Efficient Planning in MDPs by Small Backups

Harm van Seijen HARM.VANSEIJEN@UALBERTA.CA
Richard S. Sutton SUTTON@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, Alberta, T6G 2E8, Canada

Abstract

Efficient planning plays a crucial role in model-based reinforcement learning. Traditionally, the main planning operation is a full backup based on the current estimates of the successor states. Consequently, its computation time is proportional to the number of successor states. In this paper, we introduce a new planning backup that uses only the current value of a single successor state and has a computation time independent of the number of successor states. This new backup, which we call a *small backup*, opens the door to a new class of model-based reinforcement learning methods that exhibit much finer control over their planning process than traditional methods. We empirically demonstrate that this increased flexibility allows for more efficient planning by showing that an implementation of prioritized sweeping based on small backups achieves a substantial performance improvement over classical implementations.

1. Introduction

In *reinforcement learning* (RL) (Kaelbling et al., 1996; Sutton & Barto, 1998), an agent seeks an optimal control policy for a sequential decision problem in an initially unknown environment. This task is often formalized as a Markov Decision Process (MDP), where the environment provides feedback on the agent's behaviour in the form of a reward signal. The agent's goal is to maximize the expected *return*, which is the discounted sum of rewards over future time steps. An important performance measure in RL is the *sample efficiency*, which refers to the number of environment interactions that is required to obtain a good policy.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

Many solution strategies improve the policy by iteratively improving a *state-value* or *action-value function*, which provide estimates of the expected return under a given policy for (environment) states or state-action pairs, respectively. Different approaches for updating these value functions exist. In terms of sample efficiency, one of the most effective approaches is to estimate the environment model using observed samples and to compute, at each time step, the (action-)value function that is optimal with respect to the model estimate using planning techniques. A popular planning technique used for this is *value iteration* (VI) (Bellman, 1957), which performs sweeps of backups through the state or state-action space, until the (action-)value function has converged.

A drawback of using VI is that it is computationally expensive, making it impractical for domains that require a high action-selection frequency. Fortunately, efficient approximations can be obtained by limiting the number of backups that is performed per time step. A very effective approximation strategy is *prioritized sweeping* (Moore & Atkeson, 1993; Peng & Williams, 1993), which prioritizes backups that are expected to cause large value changes. This paper introduces a new backup that enables a dramatic improvement in the efficiency of prioritized sweeping.

The main idea behind this new backup is as follows. Consider that we are interested in some estimate A that is constructed from a sum of other estimates X_i. The estimate A can be computed using a *full backup*:

$$A \leftarrow \sum_i X_i \,.$$

If the estimates X_i are updated, A can be recomputed by redoing the above backup. Alternatively, if we know that only X_j received a significant value change, we might want to update A for only X_j. Let us indicate the old value of X_j, used to construct the current value of A, as x_j. A can then be updated by subtracting this old value and adding the new value:

$$A \leftarrow A - x_j + X_j \,.$$

This kind of backup, which we call a *small backup*, is computationally cheaper than the full backup. The trade-off is that, in general, more memory is required for storing the estimates x_i associated with A. In planning, where the X estimates correspond to state-value estimates and A corresponds to a state or state-action estimate, this is not a serious restriction, because a full model is stored already. The additional memory required has the same order of complexity as the memory required for storage of the model.

Small backups can be combined in different ways to form composite backups. A composite backup consisting of one small backup for each successor state is equivalent (in effect and computational complexity) as a full backup. However, by combining them in a different way, composite backups can be created with unique properties.

In this paper, we introduce a composite backup that performs one small backup for each *predecessor* of a state. This backup, which we call a *reversed full backup*, has the same computational complexity as a full backup. However, it has the advantage that an accurate estimate of the effect of the backup can be determined very efficiently. This is a critical property for prioritized sweeping. We demonstrate this empirically, by showing that a prioritized sweeping implementation based on reversed full backups yields a large performance improvement over the classical implementation, based on full backups (Moore & Atkeson, 1993).

In addition, we demonstrate the relevance of small backups in domains with severe constraints on computation time, by showing that a method that performs one small backup per time step has an equal computation time complexity as TD(0), the classical method that performs one *sample backup* per time step. Since sample backups introduce sampling variance, they require a step-size parameter to be tuned for optimal performance. Small backups, on the other hand, do not introduce sampling variance, allowing for a parameter-free implementation. We empirically demonstrate that the performance of a method that performs one small backup per time step is similar to the optimal performance of TD(0), achieved by carefully tuning the step-size parameter.

2. Markov Decision Processes

Sequential decision problems are often formalized as *Markov decision processes* (MDPs), which can be described as tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ consisting of $\mathcal{S}$, the set of all states; $\mathcal{A}$, the set of all actions; $\mathcal{P}_{sa}^{s'} = Pr(s'|s,a)$, the transition probability from state $s \in \mathcal{S}$ to state s' when action $a \in \mathcal{A}$ is taken; $\mathcal{R}_{sa} = E\{r|s,a\}$, the reward function giving the expected reward r when action a is taken in state s; and γ, the discount factor controlling the weight of future rewards versus that of the immediate reward.

Actions are taken at discrete time steps $t = 0, 1, 2, ...$ according to a *policy* $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, which defines for each action the selection probability conditioned on the state. The optimal policy π^* maximizes, for each state, the expected *return* G, which is the discounted cumulative reward:

$$G_t = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + ... = \sum_{k=1}^{\infty} \gamma^{k-1}\, r_{t+k}\, ,$$

where r_{t+1} is the reward received after taking action a_t in state s_t at time step t.

Value-function based methods find the optimal policy π^* by searching for the optimal *state-value function* $V^*(s)$ and/or optimal *action-value function* $Q^*(s,a)$, which give the expected return when following the optimal policy, conditioned on a state or state-action pair, respectively.

In *planning*, where the model is given, the optimal value functions can be determined by iteratively improving estimates V of V^* and Q of Q^* by performing *full backups*. A full backup for state s is defined as:

for all a:

$$Q(s,a) \leftarrow \mathcal{R}_{sa} + \gamma \sum_{s'} \mathcal{P}_{sa}^{s'} V(s') \qquad (1)$$

$$V(s) \leftarrow \max_a Q(s,a) \qquad (2)$$

Any ordering of full backups is guaranteed to make V converge to V^* and Q to Q^*, as long as each state is visited infinitely often.

In *reinforcement learning* (RL), the model is unknown. *Model-based* RL methods use samples to update estimates of the transition probabilities, $\hat{\mathcal{P}}_{sa}^{s'}$, and reward function, $\hat{\mathcal{R}}_{sa}$. They typically improve the policy by performing planning backups, such as full backups, using the model estimate as substitute for the true model.

3. Small Backups

This section applies the small backup principle, explained in the introduction, to the action-value backup. We start with the planning case, where the full model is given. Section 3.2 discusses the reinforcement learning case, where the model is initially unknown and has to be learned through interaction with the environment.

3.1. Known Model

To apply the small backup principle to the backup of an action-value (Equation 1), the value estimates of the successor states used to construct the current action-value need to be stored. Let $U_{sa}(s')$ be the value of successor state s' stored by state-action pair (s,a). [1] A small backup for state-action pair (s,a) should not just update the action-value of (s,a); it should also update U_{sa}, in order to ensure that it represents the successor values used to construct the current action-value of (s,a). More specifically, the following relation between $Q(s,a)$ and U_{sa} should be maintained:

$$Q(s,a) = \mathcal{R}_{sa} + \gamma \sum_{s'} \mathcal{P}_{sa}^{s'} U_{sa}(s') \,. \tag{3}$$

Note that for each state-action pair the value $U_{sa}(s')$ can be different, since backups occur at different times, and hence successor values can be different. At most, storage of U can require $\mathcal{O}(\mathcal{S}^2\mathcal{A})$ memory , the same amount as required for storing the transition probabilities.[2]

Definition A small backup for state-action pair (s,a) based on successor state s'' consists of the following backups, performed sequentially:

$$\begin{aligned} \Delta U &\leftarrow V(s'') - U_{sa}(s'') & (4)\\ U_{sa}(s'') &\leftarrow V(s'') & (5)\\ Q(s,a) &\leftarrow Q(s,a) + \gamma\, \mathcal{P}_{sa}^{s''} \Delta U & (6) \end{aligned}$$

Note that the computational cost of a small backup is $\mathcal{O}(1)$. The effect of the backup is equivalent to updating the function U_{sa} with the current value of s'', and then recomputing the action-value of (s,a) using Equation (3), as the following lemma demonstrates.

Lemma 3.1 *If Equation (3) holds, a small backup for state-action pair (s,a) based on successor state s'' has the same effect as the following backups:*

$$\begin{aligned} U_{sa}(s'') &\leftarrow V(s'') & (7)\\ Q(s,a) &\leftarrow \mathcal{R}_{sa} + \gamma \sum_{s'} \mathcal{P}_{sa}^{s'} U_{sa}(s') \,. & (8) \end{aligned}$$

Proof First, note that U_{sa} receive exactly the same backup, so its value after the backups is the same. Second, by substituting the result of backup (7), backup (8) can be written as:

$$Q(s,a) \leftarrow \mathcal{R}_{sa} + \gamma \sum_{s' \neq s''} \mathcal{P}_{sa}^{s'} U_{sa}(s') + \gamma \mathcal{P}_{sa}^{s''} V(s'') \,.$$

Substituting Equation (3) and the result of backup (4) in backup (6) results in the same backup. ∎

Note that Lemma 3.1 implies that once Equation (3) holds, it remains true under small backups. Hence, in the planning case, an initialization of Q and U that satisfies Equation (3) is sufficient for successful application of small backups. An example of such an initialization is $Q(s,a) = \mathcal{R}_{sa}$ for all s,a and $U_{sa}(s') = 0$ for all s,a,s'.

The following theorem formalizes the notion that a small backup is a more incremental version of a backup based on all successor states.

Theorem 3.1 *If Equation (3) holds, sequentially performing small backups of (s,a) for each of its successor states is equivalent to the backup:*

$$Q(s,a) \leftarrow \mathcal{R}_{sa} + \gamma \sum_{s'} \mathcal{P}_{sa}^{s'} V(s') \,.$$

The proof of this theorem follows straightforwardly from Lemma 3.1.

Note that small backups only back up action-values, based on state-values. To get convergence, the state-values also need to be updated, using backup (2).

3.2. Model Learning

In the planning case, an initialization of Q and U that satisfies Equation (3) is sufficient for successful application of small backups. In model-based RL, $\mathcal{P}_{sa}^{s'}$ and $\mathcal{R}_{sa}$ in Equation (3) are replaced by the model estimates $\hat{\mathcal{P}}_{sa}^{s'}$ and $\hat{\mathcal{R}}_{sa}$. These model estimates are updated each time a sample is observed, breaking Equation (3) for the state-action pair corresponding with the sample. Therefore, in the RL case, additional backups have to be performed to restore the equation. If sample (s,a,r,s') is observed, Equation (3) can be restored, after the model update, by performing the backup:

$$Q(s,a) \leftarrow \hat{\mathcal{R}}_{sa} + \gamma \sum_{s'} \hat{\mathcal{P}}_{sa}^{s'} U_{sa}(s') \,. \tag{9}$$

Typically, the model estimate is defined as:

$$\begin{aligned} \hat{\mathcal{P}}_{sa}^{s'} &\leftarrow N_{sa}^{s'}/N_{sa}\\ \hat{\mathcal{R}}_{sa} &\leftarrow R_{sa}^{sum}/N_{sa} \,, \end{aligned}$$

[1]We use the notation $U_{sa}(s')$ rather than $U(s,a,s')$ to emphasize the fact the U is an estimate of the value of s'.

[2]We assume tabular representations throughout this article.

where N_{sa} counts the number of times state-action pair (s, a) is visited, $N_{sa}^{s'}$ counts the number of times s' is observed as successor state of (s, a), and R_{sa}^{sum} is the sum of observed rewards for (s, a). For this model estimate, the effect of the following backup is equivalent to that of backup (9), if the model has just been updated with sample (s, a, r, s'), while its computational cost is only $\mathcal{O}(1)$:[3]

$$Q(s,a) \leftarrow \Big[Q(s,a)(N_{sa}-1) + r + \gamma\, U_{sa}(s')\Big] / N_{sa} \,.$$

Algorithm 1 shows pseudo-code for a simple RL method that performs one small backup after each observed sample. Note that $\hat{\mathcal{R}}_{sa}$ is never explicitly computed, saving time and memory. This algorithm is, to our knowledge, the first model-based RL method whose computation time per observation is independent of the number of successor states.

Algorithm 1 Model Learning with Small Backups

```
1: initialize $Q(s,a)$ arbitrarily for all $s, a$
2: initialize $V(s)$ arbitrarily for all $s$
3: initialize $U_{sa}(s') = V(s')$ for all $s, a, s'$
4: initialize $N_{sa}, N_{sa}^{s'}$ to 0 for all $s, a, s'$
5: loop {over episodes}
6:    initialize $s$
7:    repeat {for each step in the episode}
8:       select action $a$, based on $Q(s, \cdot)$
9:       take action $a$, observe $r$ and $s'$
10:      $N_{sa} \leftarrow N_{sa} + 1$;    $N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$
11:      $Q(s,a) \leftarrow \Big[Q(s,a)(N_{sa}-1) + r + \gamma U_{sa}(s')\Big] / N_{sa}$
12:      $\Delta U \leftarrow V(s') - U_{sa}(s')$
13:      $U_{sa}(s') \leftarrow V(s')$
14:      $Q(s,a) \leftarrow Q(s,a) + \gamma N_{sa}^{s'} / N_{sa} \cdot \Delta U$
15:      $V(s) \leftarrow \max_b Q(s,b)$
16:      $s \leftarrow s'$
17:   until $s$ is terminal
18: end loop
```

Algorithm 1 can be easily transformed in an algorithm for *policy evaluation*, in which the goal is to determine the state-value function V^π for some fixed policy π. This can be achieved by always selecting actions according to π (on line 8) and ignoring the action component in the variables. That is, $N_{sa}, N_{sa}^{s'}$ and U_{sa} are replaced by $N_s, N_s^{s'}$ and U_s. Ignoring the action component of $Q(s, a)$ reduces it to $Q(s) = V(s)$, so $Q(s, a)$ is replaced by $V(s)$ (making line 15 obsolete). In Section 7.1, we compare this policy evaluation variant with TD(0), the classical model-free method for policy evaluation.

[3]Note that this backup is simply a weighted average between the current action-value and $r + \gamma\, U_{sa}(s')$.

3.3. Small Backups versus Sample Backups

In *model-free learning*, samples are used to directly back up the value functions. Hence, no model is required. These backups, called *sample backups*, have in common with small backups that they are based on only a single successor value. Consequently, their computational complexity is independent on the number of successor states as well.

A disadvantage of sample backups, with respect to small backups, is that they introduce sampling variance, caused by a stochastic environment. This requires the use of a step-size parameter to enable averaging over successor states (and rewards). A small backup does not introduce sampling variance, since it is implicitly based on an expectation over successor states. Hence, methods based on small backups do not require tuning of a step-size parameter for optimal performance. In Section 7.1 we perform an empirical comparison between TD(0), which performs one sample backup per time step, and the policy evaluation variant of Algorithm 1, which performs one small backup per time step.

4. Reversed Full Backup

Small backups can be combined in different ways to produce larger, composite backups. In this section, we combine multiple small backups into a composite backup by performing, for each predecessor of a state, one small (action-value) backup, as well as one state-value backup. We call this composite backup a *reversed full backup*. Whereas the full backup backs up a single state-value using all its successor state-values, the reversed full backup backs up all predecessor state-values using one state-value.

Definition A reversed full backup for state s is defined as:

$$\begin{aligned} &\Delta U \leftarrow V(s) - U(s) \\ &U(s) \leftarrow V(s) \\ &\text{for all } (\bar{s}, \bar{a}) \in \mathit{pred}(s): \\ &\qquad Q(\bar{s}, \bar{a}) \leftarrow Q(\bar{s}, \bar{a}) + \gamma\, \mathcal{P}_{\bar{s}\bar{a}}^{s} \Delta U \\ &\qquad V(\bar{s}) \leftarrow \max_b Q(\bar{s}, b) \end{aligned}$$

where $\mathit{pred}(s) = \{(\bar{s}, \bar{a}) \,|\, \mathcal{P}_{\bar{s}\bar{a}}^{s} > 0\}$.

Note that the U function does not contain the subscript sa. Because a reversed full backup backs up all predecessor state-action pairs using the same state value, the U function does not have to be stored for each state-action pair individually. Instead, a single global U can be used, saving memory.

Next, we demonstrate that the average computational complexity of a reversed full backup is equal to that of a full backup. The computational complexity of a full backup (Equations 1 and 2) is $\mathcal{O}(B|\mathcal{A}|)$, where B is the number of successor states of an action. Let D be the number of predecessor state-action pairs of a state. Because a state-value backup occurs after every action-value backup, the max-operator can be implemented efficiently by keeping track of the greedy action:[4] for a non-greedy action a, $V(\bar{s})$ is set equal to $Q(\bar{s}, a)$ if and only if $Q(\bar{s}, a) > V(\bar{s})$. Hence, the max-operator can be implemented at $\mathcal{O}(1)$ complexity for non-greedy actions. On the other hand, if the action is greedy, all values need to be compared, resulting in $\mathcal{O}(|\mathcal{A}|)$ complexity. On average, for D predecessor state-action pairs, $D/|\mathcal{A}|$ pairs contain a greedy action. Therefore, the average computational cost of a reversed full backup is $\mathcal{O}(D \cdot 1) + \mathcal{O}(|\mathcal{A}| \cdot D/|\mathcal{A}|) = \mathcal{O}(D)$. Interestingly, for any MDP, D is on average equal to $B|\mathcal{A}|$, the number of actions multiplied by the number of successors per action, because every action successor corresponds with exactly one state predecessor. Hence, on average the computational complexity of a reversed full backup is equal to that of a full backup.

In the next section, we demonstrate the advantage of reversed full backups over full backups by combining them with prioritized sweeping.

5. Prioritized Sweeping with Reversed Full Backups

Prioritized sweeping (PS) makes the planning step of model-based RL more efficient by using a heuristic (a 'priority') for selecting backups that favours backups that are expected to cause a large value change. A priority queue is maintained that determines which values are next in line for receiving backups. PS methods perform backups in what we call *update cycles*. By adjusting the number of update cycles that is performed per observation, the computation time per observation can be controlled. The classical PS implementation of Moore and Atkeson performs one full backup per update cycle. In this section, we introduce a new PS method that performs one reversed full backup per time step.

An update cycle in the implementation of Moore and Atkeson consists of the following steps. First, the top state is removed from the queue, and receives a full backup. Let s be the top state and ΔV_s the value change caused by the backup. Then, for all predecessor state-action pairs $(\bar{s}, \bar{a})$ a priority p is computed, using:

$$p \leftarrow \hat{\mathcal{P}}^{s}_{\bar{s}\bar{a}} \cdot |\Delta V_s| \,. \qquad (10)$$

For each predecessor state-action pair, the corresponding state is added to the queue, if it is not yet on the queue. If it is already on the queue, but with a priority smaller than p, the priority is upgraded to p.

A weakness of the approach above is that the priority of a state is determined using the value change of only a *single* successor state, while the full backup is based on the values of *all* successor states. When the number of successor states is large, this can result in ineffective priorities. In contrast, when reversed full backups are used, all predecessors are backed up using the same state value. Therefore, a priority that is equal to the value change of this state is a very accurate indicator of the effect of a reversed full backup.

Algorithm 2 shows the pseudo-code for PS with reversed full backups. In Section 7.2, we empirically demonstrate that using reversed full backups instead of full backups leads to a dramatic improvement in performance.

6. Related Work

Peng & Williams (1993) developed a version of prioritized sweeping called Queue-Dyna. The variant of Queue-Dyna for stochastic environments is significantly different than Moore and Atkeson's method. Specifically, it does not use full backups, but backups based on a single successor, like the backups used by our method. However, in contrast to our method, their single-successor backup has a cost of $\mathcal{O}(B)$ instead of $\mathcal{O}(1)$, and they perform only one such backup per update cycle, instead of one for each predecessor. Consequently, the effect of an update cycle of Queue-Dyna on the value functions is a lot smaller than for our method, while the computational cost is similar. We demonstrate this empirically in the Section 7.2.

Andre et al. (1998) developed a PS version for more general models than tabular models, such as DBN models. Applying their method to a tabular model, results in a method similar to the one of Moore and Atkeson.

Wiering & Schmidhuber (1998) developed a PS version that performs a full backup for each predecessor state per update cycle. Consequently, the computational complexity of an update cycle is significantly larger: $\mathcal{O}(B^2|\mathcal{A}|^2)$ instead of $\mathcal{O}(B|\mathcal{A}|)$. Besides that, their method empties the priory queue before each new observation.

McMahan & Gordon (2005) made a version of PS de-

[4]If a state has multiple greedy actions, one of them is chosen as the 'official' greedy action.

Algorithm 2 Prioritized Sweeping with reversed full backups

```
1: initialize $V(s)$ arbitrarily for all $s$
2: initialize $U(s) = V(s)$ for all $s$
3: initialize $Q(s,a) = V(s)$ for all $s, a$
4: initialize $N_{sa}, N_{sa}^{s'}$ to 0 for all $s, a, s'$
5: loop {over episodes}
6:   initialize $s$
7:   repeat {for each step in the episode}
8:     select action $a$, based on $Q(s,\cdot)$
9:     take action $a$, observe $r$ and $s'$
10:    $N_{sa} \leftarrow N_{sa} + 1$;   $N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$
11:    $Q(s,a) \leftarrow \big[Q(s,a)(N_{sa}-1) + r + \gamma U(s')\big]/N_{sa}$
12:    $V(s) \leftarrow \max_b Q(s,b)$
13:    $p \leftarrow |U(s) - V(s)|$
14:    if $s$ is on queue, set its priority to $p$;
       otherwise, add it with priority $p$
15:    for  a number of update cycles do
16:      remove top state $\bar{s}'$ from queue
17:      $\Delta U \leftarrow V(\bar{s}') - U(\bar{s}')$
18:      $U(\bar{s}') \leftarrow V(\bar{s}')$
19:      for all $(\bar{s}, \bar{a})$ pairs with $N_{\bar{s}\bar{a}}^{\bar{s}'} > 0$ do
20:        $Q(\bar{s},\bar{a}) \leftarrow Q(\bar{s},\bar{a}) + \gamma N_{\bar{s}\bar{a}}^{\bar{s}'}/N_{\bar{s}\bar{a}} \cdot \Delta U$
21:        $V(\bar{s}) \leftarrow \max_b Q(\bar{s},b)$
22:        $p \leftarrow |U(\bar{s}) - V(\bar{s})|$
23:        if $s$ is on queue, set its priority to $p$;
           otherwise, add it with priority $p$
24:      end for
25:    end for
26:    $s \leftarrow s'$
27:  until $s$ is terminal
28: end loop
```

signed specifically for planning in stochastic shortest path problems (MDPs with only negative rewards and absorbing goal states). Their method performs multiple iterations of backups, starting from the goal state and working its way backwards. It has the property that it reduces to Dijkstra's algorithm when the environment is deterministic. Their method is not suitable for reinforcement learning, since it assumes the goal state is known.

More recently, Grzes & Hoey (2011) made a PS version for the planning step of R-max (Brafman & Tennenholtz, 2002). Their version improves upon Moore and Atkeson's method in this setting, by exploiting the fact that under R-max values only decrease. While R-max has PAC guarantees, in practise its on-line performance is often poor, because it puts too much emphasis on exploration (Rao & Whiteson, 2012).

PS is one of the main techniques to obtain a good ordering of backups, but it's not the only one. Notable other methods that aim to obtain a good ordering are RTDP (Barto et al., 1995) and LRTDP (Bonet & Geffner, 2003). Both are planning methods for stochastic shortest path problems. They generate simulated episodes based on the model, and backup the states from those episodes using full backups. A strength of these methods is that they can find optimal policies without searching the full state space, by using the initialization of the value functions as heuristic. These methods are not suitable for reinforcement learning, since the simulated episodes require that the goal states are known.

7. Experimental Results

In this section, we evaluate the performance of the policy evaluation variant of Algorithm 1, as well as the performance of Algorithm 2.

7.1. Small backup versus Sample backup

We compare the performance of TD(0), which performs one sample backup per time step, with the policy evaluation variant of Algorithm 1, which performs one small backup per time step.

Their performance is compared on two evaluation tasks, both consisting of 10 states laid out in a circle. State transitions only occur between neighbours. The transition probabilities are generated by a random process. The reward for counter-clockwise transitions is always +1. The reward for clockwise transitions is different for the two tasks. In the first task, a clockwise transition results in a reward of -1; in the second task, it results in a reward of +1. The discount factor γ is 0.95 and the initial state values are 0.

For TD(0), we performed experiments with a constant step-size for values between 0 and 1 with steps of 0.02. In addition, we performed experiments with a decaying, state-dependent step-size, according to

$$\alpha(s) = \frac{1}{d \cdot (N_s - 1) + 1} \quad , \qquad (11)$$

where N_s is the number of times state s has been visited, and d specifies the *decay rate*. We used values of d between 0 and 1 with steps of 0.02. Note that for $d = 0$, $\alpha(s) = 1$, and for $d = 1$, $\alpha(s) = 1/N_s$.

Each time a transition is observed and the corresponding backup is performed, the root-mean squared (RMS) error over all states is determined. The average RMS error over the first 10.000 transitions, normalized with the initial error, determines the performance. Figure 1 shows this performance, averaged over 100 runs. The standard error is negligible (it is in the order of the line thickness in the graphs). Note that the performance for $d = 0$ is equal to the performance for $\alpha = 1$, as it should, by definition. The normalized performance for $\alpha = 0$ is 1, since no learning occurs in this case. For the second task, the optimal step-size is 1, because while the transitions are stochastic, the observed rewards are always the same, hence no averaging is required.

These experiments demonstrate three things. First, the optimal step-size can vary a lot between different tasks. Second, selecting a sub-optimal step-size can cause large performance drops. Third, a small backup, which is parameter-free, has a performance similar to the performance of TD(0) with optimized step-size. Since the computational complexity is the same, the small backup is a better choice than the sample backup. Whether this is true for all (tabular) domains or just for some, is something that needs to be explored further. Keep in mind that a small backup does require a model estimate, so if there are tight constraints on the memory, a sample backup might still be the only option.

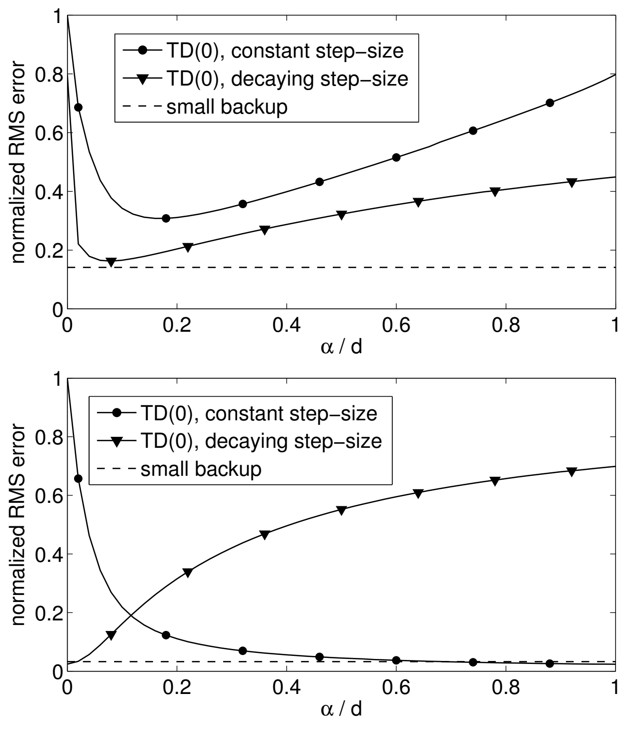

Figure 1. Average RMS error over the first 10.000 observations, normalized by the initial error, for different values of the step-size parameter α, in case of constant step-size, or different values of the decay parameter d, in case of decaying step-size. The top graph corresponds with the first evaluation task; the bottom graph with the second.

7.2. Prioritized Sweeping

We compare the performance of prioritized sweeping with reversed full backups (Algorithm 2) with the PS implementation of Moore & Atkeson, as well as the implementations of Peng & Williams and Wiering & Schmidhuber. As reference, we also show the performance of R-max and the performance obtained by doing value iteration until convergence after each observation. We indicate this last method by 'PS, ∞ update cycles', since all PS methods converge to this method when their number of update cycles goes to infinity. Both these methods required a lot more computation per time step than the PS methods (up to 400 times as much compared to a single update cycle).

We compare the methods on two variations of the maze task depicted in the left of Figure 2. In both tasks, the agent can choose between four actions, corresponding to the four compass directions. In the first task, the agent moves with 80% probability to the neighbouring square corresponding to the compass direction, and with 20% probability it moves at random to one of its four neighbouring squares. In the second task, the number of successor states is larger. The right of Figure 2 shows the relative action outcomes of a 'north' action for the second task. In free space, an action can result in 15 possible successor states, each with equal probability. When the agent is close to a wall, this number decreases. For both tasks, the reward received at each time step is -1 and the discount factor is 0.99.

As exploration strategy, the agent select with 5% probability a random action, instead of the greedy one. On top of that, we use the 'optimism in the face of uncertainty' principle, as also used by Moore & Atkeson. This means that as long as a state-action pair has not been visited for at least M times, its value is defined as some optimistic value (0 in our case), instead of the value based on the model estimate. We optimized M using the value iteration method, resulting in $M = 6$ for the first task, and $M = 4$ for the second task. These values are used for all PS methods. We optimized the parameters of R-max separately.

We performed experiments for 1, 3, 5 and 10 update cycles per time step (except for the method of Wiering & Schmidhuber, for which we skipped the experiment with 10 update cycles). Figure 3 shows the average return over the first 200 episodes for the different methods. The results are averaged over 100 runs. The maximum standard deviation is 0.1 for all methods, except for the method of Peng & Williams, which had a maximum standard deviation of 1.0.

PS with reversed full backups substantially outperforms the other methods on both tasks. In the second task, a single reversed full backup per update cycle is already enough to match the performance in the limit ('PS, ∞ update cycles'). In contrast, its closest competitor, 'PS Moore&Atkeson', has with 10 times as much computation ($5 \cdot 10^{-6}$ versus $0.5 \cdot 10^{-6}$) still not matched this performance.

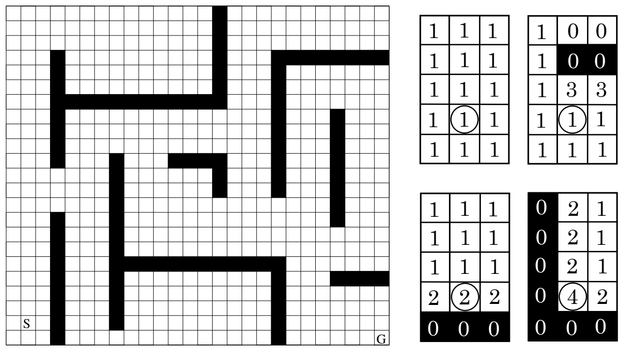

Figure 2. Left, the maze task, in which the agent must travel from S to G. Right, transition probabilities ($\cdot \frac{1}{15}$) of a 'north' action in the second task for different positions of the agent (indicated by the circle) with respect to the walls (black squares).

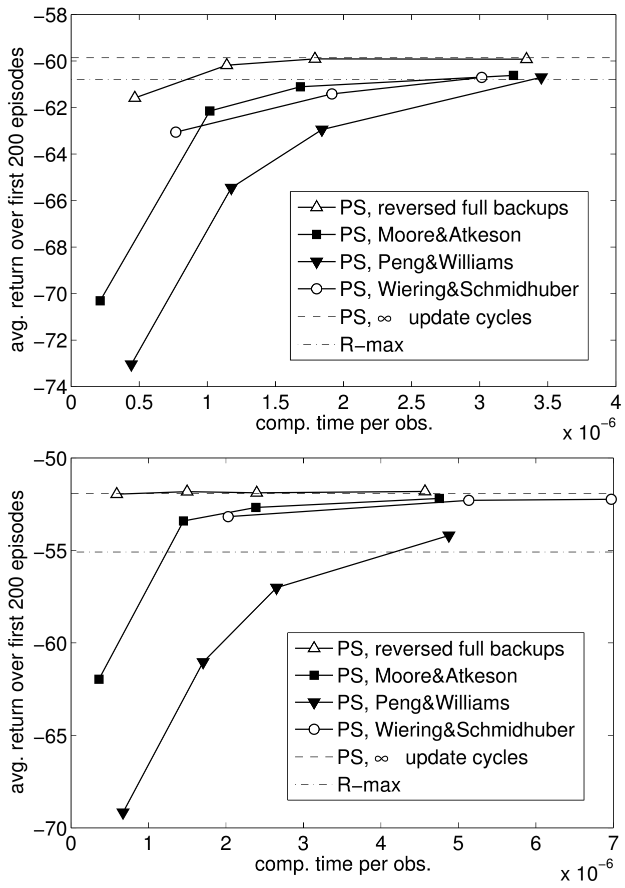

Figure 3. Performance of different PS methods on the first maze task (top), which has 4 successor states per action, and the second maze task (bottom), which has up to 15 successor states per action.

8. Discussion

The results of Figure 3 clearly demonstrate the advantage of reversed full backups over full backups in the context of prioritized sweeping: a certain level of performance can be obtained with much less computation. This is relevant in for example real-time systems, where a high action-selection frequency is often critical, placing bounds on the amount of computation that can be done in between actions. Another relevant application domain is the domain where RL is used in combination with a *simulation model*, i.e., a model that generates samples. Since the action-selection frequency in this domain is fully determined by the computation time per simulation step, improvements in this computation time directly translate in improvements of the total runtime of an experiment.

So far, we assumed that the size of the problem was small enough for the full model to be stored. If the full model cannot be stored in memory, a partial model might be used, that only stores the transition probabilities for a subset of the successor states. Such models require memory anywhere between $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ and $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|))$, depending on how large the subset is. Van Seijen et al. (2011) showed that full convergence to the optimal values can be obtained for such models when they are combined with a separate model-free value-function. Of course, for really large domains (or continuos-state domains) a memory complexity of $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ can already be prohibitive. Moreover, for data efficiency reasons, some form of data generalization might be required. This can be achieved with tabular methods by using state aggregation, where multiple states (or an area in case of continuous-state domains) are combined into a single abstract state.

9. Conclusion

We demonstrated in this paper that the planning step in model-based reinforcement learning methods can be done substantially more efficient by making use of small backups. These backups are finer-grained version of a full backup, which allow for more control over how the available computation time is spend. This makes new, more efficient, update strategies possible. In addition, small backups can be useful in domains with very tight time constraints, offering a parameter-free alternative to sample backups, which were up to now the only feasible option for such domains.

References

Andre, D., Friedman, N., and Parr, R. (1998). Generalized Prioritized Sweeping. *Advances in Neural Information Processing Systems,* 10:1001-1007.

Barto, A. G., Bradtke, S. J., and Singh, S.P. (1995). Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1): 81–138.

Bellman, R. (1957). A Markovian decision process. *Journal of Mathematical Mechanics*, 6:679–684.

Bonet, B. and Geffner, H. (2003). Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. *ICAPS*, 12–21.

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

Grzes, M. and Hoey, J. (2011). Efficient planning in R-max. *AAMAS*.

Kaelbling, L.P., Littman, M.L., and Moore, A.P. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285.

McMahan, H.B. and Gordon, G.J. (2005). Fast Exact Planning in Markov Decision Processes. *ICAPS*.

Moore, A. and Atkeson, C. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, 13:103–130.

Peng, J. and Williams, R.J. (1993). Efficient Learning and Planning Within the Dyna Framework. *Adaptive Behavior*, 1(4):437–454.

Rao, K. and Whiteson, S. (2012). V-MAX: Tempered Optimism for Better PAC Reinforcement Learning. *AAMAS*, 375–382.

Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, Massachussets.

Van Seijen, H, Whiteson, S, van Hasselt, H, and Wiering, M. (2011). Exploiting Best-Match Equations for Efficient Reinforcement Learning. *Journal of Machine Learning Research*, 12:2045–2094.

Wiering, M. and Schmidhuber, J. (1998). Efficient Model-Based Exploration. *SAB*, 223-228.