

Reinforcement Learning: a brief overview.

Jeremy Wyatt¹

School of Computer Science, University of Birmingham, Edgbaston, Birmingham,
B15 2TT, jlw@cs.bham.ac.uk

1 Introduction

Learning techniques can be usefully grouped by the type of feedback that is available to the learner. A commonly drawn distinction is that between *supervised* and *unsupervised* techniques. In supervised learning a teacher gives the learner the correct answers for each input example. The task of the learner is to infer a function which returns the correct answers for these exemplars while generalising well to new data. In unsupervised learning the learner's task is to capture and summarise regularities present in the input examples. Reinforcement learning (RL) problems fall somewhere between these two by giving not the correct answer, but an indication of how good an answer is. The learner's task in this framework is to learn a function that produces answers that maximise the goodness of its answers.

Most commonly in RL we are concerned with an *agent* acting in an *environment*, where the principal form of feedback is a measure of *immediate* performance, and the goal of the agent is to learn to act so as to maximise some *long term* measure of performance based on this. There are further important differences between this and typical problems in supervised or unsupervised learning. Because the agent is learning how to act, the actions it selects while it is learning affect the examples it sees in the future. Furthermore, the outcomes of actions are not certain so that the agent cannot select its next experience, only influence it. We concentrate on RL problems with both these characteristics in this chapter. The RL framework as described here is suited in some respects to studying problems properly characterised as involving on-going interaction: such as those in robotics, animal learning, optimal foraging, and optimal learning.

There is a comprehensive body of mathematics for modelling agent-environment interactions that are stochastic. It is this that underpins current work in RL and while it has been set out previously elsewhere (see [32,12,21,4]) this chapter summarises the main results and algorithms. The environment-agent interaction is typically modelled as a Markov decision process (MDP), or a partially observable MDP (POMDP) in which the agent observes and controls the process. I shall describe methods for prediction and control in both known and unknown MDPs. The prediction problem is the problem of inferring the long term behaviour of the process in terms of reward, and the control problem is that in which we must determine which actions maximise the agent's performance. Solution methods can be seen as falling into three categories, policy modification techniques, value function based techniques, and model based techniques. I have not made any

assumptions in terms of mathematical knowledge other than a grasp of basic probability theory.

2 Markov Processes

Markov Processes are a form of stochastic process, and a stochastic process is simply a sequence of random events. Stochastic processes can be used to model many phenomena: the motion of particles in a liquid or gas; the fluctuations of the stock market; the motion of a robot; or the sequence of moves in a game of chance like backgammon or cards. We are concerned here with random processes that evolve in discrete time, and which have a countable number of outcomes. As an example let us imagine a frog in a pond full of lily pads. The lily pads are the outcomes (or states) of our process. The frog hops from pad to pad at regular intervals, and which pad it jumps to next is uncertain. We can describe this mathematically.

In a discrete stochastic process like this we take the random variable X_t to denote the outcome at the t^{th} stage or time step. The stochastic process is defined by the set of random variables $\{X_t, t \in T\}$, where $T = \{0, 1, 2, \dots\}$ is the set of possible times. The domain of X_t is the set of possible outcomes denoted $\mathcal{S} = \{s_1, s_2, \dots, s_{|\mathcal{S}|}\}$. In the general case the outcome at time t is dependent on the prior sequence of outcomes x_0, x_1, \dots, x_{t-1} . The likelihood of the outcome at time t being s_j is therefore written:

$$\Pr(X_t = s_j | x_{t-1} \wedge x_{t-2} \wedge \dots \wedge x_0) \quad (1)$$

A process can be said to be an *independent process* if the outcome at each time t is independent of the outcomes at all prior stages:

$$\Pr(X_t = s_j | x_{t-1} \wedge x_{t-2} \wedge \dots \wedge x_0) = \Pr(X_t = s_j) \quad (2)$$

A Markov process weakens this independence assumption minimally by requiring that the outcome at time t is independent of all events prior to $t - 1$:

$$\Pr(X_t = s_j | (X_{t-1} = s_i) \wedge x_{t-2} \wedge \dots \wedge x_0) = \Pr(X_t = s_j | X_{t-1} = s_i) \quad (3)$$

Equation 3 is known as the *Markov property*. The probability $\Pr(X_t = s_j | X_{t-1} = s_i)$ can be regarded as a transition probability from the outcome s_i at $t - 1$ to the outcome s_j at time t , denoted by $s_i \rightsquigarrow s_j$. If the transition probabilities are independent of time then the process is a stationary Markov chain. The outcomes are referred to as the *states* of the process. We use the following shorthand to denote the probability of the transition from state $s_i \rightsquigarrow s_j$:

$$p_{ij} = \Pr(X_t = s_j | X_{t-1} = s_i) \quad (4)$$

Given the current state of a Markov chain and its transition probabilities we can predict its behaviour any number of steps into the future. The transition probabilities are represented in the form of a transition matrix, \mathbf{P} , the i, j^{th}

element of which is p_{ij} . We also define a probability distribution across the starting states (i.e. when $t = 0$), denoted by the row vector $\mathbf{x}_0 = [\Pr(X_0 = s_1), \Pr(X_0 = s_2), \dots, \Pr(X_0 = s_n)]$, where n is the number of states. I denote the probability distribution across \mathcal{S} at any time t by \mathbf{x}_t . Given \mathbf{x}_0 and \mathbf{P} , \mathbf{x}_t can be expressed elegantly as the product:

$$\mathbf{x}_t = \mathbf{x}_0 \cdot \mathbf{P}^t \quad (5)$$

The significance of this is that the study of the state of the process n steps into the future is the study of the n^{th} power of the transition matrix. It is worth noting for practical purposes that the notion of the future behaviour of the process being dependent solely on the current state of the process is a representational device. Processes whose future behaviour relies on knowing some or all of the process history can be made to satisfy the Markov property by including sufficient record of that history in the description of the current state. This may be expressed in the following manner. If the description of the state at time t is denoted by the column vector κ_t then we can denote the supplemented description of the current state by the concatenation of two vectors

$$\kappa'_t = [\kappa_t^T, f(\kappa_{t-1} \dots \kappa_{t-k})^T]^T \quad (6)$$

where T means transposition and $f(\cdot)$ is a function summarising the process history in the form of a new vector from states as far back in time as necessary, here k steps. In many cases the additional information may not add excessively to the length of the state description. If for example, we wish to predict the trajectory of a ball thrown through the air, then we use first and second order derivatives of position to summarise the history of the process necessary for the prediction of the future. If we use this information to control a process then we say that the controller has state. One of the primary problems with optimization methods relying on the Markov assumption is that we do not always know how much information it is necessary to supplement the description of the current state with. This is referred to as the question of how much state to include in the controller. State that is not directly observable by an agent is referred to as *hidden state*.

It can, however, be seen that this ability in principle to represent *any* stochastic process as a Markov process is a potentially powerful one. The inferential power gained is achieved by the way the Markov property separates the past and the future. The *necessary* history of the process is encapsulated in the description of the current state and this state completely determines future behaviour. We will now outline Markov decision processes.

2.1 Markov Decision Processes

We have said previously that we are interested in problems where the agent can select an action at each step to influence the evolution of the process. To incorporate this the finite state, discrete-time Markov chain model needs to be extended by making the transition matrix at time t depend on an *action* a_t

chosen at that time. The set of possible actions may vary from state to state so we write,

- \mathcal{A} for the set of possible actions across all states
- $\mathcal{A}_x \subseteq \mathcal{A}$ for the set of actions allowable in state x .

The transition probabilities that depend on the action chosen are denoted $p_{ij}(a)$, where $a \in \mathcal{A}$. There are now m transition matrices (where the size of the set \mathcal{A} is m), one for each action: $\mathbf{P}_a = [p_{ij}(a)]$. If an action a is not possible in a particular state s_i , then $p_{ij}(a) = 0$. We may regard the transition function \mathbf{P} as a function specified by these m transition matrices, mapping from all possible pairs of states and actions into a probability distribution across the set of states. We denote the transition from s_i to s_j following the selection of action a in state s_i by $s_i \xrightarrow{a} s_j$. Finally we define a *reinforcement function* R which in the most general case is defined as a mapping from the state, action and next state into a probability density over the set of possible rewards $\mathcal{R} \subseteq \mathfrak{R}$,

$$R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \rightarrow [0, 1]$$

At each observation, a *reward* is generated dependent upon the state, the action performed, and the next state. The random variable denoting the reward at time t is $R_t = R(x_t, a_t, x_{t+1})$. The actual reward generated during the transition $x_t \xrightarrow{a_t} x_{t+1}$ is r_t . For some problems it is important to distinguish whether or not the reward function is known to the agent.

Reward functions The simplest possible reward function is when the set of possible rewards is Boolean, $\mathcal{R} = \{0, 1\}$. In this case the reward model is termed a *P-model*. Any problem with well-defined criteria for success and failure can be represented as a *P-model*. If for example the aim of a process is to track a set-point ω , and a certain magnitude of error ε is acceptable then taking 1 to be success¹ the reward at time t is

$$r_t = \begin{cases} 1 & \text{if } |\hat{\omega}_t - \omega| \leq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{\omega}_t$ is the system's approximation to the set point at time t . The minimal extension of this model is to allow any finite number of reward values in the interval $[0, 1]$. Such a model is termed a *Q-model*. Problems with real-valued rewards can be expressed in this form by means of normalisation and quantisation. The most general case is when the reward can take any real value in the interval $[0, 1]$. Such a reward model is termed an *S-model*. By normalisation any problem with bounded reward can be expressed as an *S-model*.

The reward function merely specifies the reward generated at a particular instant. Using the notion of immediate reward we can construct measures of performance over many time steps. Following Barto et al. [2], I refer to a measure

¹ Usually in Learning Automata Theory 0 is taken to be success and 1 to be failure. Since the convention in RL is to maximise reward I reverse this for convenience.

of long-term reward as a measure of *return*. \mathbf{R}_t is the random variable denoting return at time t . There are several measures of return investigated in the literature. All can be expressed in terms of a discount vector $G = \{\gamma_0, \gamma_1, \gamma_2, \dots\}$, where the return at time t is,

$$\mathbf{R}_t = \sum_{k=0}^{\infty} \gamma_k r_{t+k} \quad (7)$$

The vector G may in principle be arbitrary, but in practice one of three *discount* schemes is used. The first is the *finite horizon* model of return, where the horizon is a finite number h of steps into the future,

$$\gamma_k = \begin{cases} 1 & \text{if } k \leq h \\ 0 & \text{if } k > h \end{cases}$$

This model has been studied extensively in the bandit literature². Alternatively we may use the *average-reward* model [24,26].

$$\gamma_k = \begin{cases} 1/h, & \text{if } k \leq h \\ 0 & \text{if } k > h \end{cases}$$

By far the most widely studied measure of return however, particularly within work on learning from delayed reinforcement, is an infinite horizon model termed the *geometric discount* model of return.

$$\gamma_k = \gamma^k, \text{ where } 0 \leq \gamma < 1$$

The value of γ chosen determines the relative weighting of short and long term rewards. As $\gamma \rightarrow 0$ short-term rewards become more important. When $\gamma = 0$ the only reward that matters is the immediate reward. As well as being attractive for its elegance this method has been shown to make certain problems in learning from reinforcement more tractable, e.g. bandit tasks. Here on I consider only the geometric discount model of return.

The importance of our definition of return is that we now have a model that enables us to take an immediate measure of performance and turn it into a long term measure of performance. It is not a trivial task to design a reward function that will give an appropriate return function for real world tasks. The assumption that this conversion of a short term measure into a long term measure is a useful thing to do lies at the heart of almost all modern work on reinforcement learning. Next we will consider ways an agent can act, and how we can say that one way of acting is better than another according to our measure of long term performance.

² A k -armed bandit is an MDP with a single state and k actions available. The actions generate stochastic rewards. A bandit problem is one in which the stochastic effects of the actions are unknown, and the learner must maximise its performance while it is learning. This is the simplest problem in the optimal learning literature, also known as the exploration-exploitation trade-off.

3 Policies and Optimal Policies

A policy π is a mapping that specifies the actions the agent takes in each state of the environment, and is thus the sole essential component of an agent. π says what to do in every possible state. Thus it can also be seen as a universal plan, i.e. a plan with no explicitly specified sequence of actions. A *stationary* policy specifies an action to be taken for each state, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, this means that the action taken in a state is always the same. There is an optimal stationary policy for any completely observable MDP. A *stochastic* policy is a mapping $\pi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. π is specified by a stochastic matrix, where the i, j^{th} element of π is,

$$\pi_{ij} = \Pr(a_j | s_i), \text{ and } \sum_j \pi_{ij} = 1, \forall i$$

Under a stochastic policy the action an agent selects in a given state may vary (hence it is not stationary), but is always selected according to the same distribution. Some Partially Observable MDPs (POMDPs) which have no optimal stationary policy have an optimal stochastic policy. A *non-stationary* policy is one in which the policy is indexed by time. In finite horizon models of reward optimal policies are typically non-stationary. All the agents we are interested in modify π directly or indirectly as a function of their experience.

For our purposes it is obviously necessary to be able to order policies according to some index of performance. The reward models I have discussed can be used to derive just such an ordering on the set of possible policies. A policy π_1 is said to be at least as good as a policy π_2 if it has an expected return which is greater than or equal to that of policy π_2 in each of the possible initial states of the process:

$$E[\mathbf{R}(s_i) | \pi_1] \geq E[\mathbf{R}(s_i) | \pi_2], \forall s_i \in \mathcal{S}_0 \Leftrightarrow \pi_1 \succeq \pi_2 \quad (8)$$

where $a \succeq b$ is a preference operator, indicating that a is at least as good as b ; and $\mathcal{S}_0 \in \mathcal{S}$ is the set of possible initial states of the process. An *optimal* policy is defined as any policy π^* which is at least as good as any other policy,

$$E[\mathbf{R}(s_i) | \pi^*] \geq E[\mathbf{R}(s_i) | \pi_j], \forall s_i \in \mathcal{S}_0, \pi_j \in \Pi \Leftrightarrow \pi^* \succeq \pi_j, \forall \pi_j \in \Pi \quad (9)$$

where Π is the set of possible policies. It is an important result that for both the finite horizon and geometrically discounted models of return there is at least one policy that is optimal for any MDP. There may be more than one optimal policy, and so I will denote the set of optimal policies Π^* . The aim of any policy-modifying agent is to converge to such a policy. Our aim is to design learning agents which converge to an optimal policy quickly and reliably. The first step in designing such agents is to be able to estimate the expected return for a given policy in order that we may compare policies by Equations 8 and 9.

4 Prediction

The prediction problem is concerned with estimating the mean goodness of each state of environment given that we follow a certain policy. Given a policy π , a transition function \mathbf{P} , and a reward function R for a Markov decision process we can calculate the expected return. Before discussing this we need some additional notation. The random variable $R^\pi(s_i, n)$ denotes the reward received on the n -th step after starting in s_i and following policy π for n steps. This random variable captures the stochastic effects of the MDP up to n steps into the future. The *value* $V^\pi(s_i)$ of policy π in state s_i is the expected return under that policy. Hence $V^\pi(s_i)$ can be written,

$$V^\pi(s_i) = E[R^\pi(s_i, 1) + \gamma R^\pi(s_i, 2) + \gamma^2 R^\pi(s_i, 3) + \dots + \gamma^n R^\pi(s_i, n) + \dots]$$

Where $E[X]$ is the expectation of the random variable X . This can be expressed recursively for all $s_i \in \mathcal{S}$,

$$V^\pi(s_i) = E[R^\pi(s_i, 1)] + \gamma \sum_j p_{ij}(\pi(s_i)) V^\pi(s_j), \forall s_i \in \mathcal{S} \quad (10)$$

The $V^\pi(s_i)$ for all s_i define the *value function* under the policy π . If we know $E[R^\pi(s_i, 1)]$ as well as \mathbf{P} then the value function can be calculated off-line by solving this set of linear equations. Probably the most widely used technique is some form of dynamic programming. We will return to describe this in detail in Section 5.

Methods employing a known transition function \mathbf{P} to derive the value function are commonly termed *model based* methods for predicting the value of a policy. To be more accurate we refer to models that explicitly represent the probability distributions over outcomes as *distribution models*. If the agent does not possess a distribution model in advance it may estimate such a model from its own experience as it proceeds through the environment. We can then use the estimated model to estimate the value function. Such estimates are often referred to as *certainty equivalent* estimates, as they assume that the model is essentially correct. A simple approach is to pick the maximum likelihood estimates of the model parameters. The certainty equivalent value function constructed from these will be the maximum likelihood value function. Methods that estimate the value function using either learned or a priori models are also referred to as *indirect methods*.

If the agent has neither $E[R^\pi(s_i, 1)]$ nor \mathbf{P} and we do not want to learn a model then we can use a *direct* or *model-free* method for predicting the value of the policy on-line. Model-free methods build an estimate of the value function directly from their experience, i.e. from the sequence of perceived states and rewards generated. There are two classes of direct methods that have been carefully studied: simple Monte-Carlo methods, and temporal difference methods.

In basic Monte-Carlo approaches we sample a sequence of observations and rewards from the world, and calculate the actual return from each state. Many such samples are taken, and we then calculate the average return from each

state over those samples. Such methods are often simply referred to as Monte-Carlo methods. There are two studied Monte-Carlo estimates of expected return: the every visit Monte-Carlo (EVMC) estimate and the first visit Monte-Carlo (FVMC) estimate [25]. The FVMC estimate has been shown to have a connection to the maximum likelihood certainty equivalent estimate of value for an MDP under a given policy. Simple Monte-Carlo methods are important in that they give a performance baseline from which to work, and aspects of them have been important in developing more sophisticated algorithms. In particular temporal difference algorithms can be seen as a combination of ideas from dynamic programming and Monte-Carlo methods. The difficulty with simple Monte-Carlo estimators is that their standard error declines very slowly as the sample size rises.

A well-known and elegant model-free method for estimating expected return for an MDP under a policy is Sutton’s *temporal difference* method [29]. This works in roughly the following manner. Given that the transition $x_t \rightsquigarrow x_{t+1}$ occurs, the reward r_t is received during the course of this transition. At time t we have estimates of the value of each state, $\hat{V}_t(x_t)$ and $\hat{V}_t(x_{t+1})$, where we have dropped the explicit reference to the policy π in our notation. It turns out that a better estimate of $V(x_t)$ than $\hat{V}_t(x_t)$ can be provided by

$$r_t + \gamma \hat{V}_t(x_{t+1})$$

The temporal difference is the difference between these two estimates,

$$r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$$

The basic temporal difference equation uses this to update the estimate of $V(x_t)$ each time the transition is made:

$$\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \alpha_t [r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)] \quad (11)$$

$0 < \alpha_t \leq 1$, is the learning rate at time t . As $\alpha_t \rightarrow 1$ so $\hat{V}_{t+1}(x_t)$ depends more on $r_t + \gamma \hat{V}_t(x_{t+1})$ and less on $\hat{V}_t(x_t)$. α_t acts as a filter damping the variance in $r_t + \gamma \hat{V}_t(x_{t+1})$. As $t \rightarrow \infty$ the estimates $\hat{V}_t(x_t)$ are guaranteed to converge to $V_t(x_t)$ if it is the case that $\sum_{t=0}^{\infty} \alpha_t = \infty$; that $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$; and that $\alpha_t > 0, \forall t \in T$. Equation 11 forms the basis of the TD(0) algorithm which is specified in Figure 2.

Updating $\hat{V}(x)$ only on making a transition from x is comparatively inefficient. Because $V(x)$ depends to some extent on $V(y)$ for all y which can be reached eventually from x , $\hat{V}(x)$ may not only be updated when it occurs, but also on the basis of the temporal difference for *any* subsequent transition. Using this insight Sutton generalised TD(0) by defining a class of prediction algorithms called TD(λ) (see Figure 2). In TD(λ) the extent to which a change in $\hat{V}_t(x_{t+1})$ is mirrored in other states is determined by the value of a function \bar{e} called an eligibility trace, defined on the domain \mathcal{S} . It is included in the temporal difference update equation as follows,

$$\hat{V}_{t+1}(s) = \hat{V}_t(s) + \alpha_t [r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)] \bar{e}_t(s), \forall s \in \mathcal{S} \quad (12)$$

The value of $\bar{e}_t(s)$ is updated each transition for all $s \in \mathcal{S}$. There are two forms of the update equation. An *accumulating* trace updates the eligibility of a state using,

$$\bar{e}_t(s) = \begin{cases} \gamma\lambda\bar{e}_{t-1}(s) + 1 & \text{if } s = x_t \\ \gamma\lambda\bar{e}_{t-1}(s) & \text{otherwise} \end{cases} \quad (13)$$

A *replacing* trace update is defined by,

$$\bar{e}_t(s) = \begin{cases} 1 & \text{if } s = x_t \\ \gamma\lambda\bar{e}_{t-1}(s) & \text{otherwise} \end{cases} \quad (14)$$

Under both mechanisms the eligibility of a state decays away exponentially when the state is unvisited. Under an accumulating trace the eligibility is increased by a constant every time the state is visited, and under a replacing trace the eligibility is reset to a constant on each visit. The effect of each update rule on the eligibility of a state according to the frequency of visits is illustrated qualitatively in Figure 1. An eligibility trace can be thought of as a short-term

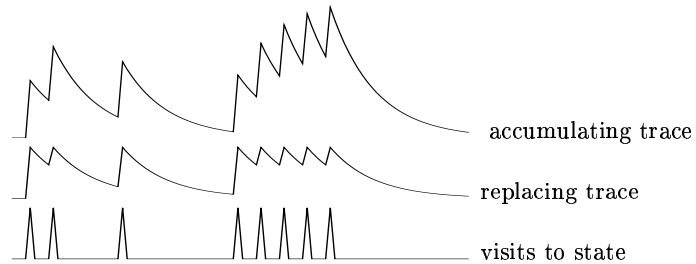


Fig. 1. Behaviour of accumulating and replacing traces.

memory process initiated the first time a state is visited by an agent. The degree of activation depends on the recency of the most recent visit and on the frequency of visits. Thus eligibility traces implement two heuristics, a *recency heuristic* and a *frequency heuristic*. These can be stated informally as saying that reinforcement received now is probably caused to a greater degree by more recently and frequently occurring states than by less recently and frequently occurring states. Accumulating traces implement both these heuristics, while replacing traces implement just the recency heuristic [28]. The rate of decay of the trace is determined by $0 \leq \lambda \leq 1$. Hence the class of algorithms defined is referred to as TD(λ). If $\lambda = 0$ then Equation 12 simplifies to 11. If $\lambda = 1$ then the estimate of $V(x_t)$ ignores the estimated values $\hat{V}(x_{t+k})$ of any subsequent states and is based entirely on the actual rewards received at each step. If a perceptron is being used for structural credit assignment then using TD(1) makes its updates equivalent to those of the Widrow-Hoff rule. In general high values of λ give fast initial convergence to $V(x)$, and low λ values give low standard error. Better performance than by using fixed λ is therefore obtained by declining

Algorithm 1 TD(λ) $0 < \alpha$

```

 $t := 0$ 
 $\hat{V}_t(x) := 0, \bar{e}_0(x) := 0, \forall x \in \mathcal{S}$ 
repeat
  observe the transition  $x_t \rightsquigarrow x_{t+1}$ 
  update  $\bar{e}(x)$  for all  $x \in \mathcal{S}$  according to Eq. 13 or 14
  update  $\hat{V}(x)$  for all  $x \in \mathcal{S}$  according to
    
$$\hat{V}_{t+1}(x) := \hat{V}_t(x) + \alpha[r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)]\bar{e}_t(x)$$

   $t := t + 1$ 

```

Fig. 2. The TD(λ) algorithm.

the value of λ from 1 to 0. Procedures for declining λ systematically have been investigated [31]. These work well for acyclic processes, but do not appear likely to extend to the more general cyclic case.

It has been shown that replacing and accumulating traces converge to different estimates of expected return. In particular accumulating traces are related to the Every Visits Monte Carlo estimate and replacing traces are related to the First Visits Monte Carlo estimate [25]. Some empirical results suggest that replacing traces used with TD(λ) outperform accumulating traces [25]. Although the TD(λ) algorithm applies strictly to estimating the value function for a Markov chain, any Markov decision process under a given policy can be modelled as a Markov chain.

In summary I have reviewed methods for the estimation of the value function for any given policy based on dynamic programming; temporal difference learning and eligibility traces. It can be seen that in principle such techniques could be used to identify the optimal policy by estimating the value function for all possible policies. Such an approach would be grossly inefficient however, and there are much faster methods which I now discuss.

5 Control

If the prediction problem is concerned with estimating expected future performance the control problem is concerned with finding a policy that optimises our predicted performance. As for the problem of predicting the value function there are both model-based and model-free methods for computing the optimal policy for a Markov decision process. If the agent has a model then we can employ one of a number of dynamic programming techniques to achieve this. I shall initially discuss two of these, *policy iteration* and *value iteration*. These algorithms

are suitable for derivation of optimal controllers off-line given accurate process models. They can however, be extended to work on-line employing an adaptive process model.

Policy iteration commences with an arbitrary policy f and calculates the value function V^f by solving the set of linear equations given by Equation 10. It then improves this policy and recalculates the value function, repeating this process until the optimal policy is reached. The key lies in the way f is improved each iteration. Consider a particular state s_i . Suppose the agent takes any action a in state s_i and follows the policy f elsewhere. The value of this modification to policy f is termed the *action-value* of the action a in state s_i with respect to policy f [32], denoted $Q^f(s_i, a)$,

$$Q^f(s_i, a) = E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a) V^f(s_j) \quad (15)$$

We choose g as the policy which selects action a in state s_i and follows policy f in all other states, where:

$$a = \arg \max_a \{Q^f(s_i, a)\}$$

Because the set \mathcal{A}_{s_i} includes the best action chosen by policy f we know that g must be at least as good as f . We can carry out this improvement process in all states simultaneously, so that

$$V^g(x) \geq V^f(x), \forall x \in \mathcal{S}$$

Because a change in the policy at any single state can impact on the value function across the entire state space the entire value-function must then be recalculated for the new policy. This constitutes a single iteration of the policy iteration algorithm (see Figure 3). The algorithm iterates in this manner until no further improvements can be made to the policy in any state. Any such policy must be an optimal policy. Policy iteration can be fast if the action space is quite small, as in such a case it will usually converge in few iterations. Even so recalculating the entire value function from scratch each iteration is expensive. Value iteration avoids this problem by solving the optimality equation for a series of geometrically discounted finite horizon problems defined on the same Markov decision process. Thus it only calculates the value function for the infinite horizon problem once. The value function for the optimal policy π^* is written V^* . Its solution for a problem with horizon $h = 0$ is given by,

$$V_0^*(s_i) = \max_a \{E[R^a(s_i, 0)]\}, \forall s_i \in \mathcal{S} \quad (16)$$

Using Bellman's optimality equation the solution for a finite horizon problem with $h = 1$ is given in terms of V_0^* . In general V_n^* is written in terms of V_{n-1}^* .

$$V_{n+1}^*(s_i) = \max_a \{E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a) V_n^*(s_j)\}, \forall s_i \in \mathcal{S} \quad (17)$$

Algorithm 2 (POLICY ITERATION)

```

f := arbitrary policy
repeat
  solve    $V^f(s_i) = E[R^f(s_i, 0)] + \gamma \sum_j p_{ij}(f(s_i))V^f(s_j)$             $\forall s_i \in \mathcal{S}$ 
  choose   $g(s_i) := \arg \max_a [Q^f(s_i, a)]$                                       $\forall s_i \in \mathcal{S}$ 
  where    $Q^f(s_i, a) = E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a)V^f(s_j)$ 
  let      $f(s_i) := g(s_i)$                                                       $\forall s_i \in \mathcal{S}$ 
until there is no change in  $f(s_i), \forall s_i \in \mathcal{S}$ 

```

Fig. 3. The policy iteration algorithm.

As $n \rightarrow \infty$ so $V_n^* \rightarrow V^*$. The difficulty with value iteration is defining a stopping criterion for n . Recently this problem has been partly solved. A upper bound on the error of the final estimate can be expressed given the maximum error between the last two approximations to the value function [35]. Worst case performance can thus be improved to a measurable point arbitrarily close to the optimum. The value iteration algorithm is given in Figure 4. There is some debate as to whether policy or value iteration converges faster for large problems.

There are a number of variations to value iteration as presented above, which is strictly termed *synchronous value iteration*. Synchronous refers simply to the fact that each iteration all the $V_n^*(x)$ are updated using the estimates $V_{n-1}^*(x)$. Gauss-Seidel iteration changes this by updating the states in a particular order. States updated later in the sequence use the already updated values for other states. Asynchronous value iteration relaxes the rules concerning the order in which the states are updated still further by allowing that an arbitrary subset of states is backed up synchronously each iteration. The set of states may vary in each iteration. In the completely asynchronous case this means that one state is updated each iteration. Both synchronous value-iteration and Gauss-Seidel iteration are guaranteed to converge to the optimal policy; asynchronous value-iteration is only guaranteed to converge if each state has a non-zero probability of being updated each iteration.

The theory of asynchronous dynamic programming was originally developed in order to facilitate multi-processor implementation. More recently this theory has been used to show that dynamic programming algorithms can work on-line, interleaving estimation of the model, policy modification, and control [1]. Asynchronous DP algorithms thus form the basis of model-based reinforcement learning systems. A class of value-iteration algorithms, termed adaptive real-time dynamic programming (ARTDP) algorithms, can be defined (see Figure 5). The set \mathcal{S}_t is the set of states whose values are to be updated at time t . It should

Algorithm 3 (VALUE ITERATION) Choose ϵ so that $\frac{2\epsilon\gamma}{(1-\gamma)}$ is sufficiently small, where $\max_{\mathcal{S}}(|\hat{V}^*(s_i) - V^*(s_i)|) \leq \frac{2\epsilon\gamma}{(1-\gamma)}$.

$V_0 :=$ arbitrary bounded function
 $n := 1$
repeat
 let $V_n^*(s_i) := \max_a \{E[R^a(s_i, 0)] + \gamma \sum_j p_{ij}(a) V_{n-1}^*(s_j)\} \quad \forall s_i \in \mathcal{S}$
until $\max_{\mathcal{S}}(|V_n^*(s_i) - V_{n-1}^*(s_i)|) < \epsilon$
let $\hat{V}^*(s_i) := V_n^*(s_i) \quad \forall s_i \in \mathcal{S}$

Fig. 4. The value iteration algorithm.

be chosen so that the algorithm satisfies the conditions for the convergence of asynchronous value-iteration, and also that there is sufficient computation available to carry out all the updates. In addition because the agent is estimating the model on-line it must choose a suitable sequence of control actions so that it experiences each transition a sufficient number of times for the estimates \hat{p}_{ij} to be good.

In a similar manner to Gauss-Seidel iteration, the choice of \mathcal{S}_t at each stage can speed convergence to the optimal value function. There are a number of possible schemes. One of the simplest is to choose k additional members of \mathcal{S}_t at random³. A good choice of \mathcal{S}_t can speed convergence considerably. Moore and Atkeson have published a technique which can be seen as a variant of ARTDP. They call it Prioritised Sweeping. Its central feature is that states leading to states with large changes in the value function are processed first. It is considerably more efficient than the other model based methods discussed previously. Wiering and Schmidhuber have more recently published a version of the algorithm which is capable of detecting large changes in the value function which occur as a result of a succession of small cumulative changes in the model [34].

There are a number of model-free policy-modification algorithms; the earliest of these [28,11] used either the TD(λ) algorithm, or temporal difference methods similar to it, to convert a delayed reward signal into a heuristic reward signal (the $\hat{V}^\pi(x_t)$ under the current policy π). The heuristic reward signal is then fed to any algorithm which modifies its policy on the basis of immediate reward, in place of the immediate reward signal r_t . Thus the class of algorithms compatible with this method is large. The policy of the agent is modified using the estimate $\hat{V}^{\pi_t}(x_t)$. In the next step the TD(λ) algorithm estimates V with respect to a different policy π_{t+1} . Thus the value function is changing as the policy changes. Examples of

³ See [13] for a description of such an algorithm. They refer to it as a Dyna method, although it is rather different from Sutton's original method of the same name [30].

Algorithm 4 (ADAPTIVE REAL-TIME VALUE ITERATION)

$\hat{Q}_t(x, a) = \bar{r}(x, a) + \gamma \sum_{y \in \mathcal{S}} \hat{p}_{xy}(a) \hat{V}_t^*(y)$. `explore` is a function mapping from estimated Q-values to a probability distribution across actions. $\bar{r}(i, a)$ is an estimate of $E[R^a(i, 0)]$.

```

t := 0
 $\hat{V}_t^*$  := arbitrary bounded function
observe  $x_t$ 
repeat
  choose  $a_t$  from explorea( $\hat{Q}_t(x_t, a)$ )
  observe the transition  $x_t \xrightarrow{a_t} x_{t+1}$ 
  update  $\hat{p}_{x_t x_{t+1}}(a_t)$  and  $\bar{r}(x_t, a_t)$ 
  for some  $\mathcal{S}_t \subseteq \mathcal{S}$  such that  $x_t \in \mathcal{S}_t$  update
     $\hat{V}_{t+1}^*(i) := \max_a \{\bar{r}(i, a) + \gamma \sum_{j \in \mathcal{S}} \hat{p}_{ij}(a) \hat{V}_t^*(j)\}$             $\forall i \in \mathcal{S}_t$ 
     $\hat{V}_{t+1}^*(i) := \hat{V}_t^*(i)$                                             $\forall i \notin \mathcal{S}_t$ 
t := t + 1

```

Fig. 5. The adaptive real-time value iteration algorithm.

policy-modification algorithms based on the TD(λ) algorithm include [12,15,28]. To my knowledge there are currently no proofs for the convergence of any such policy-modification algorithms.

A model-free method which is guaranteed to converge is Q-learning [32]. This is a model-free approximation to adaptive real-time value iteration. The primary structural difference between Q-learning and methods based on the TD(λ) algorithm is that whereas the latter maintain estimates of the values of states under the current policy, Q-learning maintains estimates of action-values. It adjusts these estimates each step using a temporal difference mechanism,

$$\hat{Q}_{t+1}(x_t, a_t) = (1 - \alpha_t) \hat{Q}_t(x_t, a_t) + \alpha_t [r_t + \gamma \max_a \{\hat{Q}_t(x_{t+1}, a)\}] \quad (18)$$

where $0 < \alpha_t < 1$ is the learning rate at time t . The update equation is fundamentally of the same form as the value iteration update, replacing the estimates \hat{p}_{ij} with α_t . Q-learning is guaranteed to converge asymptotically given that each state-action pair is tried infinitely often, and similar criteria to TD-learning for the reduction of the learning rate [33]. The other nice property of Q-learning is that the estimates of the Q-values are independent of the policy followed by the agent, the consequence of this being that the agent may deviate

Algorithm 5 $Q(\lambda)$ -LEARNING

$\hat{V}^*(x) = \max_a \hat{Q}(x, a)$. $0 \leq \lambda \leq 1$. ε'_t and ε_t are error signals. `explore` is a function mapping from estimated Q -values to a probability distribution across actions.

```

t := 0
 $\hat{Q}(x, a) := 0$  and  $\bar{e}_t(x, a) := 0, \forall x, a$ 
observe  $x_t$ 
repeat
  choose  $a_t$  from explorea( $\hat{Q}_t(x_t, a)$ )
  observe the transition  $x_t \xrightarrow{a_t} x_{t+1}$ 
   $\varepsilon'_t := r_t + \gamma \hat{V}_t^*(x_{t+1}) - \hat{Q}_t(x_t, a_t)$ 
   $\varepsilon_t := r_t + \gamma \hat{V}_t^*(x_{t+1}) - \hat{V}_t^*(x_t)$ 
  update  $\bar{e}(x, a)$  for all  $x \in \mathcal{S}, a \in \mathcal{A}$  according to Eq. 19 or 20
  update  $\hat{Q}_{t+1}(x, a)$  for all  $x \in \mathcal{S}, a \in \mathcal{A}$  using
     $\hat{Q}_{t+1}(x_t, a_t) := \hat{Q}_t(x_t, a_t) + \alpha_t \varepsilon'_t \bar{e}_t(x_t, a_t)$ 
     $\hat{Q}_{t+1}(x, a) := \hat{Q}_t(x, a) + \alpha_t \varepsilon_t \bar{e}_t(x, a)$  for all  $\hat{Q}(x, a)$  except  $\hat{Q}(x_t, a_t)$ 
t:=t+1

```

Fig. 6. The $Q(\lambda)$ algorithm.

from the optimal policy at any stage while still constructing unbiased estimates of the Q -values.

Q -learning may also be extended to take advantage of eligibility traces. $Q(\lambda)$ [22] contains one-step Q -learning as a special case ($\lambda = 0$) and so I give the full algorithm for this generalised version (Figure 6). Convergence has only been proved for $Q(\lambda)$ with $\lambda = 0$. The eligibility traces used in $Q(\lambda)$ are necessarily defined over the domain formed by the Cartesian product $\mathcal{S} \times \mathcal{A}$. The update equations are, however, fundamentally the same:

$$\bar{e}_t(x, a) = \begin{cases} \gamma \lambda \bar{e}_{t-1}(x, a) + 1 & \text{if } x = x_t \text{ and } a = a_t \\ \gamma \lambda \bar{e}_{t-1}(x, a) & \text{otherwise} \end{cases} \quad (19)$$

$$\bar{e}_t(x, a) = \begin{cases} 1 & \text{if } x = x_t \text{ and } a = a_t \\ \gamma \lambda \bar{e}_{t-1}(x, a) & \text{otherwise} \end{cases} \quad (20)$$

The version of $Q(\lambda)$ learning given in Figure 6 was devised by Peng and Williams and uses the eligibility traces to propagate the temporal difference

error for the value function under the optimal policy across the state and action space. This update is incorrect if the agent follows non-greedy actions. This means that the algorithm is not guaranteed to converge. A simple solution is to zero all the eligibilities whenever a non-greedy action is selected. This version of $Q(\lambda)$ learning was originally suggested by Watkins.

However this method effectively removes the principle benefit of the idea of combining Q-learning with eligibility traces: that you can learn quickly about the effects of one policy while following another. A different approach to the problem is to use modified Q-learning [23], also known as SARSA. This removes the assumption that the agent follows a greedy policy after executing the current action. It does this by removing the max operator from the temporal difference update rule.

$$\hat{Q}_{t+1}(x_t, a_t) = (1 - \alpha_t)\hat{Q}_t(x_t, a_t) + \alpha_t[r_t + \gamma\hat{Q}_t^\pi(x_{t+1}, a_{t+1})] \quad (21)$$

Under this rule the agent is now estimating the value of the action given that it follows some policy π after. This policy can obviously be stochastic and non-greedy. SARSA can thus be combined with full eligibility traces in a clean way.

In this section we have concentrated on describing in detail methods that search for a value function in Markovian tasks. These techniques are all subject to Bellman’s curse of dimensionality. This arises because as the number of features in a problem increases, the number of combinations of feature values, and hence distinct states rises exponentially. In order to beat this problem Reinforcement learning algorithms employ function approximators. Two very simple forms of function approximation are state aggregation and linear function approximators. However, even for these simple function approximators the convergence properties of the algorithms discussed previously can break down quickly. TD learning is known to converge with linear function approximation, but Q-learning is known to diverge with linear function approximators in a number of counter-examples and also often in practice with a wide range of function approximators. The problem of how to approximate the value function in general is a difficult open problem in reinforcement learning and a large number of papers have published. There is not space in this chapter to cover the different approaches in any depth. The interested reader is referred to [5] for a comprehensive coverage of these issues.

6 Optimal Learning

So far we have been concerned with algorithms that learn to behave optimally. However, while learning to behave optimally these algorithms may well perform rather badly, particularly in the early stages of learning. Sometimes it matters how well we perform while we are learning. Suppose we are adapting our strategy during a game of robot football, for example, or learning to control a robot while travelling through an office building. In summary how should an agent act *while*

it is learning? This question falls into the field studying what is sometimes known as *optimal learning*.

In addition to providing a framework for learning optimal behaviour it conveniently transpires that reinforcement learning also provides an elegant framework for studying optimal learning. Once we have a clear mathematical framework for optimal learning we can ask and sometimes answer questions such as how should I act so as to maximise my performance over a limited lifetime given that I will continue to learn throughout that lifetime? The problem of how to act while learning is a class of optimal control problems with a long history [10,3], and includes topics such as adaptive dual control from control theory and bandit problems from statistics. In RL it has typically taken the form of two problems: (i) how to act so as to maximise performance during the learning agent's lifetime [17,12]; and (ii) how to act to identify as good a policy as possible within the learning period [14,9,8]. These problems, while related, are not the same [36]. Either is more commonly known as the exploration-exploitation problem, and there is some ambiguity in the RL literature as to which one we are referring when we use this term. The first problem is, however, currently the only one for which we have a clear formulation, and it is therefore the solution of this that I describe here.

Arguably the cleanest framework for understanding optimal learning and the exploration-exploitation trade-off is a Bayesian one. Bayesian approaches model our uncertainty about the form of the transition and reward functions. When we know the transition and reward functions there is after all, no exploration-exploitation trade-off, the optimal way to behave is to act greedily since no information can be gained. If there is uncertainty however, Bayesian approaches can then take that uncertainty into account in calculating value functions which tell us how to act so as to optimise our performance while we are learning.

For MDPs the optimal Bayesian solution to problem (i) is well known, but intractable [17,3]. Many approximations have been proposed. The domain is a finite state MDP with an unknown transition function P and a known reward function R . This differs slightly from the assumption of some RL researchers that the reward function is essentially part of the environment, and thus unknown (but observable) by the agent. Clearly if we knew P then the problem would reduce to finding the value function using a standard dynamic programming technique and we would have no learning problem at all. Our optimal learning problem is thus concerned with the uncertainty there is about the parameters of P (the transition probabilities) and how this uncertainty changes as the learner gathers information. The essential trade-off occurs because at each step we can choose actions that exploit information we already have about P to gather reward, or actions that explore and gather information about P that we can use later on to gather even higher rewards. In other words our dilemma is "should I sacrifice reward now to gather information that may let me gather greater rewards later on?"

The Bayesian approach is based on there being a space \mathcal{P} of possible transition functions (or models) P for the MDP, and a well-defined prior probability

density over that space. The probability density over the space of possible finite state MDPs for a known state space \mathcal{S} is constructed as follows. First let us think about the density over the possible one-step transition functions from a single state action pair. If state $i \in \mathcal{S}$ has N possible succeeding states when action a is taken, then we know already that the transition function from that state action pair is a multinomial distribution over the outcomes:

$$\mathbf{p}_i^a = \{p_{i1}^a, p_{i2}^a \cdots p_{iN}^a\} \quad (22)$$

The possible transition functions from i, a are the possible multinomials \mathbf{p}_i^a . We want a probability density over this space which is closed under sampling from any such multinomial⁴. The Dirichlet density has this property for multinomials:

$$f(\mathbf{p}_i^a | \mathbf{m}_i^a) = \frac{\Gamma(\sum_{j=1}^N m_{ij}^a)}{\prod_{j=1}^N \Gamma(m_{ij}^a)} \prod_{j=1}^N (p_{ij}^a)^{m_{ij}^a - 1} \quad (23)$$

where $\Gamma(\cdot)$ is the Gamma function. The density is parameterised by the $m_{ij}^a > 0$ for all states j to which the process can transition in one step from state i under action a . Effectively each parameter m_{ij}^a represents the number of observations of that outcome. The experimenter can set a prior $m_{ij}^{a'}$ to reflect their beliefs about the likelihood of each outcome before making any actual observations. On making observations the parameter vector is updated as follows: if a single observation of a transition $i \xrightarrow{a} j$ is made, then the new density is also Dirichlet with $m_{ij}^{a''} = m_{ij}^{a'} + 1$. The Bayesian estimate of the likelihood of each transition is simply:

$$\bar{p}_{ij}^a = \frac{m_{ij}^a}{\sum_{k=1}^N m_{ik}^a} \quad (24)$$

We can compare this estimate to the maximum likelihood estimate of the transition probabilities which is of the same form, but with the initial $m_{ik}^a = 0$ for all successor states k .

Since the transition probabilities from a single state,action pair are a multinomial distribution, we can see that using a Dirichlet to express the uncertainty we have about the precise transition probabilities is sensible. The density over the space \mathcal{P} of models for the multi-state case follows directly from that for the transitions from a single state action pair. The densities over the one step

⁴ We say that a family of densities is closed under sampling from a distribution. This means that I have a prior density from a certain family (e.g. Gaussian, Dirichlet) over the (unknown) parameters of the distribution I am sampling from. If I sample from the distribution and incorporate that sample information using Bayes rule then I am guaranteed to end up with a posterior from the same family of densities. Choosing a density that is closed under sampling is appealing because it makes calculating the posterior mathematically straightforward and computationally tractable.

transition functions for the different state action pairs are mutually independent. The density $f(P|M)$ for a possible transition function $P \in \mathcal{P}$ for the whole MDP (i.e. for all the state action pairs at once) is therefore simply the product of the $f(\mathbf{p}_i^a | \mathbf{m}_{ij}^a)$ over all i, a . This density is now parameterised by a matrix $M = [m_{ij}^a]$, where $M \in \mathcal{M}$. In a Bayesian framework we now choose a prior matrix M' , which specifies our prior density over the space of possible models. The additional information from a sequence of observations is captured in a count matrix F . The posterior density given these observations is therefore simply parameterised by $M'' = M' + F$. For convenience the transformation on M due to a single observed transition $i \xrightarrow{a} j$ is denoted $T_{ij}^a(M)$.

Now we have a parametric density over the space of possible MDPs and a clear way of updating its parameters given samples from the true MDP we can turn to the problem of estimating the value function. The value function in an MDP with unknown transition probabilities is clearly a random variable, \tilde{V}_i since it is a function of P which is itself a random variable. Given the usual squared error loss function the Bayesian estimator of expected return under the optimal policy is the simply the expectation of \tilde{V}_i :

$$V_i(M) = \mathbb{E}[\tilde{V}_i | M] = \int_{\mathcal{P}} V_i(P) f(P|M) dP \quad (25)$$

where $V_i(P)$ is the value of i given the transition function P . The central result of both Bellman and Martin was that when this integral is evaluated we transform our problem into one of solving an MDP with known transition probabilities, defined on the information space $\mathcal{M} \times \mathcal{S}$:

$$V_i(M) = \max_a \left\{ \sum_j \bar{p}_{ij}^a(M) (r_{ij}^a + \gamma V_j(T_{ij}^a(M))) \right\} \quad (26)$$

where $\bar{p}_{ij}^a(M)$ is the marginal expectation of the Dirichlet as given above, $0 \leq \gamma < 1$ is the discount rate, and r_{ij}^a is the reward associated with the transition $i \xrightarrow{a} j$. The value function $V(M)$ is known as the Bayes value function. The Bayes Q-values are obviously given by the inner part of Equation 26:

$$Q_{ia}(M) = \sum_j \bar{p}_{ij}^a(M) (r_{ij}^a + \gamma V_j(T_{ij}^a(M))) \quad (27)$$

The Bayes Q-values naturally take into account the uncertainty about the process parameters in P . Thus the Bayesian estimate of value elegantly incorporates the value of future information. The optimal solution to the well-known exploration-exploitation trade-off (problem (i) above) is thus simply to act greedily with respect to the Bayes Q-values. Because the solution involves dynamic programming over a graph of information states the problem of actually obtaining the Bayes Q-values is intractable. A simple approximation to this is the certainty equivalent (CE) estimate constructed by replacing $T_{ij}^a(M)$ with M in (26).

There are a number of possible ways of coming up with an estimate of, or approximation to, the Bayes value function in a reasonable amount of time. One

way is to estimate the value of the integral by random sampling [8,27] directly from the probability density over the space of models. For each sampled model we obtain the Q-value function using dynamic programming and then calculate the average of the sampled Q-value functions for each state of the MDP. Unfortunately this will give us an estimate that has a standard error that declines slowly with the sample size. However, there are now value iteration techniques that work on what are sometimes referred to as structured representations of MDPs. These structured representations encode the states of the MDP as combinations of feature values. It has been shown [7] that the transitions between states can therefore be represented as a dynamic Bayes network (DBN). We can define a density over the space of possible transition models represented by the DBN in exactly the same way as we can for a standard (unfactored) MDP. The space of transition models P is constrained greatly by the structure represented by the DBN, and hence we would expect that the estimates produced by sampling from the resulting density over P would have a lower standard error. It is therefore reasonable to hypothesise that we might improve the performance of Monte Carlo approaches to the optimal learning problem by employing structured models.

Most solutions try to avoid the difficulties of dealing with the information state space in any form and essentially seek to approximate the right answer by solving a different MDP defined on the original state space [20,12,19,34,37]. The new MDP may differ from the old MDP in a number of ways. Either we can pick a different reward function, which reflects the uncertainty we have about the transition matrix, or we can pick a different transition matrix P . Whichever route we choose we typically select according to the heuristic “be optimistic in the face of uncertainty”. If we have a learner which directly estimates the parameters of the MDP then a good solution is to pick a model P which is optimistic with respect to the value function [34,37]. If we wish to do temporal difference learning then we are probably better off picking a reward function which gives bonuses to states which have been visited infrequently or where there appears to be uncertainty about the value function [19]. It is also worth noting that we do know of quite simple algorithms which are guaranteed to find ϵ -optimal policies for unknown MDPs in strictly polynomial time and space [14]. However the practical performance of these methods is likely to be much worse than that of most heuristic algorithms.

7 Summary

In this chapter we have outlined the simplest mathematical framework within which we can conduct reinforcement learning over many steps, a finite state Markov Decision Process. We have described three approaches to solving the MDP: Monte-Carlo sampling, temporal difference learning and adaptive dynamic programming. Clearly all three are related through Monte-Carlo sampling in the learning case in that we are sampling our experiences directly from a process we believe to be stochastic. Adaptive dynamic programming and temporal difference

approaches, however also allow us to take advantage of the conditional probability structure of the Markov chain and so construct estimates of value which have a rapidly decreasing standard error. It is notable that here we have concentrated solely on model-based and value based techniques for solving MDPs. There is burgeoning subfield working on searching directly in the policy space, and this approach may prove to be better for a number of problems. Finally we have also extended the MDP framework to cover the case of MDPs with unknown transition probabilities. This has allowed us to represent the problem of optimal learning, which unlike the problem of learning optimal policies for MDPs is essentially intractable.

Many hard questions remain before we have developed reinforcement learning algorithms for practical tasks. One of the most important questions is whether we can have algorithms that can learn from reinforcement in the face of hidden state. One mathematical framework for such problems is that of partially observable Markov Decision Processes (POMDPs). In these there is an underlying MDP which is not directly observable. Instead each state generates observations probabilistically, and there are typically fewer observations than states; so that many states can generate the same observations. Even solving these for the case where we know the parameters of the POMDP turns out to be intractable [16]. However, for solving known POMDPs we at least have algorithms that are guaranteed to converge in the limit if the value function is representable in finite space. If we want to address the learning case we are trying solve a POMDP where the parameters are unknown. Here the most successful approaches either fall into the model learning or policy modification approaches. Model learning approaches typically try to learn either a hidden Markov model or a k^{th} order Markov model [6,18]. These rely on statistical tests to try to identify the number of underlying states in the model. However, it now appears that at least for some problems that searching directly in the policy space can produce much better results.

Reinforcement learning provides a perspective on learning which combines a number of assumptions. We work with a short term measure of performance and convert this into some long term measure. We assume that the problem is a sequential decision making problem, and try to exploit the conditional independence structure that may exist. Finally we have a class of algorithms that use randomised interaction with the decision process (Monte Carlo sampling) to learn directly or indirectly how to behave. These themes: sequential decision making, randomised interaction, and the exploitation of conditional independence; are ideas that crop up not just in reinforcement learning, but throughout the study learning mechanisms and artificial intelligence in general. In reinforcement learning they are studied specifically within the framework imposed by learning from a reward signal. As we have seen this can give us a framework not only for learning optimal behaviour, but also for how to act while learning.

References

1. A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. Technical Report CMPSCI-TR-93-02, University of Massachusetts, March 1993. Revised version of TR-91-57, Real-time learning and control using asynchronous dynamic programming.
2. A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins. Learning and sequential decision making. COINS Technical Report 89-95, University of Massachusetts, September 1989. Later published in '*Learning and Computational Neuroscience*' edited by M.Gabriel & J.W. Moore.
3. R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
4. D. Bertsekas. *Dynamic Programming and Stochastic Control*. Academic Press, 1976.
5. Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.
6. Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188, 1992.
7. Richard Dearden, Craig Boutilier, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.
8. Richard Dearden, Nir Friedman, and David Andre. Model-based Bayesian exploration. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 150–159, San Francisco, CA, 1999. Morgan Kaufmann Publishers.
9. C.N. Fiechter. Expected mistake bound model for on-line reinforcement learning. In Douglas H. Fisher, editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 116–124. Morgan Kaufmann, 1997.
10. J.C. Gittins. *Multi-armed Bandit Allocation Indices*. Interscience Series in Systems and Optimization. John Wiley & Sons, 1989.
11. J.H. Holland. Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning II*, pages 593–623. Kaufman, 1986.
12. Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, Dept of Computer Science, Stanford, 1990.
13. Littman M.L. Kaelbling L.P. and Moore A.W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1995.
14. M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In Jude Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 260–268. Morgan Kaufmann, 1991.
15. Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3/4):293–321, 1992.
16. Michael Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, March 1996.
17. J.J Martin. *Bayesian Decision Problems and Markov Chains*. Wiley, New York, 1967.
18. Andrew Kachites McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, 1996.

19. N. Meuleau and P. Bourguine. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Machine Learning*, 35:117–154, 1999.
20. Andrew W. Moore and Christopher G Atkeson. Prioritised sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.
21. K. Narendra and M.A.L. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall, 1989.
22. Jing Peng and Ronald J. Williams. Incremental multi-step Q-learning. In W.W.Cohen and H.Hirsh, editors, *Machine Learning: Proceedings of the 11th International Conference*, pages 226–232, 1994.
23. Gavin Rummery. *Problem Solving with Reinforcement Learning*. Ph.D. dissertation, University of Cambridge, 1995.
24. A Schwartz. A reinforcement learning method for maximising undiscounted rewards. In *Machine Learning: Proceedings of the Tenth International Conference*. Morgan Kaufmann, 1993.
25. Satinder Singh and Richard Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, (22):123–158, 1996.
26. S.P. Singh. Reinforcement learning algorithms for average-payoff Markovian decision processes. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. AAAI Press/MIT Press, 1994.
27. Malcolm Strens. A Bayesian framework for reinforcement learning. In Pat Langley, editor, *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 943–950. Morgan Kaufmann, 2000.
28. R.S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, School of Computer and Information Sciences, 1984.
29. R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.
30. R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Bruce W. Porter and Ray J. Mooney, editors, *Machine Learning: Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann, 1990.
31. R.S. Sutton and S.P. Singh. On step-size and bias in temporal difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96, 1994.
32. C.J.C.H Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, King's College, Cambridge, England, May 1989.
33. C.J.C.H. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.
34. M. Wiering and J. Schmidhuber. Efficient model-based exploration. In R. Pfeiffer, B. Blumberg, J. Meyer, and S. W. Wilson, editors, *From Animals to Animals 5: Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, 1998.
35. R.J. Williams and L.C. Baird. Tight performance bounds on greedy policies based on imperfect value functions. Technical Report NU-CCS-93-11, Northeastern University, College of Computer Science, November 1993.
36. Jeremy Wyatt. *Exploration and Inference in Learning from Reinforcement*. PhD thesis, University of Edinburgh, Dept. of Artificial Intelligence, Edinburgh University, May 1997.

37. Jeremy Wyatt. Exploration control in reinforcement learning using optimistic model selection. In A. Danyluk and C. Brodley, editors, *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.