Chapter 4
An Introduction to Reinforcement Learning

Eduardo F. Morales
National Institute of Astrophysics, Optics and Electronics, México

Julio H. Zaragoza
National Institute of Astrophysics, Optics and Electronics, México

ABSTRACT

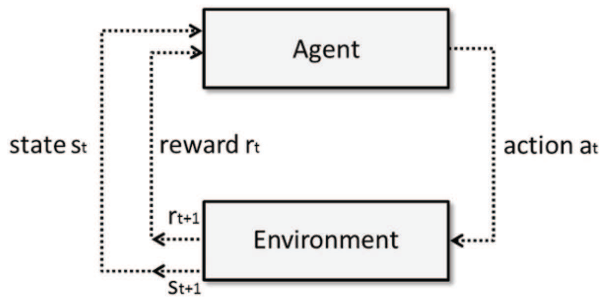
This chapter provides a concise introduction to Reinforcement Learning (RL) from a machine learning perspective. It provides the required background to understand the chapters related to RL in this book. It makes no assumption on previous knowledge in this research area and includes short descriptions of some of the latest trends, which are normally excluded from other introductions or overviews on RL. The chapter provides more emphasis on the general conceptual framework and ideas of RL rather than on presenting a rigorous mathematical discussion that may require a great deal of effort by the reader. The first section provides a general introduction to the area. The following section describes the most common solution techniques. In the third section, some of the most recent techniques proposed to deal with large search spaces are described. Finally, the last section provides some final remarks and current research challenges in RL.

INTRODUCTION

Reinforcement Learning (RL) has become one of the most active research areas in Machine Learning[1]. In general terms, its main objective is to learn how to map states to actions while maximizing a reward signal. In reinforcement learning an autonomous agent follows a trial-and-error process to learn the optimal action to perform in each state in order to reach its goals. The agent chooses an action in each state, which may take the agent to a new state, and receives a reward. By repeating this process, the agent eventually learns which is the best action to perform to obtain the maximum expected accumulated reward. In general, in each iteration (see Figure 1), the agent perceives its cur-

DOI: 10.4018/978-1-60960-165-2.ch004

Figure 1. Reinforcement learning process



rent state ($s \in S$), selects an action ($a \in A$), possibly changing its state, and receives a reward signal ($r \in R$). In this process, the agent needs to obtain useful experiences regarding states, actions, state transitions and rewards to act optimally and the evaluation of the system occurs concurrently with the learning process.

This approach appeals to many researchers because if they want to teach an agent how to perform a task, instead of programming it, which may be a difficult and time-consuming process, they only need, in principle, to let the agent learn how to do it by interacting with the environment.

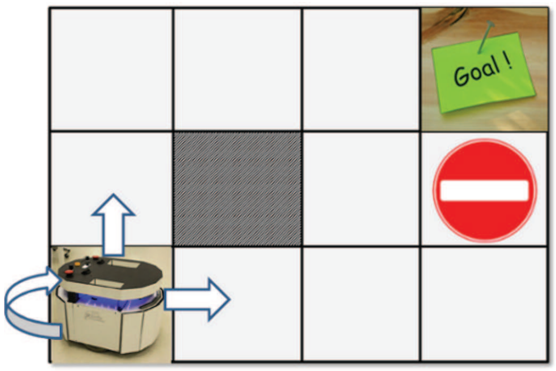
To illustrate this learning process, suppose that we want a mobile robot to learn how to reach a particular destination in an indoor environment. We can characterize this navigation problem as a RL problem. The states can be defined in terms of the information provided by the sensors of the robot, for instance, if there is an obstacle in front of the robot or not. We can have a finite set of actions per state, such as *go-forward*, *go-backward*, *go-left* and *go-right*, and the goal may be to go to a particular place (see Figure 2). Each time the robot executes an action there is some uncertainty on the actual next state as the wheels of the robot often slip on the ground or one wheel may turn faster than another, leaving the robot in a possibly different expected state. Upon reaching a goal state, the robot receives a positive reward and similarly receives negative regards in undesirable states. The robot must choose its actions in order to increase in the long turn the total accumulated rewards. By following a trial and error process the robot learns after several trials which is the best action to perform on each state to reach the destination point and obtain the maximum expected accumulated reward.

Deciding which action to take in each state is a sequential decision process. In general, we have a non-deterministic environment (the same action in the same state can produce different results). However, it is assumed to be stationary (i.e., the transition probabilities do not change over time). This sequential decision process can be characterized by a Markov Decision Process or MDP. As described in Chapter 3, an MDP, M=<S,A,P,R>, can be described as follows:

- A finite set of states (S)
- A finite set of actions (A_s) per state (s)
- A reward function $R : S \times A \rightarrow R$
- A transition function P that represent the probability of reaching state $s' \in S$ given an action $a \in A$ taken at state $s \in S$:$P(s'|s,a)$.

In many settings, the agent receives a constant reward until reaching a terminal state. For instance, the robot moves through the environment and it is until it crashes or reaches a goal state that it receives a distinctive reward. This introduces the credit-assignment problem, i.e., if the robot

Figure 2. Reinforcement Learning process with a mobile robot where the robot may have different possible actions per state and the goal is to learn how to go to a particular destination while receiving the maximum total expected reward



eventually reaches a good (bad) state, we would like to know which actions are responsible for that. Two distinctive features of RL are that: it follows a trial and error process, which means that it should follow an exploration strategy, and that there is a delayed reward, which introduces this credit-assignment problem. We will first introduce different reward models, then talk about exploration-exploitation strategies and then describe the concepts of value functions and policies that are needed to understand how the RL algorithms work.

Reward Models

Given a state $s_t \in S$ at some time *t* and an action $a_t \in A_{s_t}$, the agent receives a reward r_{t+1} and moves to the next state s_{t+1}. The idea in RL is not to maximize the immediate rewards but the long term accumulated reward. We can denote the total received rewards after certain time *i* as:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_{t+i}$$

If there is a terminal state, it is said that we have *episodic* tasks, otherwise these tasks are considered *continuous*. In this last case, we cannot impose an upper limit in the accumulated rewards, so an alternative way is to geometrically reduce the contributions of the rewards as they become more distant: <<:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where γ is known as the *discount rate* and it is between: $0 \leq \gamma < 1$.

Since we have a probabilistic transition function, what we want is to maximize the expected accumulated reward, and in general, we can have the following models:

Finite Horizon: the agent tries to optimize the expected accumulated reward on the next *h* steps, without considering what happens afterward:

$$E(\sum_{t=0}^{h} r_t)$$

where r_t refers to the reward received after *t* steps in the future.

This model can be used in two forms: (i) *stationary policy*: where in the first state it takes *h* next steps, in the next, it takes *h*−1 steps, etc., until the end. The main problem is that it is not always possible to know how many steps to take. (ii) *Receding-horizon control*: it always takes the next *h* steps.

Infinite Horizon: the rewards received by the agent are geometrically reduced according to a discount factor γ ($0 \leq \gamma < 1$):

$$E(\sum_{t=0}^{\infty} \gamma^t r_t)$$

Average Reward: the idea is to optimize in the long run the average reward:

$$lim_{h \to \infty} E(\frac{1}{h} \sum_{t=0}^{h} r_t)$$

One problem with this optimization criterion is that it cannot distinguish between policies that receive a big reward at the beginning or at the end of the episode.

The most widely used model is *infinite horizon* and it is the one we will assume from now on.

Exploration and Exploitation

One important aspect in RL is that it must explore the environment to gather information in order to build a policy. We do not want to leave unexplored areas but we also want to use the accumulated knowledge to make better decisions. In this sense, there is a balance between exploration and exploitation. To gain more rewards an agent can follow certain actions that are known to produce high immediate rewards, however, in order to know which is the best action it has to explore the environment. In many cases the exploration strategy depends on the time that the agent has interacted with the environment.

Some common strategies to select actions and explore the environment are:

- ε–*greedy*: where most of the time the selected action is the one with the largest estimated accumulated reward but with probability ε an action is randomly selected.

Softmax: where the probability of selecting an action depends on its estimated accumulated reward. The most common being the Boltzmann or Gibbs distribution, which selects an action on state *s* with probability:

$$\frac{e^{Q_t(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(s,b)/\tau}}$$

where τ is a positive parameter (temperature), *n* is the number of possible actions at state *s*, and *Q* is a value function that represents the estimated accumulated rewards and which is described in the following section.

Value Functions and Policies

In general the actions determine not only the immediate reward, but also in a probabilistic way, the next state. In RL the transition model is assumed to be Markovian, so the state transitions do not depend on previous states, and the transition probabilities are given by:

$$P(s' \mid s, a) = P(s_{t+1} = s' \mid s_t = s, a_t = a)$$

The expected reward value is:

$$R(s' \mid s, a) = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

The total expected reward depends on the current state and on the selection of actions in future states. The selection of actions per state is given by a *policy*. More formally, a policy π is a mapping of each state $s \in S$ and action $a \in A(s)$ to the probability $\pi(s,a)$ of taking the action *a* in state *s*.

One of the goals in RL is to estimate how good it is to be in a state (or being in a state and perform an action). The notion of "goodness" is defined in terms of future rewards or expected accumulated rewards which are represented as *value functions*. The value function of a state *s*, denoted by $V^\pi(s)$, represents the total expected accumulated reward that the agent can receive starting at state *s* and following policy π. Similarly, the value function of a state *s* taking action *a*, is denoted as $Q^\pi(s,a)$ and represents the total expected accumulated reward that the agent can receive starting at state *s*, taking action *a* and following a policy π. The idea is to find the policy that produces the maximum value functions rather than the maximum immediate rewards. The rewards are given by the environment, but the value functions need to be estimated (learned) with experience. *Reinforcement learning learns value functions while interacting with the environment*.

The value function for a state *s* using an infinite discounted reward model can be expressed as:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

Likewise, the value function of a state *s* with action *a* and policy π ($Q^{\pi}(s,a)$) can be expressed as:

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\}$$
$$= E_{\pi}\left\{\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

If we expand the expression for $V^{\pi}(s)$:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}(R_t \mid s_t = s) \\ &= E_{\pi}(\sum_{k=o}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s) \\ &= E_{\pi}(r_{t+1} + \gamma \sum_{k=o}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s) \\ &= \sum_a \pi(s,a) \sum_{s'} P(s' \mid s,a)\Big[R(s' \mid s,a) + \gamma E_{\pi}(\sum_{k=o}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s)\Big] \\ &= \sum_a \pi(s,a) \sum_{s'} P(s' \mid s,a)\Big[R(s' \mid s,a) + \gamma V^{\pi}(s')\Big] \end{aligned} \tag{1}$$

where π(*s,a*) is the probability of taking action *s* in state *s* under policy π.

We can similarly derive an equivalent expression for $Q^{\pi}(s,a)$:

$$Q^{\pi}(s,a) = \sum_{s'} P(s' \mid s,a)\Big[R(s' \mid s,a) + \gamma V^{\pi}(s')\Big] \tag{2}$$

The last two equations form the basis of the RL algorithms as value functions are normally updated in terms of the immediate rewards and value functions of the following state.

If we choose different policies we obtain different value functions. For instance, a mobile robot may choose a right-hand wall-following policy from which particular value functions are obtained for particular mazes. A left-hand wall-following policy produces different value functions. In general, given a particular policy we can evaluate the associated value functions for each state or state-action pair.

In practice, we want to produce the best policies, i.e., those that produce the largest expected accumulated rewards. A policy π is defined to be better than or equal to a policy π' if its expected return is greater or equal to that of π' for all states.

$$\pi \geq \pi' \text{ if } V^{\pi}(s) \geq V^{\pi'}(s) \text{ for all } s \in S$$

There is always at least one policy that is better than or equal to all other policies, i.e., an *optimal policy*, π^*. The optimal policy shares the optimal state value function V^* and the optimal state-action value function Q^* and can be expressed as:

$$V^*(s) = max_{\pi} V^{\pi}(s) \text{ and } Q^*(s,a) = max_{\pi} Q^{\pi}(s,a)$$

Considering Eqs. 1 and 2, the optimal value functions can be expressed recursively with the Bellman optimality equations as:

$$V^*(s) = max_a \sum_{s'} P(s' \mid s,a)[R(s' \mid s,a) + \gamma V^*(s')]$$

Similarly, for *Q* values:

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)[R(s' \mid s,a) + \gamma V^*(s')]$$

Algorithm 1. *TD(0) algorithm*

```
Initialize $V(s)$ arbitrarily and π to the policy to evaluate
for each episode do
    Initialize $s$
    repeat
      for each step in the episode do
        $a \leftarrow$ action given by π for $s$
        Perform action $a$; observe the reward $r$ and the next state $s'$
        $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
        $s \leftarrow s'$
      end for
    until $s$ is a terminal state
end for
```

or

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)[R(s' \mid s,a) + \gamma \max_{a'} Q^*(s',a')]$$

So the obvious question now is: how can we learn such policies? In the next section we review some common techniques.

SOLUTION TECHNIQUES

There are tree principal ways of solving MDPs: (i) Dynamic Programming, (ii) Monte Carlo, and (iii) Temporal Differences or Reinforcement Learning. Dynamic Programming methods, based on policy and value iteration are clearly described in Chapter 3. The main idea behind Monte Carlo methods is to simulate experiences, collect statistics and return the averages of the accumulated rewards obtained from the simulated experiences for each state or state-action pair. In this chapter we will concentrate only on RL algorithms.

The idea of the RL techniques is to update the value functions with the next step using the error or difference between successive predictions. Their main advantage is that they represent incremental algorithms that are easy to compute.

The simplest method TD(0) updates $V(s_t)$ as follows:

$$V(s_t) \leftarrow V(s_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right]$$

The TD(0) algorithm is described in Algorithm 1. In general, TD methods update the existing value function using the immediate reward and the estimated value function of the next state. In this case the target is based on the immediate next step information ($r + \gamma V(s')$) which is compared with the actual estimate ($V(s)$), producing a *TD error* which is used to update the current value function. Updating based on existing estimates is also known as *bootstrapping*.

RL can be used for control problems, where we want to learn a value function based on state-action pairs. As previously mentioned, there is a trade-off between exploration and exploitation and the approaches fall into two categories: *on-policy* and *off-policy* strategies. An on-policy approach learns the value of the policy that is used to make the decisions and update value functions based strictly on experience. An off-policy

Algorithm 2. *SARSA algorithm*

```
Initialize $Q$ ($s$, $a$) arbitrarily
for each episode do
       Initialize $s$
       Select an $a$ from $s$ using the policy given by $Q$ (e.g., $\varepsilon$-greedy)
       repeat
             for each step in episode do
                    Perform action $a$, observe $r$, $s'$
                    Choose $a'$ and $s'$ using the policy derived from $Q$
              $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
              $s \leftarrow s'; a \leftarrow a'$
              end for
      until $s$ is a terminal state
end for
```

approach learns the value of a policy other that the one used to make the decisions, it can update the estimated value functions using hypothetical actions, which may not have actually been tried. We will first review an on-policy strategy (SARSA) and then an off-policy strategy (Q-learning).

In SARSA, we update value functions considering an action as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)$$
$$+\alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

The algorithm is almost the same as TD(0) and is described in Algorithm 2, however, in this case we are continuously estimating the *Q* values for a particular policy π, but at the same time we are changing the policy π in a greedy approach considering the *Q* values. It can be proved that this strategy converges to the optimal policy and action-value function as long as we visit all the state-action pairs an infinite number of times.

One of the most important developments in Reinforcement Learning was an *off-policy* algorithm known as Q-learning. The main idea is to update the value functions as follows (Watkins, 1989):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)$$
$$+\alpha[r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

The algorithm is described in Algorithm 3 and it follows an off-policy strategy which means that this algorithm approximates the optimal action-value function independently of the policy being followed.

Another different RL approaches are the Actor-Critic methods. In essence, these are TD methods that have two separate memory structures, one of them is used to represent the policy an the other one is used to represent the value function. In this kind of methods the policy structure is known as the actor, because its goal is to select actions, and the estimated value function is known as the critic; the critic qualifies the actions made by the actor. In Actor-Critic methods learning is always on-policy since the critic must learn about and qualify the policy currently followed by the actor. The evaluation or critique takes the form of a TD error. This (usually scalar) signal is the only output of the critic and its work is to model the learning in both the actor and the critic. The main idea in Actor-Critic methods is to use the TD

Algorithm 3. *Q-learning algorithm*

```
Initialize $Q(s, a)$ arbitrarily
for each episode do
       Initialize $s$Repeatfor each step in episode do
              Select an $a$ from $s$ using the policy given by $Q$ (e.g., $\varepsilon$-greedy)
                    Perform action $a$, observe $r$, $s'$
                    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$

                    $s \leftarrow s'$;

               end for
       until $s$ is a terminal state.
end for
```

error (r+V(s')-V(s)) to change the policy which is responsible for choosing the actions. If the TD error is positive then the tendency to select the correct action is increased, otherwise it is decreased, this tendency is normally changed by increasing (or decreasing) its probability of being selected.

SOME RECENT DEVELOPMENTS

In order to obtain an adequate policy, traditional RL techniques need to visit all the states (or state-action pairs) several times. This, however, is only possible in very restricted domains. With large state and action spaces traditional RL techniques require large computational resources and long convergence times.

Since the number of possible states grows exponentially in the number of features (*curse of dimensionality* (Bellman, 1957)), researchers have proposed different approaches to tackle this problem. Some of the most common approaches are: (i) update several value functions at the same time, (ii) approximate value functions with continuous functions, (iii) learn and use a model to generate new experiences, (iv) employ abstractions and/or hierarchies, and (v) provide additional guidance to the agent. We review these approaches in the following sections.

Update Several Value Functions

In traditional RL each step updates a single value function using the immediate reward. This means that if the agent is two steps away from a state with a very high reward it has to wait until the immediate state is updated to obtain information about such desirable state. One idea is to propagate information among the visited states in what is called *eligibility traces*.

The idea of eligibility traces is to consider the n next states (or change the n previous states) and update several value functions.

As stated previously:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-t-1} r_T$$

What temporal difference methods do is to use the estimated accumulated reward expressed as the value function:

$$R_t = r_{t+1} + \gamma V_t(s_{t+1})$$

Algorithm 4. *SARSA (λ) with eligibility traces*

```
Initialize Q (s, a) arbitrarily y e(s, a) = $0 \forall s, a$ for each episode do
     Initialize s, a
        repeat
           for each step in episode do
               Take action a and observe r, s'
                    Select an a from s using the policy derived from Q (e.g.,
$\varepsilon$-greedy)
                    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
                    $e(s, a) \leftarrow e(s, a) + 1$
                for all s, a do
                           $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
                           $e(s, a) \leftarrow \gamma\lambda e(s, a)$
                end for
                $s \leftarrow s'; a \leftarrow a'$
          end for
       until s is a terminal state.
end for
```

which makes sense since $V_t(s_{t+1})$ replace the next terms ($\gamma r_{t+2} + \gamma^2 r_{t+3} \ldots$).

However, it also makes sense to use the value function after two known rewards:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$$

and in general for *n* future steps.

In practice, rather than waiting *n* steps to update (*forward view*), you can update backwards on the visited states (*backward view*). It can be proved that both approaches are equivalent. In the backward view, you store information over the visited states (the eligibility trace) and use the TD error to update the visited value functions associated with the visited states backwards (discounted by distance).

To implement this idea, each state or state-action pair is associated with an extra variable, representing the *eligibility trace* denoted by $e_t(s)$ or $e_t(s,a)$. This value is decremented with the length of the trace created on each episode.

Initially all the $e_t(s) = 0$. For *TD*(λ) we mark only the visited states:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

and we update the value functions over the visited states using information from the TD errors.

For SARSA we mark the visited state-action pairs as following:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } s \neq s_t \\ \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \end{cases}$$

and update as described before. *SARSA*(λ) is described in Algorithm 4. In this case the reinforcements are accumulated each time the state-action pair is visited and decay gradually when the state-action pair is not visited. At each step in an episode the TD error is propagated backwards to each previously visited state-action pair according to their current eligibility trace value.

For Q-learning, however, we need to be careful since some movements are exploratory movements. The problem is that the agent can reach undesirable states through exploratory actions that can propagate negative rewards along good paths. Some options are to maintain the trace until the first exploratory action is taken, ignoring exploratory actions, or using a more complicated scheme that considers all the possible actions per state.

Approximate Value Functions

So far we have assumed that the value functions are stored in tabular forms. This works fine for small domains, but it is impractical for domains with a large number of states like chess (10^{120}) or backgammon (10^{50}) or for continuous spaces.

One option is to use an implicit representation; a function. For instance, in games a function is used to estimate the utility of a state with a weighted linear function over a set of attributes (f_i's):

$$V(i) = w_1 f_1(i) + w_2 f_2(i) + \ldots + w_n f_n(i)$$

In chess there are approximately 10 weights, so there is a significant compression. This representation also allows to generalize over non visited states.

There are many possible options to represent functions and in general there is a trade off between expressibility and tractability since models represented in more expressive languages are more difficult to learn.

There are several examples of different choices for expressing value functions, such as neural networks (Bertsekas & Tsitsiklis, 1996), decision trees (Chapman & Kaelbling, 1991), SVMs (Dietterich & Wang, 2002), different Kernels (Ormoneit & Sen, 2002) and Gaussian processes (Peters, Vijayakumar, & Schaal, 2003b).

In this chapter, we will only illustrate the approach with a linear combination of basis functions $\phi_1, \phi_2, \ldots, \phi_n$ of the form $\sum_{i=1}^{n} w_i \phi_i$ where the weights ($w_i \in \vec{\Theta}$) need to be learned.

Many supervised learning systems try to minimize the mean squared error (MSE) under certain input distribution. If $\vec{\Theta}_t$ denotes the parameters' vector of the function we want to learn, gradient descent techniques adjust the values of such parameters in the direction that produces the maximum reduction in the error.

$$\begin{aligned}\vec{\Theta}_{t+1} &= \vec{\Theta}_t - \frac{1}{2}\alpha \nabla_{\vec{\Theta}_t}[V^{\pi}(s_t) - V_t(s_t)]^2 \\ &= \vec{\Theta}_t + \alpha[V^{\pi}(s_t) - V_t(s_t)]\nabla_{\vec{\Theta}_t} V_t(s_t)\end{aligned}$$

where α is a positive parameter $0 \leq \alpha \leq 1$ and $\nabla_{\vec{\Theta}_t} f(\Theta_t)$ denotes a vector of partial derivatives.

Since we do not know $V^{\pi}(s_t)$ we have to approximate it. We can do it with eligibility traces and update the function Θ τ as follows:

$$\vec{\Theta}_{t+1} = \vec{\Theta}_t + \alpha \delta_t \vec{e}_t$$

where δ_t is the TD error:

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

and $\vec{e}_t$ is a vector of eligibility traces, one for each component $\vec{\Theta}_t$, that is updated as:

$$\vec{e}_t = \gamma \lambda \vec{e}_{t-1} + \nabla_{\vec{\Theta}_t} V_t(s_t)$$

with $\vec{e}_0 = 0$. It has been shown that in many cases trying to find an approximate value function can diverge (Baird, 1995)

Other approaches have been used to relate or represent interactions among features in different ways. They include coarse coding (Hinton, 1984), tile coding (Lim & Kim, 1991), radial basis

Algorithm 5. *Dyna-Q algorithm*

```
Initialize Q (s, a) and $Model(s,a) \forall s \in S, a \in A$
loop
    $s \leftarrow$ actual state
    $a \leftarrow \varepsilon - greedy(s,a)$
    Take action a observe $s'$ and r
    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
    $Model(s,a) \leftarrow s'r$
    for N times do
        $s \leftarrow$ previous state randomly selected
        $a \leftarrow$ random action taken in s
        $s'r \leftarrow Model(s,a)$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
    end for
end loop
```

functions (Poggio & Girosi, 1989) and Kanerva coding (Kanerva, 1993).

Learn and Use a Transition Model

With eligibility traces we propagate the TD error through visited states. If we could have a transition function, then we could propagate among all states, which is what value iteration and policy iteration do.

RL visits many state-action pairs while learning to approximate value functions, however the information about these state transitions is lost. One idea is use this information to construct a transition model while learning. This is appealing for several reasons: (i) we can converge much faster to an optimal policy with a transition model, (ii) we do not need in general a very precise transition model and we can refine it as we learn, and (iii) the transition model can guide the exploration strategy towards states with poor transition models.

With a transition model, we can predict the next state and plan. What is interesting is that we can use planning also for learning. For a learning system, it does not matter if the state-action pairs and rewards are from real or simulated experiences.

Given a model of the environment, we can randomly select a state-action pair, use the model to select the next state, obtain a reward and update a Q value as if it was part of an episode. This can be repeated until convergence to Q^*.

Dyna-Q combines experience with planning to learn faster a policy. The idea is not only to learn from experience, but also to learn and use a model while learning to simulate experience (see Algorithm 5).

Dyna-Q randomly selects previously visited state-action pairs. However, better planning can be performed if it is focused on particular state-action pairs. For instance, starting in the goal states and going backwards or in any state with a large change on its value function. The idea behind *prioritized sweeping* is to focus the simulation on states that significantly changed their value function (see Algorithm 6). The aim is to simulate new experience only on state-action pairs whose Q values change above a certain pre-defined threshold value.

More recently, some related approaches have been suggested, such as E^3 (Kearns, 1998) and R-MAX (Brafman & Tennenholtz, 2002), where again the idea is to learn while updating a model

Algorithm 6. *Prioritized sweeping*

```
Initialize $Q$ ($s$, $a$) and $Model(s,a) \forall s \in S, a \in A$ and $QueueP$ = ∅
loop
      $s \leftarrow$ actual state
      $a \leftarrow \varepsilon - greedy(s,a)$
      take action $a$ and observe $s'$ and $r$
      $Model(s,a) \leftarrow s'r$
      $p \leftarrow | r + \gamma \max_{a'} Q(s',a') - Q(s,a) |$
      if $p > \theta$ then
            then insert $s$, $a$ in $QueueP$ with priority $p$
      end if
      while $QueueP \neq \emptyset$ do
             for $N$ times do
                    $s,a \leftarrow first(QueueP)$
                    $s',r \leftarrow Model(s,a)$
                    $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$
                    for all $\overline{s},\overline{a}$ which predicts to reach $s$ do
                          $\overline{r} \leftarrow$ predicted reward
                          $p \leftarrow \overline{r} + \gamma \max_a Q(s,a) - Q(\overline{s},\overline{a}) |$
                          if $p > \theta$, then add $\overline{s},\overline{a}$ to $Queue$ with priority $p$
                   end for
             end for
      end while
end loop
```

of its environment by gathering statistics. In E^3 an internal mechanism is used to decide whether to explore or exploit while R-MAX backs up optimistic rewards through the value function so that the learned policy effectively plans to visit insufficiently explored states.

ABSTRACTIONS AND HIERARCHIES

A common approach in Artificial Intelligence to tackle complex problems is to use abstractions (Ďzeroski, Raedt, & Driessens, 2001, Chapman & Kaelbling, 1991, Cocora, Kersting, Plagemanny, Burgardy, & Raedt, 2006, Morales, 2003) and/or to divide the problem in sub-problems (Ormoneit & Sen, 2002, Ryan, 1998, Torrey, Shavlik, Walker, & Maclin, 2008, Dietterich, 2000), perhaps using a hierarchy (Govea & Morales, 2006, Cuaya & Muñoz-Meléndez, 2007). RL has not been an exception and several approaches have been suggested along these lines.

One common approach is state aggregation (Singh, Jaakkola, & Jordan, 1996, Otterlo, 2003), in which several “similar” states are joined and they all receive the same value, thereby reducing the state space.

Another possibility is to divide the problem in sub-problems (Dietterich, 2000), learn a policy for each sub-problem and then join these policies to solve the original problem. There has been several approaches around this idea. One of them learns sequences of primitive actions (policies) and treat them as an abstracted action (e.g., Macros or Options (Sutton & Barto, 1989)). Another possibility is to learn sub-policies at a low level within a hierarchy and use them to learn policies at a higher level in the hierarchy (e.g., MAXQ (Dietterich, 2000), HEXQ (Hengst, 2002)).

Actions can also have variable duration, so it is possible to have abstractions over the action sequences, in what is called semi-Markov decision processes of SMDPs (Sutton, Precup, & Singh, 1999).

An alternative approach is based on the definition of finite state machines, in which the RL task is to decide which machine to use in a hierarchical abstract machine or HAM (Parr & Russell, 1998).

More recently, authors have used first-order representations for RL, which can have more abstract states and produce transferable policies. The idea is to use first-order representations to reason about objects and relations between objects (see (Otterlo, 2009)).

In general some abstractions can introduce partial observability and at an abstract level the problem may not longer be Markovian. Even with convergence at the abstract level does not mean convergence to an optimal solution at a primitive level.

ADDITIONAL GUIDANCE

Besides dividing or abstracting the RL problem, some approaches have focused on speeding up the learning process in order to develop control policies in reasonable times for very complex domains by including additional guidance or feedback from the user.

One commonly used approach is to "observe" a person (expert) perform a task (rather than asking him/her how to do it) and save logs with information of the performed actions and then use this information to guide the RL policy search process. The same idea has been developed under different flavors known as Behavioural Cloning (BC) (Bratko, Urbančič, & Sammut, 1998), Apprenticeship Learning (Abbeel & Ng, 2004) and Programming by Demonstration (Billard, Calinon, Dillmann, & Schaal, 2008).

The simplest approach is to use the information from the trace-logs to update the value functions and then follow the RL process with the initial value functions already updated with information of the traces (Singh, Sutton, & Kaelbling, 1996).

Other approaches use the traces to try to derive a reward function in what is known as *inverse Reinforcement Learning* (Ng & Russell, 2000). In this case, rather than trying to derive a direct mapping from states to the actions of the expert, they derive a reward function that penalizes deviations from the desired trajectories, trying to recover the expert's true and unknown reward function (see also (Schaal, 1997, Abbeel & Ng, 2004)).

An alternative approach is to use the given traces to learn a set of possible actions to perform on each state and then use RL to decide which is the best action among such reduced subset of actions ((Morales & Sammut, 2004, J. Zaragoza, 2010), see also Chapter 9).

Another approach to provide additional guidance is through the reward function. Reward shaping attempts to mold the conduct of the agent by adding additional rewards that encourage a behavior consistent with some prior knowledge (Ng, Harada, & Russell, 1999, Marthi, 2007, Grzes & Kudenko, 2009). As the shaping rewards offer localized advice, the time to exhibit the intended behavior can be greatly reduced. Since given an adequate reward shaping function may not be easy for some domains, feedback from the user can be used to provide such guidance. The user may critique sub-sequences of given traces (e.g., (Judah, Roy, Fern, & Dietterich, 2010, Argall, Browing, & Veloso, 2007)) or the system may try to learn a model from the user's feedback (e.g, (Knox & Stone, 2010)) and use that information to adjust the current policy. An alternative approach, called dynamic reward shaping, is to provide on-line feedback from the user to change the reward function during the learning process (Tenorio-Gonzalez, Morales, & Pineda, 2010).

Other Approaches

Reinforcement Learning research has grown in multiple areas. In particular there is substantial research in multi-agent systems, where the idea is to develop a control policy for a group of agents that combine their efforts to perform a common task. Here an agent has to take into account the consequences of the actions of the other agents while trying to find out its right sequence of actions that, eventually, will lead the group to the goal. Similarly, RL has been used in adversarial games or competitive MDPs where there is a sequential decision problem with several decision makers. In this setting the transition probabilities depend on the current state and the actions chosen by all the agents, however, the immediate reward can be a different function for each player. Here the policy of each player is referred to as a strategy and the overall performance metric depends on the strategies selected by all the players. When all the players want to optimize their own objective normally the solutions of the algorithms arrive to the Nash equilibrium (see for example (Littman, 1994) for zero-sum games).

As described in Chapter 3, there has been an increasing interest in developing algorithms for partially observable states in MDPs in what are called POMDPs. This has also been the case for RL and it is known as Partially Observable Reinforcement Learning (PORL) in which the agent has to deal with the uncertainty on its current state while learning a policy (lbling, Littman, & Cassandra, 1998). This setting is common in mobile robots where the sensors are normally noisy and the robot may have some uncertainty of its current state.

Another productive line of research has been the incorporating of a Bayesian framework into RL (Strens, 2000, Dearden, Friedman, & Russell, 1998). This has also been extended into multiple tasks in a hierarchical approach (Wilson, Fern, Ray, & Tadepalli, 2007) and has been incorporated also into inverse RL (Ramachandran, 2007). Very recently there have been interest in combining RL with optimal control and dynamic programming techniques for inverse RL (Theodorou, Buchli, & Schaal, 2010).

Finally, there has been some interesting work on how to transfer policies to similar domains to avoid learning from scratch (Fernandez & Veloso, 2006, Sunmola & Laboratory, 2006, Ferguson & Mahadevan, 2006).

FINAL REMARKS

Reinforcement learning is a very active research area that has been used in several areas of artificial intelligence and that has produced some very interesting results in real–world applications. These range from elevator scheduling (Crites & Barto, 1995), job-shop scheduling (Zhang & Dietterich, 1995), the AGV routing problem (Tadepalli & Ok, 1998), to areas such as Backgammon (Tesauro, 1992), humanoid control (Peters, Vijayakumar, & Schaal, 2003a), and control of helicopters (Tang, Singh, Goehausen, & Abbeel, 2010, Abbeel, Coates, & Ng, 2010).

A general introductory reference is the excellent book by Sutton and Barto (Sutton & Barto, 1989). An on-line version of the book can be found on Sutton's webpage. An earlier comprehensive survey up to 1966 is (Kaelbling, Littman, & Moore, 1996) which is also available from Kaelbling's webpage. There are other more recent surveys, such as (Gosavi, 2008).

There is a large body of literature on specific topics such as transfer learning in RL (Taylor & Stone, 2009) or relational reinforcement learning (Otterlo, 2009), and a continuous flow of current research papers on mayor machine learning forums.

ACKNOWLEDGMENT

The authors would like to thank Enrique Muñoz de Cote, Enrique Sucar and Jesse Hoey for their useful comments on an earlier version of this chapter.

REFERENCES

Abbeel, P., Coates, A., & Ng, A. Y. (2010). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, *29*(13), 1608–1639. doi:10.1177/0278364910371999

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on machine learning.* ACM Press.

Argall, B., Browing, B., & Veloso, M. (2007). Learning by demonstration with critique from a human teacher. In *2nd Conf. on Human-Robot Interaction* (pp. 57–64).

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 30–37). Morgan Kaufmann.

Bellman, R. (1957). *Dynamic programming*. Princeton, NJ: Princeton University Press.

Bertsekas, D. P., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Billard, A., Calinon, S., Dillmann, R., & Schaal, S. (2008). *Robot programming program by demonstration*. MIT Press.

Brafman, R. I., & Tennenholtz, M. (2002). R-max - A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, *3*, 213–231.

Bratko, I., Urbančič, T., & Sammut, C. (1998). Behavioural cloning: phenomena, results and problems. Automated systems based on human skill. In *Proc. of the International Federation of Automatic Control Symposium.* Berlin.

Chapman, D., & Kaelbling, L. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparison. In *Proc. of the International Joint Conference on Artificial Intelligence* (p. 726-731). San Francisco, CA: Morgan Kaufmann.

Cocora, A., Kersting, K., Plagemanny, C., Burgardy, W., & Raedt, L. D. (2006). Octuber). Learning relational navigation policies. *Journal of Intelligent & Robotic Systems*, 2792–2797.

Crites, R. H., & Barto, A. G. (1995). *Improving elevator performance using reinforcement learning* (pp. 1017–1023). Proc. of the Neural Information Processing Systems.

Cuaya, G., & Muñoz-Meléndez, A. (2007, September). *Control de un robot hexapodo basado en procesos de decision de markov.* Primer Encuentro de Estudiantes en Ciencias de la Computación (CIC-IPN).

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian q-learning. In *Proc. of the AAAI Conference on Artificial Intelligence* (pp. 761–768). AAAI Press.

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

Dietterich, T. G., & Wang, X. (2002). Batch value function approximation via support vectors . In *Advances in Neural Information Processing Systems* (pp. 1491–1498). Cambridge, MA: MIT Press.

Džeroski, S., Raedt, L. D., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*(2), 5-52.

Ferguson, K., & Mahadevan, S. (2006). Proto-transfer learning in markov decision processes using spectral methods. In *Proc. of the ICML Workshop on Transfer Learning.*

Fernandez, F., & Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 720–727). ACM Press.

Gosavi, A. (2008). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 1–45.

Govea, B. V., & Morales, E. (2006). *Learning navigation teleo-operators with behavioural cloning*.

Grzes, M., & Kudenko, D. (2009). Theoretical and empirical analysis of reward shaping in reinforcement learning. In *Proc. of the International Conference on Machine Learning and Applications* (p. 337-344).

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

Hinton, G. (1984). *Distributed representations* (Tech. Rep. No. CMU-CS-84-157). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

Judah, K., Roy, S., Fern, A., & Dietterich, T. G. (2010). Reinforcement learning via practice and critique advice. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (pp. 481–486). AAAI Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kanerva, P. (1993). Sparse distributed memory and related models . In Hassoun, M. (Ed.), *Associate neural memories: Theory and implementation* (pp. 50–76). New York, NY: Oxford University Press.

Kearns, M. (1998). Near-optimal reinforcement learning in polynomial time . In *Machine learning* (pp. 260–268). Morgan Kaufmann.

Knox, W. B., & Stone, P. (2010). Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of 9th int. Conf. on Autonomous Agents and Multiagent Systems* (pp. 5–12).

lbling, L. P. K., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence, 101*, 99–134.

Lim, C., & Kim, H. (1991). Cmac-based adaptive critic self-lerning control. *IEEE Transactions on Neural Networks*, *2*, 530–533. doi:10.1109/72.134290

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 157–163). Morgan Kaufmann.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *Proceedings of the International Conference on Machine Learning.* ICML.

Morales, E. (2003). Scaling up reinforcement learning with a relational representation. In *Proc. of the Workshop on Adaptability in Multi-Agent Systems* (p. 15-26).

Morales, E., & Sammut, C. (2004). Learning to fly by combining reinforcement learning with behavioural cloning. In *Proc. of the Twenty-First International Conference on Machine Learning* (p. 598-605).

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 278–287). Morgan Kaufmann.

Ng, A. Y., & Russell, S. (2000). Algorithms for inverse reinforcement learning. In *Proc. 17th International Conf. on Machine Learning* (pp. 663–670). Morgan Kaufmann.

Ormoneit, D., & Sen, S. (2002, November-December). Kernel-based reinforcement learning. *Machine Learning*, *49*(2-3), 161–178. doi:10.1023/A:1017928328829

Parr, R., & Russell, S. (1998). Reinforcement learning with hierarchies of machines. [MIT Press.]. *Advances in Neural Information Processing Systems*, *10*, 1043–1049.

Peters, J., Vijayakumar, S., & Schaal, S. (2003a). Reinforcement learning for humanoid robotics. In *Proceedings of the Third IEEE-RAS International Conference on Humanoid Robots* (pp. 1–20).

Peters, J., Vijayakumar, S., & Schaal, S. (2003b, September). Reinforcement learning for humanoid robotics. *Proc. of the Third IEEE-RAS International Conference on Humanoid Robots, Karlsruhe, Germany*, (pp. 29-30).

Poggio, T., & Girosi, F. (1989). A theory of networks for approximation and learning. Laboratory, Massachusetts Institute of Technology, 1140.

Ramachandran, D. (2007). Bayesian inverse reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (pp. 2586–2591).

Ryan, M. (1998). Rl-tops: An architecture for modularity and re-use in reinforcement learning. In *Proc. of the Fifteenth International Conference on Machine Learning* (p. 481-487). San Francisco, CA: Morgan Kaufmann.

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

Singh, S., Jaakkola, T., & Jordan, M. (1996). Reinforcement learning with soft state aggregation . In *Neural Information Processing Systems 7*. Cambridge, MA: MIT Press.

Singh, S., Sutton, R. S., & Kaelbling, P. (1996). Reinforcement learning with replacing eligibility traces. In *Machine learning* (pp. 123–158).

Strens, M. (2000). A Bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 943–950). ICML.

Sunmola, F. T., & Laboratory, W. C. (2006). Model transfer for Markov decision tasks via parameter matching. In *Workshop of the UK Planning and Scheduling Special Interest Group.*

Sutton, R., & Barto, A. (1989). *Reinforcement learning an introduction*. Cambridge, MA: MIT Press.

Sutton, R., Precup, D., & Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211. doi:10.1016/S0004-3702(99)00052-1

Tadepalli, P., & Ok, D. (1998). Model-based average reward reinforcement learning . In *Artificial intelligence* (pp. 881–887). AAAI Press/MIT Press.

Tang, J., Singh, A., Goehausen, N., & Abbeel, P. (2010). Parameterized maneuver learning for autonomous helicopter flight. In *Proc. of the International Conference on Robotics and Automation* (pp. 1142–1148).

Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(1), 1633–1685.

Tenorio-Gonzalez, A. C., Morales, E. F., & Pineda, L. V. (2010). Dynamic reward shaping: Training a robot by voice. In *Proc. of the Ibero-American Conference on Artificial Intelligence* (pp. 483-492).

Tesauro, G. (1992, May). Practical issues in temporal difference learning. *Machine Learning*, *8*, 257–277. doi:10.1007/BF00992697

Theodorou, E., Buchli, J., & Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, *11*, 3137–3181.

Torrey, L., Shavlik, J., Walker, T., & Maclin, R. (2008). Relational macros for transfer in reinforcement learning. *ILP*, 254-268.

van Otterlo, M. (2003). Efficient reinforcement learning using relational aggregation. In *Proc. of the Sixth European Workshop on Reinfocement Learning.* Nancy, France.

van Otterlo, M. (2009). *The logic of adaptive behavior: Knowledge representation and algorithms for adaptive sequential decision making under uncertainty in first-order and relational domains*. The Netherlands: IOS Press.

Watkins, C. (1989). *Learning from delayed rewards*. Unpublished doctoral dissertation. Cambridge, MA: Cambridge University.

Wilson, A., Fern, A., Ray, S., & Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning* (p. 1015). ACM Press.

Zaragoza, J. H. (2010). Relational reinforcement learning with continuous actions by combining behavioral cloning and locally weighted regression. *Journal of Intelligent Learning Systems and Applications*, *2*, 69–79. doi:10.4236/jilsa.2010.22010

Zhang, W., & Dietterich, T. G. (1995). Value function approximations and job-shop scheduling. In *Proceedings of the Workshop on Value Function Approximation,* Carnegie-Mellon University, School of Computer Science (pp. 95–206).

ENDNOTE

1 This can be clearly appreciated by considering the percentage of articles and special workshops in this area in some of the main Machine Learning conferences e.g., *International Conference on Machine Learning (*www.icml2010.org*), Neural Information Processing Systems (nips.cc), European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (*www.ecmlpkdd2010.org*)* and special issues in some well-known journals e.g., *Machine Learning Journal (*www.springer.com/computer/ai/journal/10994*), Journal of Machine Learning Research (*jmlr.csail.mit.edu*), Computational Intelligence (*www.wiley.com/bw/journal.asp?ref=0824-7935*).*