Postponed Updates for Temporal-Difference Reinforcement Learning

Harm van Seijen
TNO Defence, Security and Safety
Oude Waalsdorperweg 63
2597 AK The Hague, The Netherlands
harm.vanseijen@tno.nl

Shimon Whiteson
Informatics Institute, University of Amsterdam
Science Park 107
1098 XG Amsterdam, The Netherlands
s.a.whiteson@uva.nl

Abstract

This paper presents postponed updates, a new strategy for TD methods that can improve sample efficiency without incurring the computational and space requirements of model-based RL. By recording the agent's last-visit experience, the agent can delay its update until the given state is revisited, thereby improving the quality of the update. Experimental results demonstrate that postponed updates outperforms several competitors, most notably eligibility traces, a traditional way to improve the sample efficiency of TD methods. It achieves this without the need to tune an extra parameter as is needed for eligibility traces.

1 Introduction

In *reinforcement learning* (RL) [4, 13], an agent seeks an optimal control policy for a sequential decision problem. Unlike in supervised learning, the agent never sees examples of correct or incorrect behavior. Instead, it receives only positive and negative rewards for the actions it tries. When the sequential decision problem is modeled as a *Markov decision process* (MDP) [2], the agent can learn an optimal policy using *temporal-difference* (TD) methods [11, 14]. Each time the agent acts, the resulting feedback is used to update estimates of its action-value function, which predicts the long-term discounted reward it will receive if it takes a given action in a given state. Once the optimal action-value function has been learned, an optimal policy can easily be derived.

TD methods are appealing due to their simplicity and computational efficiency. However, they are often criticized for a lack of sample efficiency, as the agent may need infeasibly many interactions with its environment to discover a good policy. By contrast, in *model-based RL* [12, 6], the agent uses its experience interacting with the environment to estimate a model of the MDP and then computes a policy via off-line planning techniques such as dynamic programming [1]. While model-based methods can be more sample efficient [3, 5, 9] than TD methods, they also require more computation, for planning, and space, to represent the model.

This paper presents *postponed updates*, a new strategy for TD methods. By recording the agent's last-visit experience in each state or state-action pair, it is possible to postpone the update until later. Doing so can speed learning by improving the quality of the update, as the value estimates of other state-action pairs involved in the update may have improved in the meantime.

Our experimental results demonstrate that in its most basic form, postponed updates can improve the sample efficiency at no extra computational cost. The sample efficiency can be further improved by using *provisional updates*, that allow the final updates to be further postponed. Based on provisional updates we discuss three different algorithms that trade-off computation time for an increased sample efficiency in different ways. The *extended postponing* method offers an improved sample efficiency for limited extra computation, while pushing the trade-off all the way to unlimited computation, the agent can compute the *best-match* Q-values, resulting in still larger performance gains. Finally, we demonstrate that *prioritized sweeping* [6], a method for reducing the computational costs of model-based RL, can also be applied to speed the computation of best-match Q-values. Experimental results show that this method outperforms several competitors, including eligibility traces, a traditional way to increase the sample efficiency of TD methods, while it does not require tuning an extra parameter. By contrast, eligibility traces requires tuning the trace decay parameter λ for optimal results.

2 Background

Sequential decision problems are often formalized as *Markov decision processes* (MDPs), which can be described as 4-tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ consisting of $\mathcal{S}$, the set of all states; $\mathcal{A}$, the set of all actions; $\mathcal{T}(s, a, s') = P(s'|s, a)$, the tran-

sition probability from state $s \in \mathcal{S}$ to state s' when action $a \in \mathcal{A}$ is taken; and $\mathcal{R}(s,a) = E(r|s,a)$, the reward function giving the expected reward r when action a is taken in state s. Actions are selected at discrete time steps $t = 0, 1, 2, ...$ and r_{t+1} is defined as the reward received after taking action a_t in state s_t at time step t. The goal of the agent is to find an optimal policy π^* that maximizes the expected discounted return:

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1)$$

where γ is a discount factor with $0 \leq \gamma \leq 1$.

Most solution methods are based on estimating a *value function* $V^\pi(s)$, which gives the expected return when the agent is in state s and follows policy π, or an *action-value function* $Q^\pi(s,a)$, which gives the expected return when the agent takes action a in state s and follows policy π thereafter.

Most TD methods seek to learn the optimal action-value function $Q^*(s,a)$ by iteratively updating the current estimate $Q_t(s,a)$ each time new experience is obtained. A common form for these updates is:

$$Q_{t+1}(s_t, a_t) \leftarrow (1-\alpha) Q_t(s_t, a_t) + \alpha\, v \quad (2)$$

where α is the learning rate and v is the update target. Many update targets are possible, such as the Q-learning [14] update target:

$$v_t = r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) \quad (3)$$

Alternatively, the agent can take a model-based approach [12, 6], in which its experience is used to compute maximum-likelihood estimates of $\mathcal{T}$ and $\mathcal{R}$. Using the model, the agent can compute the optimal value function V^* using dynamic programming methods [1] such as value iteration [7]. Each time new experience is gathered, the model is updated and V^* recomputed.

3 Postponed Updates

This section introduces four different algorithms that make use of last-visit experience to postpone the updates performed by TD methods, thus improving the update target and speeding learning. The algorithms differ in the trade-off they make between computation per time step and improvement of the update target.

3.1 Basic Postponing

The most simple TD method is the Q-learning algorithm [14]. It performs updates (Equation 2) based on v_t (Equation 3) right after a given state-action pair (s_t, a_t) is visited. However, this updated value is not used until s_t is revisited, at which point it is used in the update target of the preceding state and to select a new action. Therefore, the actual update of the Q-value of (s_t, a_t) with v_t can be postponed till just before it is used at the moment of revisit of s_t, without affecting the online performance.

Following this reasoning further, it is easy to see that the above is also valid when applied to all state-action pairs simultaneously. In other words, a method that postpones the update of all Q-values till its states are revisited has the same online performance as regular Q-learning. Overall, the Q-values of this method are behind in the number of updates they have received. However, the relevant Q-values, which are the Q-values from the current state, have received exactly the same updates as with regular Q-learning, resulting in the same online performance.

Although postponing the updates based on v_t results in a method with the same performance, delaying the update makes it possible to use an alternative update target that is more accurate than v_t. Before explaining why, we define the more general update target v_t^{t+n}:

$$v_t^{t+n} = r_{t+1} + \gamma \max_a Q_{t+n-1}(s_{t+1}, a) \quad (4)$$

This update target is the same as the Q-learning update target v_t, except that the Q-values of state s_{t+1} are from a later time. Note that v_t is equal to v_t^{t+1}. If we now define t^* to be the time of the first revisit of state s_t, than by postponing till the revisit of s_t, it becomes possible to use the update target $v_t^{t^*}$ instead of v_t. Comparing v_t with $v_t^{t^*}$ leads to two cases. If state s_{t+1} has not been revisited between $t+1$ and t^*, then $v_t = v_t^{t^*}$ since the last update for s_{t+1} occurred at time $t+1$. Note that this is also true for a returning action ($t^* = t+1$). On the other hand, if state s_{t+1} is revisited before t^* (like in Figure 1), one of the Q-values of s_{t+1} has received an extra update in the meantime. Since TD updates cause the expected error in the Q-values to decrease over time [14, 8], due to this extra update v_t^{t*} will be on average more accurate than v_t. So, although at time t^* the Q-values of s_t have received just as many updates with postponed updates as with regular updates, due to the different order of the updates the update target for postponed updates is more accurate, resulting in faster learning. This counterintuitive property is the key property behind the sample efficiency improvement of all our postponed updates methods.

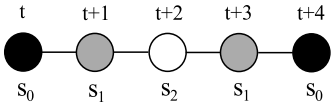

Figure 1. A state transition sequence in which postponed updates can lead to faster learning. Subscripts indicate state indices.

Simply postponing the update till the revisit of a state we call *basic postponing*. Algorithm 1 shows pseudocode for the basic postponing implementation of Q-learning with ϵ-greedy action selection. The same idea can easily be combined with other TD methods or exploration strategies. In order to compute $v_t^{t^*}$ we need to store the last-visit experience for each state, i.e., the action, reward and next state experienced at the last visit of a state. We store these values in $\hat{A}(s)$, $\hat{R}(s)$ and $\hat{S}'(s)$, respectively. If $\hat{S}(s) = -1$, then state s has not been visited yet and no update can be performed. Note that the last-visit experience is not reset at the end of an episode, but maintained across episodes.

Algorithm 1 Q-Learning with Basic Postponing

```
1:  Initialize Q(s, a) arbitrarily for all s,a
2:  Initialize Ŝ'(s) = −1 for all s
3:  loop {over episodes}
4:      Initialize s
5:      repeat {for each step in the episode}
6:          if Ŝ'(s) ≠ −1 then
7:              Q(s, Â(s)) ← (1 − α)Q(s, Â(s))+
                α [R̂(s) + γ max_a Q(Ŝ'(s), a)]
8:          end if
9:          with probability ε select random a,
            otherwise a ← argmax_a' Q(s, a')
10:         take action a, observe r and s'
11:         Ŝ'(s) ← s'; R̂(s) ← r; Â(s) ← a
12:         s ← s'
13:     until s is terminal
14: end loop
```

3.2 Extended Postponing

In the example shown in Figure 1, the update of $Q(s_0, a_t)$ is postponed until the first time state s_0 is revisited. In this section, we demonstrate that the update can be postponed even further in the case where a different action is selected upon revisit. Consider the example shown in Figure 2, which extends the previous example to include a second revisit of s_0 at time $t + 7$. Suppose that a different action is selected on the first revisit ($a_t \neq a_{t+4}$) but the original action is chosen on the second revisit ($a_t = a_{t+7}$).

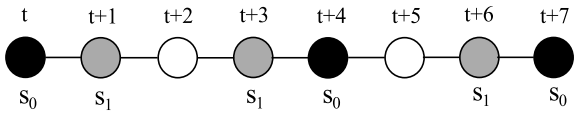

Figure 2. A state transition sequence in which provisional updates can enable further postponing. Subscripts indicate state indices; timesteps are shown above each state.

At time $t + 4$, updated Q-values of state s_0 are needed to perform action selection. Therefore, the agent can perform a *provisional update* of $Q(s_0, a_t)$. However, since no new experience about $Q(s_0, a_t)$ is obtained between time $t + 4$ and time $t + 7$, the agent can discard this provisional update and *redo* the update at time $t+7$, using the same experience, but with more recent Q-values for s_1. Since s_1 is revisited again at time $t + 6$, these Q-values may have further improved in the meantime. Since the agent reselects a_t after this update, it is also the *final update* of $Q(s_0, a_t)$.

Extending the postponing period in this way requires additional computation, as the agent typically performs multiple updates per time step. In the example, at time $t + 7$ the agent must perform both a final update of $Q(s_0, a_t)$, but also a provisional update of $Q(s_0, a_{t+4})$. In the worst case, when all state-actions pairs have been visited at least once, the agent performs $|\mathcal{A}|$ updates per time step. Furthermore, some old Q-values must be remembered in order to undo provisional updates and last-visit experience must be tracked per state-action pair instead of per state. However, this bookkeeping requires only $\mathcal{O}(|\mathcal{S}||\mathcal{A})|$ space, so the space complexity of TD learning remains unchanged.

Algorithm 2 shows the generic pseudocode of using Q-learning with provisional updates, again with ϵ-greedy exploration. The last-visit experience is stored in $\hat{R}(s, a)$ and $\hat{S}'(s, a)$ and old Q-values in $Q_{old}(s, a)$. For the extended postponing approach discussed in this section line 6 is implemented by the following code:

```
1: for all a ∈ A do
2:     if Ŝ'(s, a) ≠ −1 then
3:         Q(s, a) ← (1 − α)Q_old(s, a)+
           α [R̂(s, a) + γ max_a' Q(Ŝ'(s, a), a')]
4:     end if
5: end for
```

Algorithm 2 Q-learning with Provisional Updates

```
1:  Initialize Q(s, a) arbitrarily for all s,a
2:  Initialize Ŝ'(s, a) = −1 for all s,a
3:  loop {over episodes}
4:      Initialize s
5:      repeat {for each step in the episode}
6:          Perform provisional updates of Q-values based on Q_old, Ŝ' and R̂
7:          With probability ε select random a,
            otherwise a ← argmax_a' Q(s, a')
8:          Take action a, observe r and s'
9:          Ŝ'(s, a) ← s', R̂(s, a) ← r
10:         Q_old(s, a) ← Q(s, a)
11:         s ← s'
12:     until s is terminal
13: end loop
```

3.3 Best-Match Q-Values

In the approach described above, provisional updates occur only when a state is revisited. Consequently, when revisits do not occur, the experience from the initial state visit is not employed when updating other states. For example, Figure 3 shows a scenario in which s_1 is visited only once before s_0 is revisited. As a result, both of the algorithms described above will perform at time $t + 3$ the same update as regular Q-learning, without exploiting the experience for s_1 gathered at time $t + 2$.

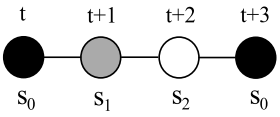

Figure 3. A state transition sequence in which s_1 is not revisited.

Fortunately, it is not necessary to wait for a revisit of s_1 to perform a provisional update. Instead, it can be performed at the moment it is needed: when s_0 is revisited. Thus, if at time $t+3$ the agent performs a provisional update of $Q(s_1, a_{t+1})$, before updating $Q(s_0, s_t)$, the latter update will exploit more recent Q-values for s_1, just as if s_1 had been revisited. However, performing these updates further increases the computational cost. In the worst case, if the agent updates the Q-values of the next state of every action of s_0, it will perform $|\mathcal{A}|^2$ updates per time step.

Taking this idea further, the agent can first update the Q-values of s_2 before updating the Q-values of s_1. In other words, the agent uses the Q-values of s_0 to perform a provisional update of s_2, then performs a provisional update of s_1 and finally s_0. However, once the Q-values of s_0 have changed, it is possible to further improve the Q-values of s_2 even more by redoing its provisional update. The new Q-values of s_2 can then be used to redo the update of s_1, which in turn can be used to re-update s_0. This process can repeat until the Q-values reach a fixed point.

From a mathematical perspective, this situation can be described using a system of $|\mathcal{S}||\mathcal{A}|$ non-linear equations based on the provisional update rule:

$$Q(s,a) = (1-\alpha)\, Q_{old}(s,a) + \alpha\,[\hat{R}(s,a) + \gamma \max_{a'} Q(\hat{S}'(s,a), a')] \quad (5)$$

for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$. The values of Q_{old}, $\hat{S}$ and $\hat{R}$ are known and there are $|\mathcal{S}||\mathcal{A}|$ unknown Q-values. These equations have a unique solution that can be found by iteratively performing updates based on these equations until all Q-values converge. The solution is the set of *best-match* Q-values, i.e., those that best match the last-visit experience from all state-action pairs. The methods described in Sections 3.1 and 3.2 can be seen as approximating this best-match solution under different computation constraints.

We can compute the best-match Q-values by implementing line 6 of algorithm 2 by the following code:

```
1:  repeat
2:      Δ ← 0
3:      for all s,a do
4:          if S'(s,a) ≠ −1 then
5:              v ← Q(s,a)
6:              Q(s,a) ← (1 − α)Q_old(s,a)+
                α [R̂(s,a) + γ max_a' Q(Ŝ'(s,a), a')]
7:              Δ ← max(Δ, |v − Q(s,a)|)
8:          end if
9:      end for
10: until Δ < θ (a small positive number)
```

The resulting algorithm is closely related to model-based methods in which the planning step is performed with value iteration [7]. In value iteration, the set of Bellman optimality equations is solved via iterative update sweeps through the state space. When $\alpha = 1$, Equation 5 reduces to the set of Bellman optimality equations with the last-visit experience treated as a deterministic model. Therefore, in a deterministic environment the best-match method with $\alpha = 1$ is equal to model-based learning. However, in the general case of a stochastic environment this deterministic model is incorrect and α has to be set smaller than 1, causing Q to be an update of Q_{old} based on the last-visit experience. Note that if $Q_{old}(s,a)$ is replaced by $Q(s,a)$ in Equation 5, the solution of the set of equations depends only on the last visit experience, just like in the $\alpha = 1$ case, and it becomes impossible to effectively deal with a stochastic environment. This illustrates the importance of Q_{old}, which ensures proper averaging over stochastic experience. The resulting algorithm is as computationally expensive as model-based RL methods that plan between each step. However, the space complexity remains $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$, whereas representing a full stochastic model requires $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$ space.

3.4 Model-Free Prioritized Sweeping

Computing the best-match Q-values can be very computationally expensive. However, it is possible to efficiently compute good approximations of these values with the same strategies used to improve the computational efficiency of model-based RL. In this section, we show how one such strategy, *prioritized sweeping* [6], can be applied to speed computation of the best-match Q-values.

The prioritized sweeping algorithm makes the planning step of model-based RL more efficient by focusing on the updates expected to have the largest effect on the value function. The algorithm maintains a priority queue of state-action pairs in consideration for updating. When a state-action pair (s,a) is updated, all predecessors (i.e., those state-action pairs whose transition probabilities to s are greater than 0) are added to the queue according to a heuristic priority estimating the impact of the update. At each time step, the top N state-action pairs from this queue are updated, with N depending on the available computation time.

Prioritized sweeping can also be applied to model-free learning to speed computation of the best-match Q-values. The resulting method, which we call model-free prioritized sweeping, can compute good approximations of these values in a computationally efficient way, without the space requirements of model-based learning. The pseudocode for this method implements line 6 of algorithm 2 as:

```
1: loop {N times, while PQueue is not empty}
2:     s, a ← first(PQueue)
```

```
3:    V_s' ← max_a' Q(Ŝ'(s,a), a')
4:    Q(s,a) ← (1−α)Q_old(s,a) + α[R̂(s,a) + γV_s']
5:    V_s ← max_a Q(s,a)
6:    for all s, a with S'(s,a) = s do
7:       p ← |(1−α)Q_old(s,a) + α[R̂(s,a) + γ V_s] − Q(s,a)|
8:       if p > θ, insert s, a into PQueue with priority p
9:    end for
10: end loop
```

Besides that, the priority queue $PQueue$ has to be initialized as an empty queue and after line 11 the line

promote pair (s, a) to top of priority queue

should be added.

This algorithm is very similar to the deterministic version of model-based prioritized sweeping from Sutton and Barto [13] with one crucial difference: the updates happen with respect to Q_{old} instead of Q. In other words, we perform provisional updates instead of regular updates. As explained in Section 3.3 this ensures proper averaging of experience making it possible to use the algorithm in a stochastic environment. We demonstrate the importance of provisional updates in Section 4.2 by comparing the performance of our model-free prioritized sweeping algorithm with the deterministic model-based version of Sutton and Barto on a stochastic task.

4 Results and Discussion

In this section we compare the performance of the various postponed updates methods and several alternatives. We pay special attention to the comparison with eligibility traces, since this method has much in common with the postponed updates approach.

4.1 Comparing Different Postponed Updates Methods

We begin our empirical evaluation by comparing the performance of Q-learning to the novel methods presented in Section 3 on a stochastic variation of the Dyna Maze problem [12]. In this navigation task, depicted in Figure 4, the agent has to find its way from start to goal. The agent can choose between four movement actions: *up*, *down*, *left* and *right*. To determine whether the methods we evaluate are robust in stochastic environments, we employ a probabilistic transition function: with a 20% probability, the agent moves in an arbitrary direction instead of the direction corresponding to the action. All actions result in 0 reward, except for when the goal is reached, which results in a reward of $+1$. The discount factor γ was set to 0.95.

To compare performance, we measure the average return each method accrued from the start state during the first 200 episodes, averaged over 1000 independent runs per method. Each method uses ϵ-greedy action selection with $\epsilon = 0.1$

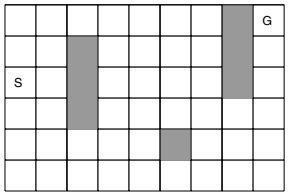

Figure 4. The Dyna Maze task, in which the agent must travel from S to G.

and an initial learning rate α_0 of 0.9 that is decayed in the following manner:

$$\alpha = \alpha_0 d^{n(s,a)} \tag{6}$$

where $n(s, a)$ is the number of times the current action a was previously selected in the current state s. We optimize d for each method individually: we first test $d = 0.9$ and then decrease it by increments of 0.05 until the average return in the 200th episode stops improving. All Q-values are initialized to $1/(1-\gamma)$ in order to be consistent with delayed Q-learning, discussed later in this section.

Figure 5 shows the results of these experiments, comparing the average return per episode of each method. The potential benefits of recording last-visit experience and using it to postpone updates is evident from the performance of the basic postponing method, which substantially outperforms regular Q-learning. Remarkably, it achieves this performance gain without any additional computational or space requirements. The results also demonstrate that computation time can be traded for even better performance, as the extended postponing method obtains a much larger performance gain over Q-learning. Since there are 4 actions in this task, the extended postponing method typically performs 4 updates per time step instead of 1.

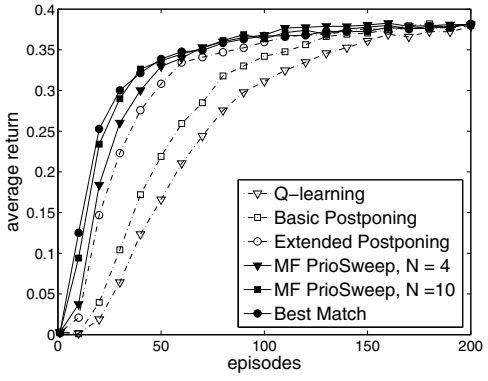

Figure 5. The average return per episode of Q-learning and various postponed updates algorithms on the stochastic Dyna Maze task.

The computational requirements of model-free prioritized sweeping are controlled by the parameter N, the num-

ber of updates performed per time step. With $N = 4$, this method requires approximately the same number of updates per step as extended postponing on this task. However, the results show that, by using a priority queue to determine which state-action pairs to update, it performs even better. Note that this performance is achieved using the same $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ space complexity as Q-learning.

The graph also shows the performance of the best-match algorithm, which can be thought of as a computationally expensive upper bound on the performance achievable with this space complexity. While computing the best-match values will be infeasible in many realistic problems, the results demonstrate that they can be effectively approximated in a computationally efficient way, as model-free prioritized sweeping requires only $N = 10$ before its performance becomes indistinguishable from the best-match approach.

4.2 Comparing Against Other Methods

In this subsection, we compare the performance of model-free prioritized sweeping to several alternative methods from the literature that balance computational, space, and sample efficiency in various ways.

We start by comparing against methods using eligibility traces [11], to which the postponed updates approach is closely related. Whereas postponed updates methods keep track of their recent history by storing last-visit experience, methods based on eligibility traces keep track of recently visited states by maintaining a trace parameter per state or state-action pair. This trace parameter is increased when a state is visited and decreased by $\gamma\,\lambda$ otherwise, where λ is the *trace-decay parameter*. At each time step, all states are updated proportional to their trace parameter. Since recently visited states have a higher trace value, they receive a larger correction. There are two commonly used variation of eligibility traces. Methods that use *accumulating traces* increase the trace parameter of a visited state by 1, while methods that use *replacing traces* set the trace parameter equal to 1.

Both postponed updates and eligibility traces aim to improve the Q-value estimates by using experience to perform multiple updates, though they do so in different ways. To illustrate the differences, we first compare their performance on a policy evaluation task. We consider the small network shown in Figure 6. The network consists of 4 equivalent states, each with only a single action pointing to a neighbor state. Each action results in a reward drawn from a normal distribution with mean 1 and a standard deviation of 0.5, making it a stochastic problem. With a discount factor of 0.95, we can easily determine analytically that the value of each state is 20. We compare the root mean-squared (RMS) error across the 4 states for the best-match method and model-free prioritized sweeping to TD(λ) for different values of λ. We initialize the Q-values at 0 and used annealing learning rates according to equation 6, with $a_0 = 1$ and optimized the decay parameter d per method. We averaged over 1000 independent runs. For TD(λ) we use accumulating traces, since it outperforms replacing traces on this task. Figure 6 shows the results for the first 50 timesteps. Although we determined the error for λ values from 0 to 1 with steps of 0.1, we only show the results for 3 λ-values, including the optimal λ-value, which was 0.9.

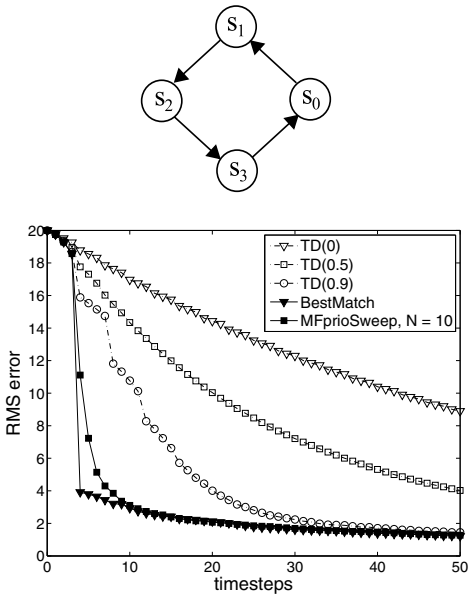

Figure 6. The small circular network (top) and the RMS error of various methods on its policy evaluation (bottom).

This example shows that model-free prioritized sweeping outperforms eligibility traces at all its λ values, without the need to tune an extra parameter.

While the previous experiment focused on the quality of value estimates, the performance of the postponed update methods can also be compared to eligibility traces in the control case. To do so, we compare our model-free prioritized sweeping algorithm to Watkins's Q(λ), the off-policy implementation of eligibility traces, on the Dyna Maze task. As before, we optimize λ for the range from 0 to 1 with steps of 0.1.

We also compare against several other alternatives. The second alternative is model-based prioritized sweeping [6]. As mentioned in Section 3.4, this method maintains a maximum-likelihood model and performs N value iteration updates per time step, prioritizing updates based on the expected impact on the value function. For a given N, model-based prioritized sweeping has similar computational requirements as our model-free alternative. However, maintaining a model requires $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$ space.

The third alternative is deterministic model-based prioritized sweeping [13], a simpler variation on prioritized sweeping that learns only a deterministic model, uses a slightly different priority heuristic, and performs Q-learning updates to its Q-values. While this method was clearly designed with deterministic tasks in mind, it can be applied to stochastic tasks, in which case updates are based on a model consistent with the last-visit experience. This approach can be viewed as a naïve alternative way of achieving the $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ space requirements of model-free prioritized sweeping.

The fourth alternative is delayed Q-learning [10], a model-free method that, like some model-based methods [3, 5, 9], is proven to be *probably approximately correct* (PAC), i.e., its sample complexity is polynomial with high probability. Delayed Q-learning works by initializing its Q-values optimistically and ensuring that value estimates are not reduced until the corresponding state-action pairs have been sufficiently explored. Because it does not maintain a model, it has the same $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ space requirements as model-free prioritized sweeping. However, to our knowledge, its empirical performance has never been evaluated before.

As in the previous experiments, we optimized the learning rate decay shown in Equation 6 for each method except model-based prioritized sweeping and delayed Q-learning, which do not use learning rates. For delayed Q-learning we optimized the two free parameters m and ϵ_1 by testing every combination of the following values: $m \in [1, 10]$ and $\epsilon_1 \in \{0.001, 0.005, 0.010, 0.015, 0.020\}$ and found that $m = 6$ and $\epsilon_1 = 0.015$ performed the best. All methods use ϵ-greedy action selection with $\epsilon = 0.1$, except for delayed Q-learning, which relies on optimistic initialization for exploration. The top graph of Figure 7 shows the average return of the start state for the first 500 episodes, averaged over 1000 runs for each method. The graph also includes the performance of regular Q-learning as a baseline for comparison.

As expected, the performance of model-free prioritized sweeping falls between that of Q-learning and model-based prioritized sweeping. Each alternative represents a different trade-off: Q-learning requires less computation but achieves inferior performance; model-based prioritized sweeping achieves superior performance but has higher space complexity. In this domain, Q(λ) performs similarly to model-free prioritized sweeping. However, achieving this performance requires optimizing the λ parameter; for other λ values the performance is considerably worse. By contrast, model-free prioritized sweeping has one fewer parameter to optimize.

Surprisingly, delayed Q-learning performs quite poorly, worse even than regular Q-learning. While the algorithm represents an important theoretical contribution, its theoretical properties do not guarantee good performance in practice. Unlike both model-based and model-free prioritized sweeping, it does not use extra computation time to speed performance. While its exploration strategy ensures the algorithm is PAC, the results show it is not always effective in practice, as simple ϵ-greedy Q-learning performs better.

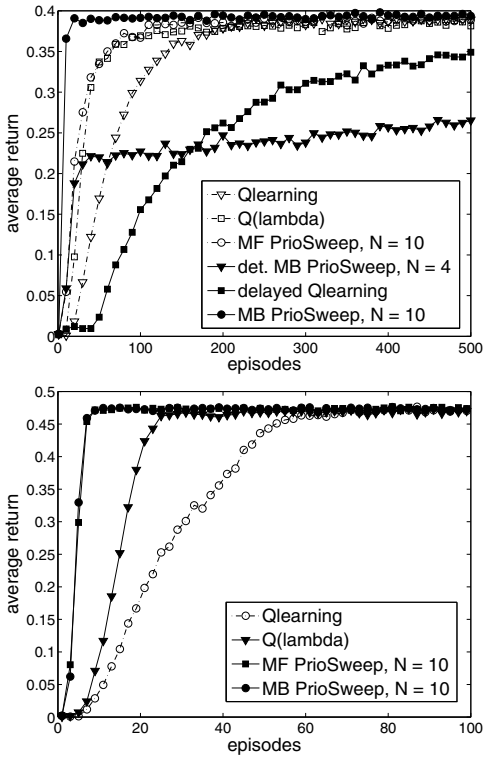

Figure 7. The average return per episode of model-free prioritized sweeping and various alternatives on the stochastic (top) and deterministic (bottom) Dyna Maze task.

The performance of deterministic model-based prioritized sweeping was very unstable on the Dyna Maze problem. At many parameter settings we were unable to complete any runs because individual episodes ran infeasibly long, as the agent was unable to find the goal. The results shown occurred on the only parameter setting for which we successfully completed 1000 runs. We also found other parameter settings that performed better (though never as well as the regular Q-learning baseline) on some runs, but failed to terminate on others.

The bottom graph of Figure 7 shows the results for the deterministic case. The results are averaged over 500 independent runs. As expected, model-free prioritized sweeping performs now similar to model-based prioritized sweeping and outperforms Q(λ) at an optimal λ of 0.7 by a large margin.

Finally, we compare the performance of model-free prioritized sweeping to that of Q-learning and Q(λ) on a larger maze problem (see Figure 8), to determine if its perfor-

mance scales to more challenging tasks. In this case, the agent receives only a reward of -0.1 per time step, but it receives a reward of -2 if a wall or the border is hit. Upon reaching the goal state, the agent receives a reward of 100. The discount factor is 0.99 and the Q-values are initialized to 0. The environment is stochastic and moves the agent with a probability of 10% in a random direction instead of the preferred direction. ϵ-greedy exploration is used with $\epsilon = 0.05$. We anneal the learning rate according to Equation 6 with α_0 set to 0.9 and optimize d to get the best average performance over the first 200 episodes. For Q(λ) we also optimize λ. Results are averaged over 100 independent runs.

The bottom graph of Figure 8 shows the results. For this task, model-free prioritized sweeping significantly outperforms Q(λ), demonstrating that model-free prioritized sweeping can significantly outperform Q(λ) in stochastic environments also. The performance of model-free prioritized sweeping very closely follows the performance of model-based prioritized sweeping for the initial learning phase, but shows no significant improvement anymore after the first 20 episodes because the optimal decay parameter is relatively large causing the learning rate to be close to zero after the initial learning phase.

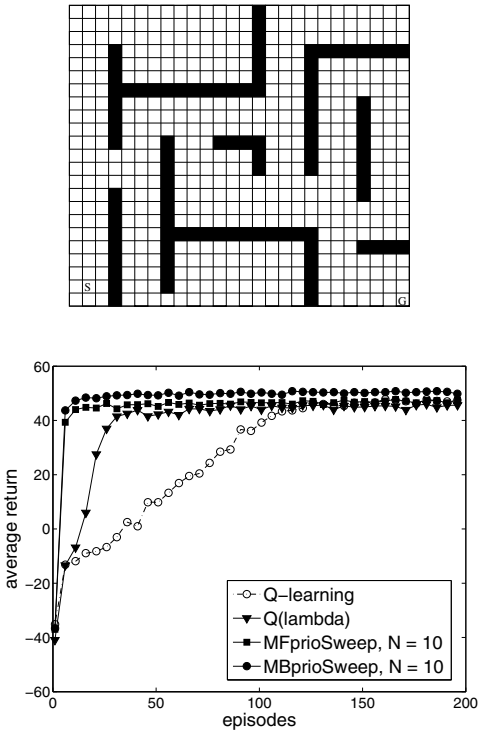

Figure 8. Large maze task (top) and the average return per episode of model-free prioritized sweeping and various alternatives on this task (bottom).

Overall, these empirical results validate the potential of postponed updates for improving the performance of temporal-difference reinforcement learning.

5 Future Work

In the future, we plan to investigate what formal convergence guarantees are obtainable for the various postponed updates methods presented here. We also plan to conduct more extensive empirical comparisons and to combine model-free prioritized sweeping with function approximation, which we hope will yield an effective algorithm for complex domains for which learning models is infeasible.

References

[1] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ., 1957.

[2] R. E. Bellman. A Markov decision process. *Journal of Mathematical Mechanics*, 6:679–684, 1957.

[3] R. I. Brafman and M. Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[4] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[5] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2):209–232, 2002.

[6] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[7] M. L. Puterman and M. C. Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24:1127–1137, 1978.

[8] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[9] A. Strehl and M. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 856–863, 2005.

[10] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. PAC model-free reinforcement learning. In *In: ICML-06: Proceedings of the 23rd international conference on Machine learning*, pages 881–888, 2006.

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

[12] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.

[13] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[14] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):9–44, 1992.