaisthatonceV_k(orQ_k)iscloseto V^{*}(resp.,Q^{*}),apolicythatisgreedywithrespecttoV_k(resps.,Q_k)willbeclose-to-optimal. Inparticular,thefollowingboundisknowntohold:Fixanaction-valuefunction$Q$andlet π be a greedy policy w.r.t. Q. Then the value of policy π can be lower bounded as follows (e.g., Singh and Yee, 1994, Corollary2):

V π (x) ≥ V ∗ (x) − 2 1 − γ kQ − Q ∗ k∞, x ∈ X . (16)

The process of policy iteration is as follows. Set a starting policy of 0 at random. At iteration k > 0, calculate the action-value function underlying k. (This is referred to as the policy evaluation stage.) Then define k+1 as a greedy strategy with respect to Qk, given Qk (this is called the policy improvement step). In terms of the value function derived using k iterations of value iteration after k iterations, policy iteration delivers a policy that is not worse than the greedy policy if the two methods start with the same initial value function. A single step in policy iteration, on the other hand, has a significantly higher computational cost than a single update in value iteration (due to the policy review phase).

Value Prediction Problems

We look at the difficulty of estimating the value function V that underpins several Markov reward processes in this section (MRP). Value prediction issues can occur in a variety of ways: Value prediction problems include estimating the likelihood of a future occurrence, the estimated time before an event occurs, and the (action-)value function underpinning a policy in an MDP. Predicting the failure likelihood of a big power system (Frank et al., 2008) or estimating taxi-out times of flights at congested airports (Balakrishna et al., 2008) are only two of the many uses**.** Because the value of a state is defined as the expectation of the random return when the process is initiated from the given state, computing an average over numerous independent realisations starting from the given state is an apparent approach of estimating this value. This is an example of what is known as the Monte-Carlo approach. Unfortunately, the returns' variance can be significant, implying that the estimates' quality will be low. Also, when dealing with others, It may be impossible to reset the state of a system in a closed-loop form (that is, when estimation occurs while interacting with the system). The Monte-Carlo method cannot be used in this scenario without introducing additional bias. Temporal difference (TD) learning (Sutton, 1984, 1988), without a doubt one of the most important ideas in reinforcement learning, is a technique that can be used to overcome these problems.

Temporal Difference Learning in Finite State Spaces: The usage of bootstrapping is a distinctive aspect of TD learning: predictions are employed as targets during the learning process. In this section, we'll go over the basics of the TD algorithm and how bootstrapping works. Then, we compare TD learning to (basic) Monte-Carlo approaches, arguing that each has its own strengths. Finally, the TD() technique is shown, which unites the two approaches. In this paper, we only consider the case of small, finite MRPs, in which the value-estimates of all states can be kept in an array or table in the main memory of a computer, a condition known as the tabular case in the reinforcement learning literature. When a tabular representation is used, extensions of the ideas provided here to huge state spaces are possible.

Fix some finite Markov Reward Process M. We wish to estimate the value function V underlying M given a realization $((X_t, R_{t+1}); t \geq 0)$ of M. Let $\hat{V}_t(x)$ denote the estimate of state x at time t (say, $\hat{V}_0 \equiv 0$). In the t^{th} step TD(0) performs the following calculations:

$$\delta_{t+1} = R_{t+1} + \gamma \hat{V}_t(X_{t+1}) - \hat{V}_t(X_t),$$

$$\hat{V}_{t+1}(x) = \hat{V}_t(x) + \alpha_t \delta_{t+1} \mathbb{I}_{\{X_t = x\}}, \; x \in X. \tag{17}$$

Here the *step-size* sequence $(\alpha_t; t \geq 0)$ consists of (small) nonnegative numbers chosen by the user. Algorithm 1 shows the pseudocode of this algorithm.

A closer inspection of the update equation reveals that the only value changed is the one associated with X_t, i.e., the state just visited (cf. line 2 of the pseudocode). Further, when $\alpha_t \leq 1$, the value of X_t is moved towards the "target" $R_{t+1} + \gamma \hat{V}_t(X_{t+1})$. Since the target depends on the estimated value function, the algorithm uses *bootstrapping*. The term "temporal difference" in the name of the algorithm comes from that δ_{t+1} is defined as the

TABULAR TD(0):

Differrence in state values corresponding to subsequent time steps A temporal difference error, in particular, is referred to as t+1. Tabular TD(0), like many other reinforcement learning methods, is a

Tochastic approximation (SA) algorithm. It's obvious that if something onverges, it must converge to a function V such that, given V, the predicted temporal difference,

$$F\hat{V}(x) \overset{\text{def}}{=} \mathbb{E}\left[R_{t+1} + \gamma \hat{V}(X_{t+1}) - \hat{V}(X_t) \,\middle|\, X_t = x \right],$$

is zero for all states x, at least for all states that are sampled infinitely often. A simple calculation shows that $F\hat{V} = T\hat{V} - \hat{V}$, where T is the Bellman-operatorun derlying the MRPconsidered.ByFact1,$F\hat{V}$

= 0 has a unique solution, the value function V . Thus, if

TD(0) converges (and all states are sampled infinitely often) then it must converge to V . To study the algorithm's convergence properties, for simplicity

MRP considered. By Fact 1, $F\hat{V} = 0$ has a unique solution, the value function V. Thus, if TD(0) converges (and all states are sampled infinitely often) then it must converge to V .
To study the algorithm's convergence properties, for simplicity, assume that $(X_t; t \in \mathbb{N})$ is a stationary, ergodic Markov chain.[7] Further, identify the approximate value functions $\hat{V}_t$ with D-dimensional vectors as before (e.g., $\hat{V}_{t,i} = \hat{V}_t(x_i)$, $i = 1, \ldots, D$, where $D = |X|$ and $X = \{x_1, \ldots, x_D\}$). Then, assuming that the step-size sequence satisfies the *Robbins-Monro (RM) conditions,*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < +\infty$$

the sequence $(\hat{V}_t \in \mathbb{R}^D; t \in \mathbb{N})$ will track the trajectories of the ordinary differential equation (ODE)

$$\dot{v}(t) = c F(v(t)), \quad t \geq 0, \tag{18}$$

where $c = 1/D$ and $v(t) \in \mathbb{R}^D$ (e.g., Borkar, 1998). Borrowing the notation used in (11), the above ODE can be written as

$$\dot{v} = r + (\gamma P - I)v.$$

Note that this is a *linear* ODE. Since the eigenvalues of $\gamma P - I$ all lie in the open left half complex plane, this ODE is globally asymptotically stable. From this, using standard results

[7] Remember that a Markov chain $(X_t, t \in \mathbb{N})$ is ergodic if it is irreducible, aperiodic and positive recurrent.

Practically, this means that the law of large number holds for sufficiently regular functions of the chain.

On step-sizes

Because many of the algorithms we'll talk about involve step-sizes, it's worth spending some time talking about them. t = c/t, with c > 0, is a simple step-size sequence that meets the aforementioned requirements. In general, any step-size sequence of the form t = ct will work for as long as 1/2 1. The smallest step-sizes are found in the = 1 series of these step-size sequences. This decision will be the best asymptotically, but in terms of the algorithm's transitory behaviour, choosing a value closer to 1/2 will be better (since with this choice the step-sizes are bigger and thus the algorithm will make larger moves). It is feasible to perform even better. In reality, Polyak developed a simple method called iterate-averaging. In fact, people frequently utilise constant step sizes in practise, which obviously violates the RM requirements. This decision is supported by two factors: first, the algorithms are frequently utilised in non-stationary environments (i.e., the policy to be evaluated might change). Second, the algorithms are frequently used only in the context of tiny samples. (When a constant step-size is used, the parameters converge in distribution, and the variance of the limiting distribution is proportional to the step-size chosen.) There is also a lot of work being done on developing methods for automatically tuning step-sizes, see (Sutton, 1992; Schraudolph, 1999; George and Powell, 2006) and the references therein. The jury is still out on which strategy is the most effective. The procedure can also be employed on an observation series of the type ((Xt, Rt+1, Yt+1); t 0), where (Xt; t 0) is an arbitrary ergodic Markov chain over X, (Yt+1, Rt+1) P0(| Xt), and (Yt+1, Rt+1) P0(| Xt). The shift is in the way temporal differences are defined:

$$\delta_{t+1} = R_{t+1} + \gamma \hat{V}(Y_{t+1}) - \hat{V}(X_t).$$

Then, without any additional circumstances, Vt still converges almost inexorably to the un converged value function. The MRP's underpinnings (X , P0). The distribution of states (Xt; t 0) in particular has no bearing on this is intriguing for a variety of reasons. We may be able to alter the distribution of the states (Xt; t 0) independently of the MRP if the samples are created using a simulator. This could be useful for balancing out any The sequence in which an episode begins inequalities in the stationary distribution underlying the Markov kernel P. Another application is to learn about a specific policy goal. in an MDP while adhering to another policy, sometimes referred to as created by employing the behaviour policy, where the action taken does not match the action that the target policy would have taken in the given state, while the remainder is kept. This method could allow you to learn about many policies at once (more generally, about multiple long-term prediction problems). Off-policy learning is when you learn about one policy while following another. Because of this, we'll refer to learning based on triplets ((Xt, Rt+1, Yt+1); t 0) as off-policy learning when Yt+1 = Xt+1. When the intention is to apply the algorithm to an episodic problem, there is a third, technical purpose. The triplets (Xt, Rt+1, Yt+1) are chosen in this example as follows: First, the transition kernel P(X) is sampled for Yt+1.

In other words, the process is resumed from the starting state distribution P0 when it reaches a terminal state. The time between a P0 restart and reaching a terminal state is referred to as an episode (hence the name of episodic problems). Continuous sampling with restarts from P0 is the name given to this method of creating a sample.

Because tabular TD(0) is a standard linear SA method, its rate of convergence will be of the order O(1/t) (for further details, see Tadi'c (2004) and the references therein). The constant component in the rate, on the other hand, will be heavily influenced by the step-size sequence chosen, the features of the kernel P0, and the value of.

Every-visit Monte-Carlo

As previously indicated, computing sample means can also be used to estimate the value of a state, giving rise to the so-called every visit Monte-Carlo approach. Here, we clarify what we mean more specifically and compare the resulting method to TD (0).

Consider an episodic problem to solidify your views (otherwise, it is impossible to finitely compute the return of a given state since the trajectories are infinitely long). Let M = (X, P0) be the underlying MRP, and ((Xt, Rt+1, Yt+1); t 0) be the result of continuous sampling in M with restarts from some distribution P0 defined over X. Let (Tk; k 0) be a function.

(thus, for each k, XTk is sampled from P0).

For a given time t, let $k(t)$ be the unique episode index such that $t \in [T_k, T_{k+1})$. Let

$$R_t = \sum_{s=t}^{T_{k(t)+1}-1} \gamma^{s-t} R_{s+1} \tag{19}$$

denote the return from time t on until the end of the episode. Clearly, $V(x) = \mathbb{E}[R_t | X_t = x]$, for any state x such that $\mathbb{P}(X_t = x) > 0$. Hence, a sensible way of updating the estimates is to use

$$\hat{V}_{t+1}(x) = \hat{V}_t(x) + \alpha_t (R_t - \hat{V}_t(x)) \mathbb{I}_{\{X_t = x\}}, \qquad x \in \mathcal{X}.$$

Because multi-step forecasts of the value are used in Monte-Carlo methods like the one above (cf. Equation (19), they are referred to as multi-step methods. Algorithm 2 is the pseudo-code for this update-rule.

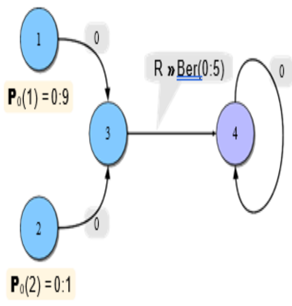
This algorithm is an example of stochastic approximation once more. As a result, the ODE v(t) = V v determines its behaviour (t). Because of the one-of-a-kind globally asymptotically stable

This ODE's equilibrium is V, and Vt usually always converges to V. Given that both algorithms achieve the same aim, one would wonder which is superior.

TD (0) OR MONTE-CARLO?

Figure 4 shows the undiscounted episodic MRP. Either 1 or 2 are the initial states. The process begins in state 1 with a high likelihood, while it begins in state 2 less frequently. Consider how TD(0) will act in the second state. State 3 has been visited 10 k times on average by the time state 2 is visited for the kth time. Assume that t = 1/(t + 1) is the case. The TD(0) update at state 3 is reduced to averaging the Bernoulli rewards incurred on leaving state 3. Var Vt(3) 1/(10 k) for the kth visit of state 2 (obviously, E Vt(3) = V (3) = 0.5). As a result, the goal of the state 2 update will be an accurate estimate of the true value of state 2. Take a look at the Monte-Carlo approach. The Monte-Carlo technique disregards the estimated value of state 3 and relies solely on the Bernoulli rewards. Var [Rt|Xt = 2] = 0.25, indicating that the target's variance does not change over time. This causes the Monte-Carlo approach take longer to converge in this case, demonstrating that bootstrapping can be beneficial in some cases.

Imagine that the challenge is changed so that the reward connected with the transition from state 3 to state 4 is made deterministically.



transitioning from condition 3 to state 4, when a Bernoulli random variable with parameter 0.5 is used, the payoff is zero. The fourth state is the final one. The process is reset to state 1 or 2 when it reaches the terminal state. Starting in state 1 has a probability of 0.9, while starting at state 2 has a probability of 0.1. the same as one Since Rt = 1 is the true target value, the Monte-Carlo method becomes faster in this situation, whereas for the value of state 2 to approach its true value, TD(0) must wait until the estimate of the value at state 3 approaches its true value. The convergence of TD is slowed as a result of this (0). In fact, for

I 1,..., N, one might envisage a longer chain of states, where state I + 1 follows state I and the only time a nonzero reward is incurred is when transitioning from state N 1 to state N. The pace of convergence of the Monte-Carlo approach is unaffected by the value of N in this example, whereas TD(0) would slow down as N increased.

TD(Λ): Unifying Monte-Carlo and TD(0)

Both Monte-Carlo and TD(0) have merits, as seen in the previous cases. Surprisingly, there is a method to bring these perspectives together. The so-called TD() family of techniques does this (Sutton, 1984, 1988). [0, 1] is a parameter that enables for interpolation between Monte-Carlo and TD(0) updates in this case: TD(0) is obtained by setting = 0 (thus the term TD(0)), but TD(1) is obtained by setting = 1, i.e., TD(1) is identical to a Monte-Carlo approach. In essence, given some > 0, the TD() update's targets are delivered as a blend of

$$R_{t:k} = \sum_{s=t} \gamma^{s-t} R_{s+1} + \gamma^{k+1} \hat{V}_t(X_{t+k+1}),$$

where the exponential weights (1)k, k 0 are the mixing coefficients As a result, TD() will be a multi-step technique for > 0. The inclusion of so-called eligibility traces makes the algorithm incremental.

In fact, the eligibility traces can be defined in a variety of ways, and as a result, TD() exists in a variety of different versions. The following is the TD() update rule for the so-called accumulating traces:

$$\begin{aligned}
\delta_{t+1} &= R_{t+1} + \gamma \hat{V}_t(X_{t+1}) - \hat{V}_t(X_t), \\
z_{t+1}(x) &= \mathbb{I}_{\{x=X_t\}} + \gamma\lambda z_t(x), \\
\hat{V}_{t+1}(x) &= \hat{V}_t(x) + \alpha_t \delta_{t+1} z_{t+1}(x), \\
z_0(x) &= 0, \\
& x \in X.
\end{aligned}$$

Here $z_t(x)$ is the *eligibility trace* of state x. The rationale of the name is that the value of $z_t(x)$ modulates the influence of the TD error on the update of the value stored at state x. In another variant of the algorithm, the eligibility traces are updated according to

$$z_{t+1}(x) = \max(\mathbb{I}_{\{x=X_t\}}, \gamma\lambda z_t(x)), \qquad x \in X.$$

The replacing traces update is what it's called. The trace-decay option regulates the amount of bootstrapping in these updates: When lim0+(1) k0 kRt:k = Rt:0 = Rt+1 + Vt(Xt+1), the above methods become equivalent to TD(0) (because lim0+(1) k0 kRt:k = Rt:0 = Rt+1 + Vt(Xt+1)). When = 1, we get the TD(1) algorithm, which simulates the previously reported every-visit Monte-Carlo technique in episodic problems using accumulating traces. (For an exact equivalent, assume that value updates occur only at the ends of trajectories, and that the updates are simply summed up to that point.) Because the discounted sum of temporal differences along a trajectory from a start to a terminal state telescopes and delivers the difference between the return along the trajectory and the value estimate at the start, the assertion follows. Replacing traces and = 1 refer to a Monte-Carlo algorithm in which a state is only updated when it is encountered for the first time in a trajectory. The first-visit Monte-Carlo method is the related algorithm. The formal relationship between the first-visit Monte-Carlo approach and TD(1) with replacing traces is known to be valid only for the undiscounted situation (Singh and Sutton, 2003).

Algorithm 3 The function that implements the tabular TD(λ) algorithm with replacing traces. This function must be called after each transition.

```
function TDLAMBDA($X, R, Y, V, z$)
Input: $X$ is the last state, $Y$ is the next state, $R$ is the immediate reward associated with
    this transition, $V$ is the array storing the current value function estimate, $z$ is the array
    storing the eligibility traces
1: $\delta \leftarrow R + \gamma \cdot V[Y] - V[X]$
2: for all $x \in X$ do
3:     $z[x] \leftarrow \gamma \cdot \lambda \cdot z[x]$
4:     if $X = x$ then
5:         $z[x] \leftarrow 1$
6:     end if
7:     $V[x] \leftarrow V[x] + \alpha \cdot \delta \cdot z[x]$
8: end for
9: return $(V, z)$
```

1996). The pseudocode for the variation with replacement traces is given by Algorithm 3. The best value of is discovered by trial and error in practise. In fact, even during the algorithm, the value of can be altered without affecting convergence. This is true for a variety of additional potential eligibility trace modifications (for precise conditions, see Bertsekas and Tsitsiklis, 1996, Section 5.3.3 and 5.3.6). In reality, the replacement traces variant of the method is thought to perform better (for some examples when this happens, consult Sutton and Barto, 1998, Section 7.8). It has been noticed that > 0 is useful when the learner only has a partial understanding of the state, or (in a related case) when function approximation is used to approximate the value functions in a mathematical model.

ALGORITHMS FOR LARGE STATE SPACES

When the state space is large (orinfinite), it is not feasible to keep a separate value for each state in the memory. In such cases, we often seek an estimate of the values in the form

$$V_\theta(x) = \theta^T \phi(x), \qquad x \in X,$$

where $\theta \in \mathbb{R}^d$ is a vector of parameters and ϕ : $X \rightarrow \mathbb{R}^d$ is a mapping of states to d-dimensional vectors. For state x, the components $\phi_i(x)$ of the vector $\phi(x)$ are called the *features* of state x and ϕ is called a *feature extraction* method. The individual functions

ϕ_i: $X \rightarrow \mathbb{R}$ defining the components of ϕ are called *basis functions*.

The features (or basic functions) can be created in a variety of ways once you have access to the state. If x R (i.e., X R), a polynomial, Fourier, or wavelet basis can be used up to a certain order. For example,

provided a suitable measure (such as the stationary distribution) over the states is available, (x) = (1, x, x2,..., xd1)T can be used in the case of a polynomial basis or an orthogonal system of polynomials. This second option may aid in the faster convergence of the incremental algorithms we will explore shortly. In the case of multidimensional statespaces, the *tensor productcon struction* is a commonly used way to construct features given features of the states'

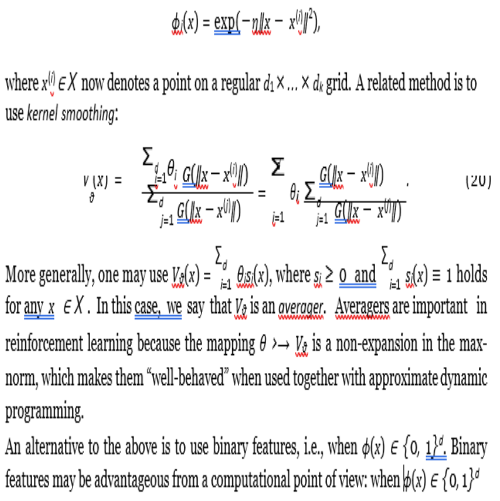
individual components. The tenso rproductcon struction works as follows: Imagine that$X \subset X_1 \times X_2 \times \ldots \times X_k$.Let $\phi^{(i)}$:$X_i \longrightarrow$Rdibea feature extractor defined for theith state component. Thetensor product $\phi = \phi^{(1)} \otimes \ldots \otimes \phi^{(k)}$feature extractor will have $d = d_1 d_2 \ldots d_k$ components, which can be conveniently indexed using multi-indices of the form $(i_1, \ldots, i_k), 1 \leq i_j \leq d_j, j = 1, \ldots, k$. Then $\phi_{(i,\ldots,i)}(x) = \phi^{(1)}(x_1)\phi^{(2)}(x_2) \ldots \phi^{(k)}(x_k)$. When $X \subset \mathbb{R}^k$, one particularly

popular choice is to use radial basis function (RBF) networks, when $\phi^{(i)}(x_i) = (G(|x_i -$

$x^{(1)}|), \ldots, G(|x_i - x^{(d_i)}|))^T$.
Here$x^{(j)} \in \mathbb{R}(j = 1, \ldots, d_i)$ is fixed by the user andG is a suitable function. A typical choice

for G is $G(z) = \exp(-\eta\ z^2\ )$ where $\eta > 0$ is a scale

parameter. The tensor product construct in this cases places Gaussians at points of a regular grid and the $i$thbasis function becomes



$$\phi_i(x) = \exp(-\eta\|x - x^{(i)}\|^2),$$

where $x^{(i)} \in X$ now denotes a point on a regular $d_1 \times \ldots \times d_k$ grid. A related method is to use *kernel smoothing*:

$$V_\theta(x) = \frac{\sum_{i=1}^{d} \theta_i\ G(\|x - x^{(i)}\|)}{\sum_{j=1}^{d} G(\|x - x^{(j)}\|)} = \sum_{i=1} \theta_i \frac{G(\|x - x^{(i)}\|)}{\sum_{j=1}^{d} G(\|x - x^{(j)}\|)}. \qquad (20)$$

More generally, one may use $V_\theta(x) = \sum_{i=1}^{d} \theta_i s_i(x)$, where $s_i \geq 0$ and $\sum_{i=1}^{d} s_i(x) \equiv 1$ holds for any $x \in X$. In this case, we say that V_θ is an *averager*. Averagers are important in reinforcement learning because the mapping $\theta \mapsto V_\theta$ is a non-expansion in the max-norm, which makes them "well-behaved" when used together with approximate dynamic programming.

An alternative to the above is to use binary features, i.e., when $\phi(x) \in \{0, 1\}^d$. Binary features may be advantageous from a computational point of view: when $\phi(x) \in \{0, 1\}^d$

Then V(x) = i:i(x)=1 I I I I I I I I I I I I I I I If (x) is s-sparse (i.e., only s elements of (x) are non-zero), the value of state x can be determined at the cost of s additions, given that the index of the non-zero components of the feature vector can be obtained directly.

This is the situation when the features are defined using state aggregation. In this instance,

(The individual features') coordinate functions correspond to markers of non-overlapping sections of the state space X whose union encompasses X. (i.e., the regions form a partition of the state space). T(x) will obviously be constant over the individual areas in this situation, therefore state aggregation effectively "discretizes" the state space. A function that aggregates states.

Tile coding (formerly known as CMAC, Albus, and others) is another option that leads to binary features.

1971 and 1981. The basis functions of correspond to indicator functions of numerous shifting partitions (tilings) of the state space in the simplest version of tile coding: if s tilings are utilised, will be s-sparse. The offsets of the tilings corresponding to different dimensions should be different to make tile coding an efficient function approximation approach.

THE CURSE OF DIMENSIONALITY

The issue with tensor product constructions, stateaggre- gation and straight forward tile coding is that when the state space is high dimensional they quickly become intractable: For example, a tiling of $[0, 1]^D$ with cubical regions with side- lengths of ε gives rise to $d = \varepsilon^{-D}$-dimensional feature- and parameter-vectors. If $\varepsilon = 1/2$ and $D = 100$, we get the enormous number $d \approx 10^{30}$. This is problematic since state- representations with hundreds of dimensions are common in applications. At this stage, onemaywonderifitispossibleatalltosuccessfully dealwithapplicationswhenthestate lives in a high dimensional space. What often comes at rescue is that the actual problem. The complexity of the state variable could be substantially lower than what is predicted by counting the number of dimensions (although, there is no guarantee that this happens). To see why this is occasionally true, consider that the same problem can have numerous representations, some of which have low-dimensional state variables and others with high-dimensional state variables. Because the user often chooses the state-representation in a conservative manner, it's possible that many of the state variables are useless in the chosen representation. It's also possible that the actual states encountered are on (or near) a low-dimensional submanifold of the specified high-dimensional "state-space."

Consider an industrial robot arm with three joints and six joints. The number of states represented will easily be in the millions, yet the inherent dimensionality will remain at 12. In fact, the higher the dimensionality, the more cameras we have. A straightforward strategy aimed at reducing dimensionality would recommend using as few cameras as possible. More information, on the other hand, isn't going to hurt! As a result, smart algorithms and function approximation methods that can deal with high-dimensional but low-complexity situations should be sought.

Strip-like tilings combined with hash functions, interpolators employing low-discrepancy grids (Lemieux, 2009, Chapters 5 and 6), and random projections are all possibilities (Das- gupta and Freund, 2008). Methods for approximating nonlinear functions (for example, neural networks with sigmoidal transfer functions in the hidden layers or RBF nets).

which should be compared to its parametric counterpart (20). Other examples include methods that work by finding an appropriate function in some large (infinite dimensional) function space that fits an empirical error. The function space is usually a Reproducing Kernel Hilbert space which is a convenient choice from the point of view of optimization

$$V_D^{(k)}(x) = \sum_{i=1} v_i \frac{K_D(x,x_i)}{\sum_{j=1}^{n} K_D(x,x_j)},$$

It needs to be contrasted with its parametric counterpart (20). Methods that work by finding a suitable function in a huge (infinite dimensional) function space that fits an empirical inaccuracy are another example. The function space is typically a Reproducing Kernel Hilbert space, which is a good choice from an optimization standpoint. Spline smoothers (Wahba, 2003) and Gaussian process regression are examples of special situations (Rasmussen and Williams, 2005). Another option is to divide the input space recursively into finer sections using a heuristic criterion and then forecast the values in the leafs using a simple approach.

Eventually, tree-based approaches emerge. The line separating parametric and nonparametric approaches is a hazy one. When the number of basis functions is permitted to change (i.e., new basis functions are introduced as needed), a linear predictor becomes a nonparametric technique. Thus, when experimenting with alternative feature extraction approaches, we may argue that we are using a nonparametric strategy from the perspective of the total tuning process. In fact, if we take this perspective, it follows that "real" parametric approaches are rarely, if ever, utilised in practise.

The inherent flexibility of nonparametric approaches is a benefit. This, however, usually comes at the cost of greater computing complexity. As a result, efficient implementations are critical when employing non-parametric approaches (e.g., one should use k-D trees when implementing nearest neighbour methods, or the Fast Gaussian Transform when imple- menting a Gaussian smoother). Nonparametric approaches must also be fine-tuned. $K_D(x,x_j)$

They have the potential to overfit or underfit. If k is too large in a k-nearest neighbour approach, for example,

If k is too large, the approach will smooth out too much (i.e., it will underfit), whereas if k is too little, it will fit to the noise (i.e., overfit). Section 3.2.4 will go over overfitting in more detail. The reader is urged to consult for more information on nonparametric regression.

Although we will explore parametric function approximation (and in many cases linear function approximation) in the next sections, many of the strategies can be extended to nonparametric methods. When such extensions are available, we will make a note of it. Until now, the debate has implicitly assumed that the state is measurable. In real-world applications, however, this is rarely the case. Fortunately, the methods we'll explore below don't require direct access to the states; they'll work just as well if any "sufficiently descriptive feature-based representation" of the states is provided (such as the camera images in the robot-arm example). Constructing state estimators (or observers, in control language) based on the data is a popular technique to arrive at such a representation.

TD(Λ) WITH FUNCTION APPROXIMATION

Letusreturntotheproblemofestimatingavaluefunction$V$ofaMarkovrewardprocess $M = (X, P_0)$, but now assume that the state space is large (or even infinite). Let $D = ((X_t, R_{t+1}); t \geq 0)$ be a realization of M. The goal, as before, is to estimate the value function of M given D in an incremental manner.

Choose a smooth parametric function-approximation method (V_θ; $\theta \in \mathbb{R}^d$) (i.e., for any

$\theta \in \mathbb{R}^d, V_\theta : X \to \mathbb{R}$ is such that $\nabla_\theta V_\theta(x)$ exists for any $x \in X$). The generalization of

Here z_t

$\in \mathbb{R}^d$.Algorithm4showsthepseudocodeofthisalgorithm.

To see that this algorithm is indeed a generalization of tabular TD(λ) assume that X =

$\{x_1,\ldots,x_D\}$ and let $V_\theta(x)=\theta^T\phi(x)$ with $\phi_i(x)=\mathbb{I}_{\{x=x_i\}}$. Note that since Vθ is linear in the parameters (i.e., $V_\theta=\theta^T\phi$), it holds that $\nabla_\theta V_\theta=\phi$. Hence, identifying $z_{t,i}(\theta_{t,i})$ with $z_t(x_i)$ (resp.,V^t(xi)) we see that the update(21), indeed, reduces to the previous one.

In the off-policy version of TD(λ), the definition of δ_{t+1} becomes

$$\delta_{t+1} = R_{t+1} + \gamma V_{\theta t}(Y_{t+1}) - V_{\theta t}(X_t)$$

Off-policy sampling, unlike tabular sampling, does not guarantee convergence; in fact, the parameters may diverge (see, for example, Bertsekas and Tsitsiklis, 1996, Example 6.7, p. 307). When the distributions of (Xt; t 0) do not match the stationary distribution of the MRP M, this is true for linear function approximation. When the approach is combined with a nonlinear function-approximation method, the algorithm may diverge (see, e.g., Bertsekas and Tsitsiklis, 1996, Example 6.6, p. 292). See Baird (1995); Boyan and Moore for more examples of instability (1995).

On the plus side, when I a linear function-based algorithm is used, practically certain convergence can be guaranteed.

The approximation method is used to the following: X Rd; (ii) the stochastic process (Xt; t 0) is utilised.

Active Learning InBandits

Direct Methods

In this part, we look at techniques that attempt to directly approximate the optimal action-value function Q. The algorithms under consideration are sample-based, approximate variants of value iteration that produce a succession of action-value functions (Qk; k 0). The assumption is that if Qk is close to Q, a greedy policy with respect to Qk will be close to optimal, as demonstrated by the bound (16).

The first algorithm we'll look at is Watkins' Q-learning (1989). We begin by discussing this approach for (small) finite MDPs, then move on to its numerous extensions, which work even in enormous MDPs.

Consider active learning, which is still possible if the MDP is in a single state. Given (say) T interactions, the goal should be to select an action with the biggest immediate payoff. Because the benefits obtained during interaction are irrelevant, the only reason not to try an action is if it can be proven to be worse than another action with enough certainty. The remaining measures should be attempted in the hopes of demonstrating that some are ineffective. Calculating upper and lower confidence boundaries for each action is a straightforward approach to accomplish this:

$$U_t(a) = Q_t(a) + R\sqrt{\frac{\log(2|A|T/\delta)}{2}},$$

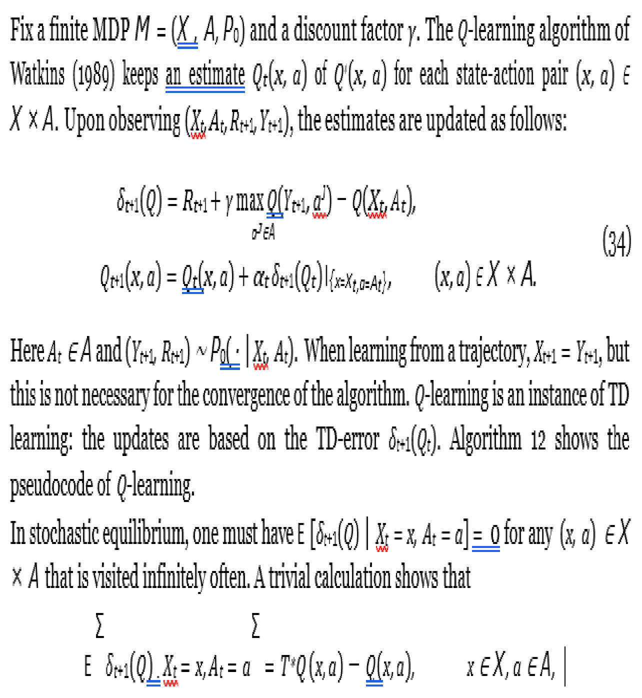
$$L^t(a) = Q^t(a) - R\sqrt{\frac{\log(2|A|T/\delta)}{2}},$$

If Ut(a) maxaJA Lt, then an action is eliminated (aJ). Here, 0 1 is a user-defined parameter that determines the goal confidence level at which the algorithm is allowed to fail to produce the greatest predicted reward action. This algorithm is unimprovable except for constant factors and using estimated variances in the confidence bounds (Even- Dar et al., 2002; Tsitsiklis and Mannor, 2004; Mnih et al., 2008).

Online learning in Markov Decision Processes

Let us now return to MDPs' online learning. One such goal is to reduce regret, which is defined as the difference between the total reward achieved by the best policy and the total reward received by the learner. This issue is discussed in the first portion of this section. Another goal could be to reduce the number of time steps when the algorithm's future projected return is less than the ideal expected return by a certain amount. In the second part of this section, we'll look at this issue.

Q-learning in Finite MDPs



Fix a finite MDP $M = (X, A, P_0)$ and a discount factor γ. The Q-learning algorithm of Watkins (1989) keeps an estimate $Q_t(x, a)$ of $Q^*(x, a)$ for each state-action pair $(x, a) \in X \times A$. Upon observing $(X_t, A_t, R_{t+1}, Y_{t+1})$, the estimates are updated as follows:

$$\begin{aligned} \delta_{t+1}(Q) &= R_{t+1} + \gamma \max_{a' \in A} Q(Y_{t+1}, a') - Q(X_t, A_t), \\ Q_{t+1}(x, a) &= Q_t(x, a) + \alpha_t \delta_{t+1}(Q_t) \mathbb{I}_{\{x = X_t, a = A_t\}}, \qquad (x, a) \in X \times A. \end{aligned} \tag{34}$$

Here $A_t \in A$ and $(Y_{t+1}, R_{t+1}) \sim P_0(\cdot \mid X_t, A_t)$. When learning from a trajectory, $X_{t+1} = Y_{t+1}$, but this is not necessary for the convergence of the algorithm. Q-learning is an instance of TD learning: the updates are based on the TD-error $\delta_{t+1}(Q_t)$. Algorithm 12 shows the pseudocode of Q-learning.

In stochastic equilibrium, one must have $\mathbb{E}[\delta_{t+1}(Q) \mid X_t = x, A_t = a] = 0$ for any $(x, a) \in X \times A$ that is visited infinitely often. A trivial calculation shows that

$$\mathbb{E}\left[\delta_{t+1}(Q) \mid X_t = x, A_t = a\right] = T^*Q(x, a) - Q(x, a), \qquad x \in X, a \in A,$$

where $T^*$is the Bellman optimality operator defined by (15). Hence, under the minimal assumptionthateverystate-actionpairisvisitedinfinitelyoften,instochasticequilibrium, one must have $T^*Q = Q$. Using Fact 3,we see that if the algorithm converges, it must converge to Q^*under the stated condition. The sequence $(Q_t;\ t \geq 0)$ is indeed known to converge to $Q^*$when appropriate local learning rates are used (Tsitsiklis,1994;Jaakkola

etal.,1994).15TherateofconvergenceofQ-learningwasstudiedbySzepesv´ari(1997)in

an a symptotics ettingandlater by Even-Darand Mansour(2003) in a finite-sample setting.

What policy to follow during learning?

One of the most appealing features of Q-learning is its simplicity, which allows for the use of any sampling technique to create training data as long as all state-action pairs are updated infinitely often in the limit. In a closed-loop situation, the Boltzmann scheme (in which the probability of selecting action an at time t is chosen to be proportional to eQt(Xt,a)) or the -greedy action selection scheme (in which the probability of selecting action an at time t is chosen to be proportional to eQt(Xt,a)) are the most commonly used strategies. With the right adjustment, the behaviour policy can reach asymptotic consistency (see Szepesv'ari, 1998, Section 5.2, and Singh et al., 2000). However, as described in Section 4.2, more systematic exploration may be required in closed-loop learning to obtain acceptable online performance.

Actor-critic methods: Actor-critic approaches are used to iterate policies in a broad way. It's important to remember that policy iteration works by alternating between a full policy evaluation and a full policy improvement step. Exact evaluation of policies using sample-based approaches or function approximation may necessitate an endless number of samples or be impossible due to the limitations of the function-approximation technique. As a result, policy iteration reinforcement learning algorithms must update the policy based on partial information of the value function.

Generalized policy iteration refers to algorithms that update the policy before it is fully evaluated (GPI). An actor and a critic are two closely interacting processes in GPI: the actor strives to improve current policy, while the critic reviews current policy, thereby assisting the actor.

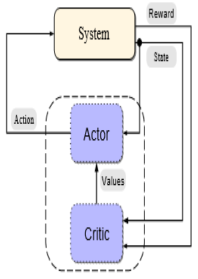


Figure 7: The Actor-Critic Architecture

IMPLEMENTING ACRITIC

The critic's task is to assess the value of the actor's current target policy. This is a problem in which you must anticipate the value of something. As a result, the critic can employ the techniques indicated in Section 3. Because the actor requires action values, the methods are usually changed to directly estimate action values. The algorithm known as SARSA() is obtained by correctly extending TD(). This is the first algorithm we'll talk about. When LSTD() is extended, we get LSTD-Q(), which is the subject of the next section. -LSPE could be extended as well, but for the purpose of brevity, we won't go over that here.

SARSA Similarly to Q-learning, SARSA keeps track of the action-value underlying finite (and small) state and action spaces.

$$\begin{aligned} \delta_{t+1}(Q) &= R_{t+1} + \gamma Q(Y_{t+1}, A'_{t+1}) - Q(X_t, A_t), \\ Q_{t+1}(x,a) &= Q_t(x,a) + \alpha_t \delta_{t+1}(Q_t) \mathbb{I}_{\{x=X_t, a=A_t\}}, \qquad (x,a) \in X \times A. \end{aligned} \tag{35}$$

For further exploration:

Inevitably, due to space constraints, this review must missalargeportion of the reinforcement Learning literature.

Further reading: Effective sampling-based planning (Kearns et al., 1999; Szepesv'ari, 2001; Kocsis and Szepesv'ari, 2006; Chang et al., 2008) is one issue that has received little attention. The main takeaway is that, in the worst-case scenario, off-line planning can scale exponentially with the dimensionality of the state space (Chow and Tsitsiklis, 1989), whereas online planning (i.e., planning for the "current state") can avoid the dimensionality curse by spreading the planning effort over multiple time steps (Rust, 1996; Szepesv'ari, 2001).

Other topics of interest include linear programming-based approaches (de Farias and Van Roy, 2003, 2004, 2006), dual dynamic programming (Wang et al., 2008), sample average approximation techniques (Shapiro, 2003), such as PEGASUS (Ng and Jordan,

Applications Learning in games (e.g., Backgammon (Tesauro, 1994) and Go (Silver et al., 2007), applications in networking (e.g., packet routing (Boyan and Littman, 1994), channel allocation (Singh and Bertsekas,

2000), and online learning in MDPs with arbitrary reward (de Farias and Van Roy, 2003, 2004, 2006). Effective sampling-based planning (Kearns et al., 1999; Szepesv'ari, 2001; Kocsis and Szepesv'ari, 2006; Chang et al., 2008) is one issue that has received little attention. The main takeaway is that, in the worst-case scenario, off-line planning can scale exponentially with the dimensionality of the state space (Chow and Tsitsiklis, 1989), whereas online planning (i.e., planning for the "current state") can avoid the dimensionality curse by spreading the planning effort over multiple time steps (Rust, 1996; Szepesv'ari, 2001).

Other topics of interest include linear programming-based approaches (de Farias and Van Roy, 2003, 2004, 2006), dual dynamic programming (Wang et al., 2008), sample average approximation techniques (Shapiro, 2003), such as PEGASUS (Ng and Jordan, 2000), and online learning in MDPs with arbitrary reward (de Farias and Van Roy, 2003, 2004, 2006).

1997)), applications to operations research problems (e.g., targeted marketing (Abe et al., 2004), job-shop scheduling (Zhang and Dietz, 2004),

REFERENCES

A. Prieditis, S. J. R., editor (1995). *Proceedings of the 12th International Conference on Machine Learning (ICML 1995)*, San Francisco, CA, USA. Morgan Kaufmann.

Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2007). An application of reinforcement learningtoaerobatichelicopterflight.InSchölkopfet al.(2007),pages1–8.

Abe,N.,Verma,N.K.,Apt´e,C.,andSchroko,R.(2004) . Crosschanneloptimizedmarketing by reinforcement learning. In Kim, W., Kohavi, R., Gehrke, J., and DuMouchel, W.,

Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, 10:25–61. Albus, J. S. (1981). *Brains, Behavior, and Robotics*. BYTE Books, Subsidiary of McGraw-

Hill, Peterborough, New Hampshire.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Antos, A., Munos, R.,and Szepesv´ari,C.(2007). Fitted Q-iterationin continuousaction- space MDPs. In Plattetal. (2008), pages9–16.

Antos,A.,Szepesv´ari, C.,andMunos,R.(2008).Learningnear-optimalpolicieswith Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129. Published Online First: 14 Nov, 2007.

Audibert,J.-Y.,Munos,R.,andSzepesv´ari,C.(2009).Exploration-exploitationtrade- off using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19):1876–1902.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Auer,P.,Jaksch,T.,andOrtner,R.(2010).Near-optimalregretboundsforreinforcement learning. *Journal of Machine Learning Research*, 11:1563—1600.

Bagnell,J.A.andSchneider,J.G.(2003).Covariantpolicysearch.InGottlob,G.andWalsh, T., editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*,pages1019–1024,SanFrancisco, CA, USA. MorganKaufmann.

Baird,L.C.(1995).Residualalgorithms:Reinforcementlearningwithfunctionapproxima- tion. In A. Prieditis (1995), pages30–37.

Balakrishna, P., Ganesan, R., Sherry, L., and Levy, B. (2008). Estimating taxi-out times with a reinforcement learning algorithm. In *27th IEEE/AIAA Digital AvionicsSystems Conference*, pages 3.D.3–1 –3.D.3–12.

Bartlett, P. L. and Tewari, A. (2009). REGAL: A regularization based algorithm for rein- forcement learning in weakly communicating MDPs. In *Proceedings of the 25th Annual Conference on Uncertainty in Artificial Intelligence*.

Barto,A.G.,Sutton,R.S.,andAnderson,C.W.(1983). Neuronlikeadaptiveelementsthat can solve difficult learning control problems. *IEEE Transactions on Systems, Man,and Cybernetics*,13:834–846.

Beleznay, F.,Gr˝obler,T.,andSzepesv´ari,C.(1999).Comparingvalue-functionestima- tion algorithms in undiscounted problems. Technical Report TR-99-02, Mindmaker Ltd., Budapest1121,KonkolyTh.M.u.29-33,Hungary.

Berman,P.(1998).On-linesearchingandnavigation.InFiat,A.andWoeginger,G.,editors,*Online Algorithms: The State of the Art*, chapter 10. Springer, Berlin, Heidelberg.

Bertsekas, D. P. (2007a). *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, Belmont, MA, 3edition.

Bertsekas, D. P. (2007b). *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, Belmont, MA, 3edition.

Bertsekas,D.P.,Borkar,V.S.,andNedi˘c,A.(2004).Improvedtemporaldifferencemethods withlinearfunctionapproximation.InSi,J.,Barto,A.G.,Powell,W.B.,andWunschII, D.,editors,*LearningandApproximateDynamic Programming*,chapter9,pages235–257. IEEEPress.

Bertsekas,D.P.andIoffe,S.(1996).Temporaldifferences-basedpolicyiterationandappli- cations in neuro-dynamic programming. LIDS-P-2349, MIT.