Speeding up Q(λ)-learning

Marco Wiering and Jürgen Schmidhuber

IDSIA, Corso Elvezia 36
CH-6900-Lugano, Switzerland

Abstract. Q(λ)-learning uses TD(λ)-methods to accelerate Q-learning. The worst case complexity for a single update step of previous online Q(λ) implementations based on lookup-tables is bounded by the size of the state/action space. Our faster algorithm's worst case complexity is bounded by the number of actions. The algorithm is based on the observation that Q-value updates may be postponed until they are needed.

Keywords: Reinforcement learning, Q-learning, TD(λ), online Q(λ), lazy learning

1 Introduction

Q(λ)-learning (Watkins, 1989; Peng and Williams, 1996) is an important 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.

Online vs. offline. We distinguish *online* RL and *offline* RL. Online RL updates modifiable parameters after each visit of a state. Offline RL delays updates until after a trial is finished, that is, until a goal has been found or a time limit has been reached. Without explicit trial boundaries offline RL does not make sense at all. But even where applicable, offline RL tends to get outperformed by online RL which uses experience earlier and therefore more efficiently (Rummery and Niranjan, 1994). Online RL's advantage can be huge. For instance, online methods that punish actions (to prevent repetitive selection of identical actions) can discover certain environments' goal states in polynomial time (Koenig and Simmons, 1996), while offline RL requires exponential time (Whitehead, 1992).

Previous Q(λ) implementations. To speed up Q-learning, Watkins (1989) suggested combining it with TD(λ) learning. Typical *online* Q(λ) implementations (Peng and Williams, 1996) based on lookup-tables or other local approximators such as CMACS (Albus 1975; Sutton, 1996) or self-organizing maps (Kohonen, 1988), however, are unnecessarily time-consuming. Their update complexity depends on the values of λ and discount factor γ, and is proportional to the number of SAPs (state/action pairs) which occurred. The latter is bounded by the size of state/action space (and by the trial length which may be proportional to this).

Lin's *offline* Q(λ) (1993) creates an action-replay set of experiences after each trial. Cichosz' *semi-online* method (1995) combines Lin's offline method and online learning. It needs fewer updates than Peng and Williams' online Q(λ), but postpones Q-updates until several subsequent experiences are available. Hence actions executed before the next Q-update are less informed than they could be. This may result in performance loss. For instance, suppose that the same state is visited twice in a row. If some hazardous action's Q-value does not reflect negative experience collected after the first visit then it may get selected again with higher probability than wanted.

The novel method. Previous methods either are not truly online and thus require more experiences, or their updates are less efficient than they could be and thus require more computation time. Our Q(λ) variant is truly online and more efficient than others because its update complexity does not depend on the number of states[1]. It uses "lazy learning" (introduced in memory-based learning, e.g., Atkeson, Moore and Schaal 1996) to postpone updates until they are needed.

Outline. Section 2 reviews Q(λ) and describes Peng and William's Q(λ)-algorithm (PW). Section 3 presents our more efficient algorithm. Section 4 presents a practical comparison on 100 $\times$ 100 mazes. Section 5 concludes.

2 Q(λ)-Learning

We consider discrete Markov decision processes, using time steps $t = 1, 2, 3, \ldots$, a discrete set of states $S = \{S_1, S_2, S_3, \ldots, S_n\}$ and a discrete set of actions A. s_t denotes the state at time t, and $a_t = \Pi(s_t)$ the action, where Π represents the learner's policy mapping states to actions. The transition function P with elements $P^a_{ij} = P(s_{t+1} = j | s_t = i, a_t = a)$ for $i, j \in S$ and $a \in A$ defines the transition probability to the next state s_{t+1} given s_t and a_t. A reward function R defines the scalar reward signal $R(i, a, j) \in \mathbb{R}$ for making the transition to state j given state i and action a. r_t denotes the reward at time t. A discount factor $\gamma \in [0, 1]$ discounts later against immediate rewards. The controller's goal is to select actions which maximize 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 an action evaluation function or Q-function is learned. The optimal Q-value of SAP (i, a) satisfies

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

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}).

[1] The method can also be used for speeding up tabular TD(λ).

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 (Watkins, 1989):

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

where e'_t is the temporal difference or TD(0)-error 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 $V(s) = \max_a Q(s, a)$ (evaluated at time-step t), 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):

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

They hold for $\alpha_k(s, a) = \frac{1}{k^\beta}$, where $\frac{1}{2} < \beta \leq 1$.

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

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

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

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_k(s_t, a_t) e^\lambda_t,$$

where the TD(λ)-error e^λ_t is defined as

$$e^\lambda_t = e'_t + \sum_{i=1}^{\infty} (\gamma\lambda)^i e_{t+i}$$

Eligibility traces. The updates above cannot be made as long as 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). 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:

$$\begin{aligned}
\Delta Q(s, a) &= \lim_{k\to\infty} \sum_{t=1}^{k} e^\lambda_t \eta^t(s, a) \\
&= \lim_{k\to\infty} \sum_{t=1}^{k} [e'_t \eta^t(s, a) + \sum_{i=t+1}^{k} (\gamma\lambda)^{i-t} e_i \eta^t(s, a)] \\
&= \lim_{k\to\infty} \sum_{t=1}^{k} [e'_t \eta^t(s, a) + \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i} e_t \eta^i(s, a)] \\
&= \lim_{k\to\infty} \sum_{t=1}^{k} [e'_t \eta^t(s, a) + e_t \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i} \eta^i(s, a)] \qquad (2)
\end{aligned}$$

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

$$l_t(s,a) = \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i}\eta^i(s,a) = \gamma\lambda(l_{t-1}(s,a) + \eta^{t-1}(s,a))$$

Then the online update at time t becomes:

$$\forall (s,a) \in S \times A \;\; do: \;\; Q(s,a) \leftarrow Q(s,a) + \alpha_k(s_t,a_t)[e'_t\eta^t(s,a) + e_t l_t(s,a)]$$

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

PW's Q(λ)-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 $(l(s,a) < \epsilon)$
     - 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 "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 behavior, which means that each SAP occurs just once (if SAPs reoccur then the history list will grow at a slower rate). At the start of the trial the number of updates increases linearly until at some time step t some SAPs get deleted from H. This will happen as soon as $t \geq \frac{\log\epsilon}{\log(\gamma\lambda)}$. Since the number of updates is bounded from above by the number of SAPs, the total update complexity increases towards $O(|S||A|)$ per update for $\gamma\lambda \rightarrow 1$.

3 Fast Q(λ)-Learning

The main contribution of this paper is an efficient, fully online algorithm with time complexity $O(|A|)$ per update. The algorithm is designed for $\lambda\gamma > 0$ — otherwise we 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 (2):

$$\sum_{t=1}^{k}[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i] =$$
$$\sum_{t=1}^{t_1}[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i] + \sum_{t=t_1+1}^{t_2}[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i] + \sum_{t=t_2+1}^{t_3}[e'_t\eta^t + e_t\sum_{i=1}^{t-1}\phi^{t-i}\eta^i] + \ldots$$

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

$$e'_{t_1} + e'_{t_2} + \sum_{t=t_1+1}^{t_2} e_t\phi^{t-t_1} + e'_{t_3} + \sum_{t=t_2+1}^{t_3} e_t(\phi^{t-t_1} + \phi^{t-t_2}) + \ldots =$$
$$e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}\sum_{t=t_1+1}^{t_2} e_t\phi^t + e'_{t_3} + (\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}})\sum_{t=t_2+1}^{t_3} e_t\phi^t + \ldots =$$
$$e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}(\sum_{t=1}^{t_2} e_t\phi^t - \sum_{t=1}^{t_1} e_t\phi^t) + e'_{t_3} + (\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}})(\sum_{t=1}^{t_3} e_t\phi^t - \sum_{t=1}^{t_2} e_t\phi^t) + \ldots$$

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

$$e'_{t_1} + e'_{t_2} + \frac{1}{\phi^{t_1}}(\Delta_{t_2} - \Delta_{t_1}) + e'_{t_3} + (\frac{1}{\phi^{t_1}} + \frac{1}{\phi^{t_2}})(\Delta_{t_3} - \Delta_{t_2}) + \ldots \qquad (3)$$

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 (3) to write down the total update of $Q(s,a)$ during a trial:

$$\Delta Q(s,a) = \lim_{k\to\infty} \sum_{t=1}^{k} e'_t\eta^t(s,a) + l'_t(s,a)(\Delta_{t+1} - \Delta_t) \qquad (4)$$

To exploit this we introduce a global variable Δ keeping track of the cumulative TD(λ) error since the start of the trial. 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 3 and 4). 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.

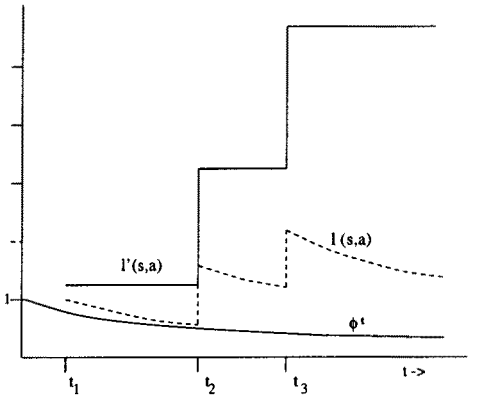

Fig. 1. *SAP* (s,a) *occurs at times* $t_1, t_2, t_3, \ldots$. *The standard eligibility trace* $l(s,a)$ *equals the product of* ϕ^t *and* $l'(s,a)$.

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 initialize 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:

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 (s_t,a_t)'s Q-value and trace variable 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
   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) $\phi^t \leftarrow \gamma\lambda\phi^{t-1}$
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) $l'(s_t, a_t) \leftarrow l'(s_t, a_t) + \frac{1}{\phi^t}$

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

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) Do $\forall (s, a) \in H$
      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 are the 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 $l < e_m$. SAPs are deleted from H once their trace falls below e_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 $O(|A|)$.

Comparison to PW. Figure 2 illustrates the difference between theoretical worst-case behaviors of both methods for $|A| = 5$, $|S| = 1000$, and $\gamma = 1$. We plot updates per time step for $\lambda \in \{0.7, 0.9.0.99\}$. The accuracy parameter ϵ (used in PW) is set to 10^{-6} (in practice less precise values may be used, but this will not change matters much). ϵ_m is set to 10^{-16}. The spikes in the plot for fast Q(λ) reflect occasional full sweeps through the history list due to limited machine precision (the corresponding average number of updates, however, is very close to the value indicated by the horizontal solid line — as explained above, the spikes hardly affect the average). No sweep is necessary in fast Q(0.99)'s plot during the shown interval. Fast Q needs on average a total of 13 update steps: 5 in choose-action, 5 for calculating $V(s_{t+1})$, 1 for updating the chosen action, and 2 for taking into account the full sweeps.

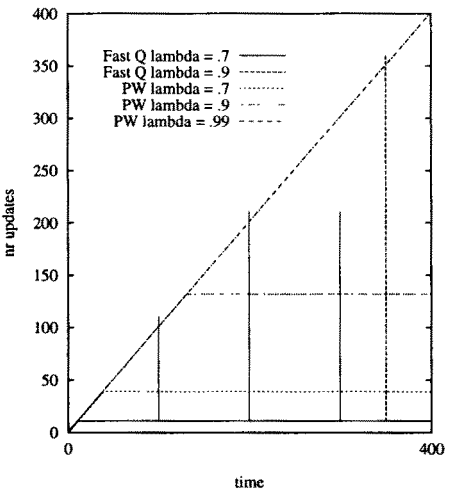

Fig. 2. *Number of updates plotted against time: a worst case analysis for our method and Peng and Williams' (PW) for different values of* λ.

Multiple Trials. We have described a single-trial version of our algorithm. One might be tempted to think that in case of multiple trials all SAPs in the

history list need to be updated and all eligibility traces reset after each trial. This is not necessary — we may use cross-trial learning as follows:

We introduce Δ^M variables, where index M stands for the M^{th} trial. Let N denote the current trial number, and let variable $visited(s,a)$ represent the trial number of the most recent occurrence of SAP (s,a). Now we slightly change *Local Update*:

Local Update(s_t,a_t) **:**
1) $M \leftarrow visited(s_t,a_t)$
2) $Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha_k(s_t,a_t)(\Delta^M-\delta(s_t,a_t))l'(s_t,a_t)$
3) $\delta(s_t,a_t) \leftarrow \Delta^N$
4) If $(M < N)$
   4a) $l'(s_t,a_t) \leftarrow 0$
   4b) $visited(s_t,a_t) \leftarrow N$

Thus we update $Q(s,a)$ using the value Δ^M of the most recent trial M during which SAP (s,a) occurred and the corresponding values of $\delta(s_t,a_t)$ and $l'(s_t,a_t)$ (computed during the same trial). In case SAP (s,a) has not occurred during the current trial we reset the eligibility trace and set $visited(s,a)$ to the current trial number. In *Global Update* we need to change lines 5 and 10b by adding trial subscripts to Δ, and we need to change line 9b in which we have to set $visited(s_t,a_t) \leftarrow N$. At trial end we reset ϕ^t to $\phi^0 = 1.0$, increment the trial counter N, and set $\Delta^N \leftarrow 0$. This allows for postponing certain updates until after the current trial's end.

4 Experiments

To evaluate competing training methods in practice we created a set of 20 different randomly initialized 100×100 mazes, each with about 20% blocked fields. All mazes share a fixed start state (S) and a fixed goal state (G) (we discarded mazes that could not be solved by Dijkstra's shortest path algorithm). See figure 3 for an example. In each field the agent can select one of the four actions: *go north, go east, go south, go west.* Actions that would lead into a blocked field are not executed. Once the agent finds the goal state it receives a reward of 1000 and is reset to the start state. All steps are punished by a reward of -1. The discount factor γ is set to 0.99. Note that initially the agent has no information about the environment at all.

Experimental set-up. To select actions we used the *max-random* selection rule, which selects an action with maximal Q-value with probability P_{max} and a random action with probability $1 - P_{max}$. A single run on one of the twenty mazes consisted of 5,000,000 steps. During each run we linearly increased P_{max} from 0.5 (start) until 1.0 (end). Every 10,000 steps the learner's performance was monitored by computing its cumulative reward so far. The optimal performance is about 41,500 = 41.5K reward points (this corresponds to 194-step paths).

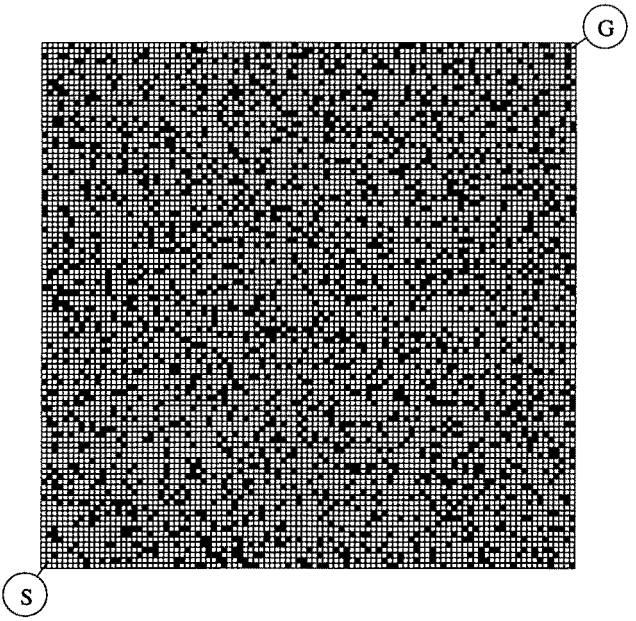

Fig. 3. *A 100 × 100 maze used in the experiments. Black states denote blocked field. The agent's aim is to find the shortest path from start S to goal G.*

To show how learning performance depends on λ we set up multiple experiments with different values for λ and β (used for annealing the learning rate). β must theoretically exceed 0.5 but may be lower in practice. If λ is large and β too small, however, then the Q-function will tend to diverge.

Parameters. We always chose the lowest beta (and hence the largest learning rate) such that the Q-function did not diverge. Final parameter choices were: Q(0) and Q(0.5) with $\beta = 0$, Q(0.7) with $\beta = 0.01$, Q(0.9) with $\beta = 0.2$, Q(0.95) with $\beta = 0.3$. Q(0.99) with $\beta = 0.4$. PW's trace cutoff ϵ was set to 0.001. Larger values scored less well; lower ones costed more time. Machine precision ϵ_m was set to 10^{-16}. Time costs were computed by measuring cpu-time (including action selection and almost neglectable simulator time) on a 50 MHz SPARC station.

Results. Learning performance for fast Q(λ) is plotted in Figure 1(A) (results with PW's Q(λ) are very similar). We observe that larger values of λ increase performance much faster than smaller values, although the final performances are best for standard Q-learning and $Q(0.95)$. Figure 1(B), however, shows that fast Q(λ) is not much more time-consuming than standard Q-learning, whereas PW's Q(λ) consumes a lot of CPU time for large λ.

Table 1 shows more detailed results. It shows that Q(0.95) led to the largest cumulative reward which indicates that its learning speed is fastest. Note that for this best value $\lambda = 0.95$, fast Q(λ) was more than four times faster than PW's Q(λ).

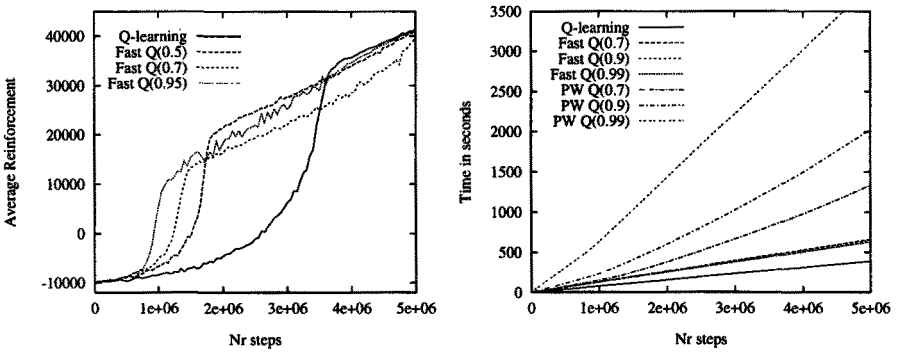

Fig. 4. *(A) Learning performance of Q(λ)-learning with different values for λ. (B) CPU time plotted against the number of learning steps.*

PW's cut-off method works only when traces decay, which they did due to the chosen λ and γ parameters. For $\lambda\gamma = 1$ (worst case), however, PW would consume about 100(!) hours whereas our method needs around 11 minutes.

5 Conclusion

While other Q(λ) approaches are either offline, inexact, or may suffer from average update complexity depending on the size of the state/action space, ours is fully online Q(λ) with average update complexity linear in the number of actions. Efficiently dealing with eligibility traces makes fast Q(λ) applicable to larger scale RL problems.

Acknowledgments

Thanks for helpful comments to Nic Schraudolph and an anonymous reviewer.

Table 1. *Average results for different online Q(λ) methods on twenty 100 × 100 mazes. Final performance is the cumulative reward during the final 20,000 steps. Total performance is the cumulative reward during a simulation. Time is measured in cpu seconds.*

System	Final performance	Total performance	Time
Q-learning	41.5K ± 0.5K	4.5M ± 0.3M	390 ± 20
Fast Q(0.5)	41.2K ± 0.7K	8.9M ± 0.2M	660 ± 33
Fast Q(0.7)	39.6K ± 1.4K	7.9M ± 0.3M	660 ± 29
Fast Q(0.9)	40.9K ± 1.0K	9.2M ± 0.3M	640 ± 32
Fast Q(0.95)	41.5K ± 0.5K	9.8M ± 0.2M	610 ± 32
Fast Q(0.99)	40.0K ± 1.1K	8.3M ± 0.4M	630 ± 39
PW Q(0.5)	41.3K ± 0.8K	8.9M ± 0.3M	1300 ± 57
PW Q(0.7)	40.0K ± 0.7K	7.9M ± 0.3M	1330 ± 38
PW Q(0.9)	41.2K ± 0.7K	9.4M ± 0.3M	2030 ± 130
PW Q(0.95)	41.2K ± 0.9K	9.7M ± 0.3M	2700 ± 94
PW Q(0.99)	39.8K ± 1.4K	8.2M ± 0.4M	3810 ± 140

References

[Albus, 1975] Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Dynamic Systems, Measurement and Control*, pages 220–227.

[Atkeson et al., 1997] Atkeson, C. G., Schaal, S., and Moore, A. W. (1997). Locally weighted learning. *Artificial Intelligence Review*, 11:11–73.

[Barto et al., 1983] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846.

[Bertsekas and Tsitsiklis, 1996] Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-dynamic Programming.* Athena Scientific, Belmont, MA.

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

[Cichosz, 1995] Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *Journal on Artificial Intelligence*, 2:287–318.

[Koenig and Simmons, 1996] Koenig, S. and Simmons, R. G. (1996). The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms. *Machine Learning*, 22:228–250.

[Kohonen, 1988] Kohonen, T. (1988). *Self-Organization and Associative Memory.* Springer, second edition.

[Lin, 1993] Lin, L. (1993). *Reinforcement Learning for Robots Using Neural Networks.* PhD thesis, Carnegie Mellon University, Pittsburgh.

[Peng and Williams, 1996] Peng, J. and Williams, R. (1996). Incremental multi-step Q-learning. *Machine Learning*, 22:283–290.

[Rummery and Niranjan, 1994] Rummery, G. and Niranjan, M. (1994). On-line Q-learning using connectionist sytems. Technical Report CUED/F-INFENG-TR 166, Cambridge University, UK.

[Singh and Sutton, 1996] Singh, S. and Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158.

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

[Sutton, 1996] Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. M. and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1045. MIT Press, Cambridge MA.

[Tesauro, 1992] Tesauro, G. (1992). Practical issues in temporal difference learning. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 4*, pages 259–266. San Mateo, CA: Morgan Kaufmann.

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

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, England.

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

[Whitehead, 1992] Whitehead, S. (1992). *Reinforcement Learning for the adaptive control of perception and action.* PhD thesis, University of Rochester.