Fast-$Q(\lambda)$ Revisited

Stuart I. Reynolds
School of Computer Science,
The University of Birmingham,
Birmingham, B15 2TT.
England
sir@cs.bham.ac.uk
http://cs.bham.ac.uk/˜sir

Marco A. Wiering
Institute of Information
and Computing Sciences,
Utrecht University,
Padualaan 14, 3508 TB Utrecht.
The Netherlands
marco@cs.uu.nl
http://www.cs.uu.nl/˜marco

Abstract

Fast $Q(\lambda)$ is a model free reinforcement learning technique for precisely implementing $Q(\lambda)$ and other eligibility trace learning algorithms at a hugely reduced computational cost. This report highlights some subtleties in the original description of Fast $Q(\lambda)$ and that are likely to lead to it being incorrectly applied. We propose changes to Fast $Q(\lambda)$, without which the behaviour of the algorithm can be significantly different (and inferior) to $Q(\lambda)$. With these changes the algorithm behaves precisely as $Q(\lambda)$ to a very high degree of precision. We also report on an empirical validation of the algorithm and also provide an exploration insensitive version (an analogue of Watkins' $Q(\lambda)$).

1 Introduction

Q(λ)-learning is an important and widely used reinforcement learning (RL) method. It combines Q-learning (Watkins, 1989; Watkins and Dayan, 1992) and TD(λ) (Sutton, 1988; Tesauro, 1992). Q(λ) is widely used — it is generally believed to outperform simple one-step Q-learning, since it uses *single* experiences to update evaluations of *multiple* state/action pairs (SAPs) that have occurred in the past.

Q(λ) learning is an unnecessarily expensive algorithm. It has a time complexity of $O(|S| \cdot |A|)$ per step for a state space S, and action set A. To improve this, the Fast Q(λ) algorithm implements Q(λ) to a very high degree of precision but at a mean computational cost of $O(|A|)$. However, the original description of Fast Q(λ) (as published in [18, 19, 17]) allows for the algorithm to be misapplied and we highlight two subtle errors that can occur. This paper states these errors and corrects them. In addition, we show how to extend the algorithm for state replacing traces, for off-policy (exploration insensitive) learning and also simplify the algorithm to avoid difficulties with action selection.

Section 2 provides a brief introduction to RL and the Q(λ) algorithm. Section 3 reviews the original description of Fast Q(λ). Section 4 presents the problems, fixes and new extensions to Fast Q(λ). Readers who are already familiar with Fast Q(λ) should skip to Section 4. In Section 5 we test how closely the new and old versions of Fast Q(λ) match standard Q(λ). Section 6 concludes.

2 Q(λ)-Learning

We consider finite Markov decision processes, using discrete time steps $t = 1, 2, 3, \ldots$, a finite set of states $S = \{S_1, S_2, S_3, \ldots, S_n\}$ and a finite set of actions A. The state at time t is denoted by s_t, and $a_t = \Pi(s_t)$ denotes the selected action, where Π represents the learner's policy mapping states to actions. The transition probability to the next state s_{t+1}, given s_t and a_t, is determined by $P_{ij}^a = P(s_{t+1} = j | s_t = i, a_t = a)$ for $i, j \in S$ and $a \in A$. A reward function R maps the SAP $(i, a) \in S \times A$ to scalar reinforcement signals $R(i, a) \in \mathbb{R}$. The reward at time t is denoted by r_t. A discount factor $\gamma \in [0, 1]$ discounts later against immediate rewards. The controller's goal is to select actions which maximise the expected long-term cumulative discounted reinforcement, given an initial state selected according to a probability distribution over possible initial states.

Reinforcement Learning. To achieve this goal, most reinforcement learning methods learn an action evaluation function or Q-function. The optimal Q-value of a SAP (i, a) satisfies

$$Q^*(i, a) = R(i, a) + \gamma \sum_j P_{ij}^a V^*(j), \tag{1}$$

where $V^*(j) = \max_a Q^*(j, a)$. To learn this Q-function, RL algorithms repeatedly do: (1) Select action a_t given state s_t, (2) Collect reward r_t and observe successor state s_{t+1}, (3) Update the Q-function using the latest experience (s_t, a_t, r_t, s_{t+1}).

Q-learning. Given (s_t, a_t, r_t, s_{t+1}), standard one-step Q-learning updates just a single Q-value $Q(s_t, a_t)$ as follows [15]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t) e'_t.$$

Here the temporal difference or TD(0)-error e'_t is given by:

$$e'_t = (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)),$$

where the value function $V(s)$ is defined as $V(s) = \max_a Q(s, a)$, and $\alpha_k(s_t, a_t)$ is the learning rate for the k^{th} update of SAP (s_t, a_t).

Learning rate adaptation. The learning rate $\alpha_k(s, a)$ for the k^{th} update of SAP (s, a) should decrease over time to satisfy two conditions for stochastic iterative algorithms (Watkins and Dayan, 1992; Bertsekas and Tsitsiklis, 1996, Sutton and Barto, 1998):

1. $\sum_{k=1}^{\infty} \alpha_k(s, a) = \infty$, and
2. $\sum_{k=1}^{\infty} \alpha_k^2(s, a) < \infty$.

The first condition ensures that the updates are large enough to overcome any initial biases or random fluctuations. The second condition ensures that the updates become small enough to ensure convergence. They hold for $\alpha_k(s, a) = 1/k^{\beta}$, where $1/2 < \beta \leq 1$.

Q(λ)-learning. Q(λ) uses TD(λ)-methods (Sutton, 1988) to accelerate Q-learning. First note that Q-learning's update at time $t + 1$ may change $V(s_{t+1})$ in the definition of e'_t. Following Peng and Williams (1996) we define the TD(0)-error of $V(s_{t+1})$ as

$$e_{t+1} = (r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})). \tag{2}$$

Q(λ) uses a factor $\lambda \in [0, 1]$ to discount TD-errors of future time steps:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t) e_t^{\lambda}, \tag{3}$$

where the TD(λ)-error e_t^λ is defined as,

$$e_t^\lambda = e_t' + \sum_{i=1}^{T} (\gamma\lambda)^i e_{t+i}, \tag{4}$$

where T denotes the time at which a terminal state in entered. T may be infinity if there are no terminal states.

Eligibility traces. The updates above cannot be made as long as the TD errors of future time steps are not known. We can compute them incrementally, however, by using eligibility traces (Barto et al., 1983; Sutton 1988). From update rule (3) and definition (4), and assuming a constant α for simplicity, the Q-function at time k is given by,

$$Q_k(s,a) \quad = \quad Q_0(s,a) + \alpha \sum_{t=1}^{k} e_t^\lambda \eta_t(s,a). \tag{5}$$

In what follows, $\eta_t(s,a)$ will denote the indicator function which returns 1 if (s,a) occurred at time t, and 0 otherwise. Omitting the learning rate for simplicity, the increment of $Q(s,a)$ for the complete trial is:

$$\begin{aligned}
\Delta Q(s,a) &= Q_T(s,a) - Q_0(s,a) \\
\frac{\Delta Q(s,a)}{\alpha} &= \sum_{t=1}^{T} e_t^\lambda \eta_t(s,a) \\
&= \sum_{t=1}^{T} \left[e_t' \eta_t(s,a) + \sum_{i=t+1}^{T} (\gamma\lambda)^{i-t} e_i \eta_t(s,a) \right] \\
&= \sum_{t=1}^{T} \left[e_t' \eta_t(s,a) + \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i} e_t \eta_i(s,a) \right] \\
&= \sum_{t=1}^{T} \left[e_t' \eta_t(s,a) + e_t \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i} \eta_i(s,a) \right]. \qquad (6)
\end{aligned}$$

To simplify this we use an eligibility trace $l_t(s,a)$ for each SAP (s,a):

$$l_t(s,a) = \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i} \eta_i(s,a). \tag{7}$$

Then the online update at time $t+1$ becomes:

$$\forall (s,a) \in S \times A \;\; do: \;\; Q(s,a) \leftarrow Q(s,a) + \alpha_k(s_t,a_t) \left[e_t' \eta_t(s,a) + e_t l_t(s,a) \right]. \tag{8}$$

Online Q(λ). We will focus on Peng and Williams' algorithm (PW) (1996), although there are other possible variants, e.g, Rummery and Niranjan's SARSA (1994). PW uses a list H of SAPs that have occurred at least once. SAPs with eligibility traces below $\epsilon_m \geq 0$ are removed from H. Boolean variables *visited*(s,a) are used to make sure no two SAPs in H are identical.

```
PW's Q($\lambda$)-update($s_t, a_t, r_t, s_{t+1}$) :
1) $e'_t \leftarrow (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$
2) $e_t \leftarrow (r_t + \gamma V(s_{t+1}) - V(s_t))$
3) For each SAP $(s, a) \in H$ Do :
    3a) $l(s, a) \leftarrow \gamma\lambda l(s, a)$
    3b) $Q(s, a) \leftarrow Q(s, a) + \alpha_k(s_t, a_t) e_t l(s, a)$
    3c) If $(l(s, a) < \epsilon_m)$
        3c-1) $H \leftarrow H \setminus (s, a)$
        3c-2) $visited(s, a) \leftarrow 0$
4) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t) e'_t$
5) $l(s_t, a_t) \leftarrow l(s_t, a_t) + 1$
6) If $(visited(s_t, a_t) = 0)$
    6a) $visited(s_t, a_t) \leftarrow 1$
    6b) $H \leftarrow H \cup (s_t, a_t)$
```

Comments.

1. The SARSA algorithm (Rummery and Niranjan, 1994) replaces the right hand side in lines (1) and (2) by $(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$.

2. For state replacing eligibility traces (Singh and Sutton, 1996), step 5 should be: $\forall a: \ l(s_t, a) \leftarrow 0$; $l(s_t, a_t) \leftarrow 1$.

3. Representing H by a doubly linked list and using direct pointers from each SAP to its position in H, the functions operating on H (deleting and adding elements — see lines (3c-1) and (6b)) cost $O(1)$.

Complexity. Deleting SAPs from H (step 3c-1) once their traces fall below a certain threshold may significantly speed up the algorithm. If $\gamma\lambda$ is sufficiently small, then this will keep the number of updates per time step manageable. For large $\gamma\lambda$, PW does not work that well: it needs a sweep (sequence of SAP updates) after each time step, and the update cost for such sweeps grows with $\gamma\lambda$. Let us consider worst-case behaviour, which means that each SAP occurs just once (if SAPs reoccur then the history list will grow at a slower rate). In the beginning of the episode the number of updates increases linearly until at some time step t some SAPs get deleted from H. This will happen as soon as $t \geq \log \epsilon_m / \log(\gamma\lambda)$. Since the number of updates is bounded from above by the number of SAPs, the total update complexity increases towards $O(|S| \cdot |A|)$ per update for $\gamma\lambda \to 1$.

The space complexity of the algorithm is $O(|S| \cdot |A|)$. We need to store for all SAPs: Q-values, eligibility traces, the "visited" bit variable and three pointers for managing the history list (one from the SAP to its place in the history list, and two for the doubly linked list).

3 Fast Q(λ)-Learning

Fast Q(λ) is intended a fully online implementation of Q(λ) but with a time complexity $O(|A|)$ per update. The algorithm is designed for $\lambda\gamma > 0$ — otherwise we can use simple Q-learning.

Main principle. The algorithm is based on the observation that the only Q-values needed at any given time are those for the possible actions given the current state. Hence, using "lazy learning", we can postpone updating Q-values until they are needed. Suppose some SAP (s, a) occurs at steps $t_1, t_2, t_3, \ldots$. Let us abbreviate $\eta_t = \eta_t(s, a)$,

$\phi = \gamma\lambda$. First we unfold terms of expression (6):

$$\begin{aligned}\sum_{t=1}^{T}\left[e'_t\eta_t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta_i\right] &= \sum_{t=1}^{t_1}\left[e'_t\eta_t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta_i\right] + \\ &\quad \sum_{t=t_1+1}^{t_2}\left[e'_t\eta_t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta_i\right] + \\ &\quad \sum_{t=t_2+1}^{t_3}\left[e'_t\eta_t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta_i\right] + \ldots\end{aligned}$$

Since η_t is 1 only for $t = t_1, t_2, t_3, \ldots$ and 0 otherwise, we can rewrite this as

$$\begin{aligned}& e'_{t_1} + e'_{t_2} + \sum_{t=t_1+1}^{t_2} e_t\phi^{t-t_1} + e'_{t_3} + \sum_{t=t_2+1}^{t_3} e_t\left(\phi^{t-t_1} + \phi^{t-t_2}\right) + \ldots = \\ & e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}\sum_{t=t_1+1}^{t_2} e_t\phi^{t} + e'_{t_3} + \left(\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}}\right)\sum_{t=t_2+1}^{t_3} e_t\phi^{t} + \ldots = \\ & e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}\left(\sum_{t=1}^{t_2} e_t\phi^t - \sum_{t=1}^{t_1} e_t\phi^t\right) + e'_{t_3} + \left(\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}}\right)\left(\sum_{t=1}^{t_3} e_t\phi^t - \sum_{t=1}^{t_2} e_t\phi^t\right) + \ldots\end{aligned}$$

Defining $\Delta_t = \sum_{i=1}^{t} e_i\phi^i$, this becomes

$$e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}(\Delta_{t_2} - \Delta_{t_1}) + e'_{t_3} + \left(\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}}\right)(\Delta_{t_3} - \Delta_{t_2}) + \ldots \tag{9}$$

This will allow for constructing an efficient online Q(λ) algorithm. We define a local trace $l'_t(s,a) = \sum_{i=1}^{t}\frac{\eta_i(s,a)}{\phi^i}$, and use (9) to write down the total update of $Q(s,a)$ during an episode:

$$\Delta Q(s,a) = \sum_{t=1}^{T}\left[e'_t\eta_t(s,a) + l'_t(s,a)(\Delta_{t+1} - \Delta_t)\right]. \tag{10}$$

To exploit this we introduce a global variable Δ keeping track of the cumulative TD(λ) error since the start of the episode. As long as SAP (s,a) does not occur we postpone updating $Q(s,a)$. In the update below we need to subtract that part of Δ which has already been used (see equations 9 and 10). We use for each SAP (s,a) a local variable $\delta(s,a)$ which records the value of Δ at the moment of the last update, and a local trace variable $l'(s,a)$. Then, once $Q(s,a)$ needs to be known, we update $Q(s,a)$ by adding $l'(s,a)(\Delta - \delta(s,a))$. Figure 1 illustrates that the algorithm substitutes the varying eligibility trace $l(s,a)$ by multiplying a global trace ϕ^t by the local trace $l'(s,a)$. The value of ϕ^t changes all the time, but $l'(s,a)$ does not in intervals during which (s,a) does not occur.

Algorithm overview. The algorithm relies on two procedures: the *Local Update* procedure calculates exact Q-values once they are required; the *Global Update* procedure updates the global variables and the current Q-value. Initially we set the global variables $\phi^0 \leftarrow 1.0$ and $\Delta \leftarrow 0$. We also initialise the local variables $\delta(s,a) \leftarrow 0$ and $l'(s,a) \leftarrow 0$ for all SAPs.

Local updates. Q-values for all actions possible in a given state are updated before an action is selected and before a particular V-value is calculated. For each SAP (s,a) a variable $\delta(s,a)$ tracks changes since the last update:

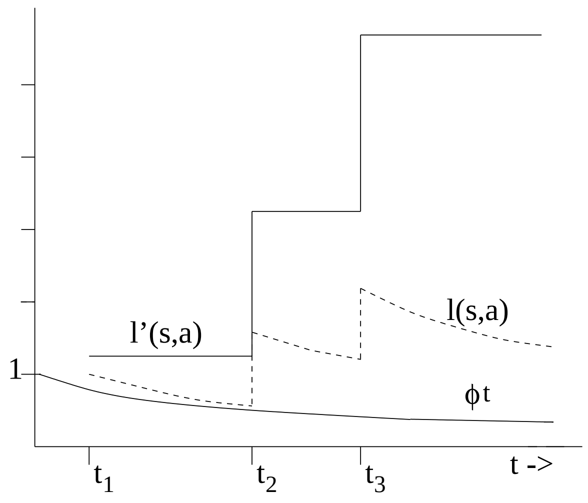

Figure 1: *SAP (s, a) occurs at times $t_1, t_2, t_3, \ldots$. The standard eligibility trace $l(s, a)$ equals the product of ϕ^t and $l'(s, a)$.*

```
**Local Update**($s_t, a_t$) :
1) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)(\Delta - \delta(s_t, a_t))l'(s_t, a_t)$
2) $\delta(s_t, a_t) \leftarrow \Delta$
```

The global update procedure. After each executed action we invoke the procedure *Global Update*, which consists of three basic steps: (1) To calculate $V(s_{t+1})$ (which may have changed due to the most recent experience), it calls *Local Update* for the possible next SAPs. (2) It updates the global variables ϕ^t and Δ. (3) It updates the Q-value and trace variable of (s_t, a_t) and stores the current Δ value (in *Local Update*).

```
**Global Update**($s_t, a_t, r_t, s_{t+1}$) :
1)$\forall a \in A$ Do                                  *Make $V(s_{t+1})$ up-to-date*
    1a) *Local Update*($s_{t+1}, a$)
2) $e'_t \leftarrow (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$
3) $e_t \leftarrow (r_t + \gamma V(s_{t+1}) - V(s_t))$
4) $\phi^t \leftarrow \gamma\lambda\phi^{t-1}$                        *Update global clock*
5) $\Delta \leftarrow \Delta + e_t\phi^t$                         *Add new TD-error to global error*
6) *Local Update*($s_t, a_t$)                        *Make $Q(s_t, a_t)$ up-to-date for next step*
7) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)e'_t$
8) $l'(s_t, a_t) \leftarrow l'(s_t, a_t) + 1/\phi^t$           *Decay Trace*
```

For state replacing eligibility traces (Singh and Sutton, 1996), step 8 should be changed as follows: $\forall a: \; l'(s_t, a) \leftarrow 0; \, l'(s_t, a_t) \leftarrow 1/\phi^t$.

Machine precision problem and solution. Adding $e_t\phi^t$ to Δ in line 5 may create a problem due to limited machine precision: for large absolute values of Δ and small ϕ^t there may be significant rounding errors. More importantly, line 8 will quickly overflow any machine for $\gamma\lambda < 1$. The following addendum to the procedure *Global Update* detects when ϕ^t falls below machine precision ϵ_m, updates all SAPs which have occurred (again we make use of a list H), and removes SAPs with $l'(s, a) < \epsilon_m$ from H. Finally, Δ and ϕ^t are reset to their initial values.

```
Global Update : addendum
9) If ($visited(s_t, a_t) = 0$)
    9a) $H \leftarrow H \cup (s_t, a_t)$
    9b) $visited(s_t, a_t) \leftarrow 1$
10) If ($\phi^t < \epsilon_m$)
    10a) $\forall (s, a) \in H$ Do
        10a-1) $Local\ Update(s, a)$
        10a-2) $l'(s, a) \leftarrow l'(s, a)\phi^t$
        10a-3) If ($l'(s, a) < \epsilon_m$)
            10a-3-1) $H \leftarrow H \setminus (s, a)$
            10a-3-2) $visited(s, a) \leftarrow 0$
        10a-4) $\delta(s, a) \leftarrow 0$
    10b) $\Delta \leftarrow 0$
    10c) $\phi^t \leftarrow 1.0$
```

Comments. Recall that *Local Update* sets $\delta(s, a) \leftarrow \Delta$, and update steps depend on $\Delta - \delta(s, a)$. Thus, after having updated all SAPs in H, we can set $\Delta \leftarrow 0$ and $\delta(s, a) \leftarrow 0$. Furthermore, we can simply set $l'(s, a) \leftarrow l'(s, a)\phi^t$ and $\phi^t \leftarrow 1.0$ without affecting the expression $l'(s, a)\phi^t$ used in future updates — this just rescales the variables. Note that if $\gamma\lambda = 1$, then no sweeps through the history list will be necessary.

Complexity. The algorithm's most expensive part is the set of calls of *Local Update*, whose total cost is $O(|A|)$. This is not bad: even simple Q-learning's action selection procedure costs $O(|A|)$ if, say, the Boltzmann rule (Thrun, 1992; Caironi and Dorigo, 1994) is used. Concerning the occasional complete sweep through SAPs still in history list H: during each sweep the traces of SAPs in H are multiplied by ϕ^t. SAPs are deleted from H once their trace falls below ϵ_m. In the worst case one sweep per n time steps updates $2n$ SAPs and costs O(1) on average. This means that there is an additional computational burden at certain time steps, but since this happens infrequently our method's total average update complexity stays $O(|A|)$.

The space complexity of the algorithm remains $O(|S| \cdot |A|)$. We need to store the following variables for all SAPs: Q-values, eligibility traces, previous delta values, the "visited" bit, and three pointers to manage the history list (one from each SAP to its place in the history list, and two for the doubly linked list). Finally we need to store the two global variables.

4 Revisions to Fast $Q(\lambda)$

In this section we show how the original version of Fast $Q(\lambda)$ is likely to be misapplied to give rise to two subtle errors. This section also introduces: *i*) what modifications, if any, are required of action selection mechanisms that intend to employ up-to-date Q-function, *ii*) the state-action replace version of Fast $Q(\lambda)$, and, *iii*) how the algorithm may be modified for off-policy learning (as Watkins' $Q(\lambda)$) [15, 12]. The new algorithms are shown in Figure 2.

Error 1. Step 1 of the original *Global Update* procedure performs the updates to the Q-values at s_{t+1} necessary to ensure that $V(s_{t+1})$ is an up-to-date estimate before steps 2 and 3 where they are used. However, $Q(s_t, a_t)$ and $V(s_t)$ are also used in steps 2 and 3 and may not be up-to-date. This is easily corrected by adding:

```
    1b) $Local\ Update(s_t, a)$
```

We shall see below that this change is not necessary if $Q(s_t, \cdot)$ is made up-to-date at the end of the *Global Update* procedure.

Error 2. When state replacing traces are employed with the original Fast $Q(\lambda)$ algorithm, it is possible that the eligibility of some SAPs are zeroed. In such a case, if these SAPs previously had non zero eligibilities then they will not receive any update making use of e_t. An exception is $Q(s_t, a_t)$, which is made up-to-date in step 6 (and so makes use of e_t). However all other SAPs at s_t with non-zero eligibilities will receive no adjustment toward e_t if their eligibilities are zeroed:

```
From the original version of Global Update:
...
3) $e_t \leftarrow (r_t + \gamma V(s_{t+1}) - V(s_t))$
...
Here, each $a \neq a_t$ with non-zero traces receive no update using $e_t$
($Q(s_t, a_t)$ is already up-to-date before this point)
8) $\forall a:\ l'(s_t, a) \leftarrow 0;\ l'(s_t, a_t) \leftarrow 1/\phi^t$.
```

To avoid this in the revised algorithm, all of the Q-values at s_t are made up-to-date before zeroing their eligibility traces (step 8*a*).

Action Selection. Steps 9 and 9*a* of the *Revised Global Update* procedure are a pragmatic change to ensure that all of the Q-values for s_{t+1} are up-to-date by the end of the procedure. If this were not so then any code needing to make use of the up-to-date Q-function at s_{t+1}, such as those for selecting the agent's next action, would need to be defined in terms of the up-to-date, Q-function instead. The up-to-date function, Q^+, is given by:

$$Q^+(s,a) \quad = \quad Q(s,a) + \alpha_k(s,a)(\Delta - \delta(s,a))l'(s_t, a_t) \tag{11}$$

From an implementation standpoint, these changes are desirable for at least three reasons. Firstly, the need to use Q^+ for action selection is easy to overlook when implementing the original version of Fast $Q(\lambda)$ as part of a larger learning agent. Secondly, it reduces coupling between algorithms; an algorithm that implements action selection based on the up-to-date Q-values of s_{t+1} does not need to use Q^+ or even care that values at different states may be out-of-date. Thirdly, it reduces the duplication of code; we are likely to already have action-selection algorithms that use $Q(s_{t+1}, \cdot)$ and so we don't need to implement others that use $Q^+(s_{t+1}, \cdot)$ instead.

The original description of Fast $Q(\lambda)$ assumed that the *Local Update* procedure was called for all actions in the current state immediately after the Global Update procedure and prior to selecting actions. However, from the original description, it was not clear that this *still* needs to be done even if the Q-values at the current state are not used by the action selection method (for example, if the actions for selected randomly or provided by a trainer). If this is done, then the new and revised algorithms are essentially identical.

State-Action Replacing Traces. The descriptions of $Q(\lambda)$ and Fast $Q(\lambda)$ in Sections 2 and 3 include only the accumulating and state replacing trace variants. In addition to these, Singh and Sutton describe a third *state-action* replacing trace variant. This is similar to the state replacing trace except that the eligibilities of the actions not followed are not zeroed but decayed as in the accumulating trace case [4]:

$$l_{t+1}(s,a) = \begin{cases} 1, & \text{if } s = s_t \text{ and } a = a_t, \\ \gamma\lambda l_t(s,a), & \text{otherwise.} \end{cases} \tag{12}$$

For Fast $Q(\lambda)$, an effect equivalent to setting an eligibility to 1 is achieved by $l'_{t+1}(s,a) \leftarrow 1/\phi^t$. We include this variant only for completeness. See [12] for a discussion of the properties of the different variants.

```
For accumulating traces:
Revised Global Update($s_t, a_t, r_t, s_{t+1}$) :
1)$\forall a \in A$ Do
    1a) Local Update($s_{t+1}, a$)
2) $e'_t \leftarrow (r_t + \gamma V(s_{t+1}) - Q(s_t, a_t))$  NB. $s_t$ was made up-to-date in step 9
3) $e_t \leftarrow (r_t + \gamma V(s_{t+1}) - V(s_t))$
4) $\phi^t \leftarrow \gamma\lambda\phi^{t-1}$
5) $\Delta \leftarrow \Delta + e_t\phi^t$
6) Local Update($s_t, a_t$)
7) $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t)e'_t$
8) $l'(s_t, a_t) \leftarrow l'(s_t, a_t) + 1/\phi^t$         Increment eligibility
9) $\forall a \in A$ Do
    9a) Local Update($s_{t+1}, a$)            Make $Q(s_{t+1}, \cdot)$ up-to-date before action selection
```

For state-action replacing traces replace step 8 with:

```
8) $l'(s_t, a_t) \leftarrow 1/\phi^t$                    Set eligibility to 1
```

For state replacing traces, replace steps 8 - 9a with:

```
8) $\forall a \in A$ Do
    8a) Local Update($s_t, a$)               Make $Q(s_t, \cdot)$ up-to-date before zeroing eligibility
    8b) $l'(s_t, a) \leftarrow 0$                   Zero eligibility
    8c) Local Update($s_{t+1}, a$)             Make $Q(s_{t+1}, \cdot)$ up-to-date before action-selection
9) $l'(s_t, a_t) \leftarrow 1/\phi^t$                    Set eligibility to 1
```

For Watkins $Q(\lambda)$ prepend the following to the *Revised Global Update* procedure.

```
0) if $Q(s_t, a_t) < V(s_t)$                 Test whether a non-greedy action was taken
    0a) Flush Updates()
```

```
Flush Updates()
1)  $\forall (s, a) \in H$ Do
2)      $Q(s, a) \leftarrow Q(s, a) + \alpha_k(s_t, a_t)(\Delta - \delta(s, a))l'(s, a)$
3)      $\delta(s, a) \leftarrow 0$
4)      $l'(s, a) \leftarrow 0$
5)  $H \leftarrow \{\}$
6)  $\Delta \leftarrow 0$
7)  $\phi^t \leftarrow 1$
```

Figure 2: The revised Fast $Q(\lambda)$ algorithm for accumulating, state replacing and state-action replacing traces and for Watkins' $Q(\lambda)$. The machine precision addendum should be appended to each algorithm. The *Flush Updates* procedure can also be called upon entering a terminal state to make the entire Q-function up-to-date and also reinitialise the eligibility and error values of each SAP ready for learning in the next episode.

Watkins' $Q(\lambda)$. $Q(\lambda)$ was originally conceived by Watkins and appeared in his thesis [15]. Like Q-learning, Watkins' $Q(\lambda)$ is an *off-policy* method; it can learn from the return obtainable under one policy (the greedy policy) while, in practice, almost any policy can be used to generate experience. It achieves this by ignoring the observed return that follows from a non-greedy action in updates to the states visited prior to that action. In an eligibility trace method the credit following a non-greedy action can be removed from the error signal by setting the eligibilities of all SAPs to zero after taking that action. The new Fast $Q(\lambda)$ version works in the same way except that here we must ensure that all non-up-to-date SAPs are updated before zeroing their traces (see the *Flush Updates* procedure).

Frequently zeroing the trace will mean that credit for receiving a particular reward will immediately be assigned only to the few recent SAPs since the last non-greedy action. A recent discussion of methods to overcome this inefficiency can be found in [6].

Unlike Watkins' $Q(\lambda)$, Peng and Williams' $Q(\lambda)$ is not off-policy. The Q-function that it will converge upon (if it converges at all) will be biased by the distribution of experience; it can be shown that if the method converges then the final Q-function may be different from Q^* if non-greedy actions are continually taken.[1] However, learning an accurate Q-function is different from learning a good policy. Because Peng and Williams' $Q(\lambda)$ does not zero its trace, each experience may immediately affect many more SAPs than Watkins' $Q(\lambda)$. Thus, in many instances it may learn more quickly.

5 Validation

In this section we empirically test how closely the correct and erroneous implementations of Fast $Q(\lambda)$ approximate the original version of $Q(\lambda)$. We use Fast $Q(\lambda)^+$ to denote the correct implementation suggested in this paper and Fast $Q(\lambda)^-$ to denote the method that does not apply a Local Update for all actions in the new state between calls to the Global Update procedure. Note that if these updates are performed, Fast $Q(\lambda)^+$ and Fast $Q(\lambda)^-$ are identical methods.[2] We analyse what consequences the erroneous implementation has on the learning ability of the RL agent.

The algorithms were tested using the maze task shown in Figure 3. At each step the agent may choose one of four actions (N,S,E,W). Transitions have probabilities of 0.8 of succeeding, 0.08 of moving the agent laterally and 0.04 of moving in the opposite to intended direction. Episodes start in random states and continue until one of the four terminal corner states is entered. A reward of 100 is given for entering the top-right corner and 10 for the others. There are a number of penalty fields of -1 and -4 around the maze. This task was chosen as credit for actions leading to the goal can be significantly delayed and also because state revisits can frequently occur.

The action taken by the agent at each step was selected using ϵ-greedy [12]. This selects a greedy action, $\arg\max_a Q(s_t, a)$, with probability ϵ, and a random action with $1-\epsilon$. For Fast $Q(\lambda)^-$ the greedy action is chosen based upon the up-to-date Q-function: $\arg\max_a Q^+(s_t, a)$.

Figure 5 compares the results for the PW Q(λ) variants. The graphs measure the total reward collected by each algorithm and the mean squared error (MSE) in the up-to-date Q-function learned by each algorithm over the course of 200000 time steps. The squared

[1]There is, as yet, no proof that any RL method with $\lambda > 0$ converges upon Q^* [12, 9].

[2]The experiments in the original description of Fast $Q(\lambda)$ did perform these local updates and so we do not repeat the experiments in the original paper in order to compare the time-cost of Fast $Q(\lambda)^+$ and Peng's $Q(\lambda)$ [18, 19, 17].

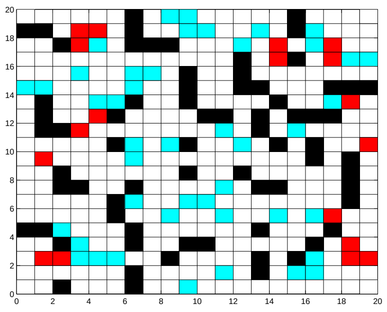

Figure 3: The large stochastic maze task. Impassable walls are marked in black and penalty fields of -4 and -1 are marked in dark and light grey respectively.

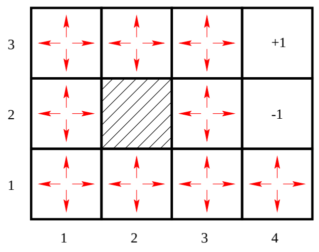

Figure 4: A small stochastic maze task taken from [8]. Rewards of -1 and $+1$ are given for entering $(4, 2)$ and $(4, 3)$, respectively. On non-terminal transitions, $r_t = 1/25$.

	Fast $Q(\lambda)^-$	**Fast** $Q(\lambda)^+$
PW-acc	0.7	$1.7 * 10^{-15}$
PW-srepl	1.3	$8.8 * 10^{-16}$
PW-sarepl	0.3	$1.7 * 10^{-15}$
WAT-acc	1.3	$7.6 * 10^{-13}$
WAT-srepl	2.5	$4.2 * 10^{-10}$
WAT-sarepl	0.6	$2.9 * 10^{-11}$

Table 1: The largest differences from Q-function learned by original $Q(\lambda)$ during the course of 2000 time steps of experience within the small maze task. The experiment parameters were $\epsilon_m = 10^{-9}$, $\alpha = 0.2$, $\lambda = 0.95$ and $\gamma = 1.0$. The experience was generated by randomly selecting actions.

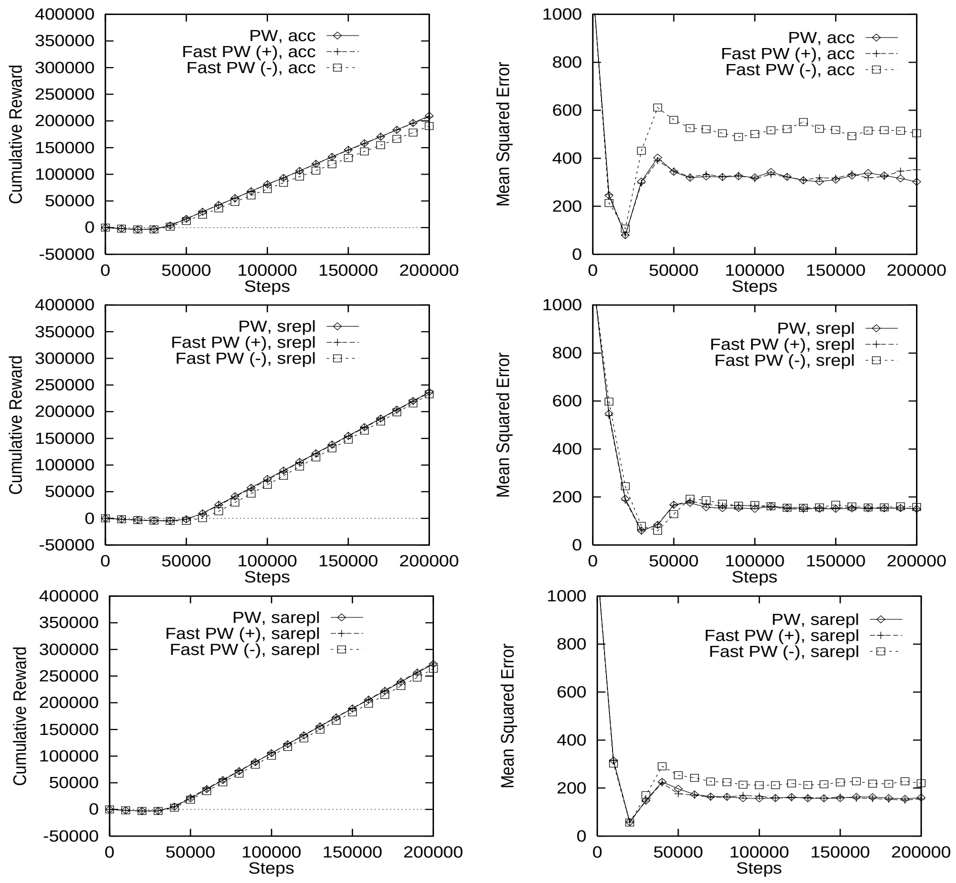

Figure 5: Comparison of PW $Q(\lambda)$, Fast PW $Q(\lambda)^+$ and Fast PW $Q(\lambda)^-$ performance profiles in the stochastic maze task. Results are the average of 20 runs. The parameters were $Q_0 = 100$, $\alpha = 0.3$, $\epsilon = 0.1$, $\lambda = 0.9$ and $\epsilon_m = 1 * 10^{-3}$ for regular $Q(\lambda)$ and $\epsilon_m = 10^{-10}$ for the Fast versions. *(left column)* Total reward collected. *(right column)* Mean squared error in the value function. *(top row)* With accumulating traces. *(middle row)* With state replacing traces. *(bottom row)* With state-action replacing traces.

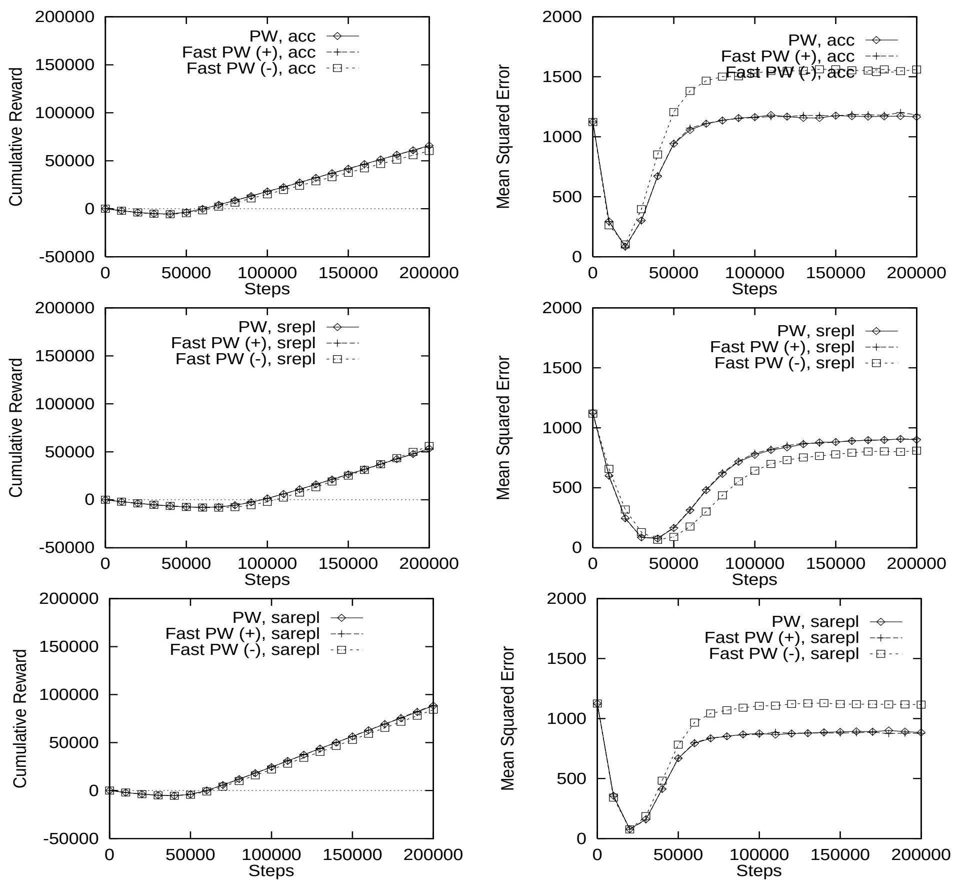

Figure 6: Comparison of Peng and Williams' $Q(\lambda)$ methods with a high exploration rate ($\epsilon = 0.5$). All other parameters are as in Figure 5. Note that the scale of the vertical axes differs between experiment sets.

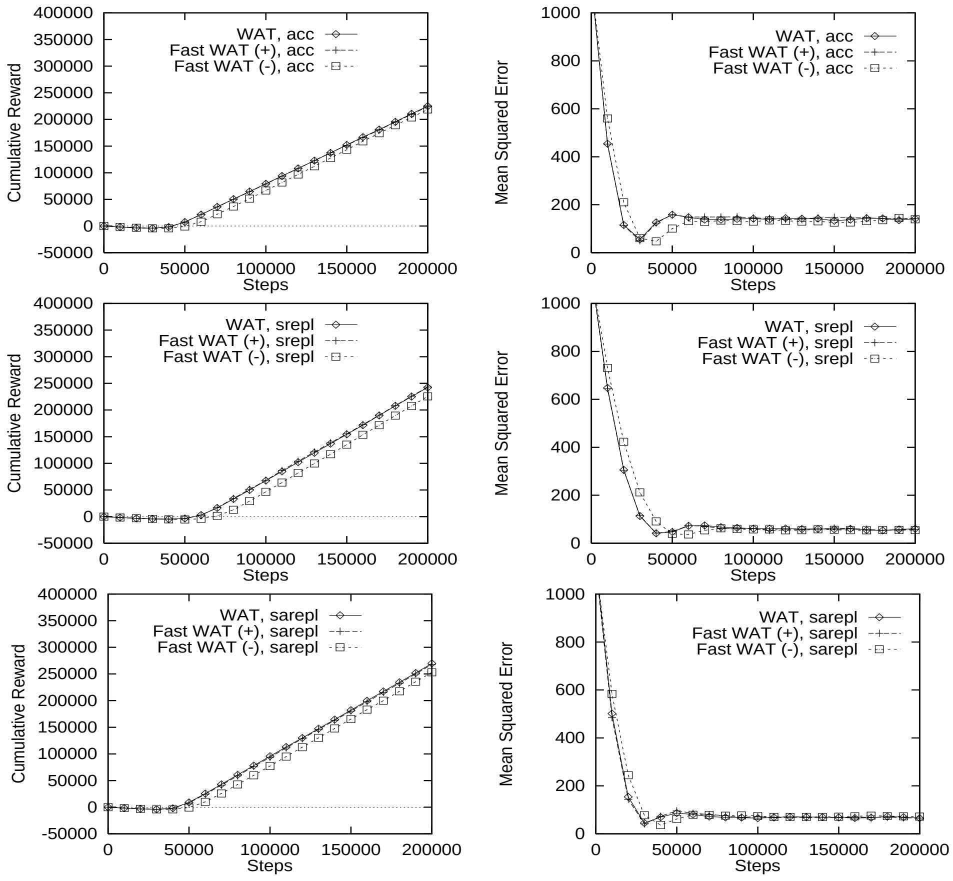

Figure 7: Comparison of Watkins' $Q(\lambda)$, Fast Watkins' $Q(\lambda)^{-}$ and Revised Fast Watkins' $Q(\lambda)^{+}$ in the stochastic maze task. All parameters are as in Figure 5.

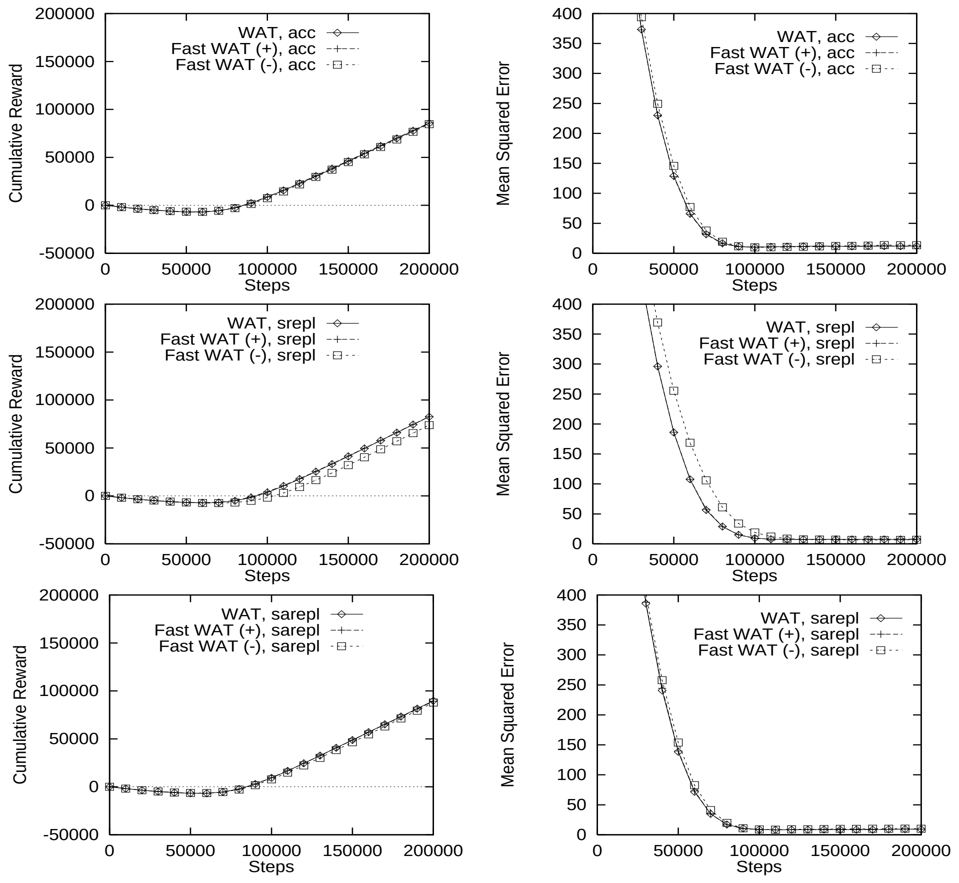

Figure 8: Comparison of Watkins' $Q(\lambda)$ methods with a high exploration rate ($\epsilon = 0.5$). All other parameters are as in Figure 5.

error was measured as,

$$SE(s) \quad = \quad \left(V^*(s) - \max_a Q(s,a)\right)^2, \tag{13}$$

for regular $Q(\lambda)$ and as,

$$SE(s) \quad = \quad \left(V^*(s) - \max_a Q^+(s,a)\right)^2, \tag{14}$$

for both versions of Fast $Q(\lambda)$. An accurate V^* was found by dynamic programming methods. All of the results in the graphs are the average of 100 runs.

Fast PW $Q(\lambda)^+$ provided equal or better performance than Fast PW $Q(\lambda)^-$ in most instances, and its results also provided an extremely good fit against the original version of PW $Q(\lambda)$ in all cases (see Figures 5 and 6). Similar results were found when comparing Watkins' $Q(\lambda)$ and its Fast variants (see Figures 7 and 8).

Fast $Q(\lambda)^-$ worked especially worse in terms of error than Fast $Q(\lambda)^+$ for PW with accumulating or state-action replacing traces. However, in one instance (with a state replacing trace) the error performance of the revised algorithm was actually worse than the original (see Figure 6).[3] This anomaly was not seen for Watkins' $Q(\lambda)$ (see Figure 8).

The effect of exploratory actions on PW $Q(\lambda)$ are also evident in these results. The PW $Q(\lambda)$ methods collected less reward and found a hugely less accurate Q-function in the case of a high exploration rate than the Watkins' methods (compare Figures 6 and 8). In contrast, the Watkins' variants collected similar or better amounts of reward but found far more accurate Q-functions than the Peng and Williams' methods in both the high and low exploration rate cases. Similar results were also reported by Wyatt in [20].

In addition to showing that the performance of Fast $Q(\lambda)^+$ is similar to $Q(\lambda)$ in the mean, we performed a more detailed test. The agents were made to learn from identical experience gathered over 2000 simulation steps in the small stochastic maze shown in Figure 4. At each time step, the difference between the Q-functions of $Q(\lambda)$ and the up-to-date Q-functions of Fast $Q(\lambda)^+$ and Fast $Q(\lambda)^-$ was measured. The largest differences at any time during the course of learning are shown in Table 1. The differences for Fast $Q(\lambda)^+$ are all in the order of ϵ_m or better. The differences for Fast $Q(\lambda)^-$ are many orders of magnitude greater.

Also, overall, state replacing traces produced best performance in terms of Q-function error in both sets of experiments. State replacing traces are the variants suggested by Singh and Sutton [10] and in the original description of Fast $Q(\lambda)$ [18, 19, 17].

6 Conclusion

Fast Q(λ) provided the means to implement Q(λ) at a greatly reduced computational cost that is independent of the size of the state space. As such, it makes it feasible for RL to tackle problems of greater scale. Although the underlying derivation of Fast Q(λ) is correct, we have shown in this paper that the original algorithmic description is likely to be misinterpreted and incorrectly implemented. This paper has provided both simplifications and clarifications of the original algorithm. The revised algorithm maintains a mean time complexity of $O(|A|)$ per step (as Fast Q(λ)). Naive implementations of Q(λ) are $O(|S| \cdot |A|)$ per step.

[3]An interpretation of this result might be that the Peng and Williams' Fast $Q(\lambda)^-$ method isn't performing as bad as quickly as the other methods since the error is actually increasing in all methods until the end of this plot.

We have also shown how Fast $Q(\lambda)$ can be modified to use state-action replacing traces or used as an exploration insensitive learning method and reported upon the relative merits of these modifications. In particular, in the experiments conducted here, the exploration insensitive versions provided similar performance in terms of the collected reward, but achieved uniformly better performance in terms of Q-function error. This was found both with high or low amounts of exploration.

References

[1] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):834–846, Septemeber 1983.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neurodynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[3] P. V. C. Caironi and M. Dorigio. Training Q agents. Technical Report IRIDIA-94-14, Université Libre de Bruxelles, 1994.

[4] Pawel Cichosz. A forwards view of replacing eligibility traces for states and state-action pairs. *Mathematical Algorithms*, 1:283–297, 2000.

[5] J. Peng and R. J. Williams. Technical note: Incremental Q-learning. *Machine Learning*, 22:283–290, 1996.

[6] Stuart I Reynolds. Experience stack reinforcment learning for off-policy control. Technical Report CSRP-02-1, University of Birmingham, School of Computer Science, January 2002. ftp://ftp.cs.bham.ac.uk/pub/tech-reports/2002/CSRP-02-01.ps.gz.

[7] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge Univeristy Engineering Department, September 1994.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, London, UK, 1995.

[9] Satinder Singh. Personal communication, 2001.

[10] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[11] Richard S. Sutton. Learning to predict by methods of temporal difference. *Machine Learning*, 3:9–44, 1988.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, MA., 1998.

[13] G. J. Tesauro. Practical issues in temporal differences learning. In *Advances in Neural Information Processing Systems* 4, pages 259–266, San Mateo, CA, 1992. Morgan Kaufamann.

[14] S. Thrun. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, PA, 1992.

[15] C.J.C.H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, May 1989.

[16] C.J.C.H. Watkins and P. Dayan. Technical note: Q-Learning. *Machine Learning*, 8:279–292, 1992.

[17] Marco Wiering. *Explorations in Efficient Reinforcement Learning.* PhD thesis, Universiteit van Amsterdam, The Netherlands, February 1999.

[18] Marco Wiering and Jürgen Schmidhuber. Fast online $Q(\lambda)$. *Machine Learning*, 33(1):105–115, 1998.

[19] Marco Wiering and Jürgen Schmidhuber. Speeding up $Q(\lambda)$-Learning. In *Proceedings of the Tenth European Conference on Machine Learning (ECML'98)*, 1998.

[20] Jeremy Wyatt, Gillian Hayes, and John Hallam. Investigating the behaviour of $Q(\lambda)$. In *In* Colloquium on Self-Learning Robots, IEE, London, February 1996.