Journal of Machine Learning Research 12 (2011) 2045-2094 Submitted 3/10; Revised 1/11; Published 6/11

Exploiting Best-Match Equations for Efficient Reinforcement Learning

Harm van Seijen HARM.VANSEIJEN@TNO.NL
Distributed Sensor Systems Group
TNO Defence, Security and Safety
P.O. Box 96864
2509 JG, The Hague, The Netherlands

Shimon Whiteson S.A.WHITESON@UVA.NL
Informatics Institute
University of Amsterdam
Amsterdam, The Netherlands

Hado van Hasselt H.VAN.HASSELT@CWI.NL
Multi-agent and Adaptive Computation Group
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

Marco Wiering MWIERING@AI.RUG.NL
Department of Artificial Intelligence
University of Groningen
Groningen, The Netherlands

Editor: Peter Dayan

Abstract

This article presents and evaluates *best-match learning*, a new approach to reinforcement learning that trades off the sample efficiency of model-based methods with the space efficiency of model-free methods. Best-match learning works by approximating the solution to a set of *best-match equations*, which combine a sparse model with a model-free Q-value function constructed from samples not used by the model. We prove that, unlike regular sparse model-based methods, best-match learning is guaranteed to converge to the optimal Q-values in the tabular case. Empirical results demonstrate that best-match learning can substantially outperform regular sparse model-based methods, as well as several model-free methods that strive to improve the sample efficiency of temporal-difference methods. In addition, we demonstrate that best-match learning can be successfully combined with function approximation.

Keywords: reinforcement learning, on-line learning, temporal-difference methods, function approximation, data reuse

1. Introduction

In *reinforcement learning* (RL) (Kaelbling et al., 1996; Sutton and Barto, 1998), an agent seeks an optimal control policy for a sequential decision problem in an unknown environment. 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. Its goal is to maximize the

expected *return*, which is the cumulative discounted reward. When the sequential decision problem is modeled as a *Markov decision process* (MDP), the agent's policy can be represented as a mapping from each state it may encounter to a probability distribution over the available actions.

There are several approaches for learning the optimal policy of an MDP. *Model-free*, or direct, methods find an optimal policy by using sample experience to directly update the *state values*, which predict the return when following a specified policy, or the *state-action values*, or *Q-values*, which predict the return when taking an action in a certain state and following a specified policy thereafter. Once the optimal state or state-action values have been found, the optimal policy can easily be constructed. A popular model-free approach is *temporal-difference* (TD) learning (Sutton, 1988), which bootstraps value estimates from other values using updates based on the *Bellman equations* (Bellman, 1957). Temporal-difference methods such as Q-learning (Watkins, 1989) and Sarsa (Rummery and Niranjan, 1994; Sutton, 1996) require only $O(|\mathcal{S}||\mathcal{A}|)$ space and are guaranteed to find optimal policies in the limit. However, they often need prohibitively many samples in practice.

Alternatively, *model-based*, or indirect, methods (Sutton, 1990; Moore and Atkeson, 1993; Brafman and Tennenholtz, 2002; Kearns and Singh, 2002; Strehl and Littman, 2005; Diuk et al., 2009) use sample experience to estimate a model of the MDP and then compute the optimal values using this model via off-line planning techniques such as *dynamic programming* (Bellman, 1957). Because the sample experience gathered by the agent is incorporated into the model, it is reused throughout learning. As a result, some model-based methods can find approximately optimal policies with high probability using only a polynomial number of samples (Brafman and Tennenholtz, 2002; Kearns and Singh, 2002; Strehl and Littman, 2005). However, representing the model requires $O(|\mathcal{S}|^2|\mathcal{A}|)$ space, which can be prohibitive in problems with large state spaces.

To avoid this limitation, methods can learn smaller, approximate models that require only a fraction of the space used by full model-based methods. Kearns and Singh (1999) show that, when using such sparse models, it is still possible to learn probably approximately correct policies. However, the performance of such methods is bounded by the quality of the model approximation. Furthermore, since the models may remain incorrect regardless of how much sample experience is gathered, such methods are not guaranteed to find optimal policies even in the limit.

In this article, we present and evaluate *best-match learning*, a new approach for trading off the strengths of model-based and model-free methods. Best-match learning works by approximating the solution to a set of *best-match equations*, which combine a sparse model with a model-free Q-value function constructed from samples not used by the model. We prove that, unlike regular sparse model-based methods, best-match learning is guaranteed to converge to the optimal policy in the tabular case. This guarantee holds even when using a *last-visit model* (LVM), which stores only the last observed reward and transition state for each state-action pair.

In addition, we present an extensive empirical analysis, comparing the performance of best-match learning to several algorithms with similar space requirements. These results demonstrate that best-match learning can outperform regular sparse model-based methods, as well as several model-free methods that strive to improve the sample efficiency of traditional TD methods. These include *eligibility traces* (Sutton, 1988; Watkins, 1989), which update recently visited states in proportion to a trace parameter; *experience replay* (Lin, 1992), which stores experience sequences and uses them for repeated TD updates; and *delayed Q-learning* (Strehl et al., 2006), which uses optimistic Q-value estimates to follow an approximately correct policy except for $O(|\mathcal{S}||\mathcal{A}|\log(|\mathcal{S}||\mathcal{A}|))$ timesteps.

The rest of this article is organized as follows. Section 2 formally defines the RL problem and summarizes some basic theoretical results. As a conceptual stepping stone, Section 3 presents *just-in-time Q-learning*, which postpones updates until the moment of revisit of the corresponding state. We prove that, although just-in-time Q-learning performs the same number of updates as regular Q-learning, the Q-values used in its update targets generally have received more updates. Thus, it can improve performance without extra computation.

Section 4 extends the idea of using improved update targets to best-match learning with an LVM, in which updates are continually revised such that the update targets constructed from them are more accurate. We show that best-match LVM learning is related to eligibility traces, by proving that under certain conditions they compute the same values. However, we also show that in arbitrary MDPs best-match LVM learning, unlike eligibility traces, performs updates that are unbiased with respect to initial state values. We demonstrate empirically that, as a result, it can substantially outperform TD(λ) despite using similar space and computation.

Section 4 also addresses the control case. We propose an efficient best-match LVM algorithm that uses *prioritized sweeping* (Moore and Atkeson, 1993), a well-known technique for prioritizing model-based updates, to trade off extra computation for improved performance. We prove that, despite the use of a sparse model, this approach converges to the optimal Q-values under the same conditions as Q-learning. In addition, we demonstrate empirically that it can substantially outperform competitors with similar space requirements.

Section 5 proposes a best-match learning algorithm that uses an *n-transition model* (NTM), which maintains an estimate of the transition probability for n transition states per state action pair. By tuning n, the space requirements can be controlled. We prove that the algorithm converges to the optimal Q-values for any value of n. We demonstrate empirically the resulting performance improvement over regular sparse model-based methods with equal space requirements, whose performance is bounded by the quality of the model approximation.

Section 6 proposes *best-match function approximation*, which demonstrates that best-match learning is useful beyond the tabular case. In particular, we combine best-match learning with gradient-descent function approximation and show empirically that it can outperform Sarsa(λ) and experience replay with linear function approximation while using similar computation.

Section 7 discusses the article's theoretical and empirical results, Section 8 outlines future work, and Section 9 concludes.

2. Background

Sequential decision problems are often formalized as *Markov decision processes* (MDPs), which can be described as 4-tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ consisting of $\mathcal{S}$, the set of all states; $\mathcal{A}$, the set of all actions; $\mathcal{P}_{sa}^{s'} = P(s'|s,a)$, the transition probability from state $s \in \mathcal{S}$ to state s' when action $a \in \mathcal{A}$ is taken; and $\mathcal{R}_{sa} = E(r|s,a)$, the reward function giving the expected reward r when action a is taken in state s. Actions are selected at discrete timesteps $t = 0, 1, 2, \ldots$ and r_{t+1} is defined as the reward received after taking action a_t in state s_t at timestep t. An optimal policy π^* is a mapping from $\mathcal{S}$ to $\mathcal{A}$ that maximizes the expected discounted return

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

where γ is 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.

In the control case, TD methods seek to learn the optimal action-value function $Q^*(s,a)$, which is the solution to the Bellman optimality equations (Bellman, 1957):

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

By iteratively updating the current estimate $Q_t(s,a)$ each time new experience is obtained, TD methods seek to approximate this function. A common form for these updates is

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

where α is the learning rate and υ_t is the update target. Many update targets are possible, such as the Q-learning (Watkins and Dayan, 1992) update target

$$\upsilon_t = r_{t+1} + \gamma \max_a Q_t(s_{t+1},a) .$$

Once the optimal action-value function has been learned, an optimal policy can be derived by taking the greedy action with respect to this function.

Alternatively, the agent can take a model-based approach (Sutton, 1990; Moore and Atkeson, 1993), in which its experience is used to compute maximum-likelihood estimates of $\mathcal{P}$ and $\mathcal{R}$. Using this model, the agent can compute Q (or the value function V) using dynamic programming methods (Bellman, 1957) such as value iteration (Puterman and Shin, 1978). Each time new experience is gathered, the model is updated and Q recomputed.

In the control case, the agent faces the *exploration-exploitation dilemma*. The agent can either exploit its current knowledge by taking the action that predicts the highest expected return given current estimates, or it can explore by taking a different action in order to improve the accuracy of the Q-value of that action.

Related to the control case is the *policy evaluation* case. In this case, the goal is to estimate the value function $V^\pi(s)$ belonging to policy π. TD methods iteratively improve the current estimate, $V_t(s)$ each time new experience is obtained using the update rule

$$V_{t+1}(s_t) \leftarrow (1-\alpha) V_t(s_t) + \alpha \upsilon_t .$$

An example of an update target for policy evaluation is the TD(0) update target

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

3. Just-In-Time Q-Learning

In this section we present just-in-time (JIT) Q-learning, whose underlying principles form a stepping stone towards best-match learning (introduced in Section 4). Like other *lazy learning* methods, for example, Atkeson et al. (1997), JIT Q-learning postpones updates until they are needed. Wiering and Schmidhuber (1998) showed that by postponing updates a computationally efficient version of Q(λ) can be constructed that does not rely on placing a bound on the trace length. We prove that by postponing Q-learning updates until a state is revisited, the update targets involved receive in general

more updates, while the total number of updates of the current state stays the same. Empirically, we demonstrate that this leads to a performance gain under a range of settings at similar computational cost.

When a Q-learning update is postponed, the values on which the update target is based are from a more recent timestep. This is advantageous, since Q-learning updates cause the expected error in the values to decrease over time (Watkins and Dayan, 1992) and therefore more recent values will be on average more accurate. However, postponing the update of a value for too long can negatively affect performance, since a value that has not been updated might be used for action selection or for bootstrapping other values. We start by showing that updates can be postponed until their corresponding states are revisited, without negatively affecting performance.

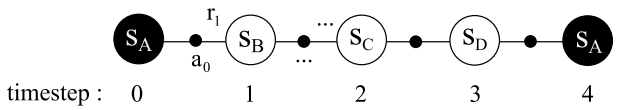

Figure 1: A state transition sequence in which the initial state s_A is revisited at timestep 4. The small black dots in between states represent actions.

Consider the state-action sequence in Figure 1. State s_A is visited at timestep 0 and revisited at timestep 4. With the regular Q-learning update, the Q-value of state-action pair (s_A, a_0) gets updated at timestep 1:

$$Q_1(s_A,a_0) = (1-\alpha)Q_0(s_A,a_0) + \alpha\left[r_1 + \gamma \max_a Q_0(s_B,a)\right],$$

while at timesteps $2-4$ no update of (s_A,a_0) occurs, and therefore $Q_4(s_A,a_0) = Q_1(s_A,a_0)$. The update of the Q-value of (s_A,a_0) at timestep 1 can be considered premature, since the earliest use of its value is in the update target for (s_D,a_3), which uses $Q_3(s_A,a_0)$. Therefore, the update of the Q-value of (s_A,a_0) can be postponed until at least timestep 3 without negatively affecting the update target for (s_D,a_3). When the update of (s_D,a_3) is also postponed, the earliest use of the Q-value of (s_A,a_0) occurs at timestep 4, where it is used for action selection. Thus, if we postpone the update of all state-action pairs, the update of the Q-value of (s_A,a_0) can be postponed until the timestep of its revisit, without causing dependent state values or the action selection procedure to use a value of (s_A,a_0) that has not been updated. We call this type of update a *just-in-time update*, since the update is postponed until just before the updated value is needed.

To denote the Q-values resulting from just-in-time updates we use $\tilde{Q}$ throughout this section. With just-in-time updates, no updates of (s_A,a_0) occur at timesteps 1-3, so $\tilde{Q}_3(s_A,a_0) = \tilde{Q}_0(s_A,a_0)$. Instead, an update occurs when s_A is revisited:

$$\tilde{Q}_4(s_A,a_0) = (1-\alpha)\tilde{Q}_3(s_A,a_0) + \alpha\left[r_1 + \gamma \max_a \tilde{Q}_3(s_B,a)\right].$$

The regular and just-in-time update for (s_A,a_0) can be written in a more similar form by expressing the value at timestep 4 in terms of the value at timestep 0:

$$\begin{aligned} Q_4(s_A,a_0) &= (1-\alpha)Q_0(s_A,a_0) + \alpha\left[r_1 + \gamma \max_a Q_0(s_B,a)\right], \\ \tilde{Q}_4(s_A,a_0) &= (1-\alpha)\tilde{Q}_0(s_A,a_0) + \alpha\left[r_1 + \gamma \max_a \tilde{Q}_3(s_B,a)\right]. \end{aligned} \tag{1}$$

This formulation highlights the difference between the two update types. At timestep 4, under both update schemes, the Q-value of (s_A, a_0) has received one update based on the same experience sample. However, a just-in-time update uses the most recent value of the Q-values of s_B, while a regular update uses the value at the timestep of the initial visit of s_A. By defining t^* as the timestep of the previous visit of state s_t, we can write the two update types more generally as

$$Q_t(s_t, a_{t^*}) = (1-\alpha)Q_{t^*}(s_t, a_{t^*}) + \alpha[r_{t^*+1} + \gamma \max_a Q_{t^*}(s_{t^*+1}, a)], \quad (2)$$

$$\tilde{Q}_t(s_t, a_{t^*}) = (1-\alpha)\tilde{Q}_{t^*}(s_t, a_{t^*}) + \alpha[r_{t^*+1} + \gamma \max_a \tilde{Q}_{t-1}(s_{t^*+1}, a)]. \quad (3)$$

Note that we express the update target using only values from the past, making an implementation easier to interpret. Note also that while $s_t = s_{t^*}$ per definition (because s_t is revisited), s_{t^*+1} does not have to be equal to s_{t+1}, since the state transition from s_t can be stochastic. Also, a_{t^*} is in general not equal to a_t.

When comparing the two update targets in more detail, two cases can be distinguished. See Figure 2 for an example of each case. In the first case, state s_B is not revisited before the revisit of state s_A. In this case, neither update type makes use of an updated Q-value for s_B in the update target for s_A. The regular update does not since it uses the values of s_B at timestep t^*, and the just-in-time update does not since s_B is not revisited and therefore no update has occurred yet at timestep $t-1$. In the second case, state s_B has been revisited before the revisit of s_A. The regular update still uses the value of s_B from timestep t^* and therefore does not use an updated value. The just-in-time update on the other hand does use an updated value, since this update occurred at the revisit of s_B. Note that for a returning action ($t^* = t-1$), both update types have exactly the same form and this can therefore be treated as an example of case 1. From these two cases, we can deduce the following theorem, which is proven in Appendix A.

Theorem 1 *Given the same experience sequence, each Q-value from the current state has received the same number of updates using JIT updates (Equation 3) as using regular updates (Equation 2). However, each Q-value in the update target of a JIT update has received an equal or greater number of updates as in the update target of the corresponding regular update.*

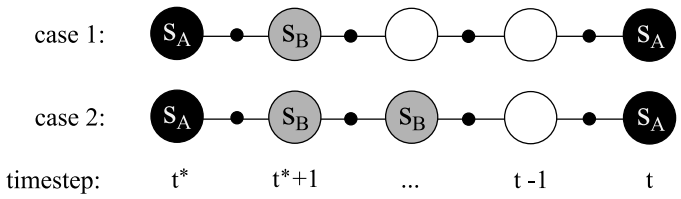

Figure 2: Two cases in which state s_A is revisited. In the first case, neither a regular update nor a just-in-time update make use of an updated value for s_B in the update target of s_A, while in the second case a just-in-time update does.

Algorithm 1 shows pseudocode for the implementation of just-in-time (JIT) Q-learning. The agent stores the reward and transition state received upon the last visit of a state, that is, the *last-visit sample*, in $R'(s)$ and $S'(s)$ respectively, while the action taken at the last visit of a state is stored

Algorithm 1 JIT Q-Learning

initialize $Q(s,a)$ arbitrarily for all s,a
initialize $S'(s) = \emptyset$ for all s
loop {over episodes}
  initialize s
  repeat {for each step in the episode}
    if $S'(s) \neq \emptyset$ **then**
      $Q(s,\bar{a}) \leftarrow (1-\alpha^{s\bar{a}}) \cdot Q(s,\bar{a}) + \alpha^{s\bar{a}} [R'(s) + \gamma \max_{a'} Q(S'(s),a')]$ $//\,\bar{a} = A(s)$
    end if
    select action a, based on $Q(s,\cdot)$
    take action a, observe r and s'
    $S'(s) \leftarrow s'$; $R'(s) \leftarrow r$; $A(s) \leftarrow a$
    $s \leftarrow s'$
  until s is terminal
end loop

in $A(s)$. If $S'(s) = \emptyset$, state s has not been visited yet and no update can be performed. Note that the last-visit sample is not reset at the end of an episode, but maintained across episodes.

Because JIT Q-learning uses more recent values in its update targets than regular Q-learning, we expect a performance improvement over regular Q-learning. We test this hypothesis by comparing the performance of JIT Q-learning with regular Q-learning on the Dyna Maze task (Sutton, 1990). In this navigation task, depicted in Figure 3, the agent has to find its way from start to goal. The agent can choose between four movement actions: up, down, left and right. All actions result in 0 reward, except for when the goal is reached, which results in a reward of +1. The discount factor γ is set to 0.95. We use a deterministic as well as a stochastic environment to test the generality of the hypothesis. In the stochastic version, 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.

To compare performance, we measure the average return each method accrues from the start state during the first 100 episodes in the deterministic case, averaged over 5000 independent runs per method. For the stochastic version, we measure the return during the first 200 episodes. Each method uses ε-greedy action selection with $\varepsilon = 0.1$. In the deterministic case, we use a constant learning rate of 1, while in the stochastic case we use an initial learning rate α_0 of 1 that is decayed in the following manner:[1]

$$\alpha^{sa} = \frac{\alpha_0}{d \cdot [n(s,a) - 1] + 1}, \tag{4}$$

where $n(s,a)$ is the total number of times action a has been selected in state s. Note that for $d = 0$, $\alpha^{sa} = \alpha_0$, while for $d = 1$, $\alpha^{sa} = \alpha_0 / n(s,a)$. We optimize the learning rate decay d between 0 and 1 by taking the decay rate with the maximum average return over the measured number of episodes. We use two different initialization schemes for the Q-values to determine whether the performance difference depends on initialization. We use optimistic initialization, by initializing the Q-values to 20, and pessimistic initialization, by setting the Q-values to 0.

1. This decay is similar to the more common form $\frac{c_1}{c_2 + n(s,a)}$, but with the free parameters re-arranged.

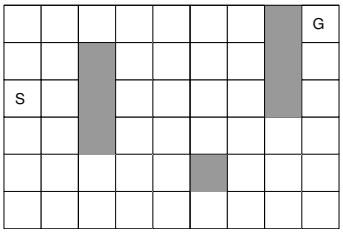

Figure 3: The Dyna Maze task, in which the agent must travel from *S* to *G*. The reward is +1 when the goal state is reached and 0 otherwise.

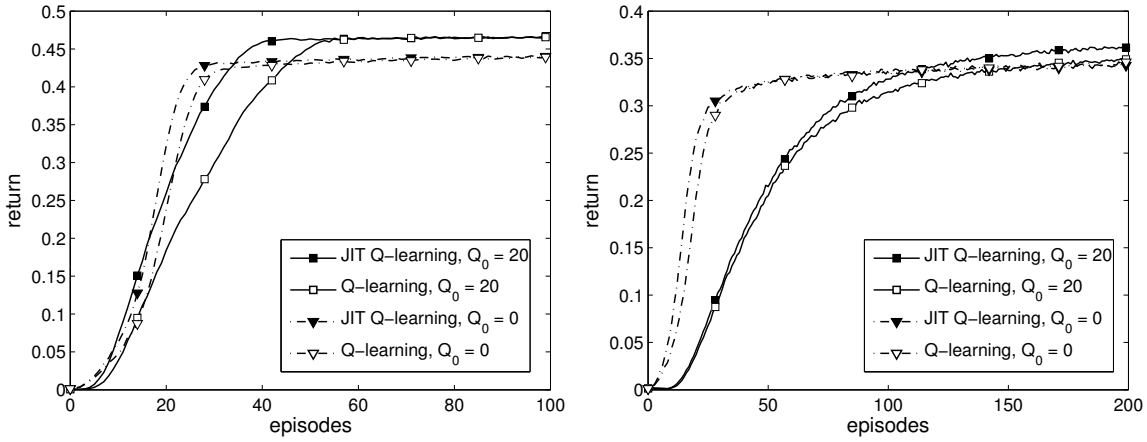

Figure 4: Comparison of the performance of JIT Q-learning and regular Q-learning on the deterministic (left) and stochastic (right) Dyna Maze task for two different initialization schemes.

	deterministic - 100 eps.			stochastic - 200 eps.		
	d	average return	standard error	d	average return	standard error
Q-learning, $Q_0 = 0$	0	0.3506	0.0004	1.0	0.3039	0.0003
JIT Q-learning, $Q_0 = 0$	0	0.3628	0.0004	1.0	0.3083	0.0003
Q-learning, $Q_0 = 20$	0	0.3438	0.0002	0.005	0.2562	0.0002
JIT Q-learning, $Q_0 = 20$	0	0.3714	0.0002	0.010	0.2674	0.0002

Table 1: The performance of JIT Q-learning and regular Q-learning on the Dyna Maze task and the optimal learning rate decay d.

Figure 4 plots the return as a function of the number of episodes, while Table 1 shows the average return and optimal learning rate. The computation time for both methods was similar. JIT Q-learning outperforms regular Q-learning in the deterministic as well as the stochastic environ-

ment and for both types of initialization, although not always by a large margin. This confirms our intuition that, since JIT Q-learning uses values from a later time which are in general more accurate, a performance benefit is gained over regular Q-learning in a broad range of settings. The performance benefit in the deterministic case can be explained by exploration, which causes the order in which states are visited to change despite the deterministic state transitions.

4. Best-Match Last-Visit Model

In this section, we demonstrate that updates can be postponed much further than is done by JIT Q-learning, without negatively affecting other updates, when *best-match updates* are performed. Best-match updates are updates that can correct previous updates when more recent information becomes available. This insight leads to the derivation of the *best-match last-visit model equations*, which combine a *last-visit model* (LVM), consisting of the last experienced reward and transition state for each state-action pair, with *model-free Q-values*, constructed from model-free updates of all observed samples, except the ones stored in the LVM. We present an evaluation as well as a control algorithm based on solving these equations and empirically demonstrate that these methods can outperform competitors with similar space requirements.

4.1 Best-Match LVM Equations

In the example presented in Section 3, the update of $Q(s_A, a_0)$ is postponed until state s_A is revisited. In this section, we demonstrate that the update can be postponed even further in the case that a different action is selected upon revisit. Since we will consider multiple updates per timestep in this section, we denote the Q-value function using two iteration indices: t and i. Each time an update occurs, i is increased, while each time an action is taken, t is increased and i is reset to 0. Therefore, if I denotes the total number of updates that occurs at time t, by definition $Q_{t,I} = Q_{t+1,0}$. Action selection at time t is based on $Q_{t,I}$. Using this convention, the regular Q-learning update can be written as

$$Q_{t+1,1}(s_t, a_t) = (1-\alpha)Q_{t+1,0}(s_t, a_t) + \alpha[r_{t+1} + \max_{a'} Q_{t+1,0}(s_{t+1}, a')]\,.$$

Now consider the example shown in Figure 5, which extends Figure 1 to include a second revisit of s_0 at timestep $t = 7$. Suppose that a different action is selected on the first revisit, that is, $a_4 \neq a_0$. Using just-in-time updates, the Q-value of state-action pair (s_A, a_0) gets updated at time $t = 4$. Using the two indices convention we can rewrite Equation 1 as[2]

$$Q_{4,1}(s_A, a_0) = (1-\alpha)Q_{1,0}(s_A, a_0) + \alpha[r_1 + \gamma \max_{a} Q_{4,0}(s_B, a)]\,. \tag{5}$$

To perform this update, the experience set (r_1, s_B) resulting from taking action a_0 in s_A is temporarily stored. With JIT Q-learning, this experience is stored per state. If the state is revisited and a new action is taken, the previous experience is overwritten and lost. However, if the experience is stored per state-action pair, then the previous experience is not overwritten until the same action is selected again. If the same action is not selected upon revisit, the experience can be used again

2. We use Q now instead of $\tilde{Q}$, since the only purpose of the tilde was to distinguish it from the Q-values of regular Q-learning.

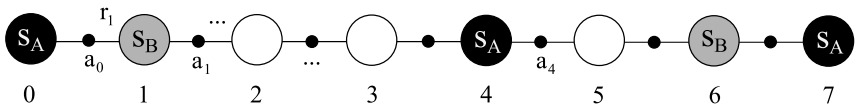

Figure 5: A state transition sequence in which best-match updates can enable further postponing. Timesteps are shown below each state.

to redo the update at a later time, using more recent values for the next state. In the example from Figure 5, the update of (s_A, a_0) can be redone at timestep 7:

$$Q_{7,1}(s_A, a_0) = (1-\alpha)Q_{1,0}(s_A, a_0) + \alpha[r_1 + \gamma \max_a Q_{7,0}(s_B, a)]. \quad (6)$$

Since state s_B is revisited at timestep 6, (s_B, a_1) has received an extra update and therefore $Q_{7,0}(s_B, a_1)$ is likely to be more accurate than $Q_{4,0}(s_B, a_1)$.

Equation 6 is not equivalent to a (postponed) Q-learning update, in contrast to Equation 5, since $Q_{1,0}(s_A, a_0)$ is not equal to $Q_{7,0}(s_A, a_0)$ due to the update at timestep 4. Equation 6 corrects the update from timestep 4, by redoing it using the most recent Q-values for the update target. We call this update a *best-match update* (this name will be explained later in the section), while we call $Q_{1,0}(s_A, a_0)$ the *model-free Q-value* of (s_A, a_0).

Before formally defining a best-match update, we define the last-visit experience and the model-free Q-values.

Definition 2 *The last-visit experience of state-action pair (s,a) denotes the last-visit reward, $R'_t(s,a)$, that is, the reward received upon the last visit of (s,a), and the last-visit transition state, $S'_t(s,a)$, that is, the state transitioned to upon the last visit of (s,a). For a state-action pair that has not yet been visited, we define $R'_t(s,a) = \emptyset$ and $S'_t(s,a) = \emptyset$.*

The LVM consists of the last-visit experience from all state-action pairs.

Definition 3 *The model-free Q-value of a state-action pair (s,a), $Q^{mf}_t(s,a)$, is a Q-value that has received updates from all observed samples except those stored in the LVM, that is, $R'_t(s,a)$ and $S'_t(s,a)$. For a state-action pair that has not yet been visited, we define $Q^{mf}_t(s,a) = Q_{0,0}(s,a)$.*

While Q can be updated multiple times per timestep, Q^{mf} is updated only once per timestep. Therefore, it is uses a single time index t. We define a best-match update as:

Definition 4 *A best-match update combines the model-free Q-value of a state-action pair with its last-visit experience from the same timestep according to*

$$Q_{t,i+1}(s,a) = (1-\alpha)Q^{mf}_t(s,a) + \alpha[R'_t(s,a) + \gamma \max_{a'} Q_{t,i}(S'_t(s,a), a')].$$

Using best-match updates to extend the postponing period of a sample update requires additional computation, as the agent typically performs multiple best-match updates per timestep. In the example, at timestep 7 the agent redoes the update of $Q(s_A, a_0)$, but also performs an update of $Q(s_A, a_4)$.

The model-free Q-value function is updated only once per timestep. Specifically, at timestep $t+1$ Q^{mf} is updated according to

$$Q^{mf}_{t+1}(s_t,a_t) = Q_{t+1,0}(s_t,a_t)\,. \tag{7}$$

Assuming (s_t,a_t) has received a best-match update at timestep t, Equation 7 is equivalent to the update

$$Q^{mf}_{t+1}(s_t,a_t) = (1-\alpha)Q^{mf}_t(s_t,a_t) + \alpha[R'_t(s_t,a_t) + \gamma \max_{a'} Q_{t,i}(S'_t(s_t,a_t),a')]\,,$$

where the value of i depends on the order of best-match updates at timestep t. After Q^{mf} has been updated, the last-visit experience for (s_t,a_t) is overwritten with the new experience

$$\begin{aligned} R'_{t+1}(s_t,a_t) &= r_{t+1}\,, \\ S'_{t+1}(s_t,a_t) &= s_{t+1}\,. \end{aligned}$$

In the approach described above, best-match updates are used to postpone the update from a sample without negatively affecting other updates or the action selection process. However, best-match updates can be exploited far beyond simply avoiding these negative effects. As an example, consider the state-action sequence in Figure 6. s_B is not revisited before the revisit of s_A. With the update strategy described above, best-match updates occur only when a state is revisited. Consequently, the experience from (s_B,a_1) is not used in the update target of (s_A,a_0). However, it is not necessary to wait for a revisit of s_B to perform a best-match update. Instead, it can be performed at the moment it is needed: when s_A is revisited. Thus, if at timestep 3 the agent performs a best-match update of $Q(s_B,a_1)$, before updating $Q(s_A,s_0)$, the latter update will exploit more recent Q-values for s_B, just as if s_B had been revisited.

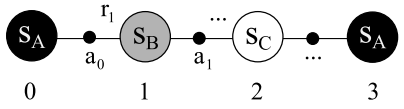

Figure 6: A state transition sequence in which s_B is not revisited. Timesteps are shown below each state.

Taking this idea further, the agent can first update the Q-values of s_C before updating the Q-values of s_B. In other words, the agent uses the Q-values of s_A to perform a best-match update of s_C, then performs a best-match update of s_B and finally updates s_A. However, once the Q-values of s_A have changed, it is possible to further improve the Q-values of s_C by performing a new best-match update. The new Q-values of s_C can then be used to redo the update of s_B, which in turn can be used to re-update s_A. This process can repeat until the Q-values reach a fixed point, which is the solution to a system of $|\mathcal{S}||\mathcal{A}|$ *best-match LVM equations*. We call this solution the *best-match Q-value function*, Q^B, which forms the best match between the LVM and the model-free Q-values.

Definition 5 *The best-match LVM equations at timestep t are defined as*

$$Q^B_t(s,a) = \begin{cases} (1-\alpha^{sa}_t)Q^{mf}_t(s,a) + \alpha^{sa}_t\,[R'_t(s,a) + \gamma \max_c Q^B_t(S'_t(s,a),c)] & \text{if } S'_t(s,a) \neq \emptyset \\ Q^{mf}_t(s,a) & \text{if } S'_t(s,a) = \emptyset\,. \end{cases}$$

There are different ways to look at these equations. One way is to see them as the limit case of redoing updates using (in general) increasingly more accurate update targets. Another way is to see them as Bellman optimality equations based on an induced model. For state-action pair (s,a) this induced model can be described as a transition with probability α to state $S'(s,a)$ with a reward of $R'(s,a)$ and a transition with probability $1-\alpha$ to a terminal state s_T (with a value of 0) and a reward of $Q^{mf}(s,a)$ (see Figure 7).[3]

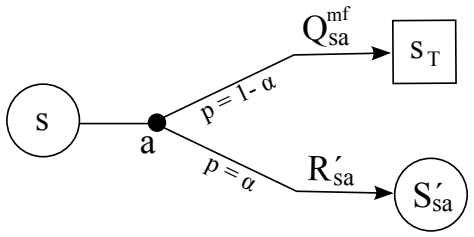

Figure 7: Illustration of the induced model for state-action pair (s,a) corresponding with the best-match LVM equations. The small black dot represents the stochastic action a leading with probability α to state $S'(s,a)$ and with probability 1-α to state s_T.

The advantage of solving the Bellman optimality equations for this induced model, compared to solving it using only the LVM, is that the bias towards the samples in the LVM can be controlled using the learning rates. With annealing learning rates, the transition probability to $S'_t(s,a)$ is decreased over time in favor of transition to the terminal state. On the other hand, when using only the LVM, the solution of the Bellman equations depends only on the samples of the LVM and does not take into account any previous samples. Clearly, in a stochastic environment, this will lead to a sub-optimal policy. Also when the solution is not computed exactly, but approximated by only performing a finite number of updates at each timestep (which is the case for any practical algorithm), using the induced model leads to a better performance, because of the strong bias towards the most recent samples that occurs when using only the LVM.

Section 4.3 discusses how to solve the best-match equations. However, we first discuss the policy evaluation case, for which analogous equations can be defined.

Definition 6 *The best-match LVM equations for state values at time t are*

$$V_t^B(s) = \begin{cases} (1-\alpha_t^s)V_t^{mf}(s) + \alpha_t^s\left[R'_t(s) + \gamma V_t^B(S'_t(s))\right] & \text{if } S'_t(s) \neq \emptyset \\ V_t^{mf}(s) & \text{if } S'_t(s) = \emptyset. \end{cases}$$

The model-free state values are updated according to $V_{t+1}^{mf}(s_t) = V_{t+1,0}(s_t)$.

While in general the value function V can be seen as a special case of the action-value function Q (with all states only having a single action), V has a linear set of best-match equations, in contrast to Q, a property we exploit in best-match LVM evaluation.

3. We assume $S'_t(s,a) \neq \emptyset$ for (s,a) in this case.

4.2 Best-Match LVM Evaluation

In the evaluation case, the best-match LVM equations form a linear set that can be solved exactly. This section proposes an algorithm that does so in a computationally efficient way, using updates that are unbiased with respect to the initial state values.

The algorithm is based on two observations. First, not all $|\mathcal{S}|$ best-match equations necessarily depend on each other. The subset of equations needed to compute the best-match value for s_t can be found by iterating through the sequence of last-visit transition states, starting with $S'(s_t)$. The corresponding N best-match equations form the linear set of equations to solve. For readability, we write s_t as $s_{[0]}$ and use the notation $s_{[n]} = S'(s_{[n-1]})$ and $r_{[n]} = R'(s_{[n-1]})$ for the subsequent transition state and reward. In addition, we use $\alpha^{[n]}$ for $\alpha^{s_{[n]}}$. The equations can now be written as

$$V^B(s_{[n]}) = (1-\alpha^{[n]})V^{mf}(s_{[n]}) + \alpha^{[n]}\left[r_{[n+1]} + \gamma V^B(s_{[n+1]})\right], \quad \text{for all } n \in [0, N-1].$$

Second, the last state of this sequence, $s_{[N]}$, is always either a terminal state or the current state. Furthermore, none of the intermediate states can appear twice, making the N equations independent. This can be proven by contradiction. First, assume that the sequence has a dead-end, that is, ends with a state for which $S' = \emptyset$. This is impossible because it would cause the agent to get stuck in this state, preventing it from reaching the current state. Since last-visit information is maintained across episodes, $s_{[N]}$ is a terminal state if the path followed after the previous visit of s_t led to a terminal state. Next, assume the sequence contains the same intermediate state twice. After the second visit of this intermediate state, the subsequent sequence would be the same as after the first visit, since there is only a single last-visit next state defined per state. This would create an infinite sequence of next states, also preventing the agent from reaching the current state.

The set of equations can be solved by backwards substituting the equations, that is, substituting the equation for $V^B(s_{[n+1]})$ in the one for $V^B(s_{[n]})$ and so on until a single equation for $V^B(s_{[0]})$ remains of the form

$$V^B(s_{[0]}) = c_A + c_B V^B(s_{[N]}),$$

with c_A and c_B defined as

$$c_A = \sum_{i=0}^{N-1} \left((1-\alpha^{[i]})V^{mf}(s_{[i]}) + \alpha^{[i]} r_{[i+1]}\right) \prod_{k=0}^{i-1} \gamma\alpha^{[k]}, \tag{8}$$

$$c_B = \prod_{i=0}^{N-1} \gamma\alpha^{[i]}. \tag{9}$$

If $s_{[N]}$ is a terminal state, its value is 0 and $V^B(s_t) = c_A$. On the other hand, if $s_{[N]} = s_t$ then $V^B(s_t) = c_A/(1-c_B)$.

Algorithm 2 shows pseudocode of the on-line policy evaluation algorithm, which computes the best-match value of the current state at each timestep. Lines 7-12 compute the values of c_A and c_B in a forward, incremental way by going from one next state to the other. Note that it is not necessary to store V^{mf} and R' separately, since they are always used in the same combination, $(1-\alpha)V^{mf}(s) + \alpha R'(s)$, which is stored in a single variable, V_r^{mf}, saving space and computation. Line 20 combines the assignments $V^{mf}(s_t) = V(s_t)$, $R'(s_t) = r_{t+1}$ and the computation of V_r^{mf} in a single update. Note that the algorithm makes use of the just-in-time learning principle, that is, updating states at the moment of their revisit. In JIT Q-learning, it is used to improve the

Algorithm 2 Best-Match LVM Evaluation

initialize $V(s)$ arbitrarily for all s
initialize $S'(s) = \emptyset$ for all s
loop {over episodes}
  initialize s
  repeat {for each step in the episode}
    if $S'(s) \neq \emptyset$ **then**
      $c_A \leftarrow V_r^{mf}(s)$; $c_B \leftarrow \gamma\alpha^s$; $s' \leftarrow S'(s)$; $n \leftarrow 0$
      while $s' \neq s \wedge s'$ is not terminal **do**
        $c_A \leftarrow c_A + c_B \cdot V_r^{mf}(s')$
        $c_B \leftarrow c_B \cdot \gamma\alpha^{s'}$
        $s' \leftarrow S'(s')$
      end while
      if $s' = s$ **then**
        $V(s) \leftarrow c_A/(1-c_B)$
      else
        $V(s) \leftarrow c_A$
      end if
    end if
    take action $\pi(s)$, observe r and s'
    $V_r^{mf}(s) \leftarrow (1-\alpha^s)V(s) + \alpha^s \cdot r$
    $S'(s) \leftarrow s'$; $s \leftarrow s'$
  until s is terminal
end loop

performance without increasing the computation cost, while in the best-match evaluation algorithm it is used to efficiently compute the best-match values.

Algorithm 2 is an on-line algorithm that computes at each timestep the best-match value of the current state. We define the off-line version as one that computes at the end of each episode the best-match values of the states that were visited during that episode. This off-line algorithm is related to off-line TD(λ), as demonstrated by the following theorem. We prove this theorem in Appendix B.

Theorem 7 *For an episodic, acyclic, evaluation task, off-line best-match LVM evaluation computes the same values as off-line TD(λ) with $\lambda_t = \alpha_t(s_t)$.*

For acyclic tasks, that is, episodic tasks with no revisits of states within an episode, $TD(\lambda)$ with $\lambda_t = \alpha_t(s_t)$ can perform TD updates that are unbiased with respect to the initial values (Sutton and Singh, 1994). Because of Theorem 7, this also holds for best-match LVM evaluation. However, in contrast to $TD(\lambda)$, best-match LVM evaluation can perform unbiased updates for any MDP, as we demonstrate with the following theorem, also proven in Appendix B.

Theorem 8 *The state values computed by the on-line best-match LVM evaluation algorithm (Algorithm 2) are unbiased with respect to the initial state values, when the initial learning rates $\alpha_0(s)$ are set to 1 for all s.*

Because best-match LVM evaluation can perform unbiased updates for any MDP, it can often substantially outperform TD(λ) while requiring similar space and computation. We demonstrate

this empirically using the two tasks shown in Figure 8. Besides comparing against TD(λ), we also compare against experience replay (Lin, 1992), which stores the n last experience samples and uses them for repeated TD updates.

Task A features a small circular network consisting of four identical states, each having a deterministic transition to a neighbor. The reward received after each transition is +1. Task B is a stochastic variation on the first task, with stochastic transitions and a reward drawn from a normal distribution with mean 1 and standard deviation 0.5. The discount factor is 0.95, resulting in a state value of 20 on both tasks for all states. We compare the RMS error of the current state value $V_t(s_t)$ for all three methods. For experience replay, we performed a TD update for each of the last 4 samples at every timestep, resulting in a computation time similar to best-match LVM and TD(λ). In addition, we implemented a version where all observed samples are stored and updated at each timestep. The learning rate is initialized to 1 and decayed according to

$$\alpha^s = \frac{\alpha_0}{d \cdot [n(s) - 1] + 1}.$$

where $n(s)$ is the total number of times state s has been visited. We optimize d as well as λ between 0 and 1. Results are averaged over 5000 runs.

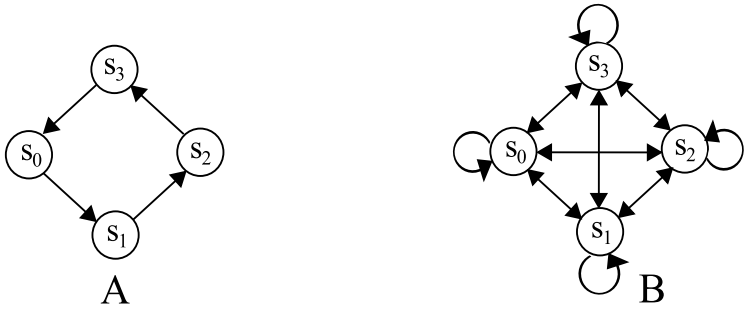

Figure 8: Two tasks for policy evaluation. Task A has deterministic state transitions and a deterministic reward of +1, while task B has stochastic transitions and a reward drawn from a normal distribution with mean +1 and standard deviation 0.5.

Figure 9 shows the experimental results in these tasks. In task A, at timestep 4 the start state is revisited and the RMS error for best-match LVM drops to 0. The reason is that in the deterministic case the last-visit model is equal to the full model once every state has been visited. Furthermore, with learning rates of 1, the best-match LVM equations reduce to the Bellman optimality equations. Therefore best-match LVM effectively performs model-based learning. TD(λ), on the other hand, has to incrementally improve upon the initial values of 0. The spiky behavior of TD(λ) is caused by the combination of a λ of 1, with zero learning rate decay (which were the optimal settings in this case). Experience replay has a performance in between best-match LVM and TD(λ). In task B, the RMS error drops more smoothly. Best-match LVM again substantially outperforms TD(λ) and experience replay, even when all samples are stored and updated. The total computation time for the 5000 runs was marginally higher for experience replay with N=4, which has to maintain a queue of recent samples, than for best-match LVM and TD(λ): on task A, around 90 ms compared

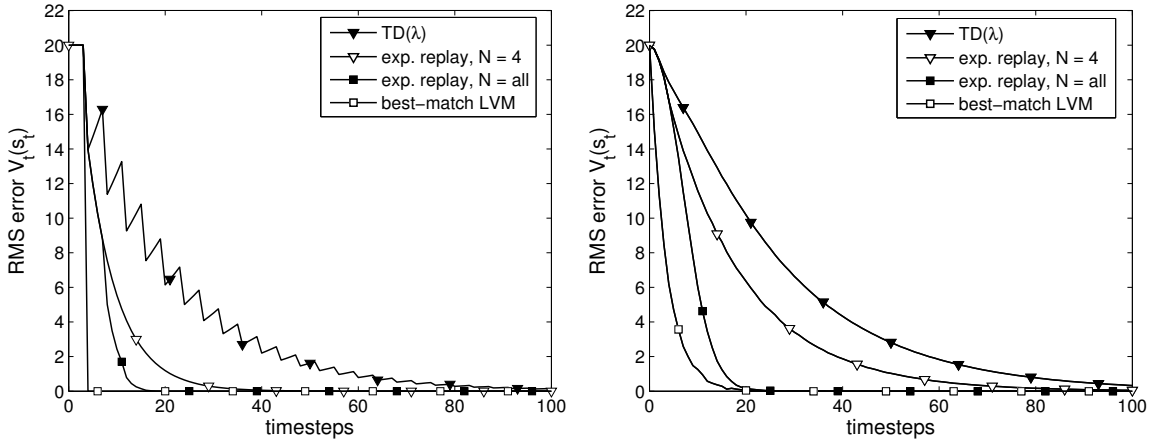

Figure 9: Comparison of the performance of best-match LVM, TD(λ) and experience replay on tasks A (left) and task B (right) of Figure 8.

to 80 ms for both best-match LVM and TD(λ). Experience replay with all samples updated had a computation time of 280 ms. On task B, all methods were about 10 ms slower.

4.3 Best-Match LVM Control

The best-match LVM equations for the control case form a nonlinear set. Therefore, it is in general not possible to compute the exact best-match Q-values at each timestep. However, they can be approximated to arbitrary accuracy via update sweeps through the state-action space, in a manner similar to value iteration, as we prove in the following lemma.

Lemma 9 *For the best-match Q-values the following equation holds for all (s,a):*

$$Q_t^B(s,a) = \lim_{i\to\infty} Q_{t,i}(s,a)\,,$$

where $Q_{t,i}$ is initialized arbitrarily for $i = 0$ and is defined for $i > 0$ as

$$Q_{t,i}(s,a) = \begin{cases} (1-\alpha)Q_t^{mf}(s,a) + \alpha\left[R'_t(s,a) + \gamma \max_{a'} Q_{t,i-i}(S'_t(s,a),a')\right] & \text{if } S'_t(s,a) \neq \emptyset \\ Q_t^{mf}(s,a) & \text{if } S'_t(s,a) = \emptyset\,. \end{cases}$$

Proof For state-action pairs (s,a) with $S'_t(s,a) = \emptyset$ the proof follows directly from the definition of Q_t^B and $Q_{t,i}$. For (s,a) with $S'_t(s,a) \neq \emptyset$, the absolute difference between $Q_{t,i}(s,a)$ and $Q_t^B(s,a)$ can be written as

$$\begin{aligned} |Q_{t,i}(s,a) - Q_t^B(s,a)| &= \alpha\gamma |\max_c Q_{t,i-i}(S'_t(s,a),c) - \max_c Q_t^B(S'_t(s,a),c)| \\ &\leq \alpha\gamma \max_c |Q_{t,i-i}(S'_t(s,a),c) - Q_t^B(S'_t(s,a),c)| \\ &\leq \alpha\gamma ||Q_{t,i-i} - Q_t^B||\,. \end{aligned}$$

From this it follows that

$$||Q_{t,i} - Q_t^B|| \leq \alpha\gamma||Q_{t,i-i} - Q_t^B||.$$

For $\alpha\gamma < 1$, it follows that for $i \to \infty$, $Q_{t,i} \to Q_t^B$. ■

Lemma 9 shows that Q_t^B can be approximated to arbitrary accuracy with a finite number of best-match updates.

Algorithm 3 shows the pseudocode for a general class of algorithms that approximate the best-match Q-values by performing best-match updates.[4] Lines 9 to 12 perform a series of best-match updates. Note that while only a single Q^{mf} value is updated per timestep, many Q-values can be updated at the same timestep. By varying the way state-action pairs are selected for updating (line 10) and changing the stopping criterion (line 12), a whole range of algorithms can be constructed that trade off computation cost per timestep for better approximations of the best-match Q-values. Note that JIT Q-learning and even regular Q-learning are members of this general class of algorithms. If the state-action pair selection criterion is the state-action pair visited at the previous timestep and the stopping criterion allows only a single update, the algorithm reduces to the regular Q-learning algorithm. Thus, Q-learning is a form of best-match control with a simplistic approximation of the best-match Q-values. However, we reserve the term 'best-match learning' for algorithms that use the same sample multiple times to redo updates.

Algorithm 3 General Best-Match LVM Control

```
1:  initialize Q(s,a) arbitrarily for all s,a
2:  initialize S'(s,a) = ∅ for all s,a
3:  loop {over episodes}
4:     initialize s
5:     repeat {for each step in the episode}
6:        select action a, based on Q(s,·)
7:        take action a, observe r and s'
8:        Q^mf(s,a) ← Q(s,a); S'(s,a) ← s'; R'(s,a) ← r
9:        repeat
10:          select some (s̄,ā) pair with S'(s̄,ā) ≠ ∅   {each pair is selected at least once before its revisit}
11:          Q(s̄,ā) ← (1 − α^{s̄ā}) Q^mf(s̄,ā) + α^{s̄ā} [R'(s̄,ā) + γ max_c Q(S'(s̄,ā),c)]
12:       until some stopping criterion has been met
13:       s ← s'
14:    until s is terminal
15: end loop
```

The following theorem states that, for any member of the best-match LVM control class, the Q-values converge to the optimal Q-values.

Theorem 10 *The Q-values of a member of the best-match LVM control class, shown in Algorithm 3, converge to Q^* if the following conditions are satisfied:*

1. *S and A are finite.*

4. Similar to the variable V_r^{mf} of Algorithm 2, a variable Q_r^{mf} can be defined that combines the variables Q^{mf} and R', saving space and computation. For readability we do not show this for Algorithm 3.

2. $\alpha_t(s,a) \in [0,1]$, $\sum_t \alpha_t(s,a) = \infty$, $\sum_t (\alpha_t(s,a))^2 < \infty$ *w.p.1* *and* $\alpha_t(s,a) = 0$ *unless* $(s,a) = (s_t, a_t)$.

3. $Var\{R(s,a,s')\} < \infty$.

4. $\gamma < 1$.

We prove this theorem in Appendix D.

4.4 Best-Match LVM Prioritized Sweeping

A wide range of methods can be constructed within the general class of best-match LVM control algorithms that trade off increased computation time for better approximation of the best-match Q-values in different ways. This section proposes one method that performs this trade-off with a strategy based on *prioritized sweeping* (PS) (Moore and Atkeson, 1993).

PS makes the planning step of model-based RL more efficient by focusing on the updates expected to have the largest effect on the Q-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 estimated transition probabilities to s are greater than 0) are added to the queue according to a heuristic estimating the impact of the update. At each timestep, the top N state-action pairs from this queue are updated, with N depending on the available computation time. Because PS maintains a full model, it requires $O(|\mathcal{S}|^2|\mathcal{A}|)$ space.

This same idea can be applied to the best-match equations for efficient approximation of the best-match values. A priority queue of state-action pairs is maintained whose corresponding best-match updates have the largest expected effect on the best-match Q-value estimates. When a state-action pair has received an update, all state-action pairs whose last-visit transition state equals the state from the updated state-action pair are placed into the priority queue with a priority equal to the absolute change an update would cause in its Q-value. Since this approach uses only an LVM, it requires only $O(|\mathcal{S}||\mathcal{A}|)$ space.

Algorithm 4 shows the pseudocode of this algorithm, which we call *best-match LVM prioritized sweeping* (BM-LVM). By always putting the state-action pair from the previous timestep on top of the priority queue (line 10), the requirement that each visited state-action pair receives at least one best-match update is fulfilled, guaranteeing convergence in the limit.

On the surface, this algorithm resembles *deterministic prioritized sweeping* (DPS) (Sutton and Barto, 1998), a simpler variation that learns only a deterministic model, uses a slightly different priority heuristic, and performs Q-learning updates to its Q-values. While clearly designed for deterministic tasks, it can also be applied to stochastic tasks, in which case updates are based on an LVM.

However, there is a crucial difference between DPS and BM-LVM. By performing updates with respect to Q^{mf} instead of Q, BM-LVM corrects previous updates instead of performing multiple updates based on the same sample. This ensures proper averaging of experience and enables convergence to the optimal Q-values using only an LVM, even in stochastic environments. This is not guaranteed for DPS since if some samples are used more often than others a bias towards these samples is created, which can prevent convergence to the optimal Q-values.

We compare the performance of PS, DPS, and BM-LVM on the deterministic and stochastic variation of the Dyna Maze task shown in Figure 3. In addition, we also compare to Q(λ) as

Algorithm 4 Best-Match LVM Prioritized Sweeping (BM-LVM)

initialize $Q(s,a)$ arbitrarily for all s,a
initialize $S'(s,a) = \emptyset$ for all s,a
initialize PQueue as an empty queue
loop {over episodes}
  initialize s
  repeat {for each step in the episode}
    select action a, based on $Q(s,\cdot)$
    Take action a, observe r and s'
    $S'(s,a) \leftarrow s'; R'(s,a) \leftarrow r; Q^{mf}(s,a) \leftarrow Q(s,a)$
    promote (s,a) to top of priority queue
    $n \leftarrow 0$
    while $(n < N) \wedge$ (*PQueue* is not empty) **do**
      $s_1, a_1 \leftarrow first(PQueue)$
      $Q(s_1,a_1) \leftarrow (1-\alpha^{s_1a_1})\, Q^{mf}(s_1,a_1) + \alpha^{s_1a_1}\,[R'(s_1,a_1) + \gamma \max_c Q(S'(s_1,a_1),c)]$
      $V_{s_1} \leftarrow max_{a'} Q(s_1,a')$
      for all $(\bar{s},\bar{a})$ with $S'(\bar{s},\bar{a}) = s_1$ **do**
        $p \leftarrow |(1-\alpha^{\bar{s}\bar{a}})\, Q^{mf}(\bar{s},\bar{a}) + \alpha^{\bar{s}\bar{a}}\,[R'(\bar{s},\bar{a}) + \gamma V_{s_1}] - Q(\bar{s},\bar{a})|$
        if $p > \theta$ **then**
          insert $(\bar{s},\bar{a})$ into *PQueue* with priority p
        end if
      end for
      $n \leftarrow n+1$
    end while
    $s \leftarrow s'$
  until s is terminal
end loop

described by Watkins (1989). This is an off-policy control version of eligibility traces. We also tried Sarsa(λ), the on-policy version, since it can sometimes outperform Q(λ) considerably, but saw no significant difference for these experiments and present only the Q(λ) results. Note that when a greedy behavior policy is used, as in the deterministic experiment, Q(λ) computes exactly the same values as Sarsa(λ). As in Section 4.2, we also compare to experience replay.

Finally, we compare to delayed Q-learning (Strehl et al., 2006), a model-free method that, like some model-based methods (Brafman and Tennenholtz, 2002; Kearns and Singh, 2002; Strehl and Littman, 2005), is proven to be *probably approximately correct* (PAC), that is, its sample complexity is polynomial with high probability. Delayed Q-learning initializes its Q-values optimistically and ensures 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 $O(|\mathcal{S}||\mathcal{A}|)$ space requirements as best-match prioritized sweeping. However, to our knowledge, its empirical performance has never been evaluated before.

For each method, the free parameters are optimized within a certain range. In the deterministic case, for Q(λ) we optimized the λ value in the range from 0 to 1, and the learning rate decay d (using Equation 4) in the range from 0 to 1, while α_0 was set to 1. We also optimized the (unbounded) trace

type (replacing versus accumulating). For delayed Q-learning we optimized m in the range from 1 to 5 with steps of 1 and e_1 in the range 0 to 0.020 with steps of 0.001. For DPS and BM-LVM, we did not optimize any parameters in the deterministic case, but simply used a constant α of 1. In the stochastic case, we also optimized the learning rate decay d for DPS and BM-LVM.

For all methods, we used optimistic initialization with $Q_0 = 20$ in order to get a fair comparison with delayed Q-learning, for which initialization to $R_{max}/(1-\gamma)$ is part of the algorithm.[5]

In the deterministic case we used a greedy behavior policy, while we used an ε-greedy policy with $\varepsilon = 0.1$ in the stochastic variant. For all prioritized-sweeping algorithms we performed a maximum of 20 updates per timestep (i.e., N = 20). For experience replay we used the last 20 samples, which also results in 20 updates per timestep. Results are averaged over 1000 independent runs.

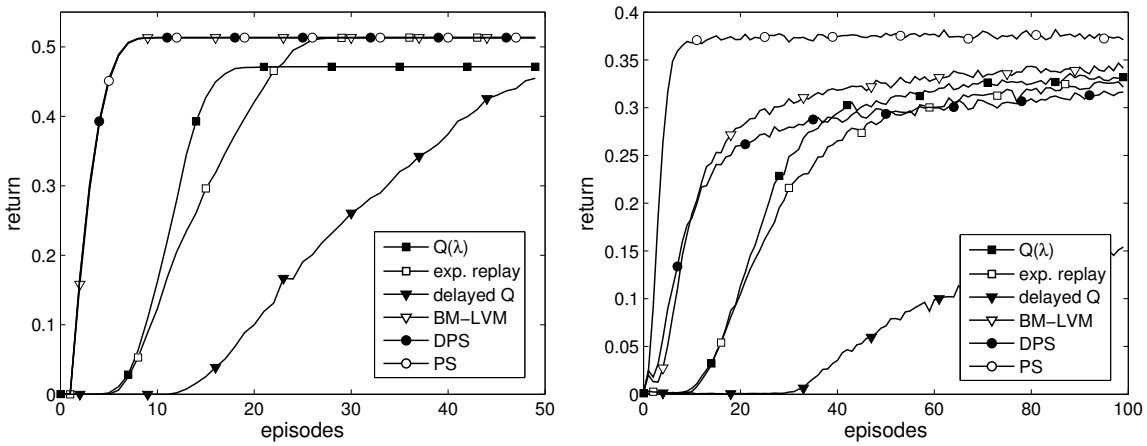

Figure 10: Comparison of the performance of BM-LVM and several competitors on the deterministic (left) and stochastic (right) Dyna Maze task.

Figure 10 shows the return as a function of the number of episodes, while Tables 2 and 3 show the average return over the measured episodes and the optimal parameter values. In the deterministic experiment, we see that the performance of PS, DPS, and BM-LVM is exactly equal, as expected when $\alpha = 1$, since the last-visit experience is equal to the model of the environment. Q(λ) performs considerably worse than the prioritized sweeping methods and does not converge to the optimal policy. In contrast, the combination of a greedy behavior policy with optimistic initialization enables the prioritized sweeping methods to converge to the optimal policy in a deterministic environment. Experience replay performs similarly to Q(λ), though it does converge to the optimal policy. Delayed Q-learning also converges to the optimal policy, as predicted by the theory, but does so much more slowly.

In the stochastic experiment, PS has a clear performance advantage. However, the goal of BM-LVM is not to match or even come close to the performance of PS. It cannot match this performance in general, since PS takes advantage of its higher space complexity. Instead, the goal of BM-LVM

5. For this task $r = R_{max}$ only when the exit is reached and 0 otherwise. Thus, the Q-values can never be higher than 1 and $Q_0 = 20$ is overly optimistic. However, since realizing that an initialization of 1 is possible would require extra prior knowledge, we initialize to 20.

	deterministic - 50 eps.			
	optimal parameters	average return	standard error	time per step $(\cdot 10^{-6} s)$
Q(λ)	λ: 0.8, d: 0	0.3606	0.0007	0.68
exp. replay	d: 0	0.3602	0.0004	0.37
delayed Q	m: 1, $e_1 = 0$	0.1878	0.0004	0.11
BM-LVM	d: 0	0.4769	0.0002	0.88
DPS	d: 0	0.4774	0.0002	0.85
PS	-	0.4772	0.0002	0.95

Table 2: Average return and optimal parameters (d = α decay rate) of best-match prioritized sweeping and several competitors on the deterministic Dyna Maze task.

	stochastic - 100 eps.			
	optimal parameters	average return	standard error	time per step $(\cdot 10^{-6} s)$
Q(λ)	λ: 0.9, d: 0.03	0.2417	0.0007	0.59
exp. replay	d: 0.18	0.2272	0.0006	0.43
delayed Q	m: 2, e_1:0.015	0.0668	0.0004	0.12
BM-LVM	d: 0.02	0.2911	0.0006	3.2
DPS	d: 0.30	0.2683	0.0008	3.7
PS	-	0.3603	0.0004	4.7

Table 3: Average return and optimal parameters (d = α decay rate) of best-match prioritized sweeping and several competitors on the stochastic Dyna Maze task.

is to optimally perform at a space complexity of $O(|\mathcal{S}||\mathcal{A}|)$. The results confirm that BM-LVM is considerably better than the other methods with this space complexity, like Q(λ) and DPS. DPS initially performs well, but cannot keep up with BM-LVM after about 10 episodes, even though BM-LVM has similar space and computation costs per timestep. Experience replay performs slightly worse than Q(λ). We tested whether doubling the size of the stored experience sequence improves the performance of experience replay, but this led to no significant performance increase. Delayed Q-learning also performs poorly in the stochastic case, despite its PAC bounds.

The computation time of BM-LVM, DPS and PS is in the deterministic experiment considerably lower than in the stochastic case. The reason for this is that while in both cases the maximum number of updates per timestep is 20, in the deterministic case the priority queue often has fewer than 20 samples, so fewer updates occur. The computation time of Q(λ) is slightly better than that of BM-LVM, while experience replay is about twice as fast as BM-LVM.

In the stochastic experiment, the computation time of Q(λ) is much better than that of any of the prioritized sweeping algorithms, which could suggest that Q(λ) is a better choice than BM-LVM when computation power is scarce. To test this hypothesis, we performed additional experiments with smaller values of N. The computation time for BM-LVM for $N = 4$ ($0.61 \cdot 10^{-6}$ s) was similar to that of Q(λ). The average return of BM-LVM dropped to 0.2598 in this case, which is still

considerably better than the average return of Q(λ). This demonstrates that BM-LVM is a better choice than Q(λ) even under severe computational constraints.

Together, these results clearly demonstrate the strength of best-match learning, since BM-LVM outperforms several competitors with similar space complexity. However, the results also show that the performance gap with full model-based learning can be considerable. Therefore, if more space is available, a better approximate model would be preferred. We address this need in the next section by applying best-match learning to an n-transition model, which estimates the transition function for n next states per state-action pair, allowing increased space requirements to be traded for improved performance.

5. Best-Match n-Transition Model

The best-match LVM equations described above combine model-free Q-values with the last-visit model. When state-action pairs have only a small number of possible next states, the last-visit model can effectively approximate the full model. In other cases, however, the last-visit model captures only a fraction of the full model and the effect of the best-match updates will be small. In this section, we combine best-match learning with the n-transition model, which estimates the transition probability for n possible next states of each state-action pair. By tuning n, increased space requirements can be traded for improved performance.

5.1 Generalized Best-Match Equations

Best-match LVM learning takes the idea of using more accurate update targets to the extreme by continuously revising update targets with best-match updates. For a specific sample, the update target is revised until the moment of revisit of the corresponding state-action pair, since at that moment the sample is overwritten with the newly collected sample. However, if space allows, the new sample can be stored along with the old sample instead of overwriting it, allowing the update target from the new as well as the old sample to be further improved. We explain with an example how this changes the best-match equations.

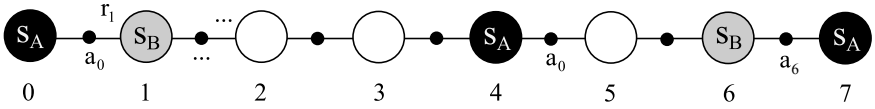

Figure 11: A state transition sequence in which best-match updates can enable further postponing. Timesteps are shown below each state.

Consider the state-action sequence from Figure 11 and assume the best-match Q-values are computed at each timestep. At the revisit of s_A, action a_0 is retaken. Therefore, when using the LVM, at timestep 5 the old experience sample is overwritten with the new experience. Before this occurs, the old experience is used in a final update of Q^{mf}. Let υ_y^x indicate the update target from the sample collected at timestep x based on the best-match Q-value of timestep y: $\upsilon_y^x = r_x + \gamma \max_a Q_y^B(s_x, a)$. Using this convention the update of Q^{mf} at timestep 5 becomes

$$Q_5^{mf}(s_A, a_0) = (1-\alpha) Q_0^{mf}(s_A, a_0) + \alpha \upsilon_4^1 .$$

At timestep 7, the best-match LVM equation for (s_A, a_0) can be written as

$$\begin{aligned} Q_7^B(s_A,a_0) &= (1-\alpha)Q_7^{mf}(s_A,a_0)+\alpha\upsilon_7^5 \\ &= (1-\alpha)Q_5^{mf}(s_A,a_0)+\alpha\upsilon_7^5 \\ &= (1-\alpha)^2 Q_0^{mf}(s_A,a_0)+\alpha(1-\alpha)\upsilon_4^1+\alpha\upsilon_7^5. \end{aligned}$$

Thus, the best-match Q-value of (s_A, a_0) at timestep 7 is equal to a weighted average of Q_0^{mf}, υ_4^1 and υ_7^5. On the other hand, if both the old and the new sample are stored, Q-values from timestep 7 could also be used for the update target of the old sample, yielding

$$Q_7^B(s_A,a_0) = (1-\alpha)^2 Q_0^{mf}(s_A,a_0)+\alpha(1-\alpha)\upsilon_7^1+\alpha\upsilon_7^5. \tag{10}$$

For the state-sequence from Figure 11 this means that the experience resulting from (s_B, a_6) is also taken into account in the update target for (s_A, a_0).

The above example shows how the best-match LVM equations can be naturally extended to two samples per state-action pair. Following the same pattern, we can define best-match equations given an arbitrary set of samples. Consider the set of samples X of size N_X, where a sample $x \in X$ has the form $\{s, a, r, s'\}$. These samples can be grouped according to their state-action pairs. We define X_{sa} as the subset of X containing all samples belonging to state-action pair (s, a) and N_{sa}^x as the size of X_{sa}. Without loss of generality, we index the samples from X_{sa} as x_k^{sa} for $1 \le k \le N_{sa}^x$. In addition, we define W_{sa} as a set consisting of $N_{sa}^x + 1$ weights $w_k^{sa} \in \mathbb{R}$ such that $0 \le w_k^{sa} \le 1$ for $0 \le k \le N_{sa}^x$ and $\sum_{k=0}^{N_{sa}^x} w_k^{sa} = 1$. We define W as the union of the weight sets from all state-action pairs.

Definition 11 *The generalized best-match equations with respect to Q_t^{mf}, X and W are*

$$Q_t^B(s,a) = w_0^{sa} Q_t^{mf}(s,a) + w_1^{sa}\upsilon_1^{sa} + w_2^{sa}\upsilon_2^{sa} + \ldots + w_{N_{sa}^x}^{sa}\upsilon_{N_{sa}^x}, \qquad \textit{for all } s,a, \tag{11}$$

where $\upsilon_k^{sa} = r + \gamma \max_c Q_t^B(s',c) \mid r, s' \in x_k^{sa}$.

Note that Equation 11 reduces to $Q_t^B(s,a) = Q_t^{mf}(s,a)$ for state-action pairs with no samples in X.

Within this context, Q^{mf} is defined as a model-free Q-value constructed from all observed samples except those in X. Consequently, when a sample is removed from X, it is used for a model-free update of Q^{mf}.

Using Definition 11, a range of algorithms can be constructed based on different sets of samples X and weights W. When the samples are combined by incremental Q-learning updates, like in Equation 10, the weights have the values

$$w_0^{sa} = \prod_{i=1}^{N_{sa}^x}(1-\alpha_i^{sa}), \tag{12}$$

$$w_k^{sa} = \alpha_k^{sa}\prod_{i=k+1}^{N_{sa}^x}(1-\alpha_i^{sa}), \qquad \text{for } 1 \le k \le N_{sa}^x. \tag{13}$$

With this weight distribution, the update targets from older samples have lower weights than more recent samples. In Q-learning, more recent samples in general have more accurate update targets so giving them higher weight makes sense. However, in best-match learning the update targets from

all stored samples have the same time index so there is no reason to use different weights for them. A better weight distribution gives all samples the same weights:

$$w_k^{sa} = (1 - w_0^{sa})/N_{sa}^x\,, \qquad \text{for } 1 \leq k \leq N_{sa}^x\,,$$

for some value of w_0^{sa}.

The last-visit model, storing one sample for each state-action pair, is one possible sample set. A straightforward extension is to store n samples per state-action pair. In the following section, however, we propose a different sample set, called the n-transition model, which can be stored more compactly.

5.2 Best-Match Learning based on the n-transition Model

While BM-LVM outperforms model-free methods with the same space complexity, it does not perform as well as PS, which stores a full model. This is symptomatic of an important limitation of BM-LVM: it offers only a single trade-off between space and performance. When there is not enough space available to store the full model, but more than enough to store the LVM, a more sophisticated method is needed to make maximal use of the available space. Using the generalized best-match equations, we can construct such a method.

An obvious approach is to store n samples per state-action pair. However, obtaining an accurate model often requires a large n, even when the number of next states per state-action pair is small. A more space-efficient solution is to group together samples that have the same next state. If we store the size of such a group in $N_{sas'}^x$ and give each sample a weight of $1/N_{sa}$, where N_{sa} is the total number of times state-action pair (s,a) is visited, then we can rewrite the contribution from all samples of X_{sa} to the best-match equations as

$$\sum_{k=1}^{N_{sa}^x} w_k \upsilon_k = \frac{1}{N_{sa}} \left[\sum_X r_{sa} + \gamma \sum_{s'} N_{sas'}^x \max_{a'} Q^B(s',a') \right],$$

where $\sum_X r_{sa}$ is the sum of the rewards from all samples in the sample set belonging to (s,a). Using $w_0^{sa} = 1 - N_{sa}^x/N_{sa}$, $\hat{\mathcal{P}}_{sa}^{s'} = N_{sas'}^x/N_{sa}^x$ and $\hat{\mathcal{R}}_{sa} = \sum_X r_{sa}/N_{sa}^x$, the generalized best-match equations can now be rewritten as

$$Q^B(s,a) = w_0^{sa} Q^{mf}(s,a) + (1 - w_0^{sa}) \left[\hat{\mathcal{R}}_{sa} + \gamma \sum_{s'} \hat{\mathcal{P}}_{sa}^{s'} \max_{a'} Q^B(s',a') \right], \qquad \text{for all } s,a\,.$$

In these equations, $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}$ constitute a sparse, approximate model, whose size can be controlled by limiting the number of next states per state-action pair for which $\hat{\mathcal{P}}$ is estimated. w_0^{sa} is the fraction of all samples belonging to (s,a) not used by the sparse model. We define an *n-transition model* (NTM) to be one that estimates the transition probability $\hat{\mathcal{P}}$ for n next states per state action pair. Once a sample enters the model, that is, is used to update $\hat{\mathcal{P}}$, it stays in the model. Each sample not used to update the model is used for a model-free update of Q^{mf}. Different strategies can be used to determine which samples enter the model. A simple approach is to use the first n unique next states that are encountered for a specific state-action pair.

Algorithm 5 shows general pseudocode for best-match NTM learning. The algorithm presents two trade-offs. First, the space complexity can be traded off with performance by selecting n.

Algorithm 5 General Best-Match NTM Control

initialize $Q(s,a) = Q^{mf}(s,a)$ arbitrarily for all s,a
initialize $N_{sa}, N^x_{sa}, R^{sum}_{sa}$ to 0 for all s,a
initialize $N^x_{sas'}$ to 0 for all s,a and $s' \in NTM(s,a)$
initialize w_0^{sa} to 1 for all s,a
loop {over episodes}
  initialize s
  repeat {for each step in the episode}
    select action a, based on $Q(s,\cdot)$
    take action a, observe r and s'
    if $s' \in NTM(s,a)$ **then**
      $N^x_{sa} = N^x_{sa} + 1; \quad N^x_{sas'} = N^x_{sas'} + 1; \quad R^{sum}_{sa} = R^{sum}_{sa} + r$
      $\hat{\mathcal{P}}^{s'}_{sa} = N^x_{sas'}/N^x_{sa}; \quad \hat{\mathcal{R}}_{sa} = R^{sum}_{sa}/N^x_{sa}$
    else
      $Q^{mf}(s,a) \leftarrow (1-\alpha^{sa})Q^{mf}(s,a) + \alpha^{sa}[r + \gamma \max_c Q(s',c)]$
    end if
    $N_{sa} = N_{sa} + 1$
    $w_0^{sa} = 1 - N^x_{sa}/N_{sa}$
    repeat
      select some $(\bar{s},\bar{a})$ pair with $N_{\bar{s}\bar{a}} > 0$ {each pair is selected at least once before its revisit}
      $Q(\bar{s},\bar{a}) \leftarrow w_0^{\bar{s}\bar{a}} Q^{mf}(\bar{s},\bar{a}) + (1 - w_0^{\bar{s}\bar{a}}) \left[\hat{\mathcal{R}}_{\bar{s}\bar{a}} + \gamma \sum_{s'} \hat{\mathcal{P}}^{s'}_{\bar{s}\bar{a}} \max_c Q(s',c)\right]$
    until some stopping criterion has been met
    $s \leftarrow s'$
  until s is terminal
end loop

Second, the computation time per simulation step can be traded off with performance by controlling the number of best-match updates performed per timestep.

Based on this general control algorithm, various specific algorithms can be constructed using different stopping criteria and strategies for selecting state-action pairs to receive best-match updates. The following theorem states that, for any member of this class, the Q-values converge to the optimal Q-values. We prove this theorem in Appendix E.

Theorem 12 *The Q-values of a member of the best-match NTM control class, shown in Algorithm 5, converge to Q^* if the following conditions are satisfied:*

1. *S and A are finite.*
2. $\alpha_t(s,a) \in [0,1]$, $\sum_t \alpha_t(s,a) = \infty$, $\sum_t (\alpha_t(s,a))^2 < \infty$ *w.p.1*
   and $\alpha_t(s,a) = 0$ *unless* $(s,a) = (s_t,a_t)$ *and* $s_{t+1} \notin NTM(s_t,a_t)$.
3. $Var\{R(s,a,s')\} < \infty$.
4. $\gamma < 1$.

5.3 Experimental Results

As in BM-LVM, prioritized sweeping can be used to trade off computation time and performance in Algorithm 5, yielding a method we call BM-NTM. We compare its performance to BM-LVM, Q-learning, and a sparse model-based method that combines prioritized sweeping with an NTM without best-match updates, which we call PS-NTM. While BM-NTM uses the samples that are not part of the NTM to update Q^{mf}, PS-NTM ignores these samples. The priority of a state-action pair (s,a) for BM-NTM is defined as

$$p = (1 - w_0^{sa})\hat{\mathcal{P}}_{sa}^{s_1} \cdot |\Delta V(s_1)|,$$

where $\Delta V(s_1)$ is the difference in the state value of s_1 before and after the best-match update of one of the Q-values of s_1. For PS-NTM, the priority is defined similarly:

$$p = \hat{\mathcal{P}}_{sa}^{s_1} \cdot |\Delta V(s_1)|.$$

The NTM we use for BM-NTM and PS-NTM is defined by the first n unique next states that are encountered for a specific state-action pair. Although more sophisticated models could be used (e.g., by estimating the n most likely transition states), this model is sufficient for our experimental setting since most transition states have similar transition probabilities.

We consider the large maze task shown at the left in Figure 12. For this maze, the reward received by the agent is -0.1 at each timestep, while reaching the goal state results in a reward of +100. The discount factor is 0.99. The agent can take four actions, 'north','south','east' and 'west'. The action outcomes are made very stochastic, in order to compare different model sizes. The right side of Figure 12 shows the relative action outcome for a 'north' action. In free space, there are 15 possible next states, each with equal transition probability. On the other hand, walls prevent not only the transition to the square the wall is located on, but also any squares behind the wall. Therefore, close to a wall the number of possible next states is less than 15. When transition to a square is blocked by a wall, the transition probability of that square is added to the transition probability of the square in front of the wall. In order to make reaching the goal feasible despite the stochastic actions, we use a goal area consisting of four goal states.

To compare performance, we measure the average return for each method over the first 500 episodes. For all methods, we use an ε-greedy policy with $\varepsilon = 0.05$ and initialize Q-values to 0. BM-NTM, PS-NTM and BM-LVM perform a maximum of 5 updates per timestep. For all learning rate based methods, we use an initial learning rate of 1 and decay the learning rate according to Equation 4, while optimizing the decay rate d. Results are averaged over 200 independent runs. An episode is stopped prematurely if the goal is not reached within 500 steps.

Table 4 presents the results, including the average return, optimal parameters, and computation time per simulation step. The model sizes used are N = 1, 3, 5, and 15. For $N = 15$, all samples enter the model. Therefore, BM-NTM has no decay rate in this case. The model weight indicates the fraction of samples that entered the model. BM-NTM has in general a slightly higher weight than PS-NTM, indicating the agent spends less time in open spaces and more time close to a wall.

For model sizes $N = 1$ and $N = 3$, the average return of BM-NTM is much better than that of PS-NTM, despite the fact that for $N = 3$ more than a third of the samples are stored in the model. For $N = 1$, the average return of PS-NTM is even worse than that of Q-learning. Figure 13 shows the return as a function of the number of episodes for BM-NTM and PS-NTM with $N = 1$ and $N = 3$. Unlike BM-NTM, the asymptotic performance for PS-NTM is clearly bounded by the size of the

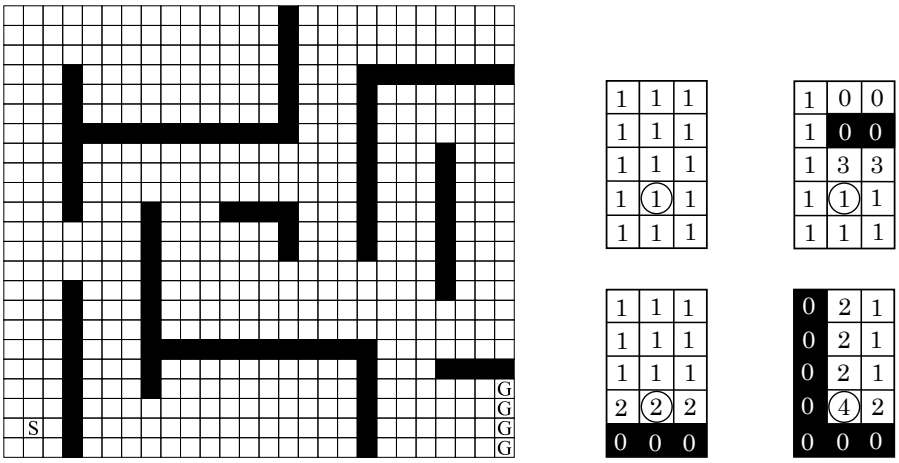

Figure 12: Left, the large maze task, in which the agent must travel from S to one of the G's. Right, transition probabilities ($\cdot\frac{1}{15}$) of a 'north' action for different positions of the agent (indicated by the circle) with respect to the walls (black squares). When the transition to a square is blocked by a wall, its transition probability is added to that of the square in front of the wall.

model. Thus, PS-NTM can match the performance of BM-NTM only when the space reduction over the full model is quite small (i.e., less than a factor of 2).

Interestingly, when $N = 1$, BM-LVM outperforms BM-NTM despite having the same space complexity. Thus, when space is scarce, BM-LVM is a good option. In contrast, BM-NTM can exploit larger models to further improve performance. The computation time per simulation step for BM-NTM is comparable to that of PS-NTM, with the exception of $N = 1$, for which it is four times larger. The reason is that the priority queue of PS-NTM is often close to empty in this case and thus the 5 updates per timestep are often not reached.

Overall, these results clearly demonstrate the strength of best-match NTM learning. When a significant space reduction over storing the full model is required, BM-NTM performs dramatically better than PS-NTM at similar computational cost.

6. Best-Match Function Approximation

The BM-NTM method described in the previous section has a space complexity of $O(n|\mathcal{S}||\mathcal{A}|)$ compared to $O(|\mathcal{S}|^2|\mathcal{A}|)$ for full model-based methods. However, in problems with large state spaces, this space complexity may be prohibitive even when $n = 1$. In addition, BM-NTM cannot be applied in problems with continuous state spaces. To address these limitations, this section demonstrates that the principles behind best-match learning can also be applied to function approximation. We show that the resulting algorithm, which combines the N most recent samples with the model-free Q-value function, outperforms both linear Sarsa(λ) and linear experience replay on the mountain car task. We start by describing best-match learning based on the N most recent samples for the tabular case, and then we show how this can be extended to the function approximation case.

	model size	model weight	optimal parameters	average return	standard error	time per step ($\cdot 10^{-6}$ s)
PS-NTM	1	0.12	-	-16.9	0.4	0.21
	3	0.36	-	9.8	0.3	1.5
	5	0.57	-	22.6	0.2	2.1
	15	1.00	-	28.9	0.2	3.1
BM-NTM	1	0.14	d = 0.04	15.4	0.3	0.85
	3	0.40	d = 0.09	19.6	0.2	1.7
	5	0.60	d = 0.06	22.3	0.2	2.2
	15	1.00	-	29.3	0.2	3.1
BM-LVM	-	-	d = 0.09	17.4	0.3	1.5
Q-learning	-	-	d = 0.03	2.4	0.2	0.09

Table 4: Average return over the first 500 episodes, optimal parameters (d: α decay rate) and computation time per simulation step on the Large Maze task.

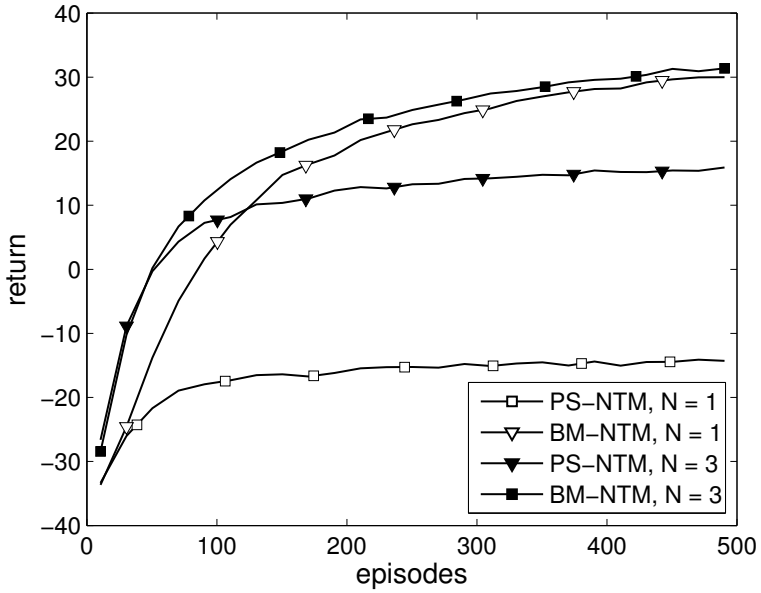

Figure 13: Performance of BM-NTM and PS-NTM on the large maze task.

6.1 Tabular Sequence Based Best-Match Learning

The generalized best-match equations are defined for an arbitrary set of samples (see Definition 11), which can be stored in a model or as an explicit set. To combine best-match principles with function approximation, we employ an explicit set consisting of the last N observed samples, an approach we call *sequence based best-match learning*. In this section we describe sequence based best-match learning for the tabular case and its advantage over experience replay, which also exploits a set of recent samples. In the next section, we extend the tabular version of sequence based best-match learning to function approximation.

Assume that a queue of the last N samples is maintained. When the queue is full and a new sample is added to the back of the queue, the sample at the front of the queue is removed and used to perform a model-free update of $Q^{mf}(s,a)$. The queue may contain multiple samples that belong to the same state-action pair. If there are N^x_{sa} samples belonging to state-action pair (s,a), then the best-match update based on these samples is

$$Q_{t,i+1}(s,a) = w_0^{sa} Q_t^{mf}(s,a) + w_1^{sa}\upsilon_1^{sa} + w_2^{sa}\upsilon_2^{sa} + \ldots + w_{N^x_{sa}}^{sa}\upsilon_{N^x_{sa}}, \tag{14}$$

where $\upsilon_k^{sa} = r + \gamma \max_c Q_{t,i}(s',c) \,|r,s' \in x_k^{sa}$. When the weights are defined according to Equations 12 and 13, this update can be implemented incrementally by performing N^x_{sa} Q-learning updates:

$$Q_{<k>}(s,a) = (1-\alpha)Q_{<k-1>}(s,a) + \alpha\left[r_k + \gamma \max_{a'} Q_{t,i}(s'_k,a')\right], \quad \text{for } 1 \le k \le N^x_{sa},$$

with $Q_{<0>}(s,a) = Q_t^{mf}(s,a)$ and $Q_{t,i+1}(s,a) = Q_{<N^x_{sa}>}(s,a)$.

By stepping through the queue from front to back and using each sample to perform an incremental Q-learning update, all state-action pairs with samples in the queue receive one full best-match update, according to Equation 14. By storing the intermediate $Q_{<k>}$ values at the same location as the final Q-value, $Q_{<N^x_{sa}>}$ automatically becomes $Q_{t,i+1}$ after all incremental updates have been performed. This implementation requires that the Q-values from the state-action pairs with samples in the queue are set equal to $Q_{<0>}$, that is, to Q_t^{mf}, before the update sweep begins. Before resetting these Q-values, the update targets of the samples must be recomputed.

Despite a superficial resemblance, sequence based best-match learning is fundamentally different from experience replay. Best-match learning uses the stored samples to correct previous updates based on those samples, whereas experience replay performs additional updates with the same sample. To illustrate the effect of this difference, suppose that sample (s,a,r,s') is observed at timestep $t = 1$ and used for an update n timesteps in a row. For simplicity, assume there are no other samples belonging to (s,a) in the sample queue and that the learning rate α is constant. We indicate the update target of the sample with $\bar{\upsilon}_i$, where i corresponds to the timestep at which the update is performed. Therefore, $\bar{\upsilon}_{i+1}$ is likely to be more accurate than $\bar{\upsilon}_i$ since it uses more recent Q-values for s'. Since experience replay performs additional updates we can express $Q_n(s,a)$, the Q-value of (s,a) at timestep n, in terms of $Q_0(s,a)$ and the update targets from the different timesteps as follows:

$$Q_n(s,a) = w_0 Q_0(s,a) + w_1\bar{\upsilon}_1 + w_2\bar{\upsilon}_2 + \ldots + w_n\bar{\upsilon}_n,$$

with $w_0 = \prod_{i=1}^{n}(1-\alpha)$ and $w_k = \alpha\prod_{i=k+1}^{n}(1-\alpha)$ for $k > 0$. If $\alpha \ll 1$, the weights can be accurately described with first-order approximations in α, yielding $w_0 \approx 1 - n\alpha$ and $w_k \approx \alpha$ for $k > 0$. Using these approximations, we can write for $Q_n(s,a)$:

$$Q_n(s,a) \approx (1-\beta)Q_0(s,a) + \beta\,\frac{\sum_{i=1}^{n}\bar{\upsilon}_i}{n}, \tag{15}$$

with $\beta = n\alpha$. On the other hand, best-match learning uses the sample for best-match updates, that is, $Q_n(s,a) = (1-\alpha)Q_n^{mf}(s,a) + \alpha\bar{\upsilon}_n$. However, since $Q_i^{mf}(s,a)$ gets updated only when a sample is removed from the queue, $Q_n^{mf}(s,a) = Q_0(s,a)$ in this case. Therefore, the following holds for best-match learning:

$$Q_n(s,a) = (1-\alpha)Q_0(s,a) + \alpha\bar{\upsilon}_n\,. \tag{16}$$

The difference between Equation 15 and Equation 16 illustrates the fundamental advantage of sequence based best-match learning, for which Q_n can be seen as an update with sample (s,a,r,s') using the most recent update target. In contrast, experience replay effectively performs an update using an update target that is an average of the update targets from the different timesteps. Therefore, the older, less accurate update targets still have an effect on Q_n.

6.2 Best-Match Gradient Descent Learning

Since tabular sequence based best-match learning can be implemented by incremental Q-learning updates, it can be easily extended to function approximation by combining it with the general gradient descent update for Q-values (Sutton and Barto, 1998)

$$\theta_{t+1} = \theta_t + \alpha\left[\upsilon_t - Q_t(s_t,a_t)\right]\nabla_{\theta_t}Q_t(s_t,a_t)\,, \tag{17}$$

where θ_t is a weight vector corresponding to the basis functions of the approximation.

Algorithm 6 shows pseudocode for general gradient descent best-match function approximation. Note that a learning rate and the most recent update target are stored per sample. The updates of θ and θ^{mf} are based on Equation 17.

We evaluate a linear version of the best-match gradient descent algorithm by comparing its performance with linear Sarsa(λ) as well as a linear version of experience replay on the mountain car task (Boyan and Moore, 1995; Sutton, 1996; Sutton and Barto, 1998) using the settings as described in Sutton and Barto (1998). This involves tile coding with ten 9x9 tilings, a discount factor of 1, an exploration parameter $\varepsilon = 0$, and Q-values optimistically initialized to 0. Additionally, to bound the run-time of an experiment, an episode is stopped prematurely if the goal is not reached within 1000 steps. Linear Sarsa(λ) is known for its good performance on this task (Sutton and Barto, 1998) and is therefore a good benchmark test. For Sarsa(λ), we use the settings that showed the best performance over the first 20 episodes: $\alpha = 0.14$ and $\lambda = 0.9$ with replacing traces. We tested whether decaying the learning rate improves the performance for a number of different α values around 0.14 but did not find a significant improvement. To make Sarsa(λ) more computationally efficient, traces are cut-off for state-action pairs that were visited longer than 20 timesteps ago. For best-match and experience replay, a queue of the 20 most recent samples is used and a single update sweep through this sample set is performed at every timestep. We optimize the initial learning rate α_0 and the learning rate decay d (see Equation 4). Results are averaged over 5000 independent runs.

Table 5 shows the average return over the first 20 episodes, the optimal parameters, and the computation time per simulation step for the 5000 runs. Figure 14 shows the return as a function of the number of episodes. For trace length/N = 20, the performance of linear best-match is about 27% better than that of linear Sarsa(λ).[6] On the other hand, Sarsa(λ) is about twice as fast.

Surprisingly, while experience replay performed comparably to Sarsa(λ) in the tabular case, in the mountain car task it performs 16% better than linear Sarsa(λ). However, as expected, it performs

6. The linear Sarsa(λ) performance is in accordance with the performance found by several other researchers (`http://webdocs.cs.ualberta.ca/~sutton/book/errata.html`).

Algorithm 6 General Gradient-Descent Best-Match

set N, γ
initialize θ, α and set $\theta^{mf} = \theta$
initialize *SampleQueue* to empty
loop {over episodes}
  initialize s
  while $s \neq$ terminal state **do**
    select action a, based on θ
    take action a, observe s',r
    if size *SampleQueue* $= N$ **then**
      pop sample x from front of the *SampleQueue*
      update θ^{mf} using x
    end if
    decay α; $\upsilon = \emptyset$
    push new sample $\{s, a, r, s', \alpha, \upsilon\}$ to back of *SampleQueue*
    for all samples x update $\upsilon_x \leftarrow r_x + \gamma \cdot V_{s'_x}$ using θ
    for all samples x **do**
      for all features from x: $\theta \leftarrow \theta^{mf}$
    end for
    for all samples x (from front to back of *SampleQueue*) **do**
      update θ using υ_x
    end for
    $s \leftarrow s'$
  end while
end loop

	optimal parameters	average return	standard error	time per step $(\cdot 10^{-6} s)$
best-match, N=20	$\alpha_0 = 0.10$, d = 0.09	-170.1	0.4	3.0
exp. replay, N=20	$\alpha_0 = 0.10$, d = 0.16	-195.1	0.4	2.5
Sarsa(λ), trace=20	$\lambda = 0.9$, $\alpha_0 = 0.14$, d = 0.0	-231.9	0.4	1.5
best-match, N=15	$\alpha_0 = 0.10$, d = 0.03	-176.3	0.4	2.5
best-match, N= 5	$\alpha_0 = 0.10$, d = 0.03	-215.1	0.4	1.5
Sarsa(λ), trace=∞	$\lambda = 0.9$, $\alpha_0 = 0.14$, d = 0.0	-228.2	0.4	6.7

Table 5: Average performance over the first 20 episodes and the computation time per simulation step on the Mountain Car task ('trace' indicates trace length)

worse than linear best-match. Thus, a substantial portion of the performance improvement linear best-match offers over Sarsa(λ) is due to the use of best-match principles, not simply the reuse of data.

Besides a comparison with equal number of samples/updates, it is interesting to make a comparison with equal computation time. To achieve this, we can either increase the sample set size used by experience replay and Sarsa(λ), or decrease the sample set size used by linear best-match, in such

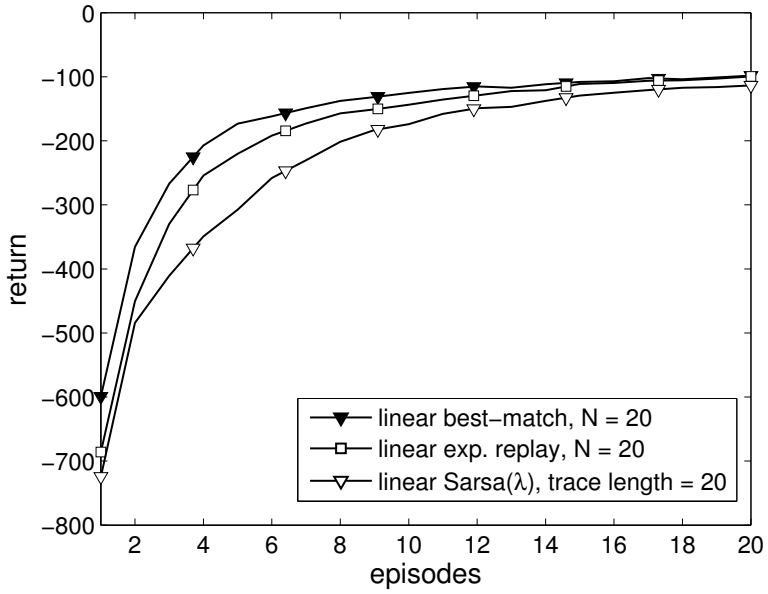

Figure 14: Performance of linear best-match, experience replay and linear Sarsa(λ) on the Mountain Car task using the 20 most recent samples.

a way that the computation times approximately match. We chose to decrease the sample set size of linear best-match. Using $N = 15$ and $N = 5$ resulted in a computation time matching that of experience replay and Sarsa(λ), respectively. Table 5 shows that the performance of linear best-match is also better with equal amount of computation time. In addition, we performed an experiment with Sarsa(λ) without bound on the trace length. This resulted in an average return of -228.2, demonstrating that the performance of Sarsa(λ) cannot be improved significantly by increasing the trace length.

Overall, these results show that best-match learning can be successfully applied to function approximation. Furthermore, they demonstrate that using samples to correct previous updates can lead to better performance that using them to perform additional updates.

7. Discussion

The methods presented in this article approximate solutions to different instantiations of the generalized best-match equations (Definition 11). These best-match equations provide a theoretical foundation for combining model-free learning (through updates of Q^{mf}) with model-based learning (through updates of Q). The resulting methods offer two trade-offs. First, the selection of a sparse, approximate model provides a trade-off between space and performance. Second, the number of best-match updates performed per timestep provides a trade-off between computation cost per timestep and performance. The performance gain offered by best-match learning can be explained from the perspective of the update targets. By performing best-match updates, the update targets from the samples stored in the model are continually recomputed and the Q-values are updated to incorporate any resulting changes.

In the case of best-match LVM, this produces an evaluation method that leads to the same values as TD(λ) with $\lambda_t = \alpha_t(s_t)$ for acyclic tasks, as proven in Theorem 7. This equivalence arises from the fact that both best-match LVM learning and eligibility traces outperform 1-step methods by correcting previous updates with newly obtained samples. However, our theoretical and empirical results suggest that the best-match LVM equations provide a much stronger basis for exploiting this principle.

Theorem 8 proves that best-match LVM evaluation can perform updates that are unbiased with respect to the initial values for an arbitrary MDP, while for TD(λ) this can only be achieved for acyclic tasks. In the control case, Theorem 10 proves convergence in the limit to the optimal Q-values for a general class of best-match LVM control algorithms. Similar converge guarantees do not exist for eligibility traces. In addition, best-match LVM learning avoids the need to choose between different trace types (accumulating or replacing) and does not require an extra λ parameter. Furthermore, in deterministic problems, best-match LVM learning, reduces to model-based learning, as one would expect for an algorithm that makes optimal use of the $O(|\mathcal{S}||\mathcal{A}|)$ space complexity.

Our empirical results show that best-match LVM evaluation substantially outperforms TD(λ) and experience replay (Figure 9), despite having similar computational costs. For the control case, we show that BM-LVM, which uses prioritized sweeping to trade-off computation cost with performance, substantially outperforms not only Q(λ), but also other methods with a space complexity of $O(|\mathcal{S}||\mathcal{A}|)$ (Figure 10). These results illustrate how best-match LVM learning efficiently exploits its stored samples.

Alternatively, best-match learning can be combined with an n-transition model, yielding space complexity between $O(|\mathcal{S}||\mathcal{A}|)$ and $O(|\mathcal{S}|^2|\mathcal{A}|)$. Without using best-match learning, the performance of an NTM is bounded by the quality of the model approximation. In contrast, Theorem 12 proves that BM-NTM converges in the limit to the optimal Q-values. Empirically, we demonstrate that, for any significant space reduction compared to the full model, BM-NTM performs much better than using only the NTM (Figure 13).

Finally, our results demonstrate that the ideas behind best-match learning can be successfully extended to function approximation by combining sequence based best-match learning with gradient descent updates (Algorithm 6). In particular, a linear implementation outperforms Sarsa(λ) and experience replay on a benchmark task (Figure 14).

8. Future Work

Several avenues of future research are suggested by the work presented in this article. For example, in Section 4.2 we proved that the best-match LVM evaluation algorithm can eliminate bias with respect to the initial values. It may be possible to extend this result to the control case. One approach would be to define a state value as the maximum of the Q-values over previously taken actions instead of the maximum over all available actions. However, a potential problem is that the control algorithms compute an approximation of the best-match Q-values, instead of the exact values. It is an open question whether efficient approximations exist that are also unbiased. A second potential problem is that many exploration schemes, such as optimistic initialization, depend on the Q-values and might not work as well when updates are unbiased.

The convergence results for the tabular best-match methods are similar to those of Q-learning: convergence in the limit to the optimal policy. It may be possible, however, to construct best-match

methods that are probably approximately correct (PAC). Since Strehl et al. (2006) showed that a full model is not required for a method to be PAC, we are optimistic that such methods exist.

Finally, it may be possible to develop novel combinations of best-match function approximation with other sample-based approaches such as fitted Q-iteration (Ernst et al., 2005) or LSPI (Lagoudakis and Parr, 2003). By combining the strengths of each approach, such methods could yield even better on-line performance. Fitted Q-iteration, for example, is an off-line algorithm that computes a policy based on a large set of samples, by performing iterative update sweeps through the sample set. For a good approximation, the number of samples should be much larger than the number of parameters of the approximation. By using a combination between a model-free Q-value function and a sample set, a smaller sample set might be possible, reducing the requirements with respect to space and computation, and potentially producing an efficient on-line version of fitted Q-iteration.

9. Conclusion

This article introduced best-match learning, a reinforcement learning approach that combines model-free and model-based learning by using some samples to update a sparse model and the rest to update a model-free Q-value. The final Q-values are computed from best-match updates that combine the model-free Q-values with the sparse model. By controlling which samples enter the model, the size of the model, and hence the space requirements, can be controlled. In the tabular case, the combination with the model-free Q-values ensures convergence to the optimal Q-values for a variety of model approximations.

Our empirical results demonstrate that in the tabular case, when there is not enough space available to store the full model, methods that exploit the best-match equations perform substantially better than methods based on only model-free learning or sparse model-based methods. This suggests that best-match learning should be the strategy of choice when limited space is available.

In addition, we demonstrated that best-match learning can be successfully extended to the function approximation domain, where the sparse model is replaced by an explicit set of samples. An interesting result in this domain is that best-match learning, which uses the sample set to correct previous updates, outperforms experience replay, which uses the same sample set but performs additional updates.

Overall, we believe that best-match learning provides an important missing link between model-free and model-based learning and that the methods introduced in this article constitute a new benchmark for reinforcement learning algorithms that are efficient with respect to both space and computation.

Acknowledgments

The research reported here is part of the Interactive Collaborative Information Systems (ICIS) project, supported by the Dutch Ministry of Economic Affairs, grant nr: BSIK03024.

Appendix A. Proof of Theorem 1

Theorem 1 *Given the same experience sequence, each Q-value from the current state has received the same number of updates using JIT updates (Equation 3) as using regular updates (Equation*

2). However, each Q-value in the update target of a JIT update has received an equal or greater number of updates as in the update target of the corresponding regular update.

Proof To prove the theorem, we need to prove

$$U[\tilde{Q}_t(s_t,a)] = U[Q_t(s_t,a)], \quad \text{for all } a, \tag{18}$$

$$U[\tilde{Q}_{t-1}(s_{t^*+1},a)] \geq U[Q_{t^*}(s_{t^*+1},a)], \quad \text{for all } a, \tag{19}$$

where $U[Q_k]$ is the total number of updates a Q-value has received at time k. From Equation 2 and 3 it follows that for both update types (s_t, a_{t^*}) is updated once between timestep t^* and timestep t, while the Q-values of the other actions of s_t are not updated during this period. Since this applies to all visits and $U[\tilde{Q}_0(s,a)] = U[Q_0(s,a)] = 0$ for all s and a, the total number of updates for a state-action pair is always equal for just-in-time updates and regular updates, when the state is the current state, proving (18).

To prove (19), first assume that a_{t^*} is a returning action, that is, $t-1 = t^*$. In this case clearly (19) is true. Now, assume a_{t^*} is not a returning action, that is, $t-1 > t^*$. From (18) it follows that $U[\tilde{Q}_{t^*+1}(s_{t^*+1},a)] = U[Q_{t^*+1}(s_{t^*+1},a)]$. Since $t-1 \geq t^*+1$ and $U[\tilde{Q}]$ increases monotonically over time, it follows that (19) is true. When state s_{t^*+1} is revisited before t, an extra update is performed and there is at least one action a, for which $U[\tilde{Q}_{t-1}(s_{t^*+1},a)] > U[Q_{t^*}(s_{t^*+1},a)]$. ■

Appendix B. Relationship between Best-Match LVM and TD(λ)

Sutton and Singh (1994) showed that it is possible to perform TD updates that are unbiased with respect to the initial values, by using TD(λ) where λ is made time-dependent and set equal to $\alpha_t(s_t)$. However, TD(λ) can be made unbiased only for acyclic tasks, that is, episodic tasks with no revisits of states within an episode. In this appendix, we prove that best-match LVM evaluation and TD(λ) can lead to the same values for acyclic tasks and that best-match LVM evaluation can perform unbiased updates for all MDPs.

B.1 Background on TD(λ)

The forward view of TD(λ) relates it to the λ-*return* (Watkins, 1989; Jaakkola et al., 1994), defined by

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)},$$

where $R_t^{(n)}$ indicates the *n-step return*, defined by

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}).$$

The λ-*return algorithm* updates state s_t with R_t^λ. It can only be implemented off-line, since it makes use of values from timesteps larger than t for the update of state s_t. While the off-line version of TD(λ) computes the same state values as the λ-return algorithm (Sutton and Barto, 1998), TD(λ) can also be implemented on-line, since it does not rely on values from the future. On-line TD(λ) can lead to more accurate updates than off-line TD(λ), although the interpretation as an incremental

implementation of the λ-return holds only by approximation for the on-line case (Sutton and Barto, 1998).

The backward view of TD(λ) interprets λ as the trace decay parameter of an eligibility trace. Each sample is used to update, not just the current state, but all states, proportional to their *trace* parameter. At each timestep the trace of the current state is increased, while the other traces are decreased by γλ. *Accumulating traces* increase the trace parameter of a visited state by 1, while *replacing traces* set it equal to 1.

Sutton and Singh (1994) proposed several ways for setting α and λ that eliminate bias towards initial state values, normally inherent to temporal-difference methods. One of these is to use TD(λ) where $\lambda_t = \alpha_t(s_t)$ and $\alpha_0(s) = 1$ for all s. This produces the same values as processing a state backwards with TD(0). All the proposed methods eliminate the bias only for acyclic tasks.

The equation for the λ-return with time-dependent λ is (Sutton and Barto, 1998)

$$\begin{aligned} R_t^{\lambda_t} &= \sum_{n=1}^{\infty} R_t^{(n)} (1-\lambda_{t+n}) \prod_{i=1}^{n-1} \lambda_{t+i} \\ &= \sum_{n=1}^{T-t-1} R_t^{(n)} (1-\lambda_{t+n}) \prod_{i=1}^{n-1} \lambda_{t+i} + R_t \prod_{i=1}^{T-t-1} \lambda_{t+i}\,, \end{aligned} \tag{20}$$

where T is the last timestep of the episode and R_t is the complete return. Note that $R_t = R_t^{(T-t)}$.

B.2 Forward View Best-Match LVM Values

The λ-return is based on the experience sequence encountered by the agent when interacting with the environment. We can define for each visited state a *last-visit experience sequence* based on the LVM by going through the transition states defined in the LVM. Using this sequence we define a last-visit version of the n-step return and of a special version of the λ-return.

Definition 13 *The last-visit experience sequence for state s is*

$$s_{[0]}, r_{[1]}, s_{[1]}, r_{[2]}, s_{[2]}, ..., r_{[N]}, s_{[N]}\,,$$

where $s_{[0]} = s$, $s_{[n]} = S'(s_{[n-1]})$ for $n > 0$ and $r_{[n]} = R'(s_{[n-1]})$. The sequence ends when a state is encountered that is terminal, equal to $s_{[0]}$ or that has no transition state. We call $s_{[N]}$ the last-visit sequence end state.

Using this definition, we define a last-visit version of the n-step return.

Definition 14 *The last-visit n-step return of s is the n-step return applied to the last-visit experience sequence of s:*

$$\breve{R}_s^{(n)} = r_{[1]} + \gamma r_{[2]} + \gamma^2 r_{[3]} + ... + \gamma^{n-1} r_{[n]} + \gamma^n V^{mf}(s_{[n]})\,. \tag{21}$$

We can now define a special version of the λ-return, which we call the *last-visit* α*-return*: a last-visit version of the time dependent λ-return (Equation 20) with $\lambda_t = \alpha_t(s_t)$.

Definition 15 *The last-visit* α*-return of s is*

$$\breve{R}_s^{\alpha} = \sum_{n=1}^{N-1} \breve{R}_s^{(n)} (1-\alpha^{[n]}) \prod_{i=1}^{n-1} \alpha^{[i]} + \breve{R}_s^{(N)} \prod_{i=1}^{N-1} \alpha^{[i]}\,, \tag{22}$$

where $\alpha^{[k]}$ is shorthand for $\alpha(s_{[k]})$, $s_{[k]}$ is the k^{th} state from the last-visit experience sequence of s and N is the index of the last-visit sequence end state.

The following lemma relates the last-visit α-return of s to the best-match value of s. The lemma is proven in Appendix C.

Lemma 16 *If the last-visit sequence end state of s is a terminal state, the following equation holds for the best-match value of s:*

$$V^B(s) = (1-\alpha^s)V^{mf}(s) + \alpha^s \breve{R}_s^\alpha .$$

This lemma forms the basis for the proof of the following theorem.

Theorem 7 *For an episodic, acyclic, evaluation task, off-line best-match LVM evaluation computes the same values as off-line TD(λ) with $\lambda_t = \alpha_t(s_t)$.*

Proof Let V_k be the state value function after the off-line updates at the end of episode k. For all states that are visited during an episode, V is updated according to Lemma 16, since the last-visit sequence end state is a terminal state for all these visited states. For the off-line algorithm, before $V_k(s)$ is computed, the update $V_k^{mf}(s) = V_{k-1}(s)$ is performed for all visited states. Therefore, the value updates of the visited states can be written as

$$V_k(s) = (1-\alpha^s)V_{k-1}(s) + \alpha^s \breve{R}_s^\alpha .$$

If the task is acyclic, the last-visit experience sequence of a visited state s is equal to the experience sequence followed by the agent from this state to the terminal state. Therefore, $\breve{R}_s^\alpha = R_t^{\lambda=\alpha_t(s_t)}$. Finally, since the values computed by off-line TD(λ) are equal to the values computed by the λ-return algorithm, off-line TD(λ) with $\lambda_t = \alpha_t(s_t)$ performs the same updates as off-line best-match LVM evaluation. ■

It follows from Theorem 7 that best-match evaluation can also eliminate the bias for acyclic tasks. The next theorem extends this property to a general MDP.

Theorem 8 *The state values computed by the on-line best-match LVM evaluation algorithm (Algorithm 2) are unbiased with respect to the initial state values, when the initial learning rates $\alpha_0(s)$ are set to 1 for all s.*

Proof Algorithm 2 computes at each timestep the best-match value of the current state. We will prove that if the best-match values of visited states computed at timesteps smaller than t are unbiased with respect to the initial state values, then so is the best-match value computed at timestep t. Since for $t = 0$ there are no visited states, it follows by induction that the values computed for all timesteps t are unbiased.

The best-match values are computed using $V^B(s_{[0]}) = c_A + c_B V^B(s_{[N]})$ with c_A and c_B defined as in (8) and (9) respectively. In Section 4.2 we showed that for the current state, $s_{[N]}$ is either a terminal state or equal to $s_{[0]}$. If $s_{[N]}$ is a terminal state, $V^B(s_{[0]}) = c_A$, while if $s_{[0]} = s_{[N]}$, then $V^B(s_{[0]}) = c_A/(1-c_B)$. In either case, the computed best-match value depends only on the variables in c_A and c_B, which consists of the learning rates, $V^{mf}(s_{[i]})$, $s_{[i]}$ and $r_{[i]}$ for $0 \le i \le N$. Clearly, only $V^{mf}(s_{[i]})$ can be affected by the initial state values. $s_{[i]}$ has been visited at least once, else it would not appear in the last-visit experience sequence. If $s_{[i]}$ has been visited once, $V^{mf}(s_{[i]})$ is equal to the

initial value $V_0(s_{[i]})$. However, since we assumed initial learning rates of 1, this value of $V^{mf}(s_{[i]})$ is removed from c_A. If $s_{[i]}$ has been visited more than once, it is equal to the best-match value of $s_{[i]}$ computed at a timestep smaller than t. From this it follows that if the best-match values computed at timesteps smaller than t are unbiased with respect to the initial values, then so is the best-match value computed at timestep t. ■

Appendix C. Proof of Lemma 16

For the sake of brevity, we present only the proof of Lemma 16 for constant α. The proof for state dependent α follows the same pattern.

Lemma 16 *If the last-visit sequence end state of s is a terminal state, the following equation holds for the best-match value of s:*

$$V^B(s) = (1-\alpha^s)V^{mf}(s) + \alpha^s \breve{R}_s^{\alpha}.$$

Proof The best-match values in case of an LVM are defined as the solution of the set of best-match LVM equations (Definition 6). In Section 4.2 we showed that by backward substitution of best-match equations we can express the best-match value of $s_{[0]}$ in terms of the best-match value of $s_{[N]}$. If $s_{[N]}$ is a terminal state, $V^B(s_{[N]}) = 0$ and $V^B(s_{[0]})$ is equal to c_A defined as in (8). This yields

$$\begin{aligned} V^B(s_{[0]}) &= \sum_{i=0}^{N-1} \left((1-\alpha)V^{mf}(s_{[i]}) + \alpha r_{[i+1]}\right) \prod_{k=0}^{i-1} \gamma\alpha, \\ &= \alpha \sum_{k=1}^{N} (\alpha\gamma)^{k-1} r_{[k]} + (1-\alpha) \sum_{k=0}^{N-1} (\alpha\gamma)^k V^{mf}(s_{[k]}). \end{aligned} \tag{23}$$

On the other hand, by substituting the definitions of the last-visit α-return (22) and the last-visit n-step return (21) into the lemma, the following equation for $V^B(s_{[0]})$ appears:

$$\begin{aligned} V^B(s_{[0]}) &= (1-\alpha)V^{mf}(s_{[0]}) + \alpha \Bigg[(1-\alpha) \sum_{k=1}^{N-1} \alpha^{k-1} \left(\sum_{p=1}^{k} \gamma^{p-1} r_{[p]} + \gamma^k V^{mf}(s_{[k]}) \right) \\ &\quad + \alpha^{N-1} \sum_{p=1}^{N} \gamma^{p-1} r_{[p]} \Bigg]. \end{aligned} \tag{24}$$

The rest of this proof shows that (23) is equal to (24).

We start by separating (24) into its state value components (V^c) and its reward components (R^c). We then simplify these components separately:

$$\begin{aligned} V^c &= (1-\alpha)V^{mf}(s_{[0]}) + \alpha(1-\alpha) \sum_{k=1}^{N-1} \alpha^{k-1}\gamma^k V^{mf}(s_{[k]}) \\ &= (1-\alpha)\left(V^{mf}(s_{[0]}) + \sum_{k=1}^{N-1} (\alpha\gamma)^k V^{mf}(s_{[k]}) \right) \\ &= (1-\alpha) \sum_{k=0}^{N-1} (\alpha\gamma)^k V^{mf}(s_{[k]}), \end{aligned}$$

$$
\begin{aligned}
R^c &= (1-\alpha)\sum_{k=1}^{N-1}\sum_{p=1}^{k}\alpha^k\gamma^{p-1}r_{[p]} + \alpha^N\sum_{p=1}^{N}\gamma^{p-1}r_{[p]} \\
&= (1-\alpha)\sum_{p=1}^{N-1}\sum_{k=p}^{N-1}\alpha^k\gamma^{p-1}r_{[p]} + \alpha^N\sum_{p=1}^{N-1}\gamma^{p-1}r_{[p]} + \alpha^N\gamma^{N-1}r_{[N]} \\
&= \sum_{p=1}^{N-1}\left[(1-\alpha)\sum_{k=p}^{N-1}\alpha^k\gamma^{p-1}r_{[p]} + \alpha^N\gamma^{p-1}r_{[p]}\right] + \alpha^N\gamma^{N-1}r_{[N]} \\
&= \sum_{p=1}^{N-1}\left[\left(\sum_{k=p}^{N-1}\alpha^k - \sum_{k=p}^{N-1}\alpha^{k+1} + \alpha^N\right)\gamma^{p-1}r_{[p]}\right] + \alpha^N\gamma^{N-1}r_{[N]} \\
&= \sum_{p=1}^{N-1}\left[\left(\sum_{k=p}^{N}\alpha^k - \sum_{k=p}^{N-1}\alpha^{k+1}\right)\gamma^{p-1}r_{[p]}\right] + \alpha^N\gamma^{N-1}r_{[N]} \\
&= \sum_{p=1}^{N-1}\left[\left(\sum_{j=p-1}^{N-1}\alpha^{j+1} - \sum_{k=p}^{N-1}\alpha^{k+1}\right)\gamma^{p-1}r_{[p]}\right] + \alpha^N\gamma^{N-1}r_{[N]} \\
&= \sum_{p=1}^{N-1}\left[\alpha^p\gamma^{p-1}r_{[p]}\right] + \alpha^N\gamma^{N-1}r_{[N]} \\
&= \alpha\sum_{p=1}^{N}(\alpha\gamma)^{p-1}r_{[p]}\,.
\end{aligned}
$$

Adding these simplified components back together yields Equation 23. ■

Appendix D. Proof of Theorem 10

Theorem 10 *The Q-values of a member of the best-match LVM control class, shown in Algorithm 3, converge to Q^* if the following conditions are satisfied:*

1. *S and A are finite.*
2. $\alpha_t(s,a) \in [0,1]$, $\sum_t \alpha_t(s,a) = \infty$, $\sum_t(\alpha_t(s,a))^2 < \infty$ *with probability 1 (w.p.1) and* $\alpha_t(s,a) = 0$ *unless* $(s,a) = (s_t,a_t)$.
3. $Var\{R(s,a,s')\} < \infty$.
4. $\gamma < 1$.

Proof We prove that the Q-values of an arbitrary instantiation of Algorithm 3 converge in the limit w.p.1 to those of the regular Q-learning algorithm. Because the algorithm requires that each visited state action pair is updated at least once before its revisit, the following equation holds

$$
Q^{mf}_{t+1}(s_t,a_t) = (1-\alpha_t(s_t,a_t))\,Q^{mf}_t(s_t,a_t) + \alpha_t(s_t,a_t)\left(r_{t^*+1} + \max_{a'} Q_{\tau,i}(s_{t^*+1},a')\right),
$$

where t^* is the timestep of the previous visit of (s_t, a_t) and $Q_{\tau,i}$ is the Q-value of s_{t^*+1} that is used in the update target of the last best-match update of (s_t, a_t), at timestep τ. Note that $t^* + 1 \leq \tau \leq t$. Assume that Q-learning is applied to the same state-action sequence produced by the given instantiation of Algorithm 3. We denote the Q-values from Q-learning by $\tilde{Q}$. Subtracting the update equation for Q-learning at time $t^* + 1$ using learning rate $\alpha_t(s_t, a_t)$ and defining $\Delta_t(s,a) = Q_t^{mf}(s,a) - \tilde{Q}_{t^*}(s,a)$ yields

$$\Delta_{t+1}(s_t, a_t) = (1 - \alpha_t(s_t, a_t))\Delta_t(s_t, a_t) + \alpha_t(s_t, a_t)F_t(s_t, a_t), \tag{25}$$

where $F_t(s_t, a_t) = \gamma\left(\max_c Q_{\tau,i}(s_{t^*+1}, c) - \max_c \tilde{Q}_{t^*}(s_{t^*+1}, c)\right)$.

We now prove that Q_t^{mf} and Q_{t^*} converge in the limit to each other using the same lemma used to prove the convergence of Sarsa (Singh et al., 2000):

Lemma 17 *Consider a stochastic process $(\alpha_t, \Delta_t, F_t)$, $t \geq 0$, where $\alpha_t, \Delta_t, F_t : X \to \mathbb{R}$ satisfy the equations:*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x) \ ,$$

where $x \in X$ and $t = 0, 1, 2, \ldots$. Let P_t be a sequence of increasing σ-fields such that α_0 and Δ_0 are P_0-measurable and ζ_t, Δ_t and F_{t-1} are P_t-measurable, $t = 1, 2, \ldots$. Assume that the following conditions hold:

1. *The set X is finite.*
2. *$\alpha_t(x) \in [0,1]$, $\sum_t \alpha_t(x) = \infty$, $\sum_t (\alpha_t(x))^2 < \infty$ w.p.1.*
3. *$\|E\{F_t|P_t\}\| \leq \kappa\|\Delta_t\| + c_t$, where $\kappa \in [0,1)$ and c_t converges to zero w.p.1, and*
4. *$Var\{F_t(x_t)|P_t\} \leq K(1 + \kappa\|\Delta_t\|)^2$, where K is some constant,*

where $\|\cdot\|$ denotes a maximum norm. Then Δ_t converges to zero with probability one.

The correspondence of (25) to Lemma 17 follows from associating X with the set of state-action pairs (s,a) and $\alpha_t(x)$ with $\alpha_t(s,a)$. We now prove that the 4 conditions hold.

The first two conditions follow from the first two conditions of Theorem 10. We define P_t as the set $\{Q_0, \alpha_0, a_0, s_0, ..., r_{t-1}, \alpha_t, a_t, s_t\}$. With this definition, $\text{Var}\{F_t(s_t, a_t)|P_t\} = 0$, satisfying condition 4, and $E\{F_t(s_t, a_t)|P_t\} = F_t(s_t, a_t)$. For $|F_t(s_t, a_t)|$ the following holds:

$$\begin{aligned}
|F_t(s_t, a_t)| &= \gamma|\max_b Q_{\tau,i}(s_{t^*+1}, b) - \max_b \tilde{Q}_{t^*}(s_{t^*+1}, b)| \\
&\leq \gamma||Q_{\tau,i}(u,b) - \tilde{Q}_{t^*}(u,b)|| \\
&= \gamma||\Delta_t(u,b) + Q_{\tau,i}(u,b) - Q_t^{mf}(u,b)|| \\
&\leq \gamma||\Delta_t|| + ||Q_{\tau,i}(u,b) - Q_t^{mf}(u,b)||.
\end{aligned}$$

We further define $F_t(s,a) = 0$ if $(s,a) \neq (s_t, a_t)$. Therefore, $||F_t(s,a)|| = |F_t(s_t, a_t)| \leq \gamma||\Delta_t|| + C_t$, where $C_t = ||Q_{\tau,i}(u,b) - Q_t^{mf}(u,b)||$. We now show that C_t converges to zero w.p.1. For C_t, the following holds:

$$C_t \leq ||Q_{\tau,i}(u,b) - Q_{\tau^*}^{mf}(u,b)|| + ||Q_{\tau^*}^{mf}(u,b) - Q_t^{mf}(u,b)||,$$

where τ^* is the timestep of the last visit of (u,b) before timestep τ. $Q_{\tau,i}(u,b)$ is the result of a best-match update of $Q^{mf}_{\tau^*}(u,b)$ or is equal to it if no best-match update has been performed yet. In the latter case, the first term is zero; in the former case it is

$$Q_{\tau,i}(u,b) = (1-\alpha_\tau(u,b))Q^{mf}_{\tau^*}(u,b) + \alpha_\tau(u,b)\upsilon^{ub}_\tau .$$

Because of condition 2 of Theorem 10, $\alpha_\tau(u,b)$ converges to 0 w.p.1 and $Q_{\tau,i}(u,b)$ converges to $Q^{mf}_{\tau^*}(u,b)$ w.p.1. Therefore, the first term converges to 0 w.p.1. For the same reason, the second term converges to zero.

Thus, the third condition of the lemma also holds and $Q^{mf}(s,a)$ converges to $\tilde{Q}(s,a)$, the Q-values from Q-learning. Because of the convergence guarantee of Q-learning, $Q^{mf}(s,a)$ also converges to $Q^*(s,a)$. Finally, since the Q-values of the given instantiation are a best-match update of $Q^{mf}(s,a)$ and because $\alpha_t(s,a)$ converges to zero w.p.1, this also proves that the Q-values of the instantiation converge to Q^*. ■

Appendix E. Proof of Theorem 12

Theorem 12 *The Q-values of a member of the best-match NTM control class, shown in Algorithm 5, converge to Q^* if the following conditions are satisfied:*

1. *S and A are finite.*
2. $\alpha^{sa}_t \in [0,1]$, $\sum_t \alpha^{sa}_t = \infty$, $\sum_t (\alpha^{sa}_t)^2 < \infty$ *with probability 1 (w.p.1), and* $\alpha^{sa}_t = 0$ *unless* $(s,a) = (s_t,a_t)$ *and* $s_{t+1} \notin NTM(s_t,a_t)$.
3. $Var\{R(s,a,s')\} < \infty$.
4. $\gamma < 1$.

E.1 Preliminaries

In this proof, we indicate the NTM by $\mathcal{M}$. Also, we indicate the model-free Q-value, Q^{mf}, by $\breve{Q}$. In addition, we use a single iteration index j for Q as well as $\breve{Q}$. This global index is increased each time an update (of either $\breve{Q}$ or Q) occurs. Thus, j is equal to the total number of model-free updates plus best-match updates that have occurred since the start of an episode. Clearly, $t \to \infty$ implies $j \to \infty$.

By denoting the state-action pair that gets updated by the j-th update as (s_j,a_j), we can write the model-free (mf) update as

$$\breve{Q}_{j+1}(s_j,a_j) = (1-\alpha^{s_ja_j})\breve{Q}_j(s_j,a_j) + \alpha^{s_ja_j}\left[r_{j+1} + \gamma \max_{a'} Q_j(s'_{j+1},a')\right], \tag{26}$$

where r_{j+1} and s'_{j+1} are the reward and transition state from the sample $(s_t,a_t,r_{t+1},s_{t+1})$ corresponding to (s_j,a_j). We use s'_{j+1} instead of s_{j+1}, since s'_{j+1}, the transition state for s_j, is in general not equal to s_{j+1}, the state that receives an update at iteration step $j+1$. The best-match (bm) update is

$$Q_{j+1}(s_j,a_j) = w_0^{s_j,a_j}\breve{Q}_j(s_j,a_j) + (1-w_0^{s_ja_j})\left[\hat{\mathcal{R}}_{s_ja_j} + \gamma\sum_{s'}\hat{\mathcal{P}}^{s'}_{s_ja_j}\max_{a'} Q_j(s',a')\right].$$

Note that there is no specific sample corresponding to a best-match update, since the update is based on the model estimate and can occur multiple times per timestep.

Let $\mathcal{P}_{sa}^{\mathcal{M}} = \sum_{s' \in \mathcal{M}} \mathcal{P}_{sa}^{s'}$. If $\mathcal{P}_{sa}^{\mathcal{M}} = 0$, w_0^{sa} will always be 1 and the best-match update reduces to $Q_{j+1}(s_j,a_j) = \breve{Q}_j(s_j,a_j)$. We make this explicit by the following equation:

$$Q_{j+1}(s_j,a_j) = \begin{cases} \breve{Q}_j(s_j,a_j) & \text{if } \mathcal{P}_{sa}^{\mathcal{M}} = 0 \\ Y_j(s_j,a_j) & \text{if } \mathcal{P}_{sa}^{\mathcal{M}} > 0\,, \end{cases} \tag{27}$$

with

$$Y_j(s_j,a_j) = w_0^{s_j,a_j} \breve{Q}_j(s_j,a_j) + (1 - w_0^{s_j a_j}) \left[\hat{\mathcal{R}}_{s_j a_j} + \gamma \sum_{s'} \hat{\mathcal{P}}_{s_j a_j}^{s'} \max_{a'} Q_j(s',a') \right].$$

Each time a sample is observed by the algorithm, w_0 gets updated. In addition, when the sample is part of $\mathcal{M}$, $\hat{\mathcal{R}}$ and $\hat{\mathcal{P}}$ get updated. Therefore, the values of these variables can change between iteration steps. However, for readability, we omit the j subscript for these variables. From the definition of w_0, $\hat{\mathcal{R}}$ and $\hat{\mathcal{P}}$, and the law of large numbers, it follows that in the limit the following holds:[7]

$$\lim_{j\to\infty} w_0^{sa} = 1 - \mathcal{P}_{sa}^{\mathcal{M}}\,, \tag{28}$$

$$\lim_{j\to\infty} \hat{\mathcal{R}}_{sa} = \sum_{s' \in \mathcal{M}} \mathcal{P}_{sa}^{s'} \mathcal{R}_{sa}^{s'} / \mathcal{P}_{sa}^{\mathcal{M}}\,, \tag{29}$$

$$\lim_{j\to\infty} \hat{\mathcal{P}}_{sa}^{s'} = \mathcal{P}_{sa}^{s'} / \mathcal{P}_{sa}^{\mathcal{M}}\,. \tag{30}$$

In general, the model-free Q-values, $\breve{Q}$, will not converge to Q^*, since they do not receive updates from samples corresponding to the next states stored by the NTM. However, as part of the proof, we show that the model-free Q-values converge to an alternative value, which we indicate by $\breve{Q}^*$. This value is defined as[8]

$$\breve{Q}^*(s,a) = \sum_{s' \notin \mathcal{M}} \mathcal{P}_{sa}^{s'} [\mathcal{R}_{sa}^{s'} + \gamma \max_{a'} Q^*(s',a')] / (1 - \mathcal{P}_{sa}^{\mathcal{M}})\,. \tag{31}$$

Using this equation, we can express Q^* as

$$\begin{aligned} Q^*(s,a) &= \sum_{s' \notin \mathcal{M}} \mathcal{P}_{sa}^{s'} [\mathcal{R}_{sa}^{s'} + \gamma \max_{a'} Q^*(s',a')] + \sum_{s' \in \mathcal{M}} \mathcal{P}_{sa}^{s'} [\mathcal{R}_{sa}^{s'} + \gamma \max_{a'} Q^*(s',a')] \\ &= (1 - \mathcal{P}_{sa}^{\mathcal{M}}) \breve{Q}^*(sa) + \sum_{s' \in \mathcal{M}} \mathcal{P}_{sa}^{s'} [\mathcal{R}_{sa}^{s'} + \gamma \max_{a'} Q^*(s',a')]\,. \end{aligned} \tag{32}$$

Note that it follows from (32), that

$$Q^*(s,a) = \breve{Q}^*(s,a)\,, \qquad \text{if } \mathcal{P}_{sa}^{\mathcal{M}} = 0\,. \tag{33}$$

Convergence of Q_j to Q^* requires convergence of $\breve{Q}_j$ to $\breve{Q}^*$, and vice versa. To deal with this mutual dependence relation, we simultaneously prove their convergence. To achieve this, we define

7. Note that $\hat{\mathcal{R}}_{sa}$ and $\hat{\mathcal{P}}_{sa}^{s'}$ do not converge to $\mathcal{R}_{sa}$ and $\mathcal{P}_{sa}^{s'}$, but to normalized values of these variables.

8. For $\mathcal{P}_{sa}^{\mathcal{M}} = 1$, that is, when all samples are stored by the NTM, $\breve{Q}^*(s,a)$ is not defined. However, in this case, $\breve{Q}(s,a)$ does not receive any updates, nor is it used by any other update. Therefore, we can safely ignore the value $\breve{Q}(s,a)$, and consequently $\breve{Q}^*(s,a)$, if $\mathcal{P}_{sa}^{\mathcal{M}} = 1$.

a function $U : \mathcal{S} \times \mathcal{A} \times \mathcal{B} \to \mathbb{R}$ that encompasses both functions Q and $\breve{Q}$. $\mathcal{B}$ is a set consisting of only two elements: 'mf' and 'bm', which indicate the *Q-value type*. We define U_j as

$$U_j(s,a,b) = \begin{cases} \breve{Q}_j(s,a) & \text{if b = 'mf'} \\ Q_j(s,a) & \text{if b = 'bm'}. \end{cases} \tag{34}$$

Both updates (26) and (27) can now be interpreted as updates of $U_j(s_j,a_j,b_j)$. It follows from (34) that when the model-free update is performed, b_j = 'mf', while for the best-match update b_j = 'bm'.

We will prove convergence of U_j to U_j^*, defined as

$$U^*(s,a,b) = \begin{cases} \breve{Q}^*(s,a) & \text{if b = 'mf'} \\ Q^*(s,a) & \text{if b = 'bm'}. \end{cases}$$

The difficulty with this proof is that we cannot simply apply Lemma 17 (or similar stochastic approximation lemmas), used to prove convergence of BM-LVM, since the $\sum_t(\alpha_t(x_t))^2 < \infty$ condition of Lemma 17 is not met for b = 'bm'. On the other hand, a related lemma can be deduced (see Appendix F), that does not require $\sum_t(\alpha_t(x_t))^2 < \infty$, however, it requires that the contraction condition holds for the value of F_t, instead of its expected value. Hence, also this lemma cannot be directly applied.

To deal with this, we define a related function U'_j, that does comply with the $\sum_t(\alpha_t(x_t))^2 < \infty$ condition, hence we can prove convergence of it to U^* using Lemma 17. On the other hand, the difference between U'_j and U_j complies with all the conditions of Lemma 20, hence we can prove that U_j converges to U'_j using Lemma 20. Adding these two results together, proves the theorem.

We define U'_j as

$$U'_j(s,a,b) = \begin{cases} \breve{Q}'(s,a) & \text{if b = 'mf'} \\ Q'(s,a) & \text{if b = 'bm'}. \end{cases}$$

$\breve{Q}'$ and Q' are updated using the same sample sequence as used for $\breve{Q}$ and Q. The update for $\breve{Q}'$ is

$$\breve{Q}'_{j+1}(s_j,a_j) = (1-\alpha^{s_ja_j})\breve{Q}'_j(s_j,a_j) + \alpha^{s_ja_j}\left[r_{j+1} + \gamma \max_{a'} Q'_j(s'_{j+1},a')\right],$$

while the update for Q' is

$$Q'_{j+1}(s_j,a_j) = \begin{cases} \breve{Q}'_j(s_j,a_j) & \text{if } \mathcal{P}^{\mathcal{M}}_{sa} = 0 \\ (1-\beta^{s_ja_j})Q'_j(s_j,a_j) + \beta^{s_ja_j}Y'_j(s_j,a_j) & \text{if } \mathcal{P}^{\mathcal{M}}_{sa} > 0, \end{cases} \tag{35}$$

with

$$Y'_j(s_j,a_j) = w_0^{s_j,a_j}\breve{Q}'_j(s_j,a_j) + (1-w_0^{s_ja_j})\left[\hat{\mathcal{R}}_{s_ja_j} + \gamma\sum_{s'}\hat{\mathcal{P}}^{s'}_{s_ja_j}\max_{a'}Q'_j(s',a')\right].$$

Note that the only difference with the updates of Q and $\breve{Q}$ is the way Q' is updated for $\mathcal{P}^{\mathcal{M}}_{sa} > 0$. Instead of setting $Q'_{j+1}(s_j,a_j)$ equal to $Y'_j(s_j,a_j)$, it is set equal to a weighted average of $Y'_j(s_j,a_j)$ and $Q'_j(s_j,a_j)$. The weighting is controlled by β_j, which is an arbitrary learning rate with properties $\beta_j^{sa} \in [0,1]$, $\sum_j \beta_j^{sa} = \infty$, $\sum_j(\beta_j^{sa})^2 < \infty$ w.p.1., and $\beta_j^{sa} = 0$ unless $(s,a) = (s_j,a_j)$ and b_j = 'bm'.[9] Because of this learning rate, Lemma 17 can be used to prove convergence of U'_j to U^*.

9. Note that such a β always exists.

E.2 Convergence of U'_j to U^*

Lemma 18 *$U'_j(s,a,b)$ converges in the limit to $U^*(s,a,b)$ w.p.1.*

Proof We define $\Delta'(s,a,b) = U'_j(s,a,b) - U^*_j(s,a,b)$ and will prove that $\Delta'(s,a,b)$ converges to 0 using Lemma 17. For $b_j =$ 'bm', we use the contraction factor κ^{sa}, defined as

$$\kappa^{sa} = (1 - \mathcal{P}^{\mathcal{M}}_{sa}) + \gamma \mathcal{P}^{\mathcal{M}}_{sa} \,. \tag{36}$$

To ensure that $\kappa^{sa} < 1$, $\mathcal{P}^{\mathcal{M}}_{sa}$ has to be larger than 0. Therefore, we exclude (s,a,b) triples for which $b =$ 'bm' $\wedge\, \mathcal{P}^{\mathcal{M}}_{sa} = 0$ from the domain of Δ'. This can be done, because Algorithm 5 states that at least one best-match update occurs in between two model-free updates. Therefore, if $\mathcal{P}^{\mathcal{M}}_{sa} = 0$, $Q'_j(s,a)$ is either equal to $\breve{Q}'_j(s,a)$ or one (model-free) update apart. Since α^{sa}_j converges to 0, it follows that $Q'_j(s,a)$ converges in the limit to $\breve{Q}'_j(s,a)$. Alternatively, we can say

$$Q'_j(s,a) = \breve{Q}'_j(s,a) + c'_j(s,a)\,, \qquad \text{if } \mathcal{P}^{\mathcal{M}}_{sa} = 0\,, \tag{37}$$

with $c'_j(s,a)$ converging to 0 w.p.1.[10] Combining this with (33), the following holds:

$$\lim_{j\to\infty} \breve{Q}'_j(s,a) = \breve{Q}^*(s,a) \quad \Rightarrow \quad \lim_{j\to\infty} Q'_j(s,a) = Q^*(s,a)\,, \qquad \text{if } \mathcal{P}^{\mathcal{M}}_{sa} = 0\,. \tag{38}$$

Note, $\|\breve{Q}'_j - \breve{Q}^*\| \leq \|\Delta'_j\|$. However, because of the exclusion of $(s,a,$'bm'$)$ triples with $\mathcal{P}^{\mathcal{M}}_{sa} = 0$, $\|Q'_j - Q^*\| \leq \|\Delta_j\|$ does not hold in general. Instead, the following holds:

$$\begin{aligned}
\|Q'_j - Q^*\| &= \max(\|Q'_j - Q^*\|_{P^{\mathcal{M}}_{sa}>0}, \|Q'_j - Q^*\|_{P^{\mathcal{M}}_{sa}=0}) \\
&\leq \max(\|Q'_j - Q^*\|_{P^{\mathcal{M}}_{sa}>0}, \|\breve{Q}'_j - \breve{Q}^*\|_{P^{\mathcal{M}}_{sa}=0} + \|c'_j\|) \\
&\leq \max(\|U'_j - U^*\|, \|U'_j - U^*\| + \|c'_j\|) \\
&= \|U'_j - U^*\| + \|c'_j\| \\
&= \|\Delta'\| + \|c'_j\|\,.
\end{aligned}$$

Because of the exclusion of the (s,a,b) triples mentioned above, for all $(s,a,$'bm'$)$ triples in the domain of Δ'_j, $\mathcal{P}^{\mathcal{M}}_{sa} > 0$.

Δ'_j is updated according to

$$\Delta'_{j+1}(s,a,b) = (1 - \zeta'_j(s,a,b))\Delta'_j(s,a,b) + \zeta'_j(s,a,b)\, F'_j(s,a,b)\,.$$

For $(s,a,b) \neq (s_j,a_j,b_j)$, $\zeta'_j(s,a,b) = 0$ and $F'_j(s,a,b) = 0$. For (s_j,a_j,b_j) the following holds:

$$\zeta'_j(s_j,a_j,b_j) = \begin{cases} \alpha^{s_j a_j}_j & \text{if } b_j = \text{'mf'} \\ \beta^{s_j a_j}_j & \text{if } b_j = \text{'bm'}\,, \end{cases}$$

$$F'_j(s_j,a_j,b_j) = \begin{cases} r_{j+1} + \gamma \max_{a'} Q'_j(s'_{j+1},a') - \breve{Q}^*(s_j,a_j) & \text{if } b_j = \text{'mf'} \\ Y'_j(s_j,a_j) - Q^*(s_j,a_j) & \text{if } b_j = \text{'bm'}\,. \end{cases}$$

10. We use the notational convention to indicate variables that converge to 0 with probability 1 with lowercase, Latin letters: c, d, e, ...

We now prove that Δ'_j converges to zero, by showing the conditions for Lemma 17 hold, using the σ-field P_j, defined as[11]

$$\begin{aligned} P_0 &= \{Q'_0, \breve{Q}'_0, \zeta_0, w_{0,0}, \breve{\mathcal{P}}_0, \breve{\mathcal{R}}_0, s_0, a_0\}, \\ P_j &= P_{j-1} \cap \{r_j, s'_j, \zeta_j, w_{0,j}, \breve{\mathcal{P}}_j, \breve{\mathcal{R}}_j, s_j, a_j\}. \end{aligned}$$

Conditions 1, 2 and 4 of the Lemma 17 follow from conditions 1,2, and 3 of Theorem 12 and the conditions that hold for β_j^{sa}. Condition 3 of the lemma, we prove below.

For b_j = 'mf', using (31), the following holds:

$$\begin{aligned} |E\{F'_j(s_j,a_j,\text{'mf'})|P_j\}| &= \Big| \sum_{s' \notin \mathcal{M}} \mathcal{P}^{s'}_{s_j a_j} [\mathcal{R}^{s'}_{s_j a_j} + \gamma \max_{a'} Q'_j(s',a')]/(1-\mathcal{P}^{\mathcal{M}}_{s_j a_j}) - \breve{Q}^*(s_j,a_j) \Big| \\ &= \gamma \sum_{s' \notin \mathcal{M}} \mathcal{P}^{s'}_{s_j a_j} \Big| \max_{a'} Q'_j(s',a') - \max_{a'} Q^*(s',a') \Big| / (1-\mathcal{P}^{\mathcal{M}}_{s_j a_j}) \\ &\leq \gamma \|Q'_j - Q^*\| \\ &\leq \gamma \|\Delta'_j\| + \gamma \|c'_j\|. \end{aligned} \tag{39}$$

For b_j = 'bm', using (32), we can write

$$\begin{aligned} |F'_j(s_j,a_j,\text{'bm'})| &= |Y'_j(s_j,a_j) - Q^*(s_j,a_j)| \\ &\leq \Big| (1-\mathcal{P}^{\mathcal{M}}_{s_j a_j})(\breve{Q}'_j(s_j,a_j) - \breve{Q}^*(s_j,a_j)) \\ &\quad + \gamma \sum_{s' \in \mathcal{M}} \mathcal{P}^{s'}_{s_j a_j} [\max_{a'} Q'_j(s',a') - \max_{a'} Q^*(s',a')] \Big| \\ &+ \Big| \Big[w_0^{s_j a_j} - (1-\mathcal{P}^{\mathcal{M}}_{s_j a_j}) \Big] \cdot \breve{Q}'_j(s_j,a_j) \Big| \\ &+ \Big| (1-w_0^{s_j a_j}) \hat{\mathcal{R}}_{s_j a_j} - \sum_{s' \in \mathcal{M}} \mathcal{P}^{s'}_{s_j a_j} \mathcal{R}^{s'}_{s_j a_j} \Big| \\ &+ \gamma \Big| \sum_{s' \in \mathcal{M}} \Big[(1-w_0^{s_j a_j}) \hat{\mathcal{P}}^{s'}_{s_j a_j} - \mathcal{P}^{s'}_{s_j a_j} \Big] \cdot \max_{a'} Q'_j(s',a') \Big|. \end{aligned}$$

The sum of the last three terms we call $d_j(s_j,a_j)$. By substituting (28), (29) and (30) in these three terms, it follows that $\lim_{j\to\infty} d_j(s_j,a_j) = 0$. We can further bound $|F'_j(s_j,a_j,\text{'bm'})|$ as follows:

$$\begin{aligned} |F'_j(s_j,a_j,\text{'bm'})| &\leq (1-\mathcal{P}^{\mathcal{M}}_{s_j a_j}) \|\breve{Q}_j - \breve{Q}^*\| + \gamma \mathcal{P}^{\mathcal{M}}_{s_j a_j} \|Q_j - Q^*\| + d_j(s_j,a_j) \\ &\leq (1-\mathcal{P}^{\mathcal{M}}_{s_j a_j}) \|\Delta'_j\| + \gamma \mathcal{P}^{\mathcal{M}}_{s_j a_j} \left(\|\Delta'_j\| + \|c_j\| \right) + d_j(s_j,a_j) \\ &\leq \left((1-\mathcal{P}^{\mathcal{M}}_{s_j a_j}) + \gamma \mathcal{P}^{\mathcal{M}}_{s_j a_j} \right) \cdot \|\Delta'_j\| + \gamma \mathcal{P}^{\mathcal{M}}_{s_j a_j} \|c'_j\| + d_j(s_j,a_j) \\ &= \kappa^{s_j a_j} \|\Delta'_j\| + \gamma \mathcal{P}^{\mathcal{M}}_{s_j a_j} \|c'_j\| + d_j(s_j,a_j). \end{aligned} \tag{40}$$

Note $\|c'_j\|$, as well as $d_j(s_j,a_j)$, converge to 0. Note also that $\kappa^{s_j a_j} < 1$, since $\mathcal{P}^{\mathcal{M}}_{s_j a_j} > 0$ and $\gamma < 1$. From (39) and (40) it follows that the third condition of Lemma 17 is also satisfied. Hence, all conditions hold and Δ'_j converges to 0 w.p.1. Combining this with (38), proves Lemma 18.

■

11. There is no explicit sample related to a best-match update. For consistency, we define $r_j = \emptyset$ and $s'_j = \emptyset$ if $b_{j-1} =$ 'bm'.

E.3 Convergence of U_j to U'_j

Lemma 19 *$U_j(s,a,b)$ converges in the limit to $U'_j(s,a,b)$ w.p.1.*

Proof We define $\Delta(s,a,b) = U'_j(s,a,b) - U_j(s,a,b)$ and will prove that $\Delta(s,a,b)$ converges to 0 using Lemma 20. We exclude $(s,a,\text{'bm'})$ triples for which $\mathcal{P}^{\mathcal{M}}_{sa} = 0$ from the domain of Δ. Similar to the reasoning behind (38) and (37), we can deduce

$$Q_j(s,a) = \breve{Q}_j(s,a) + c_j(s,a)\,, \qquad \text{if } \mathcal{P}^{\mathcal{M}}_{sa} = 0\,,$$

with $c_j(s,a)$ converging to 0 in the limit, as well as

$$\lim_{j\to\infty}\Big(\breve{Q}'_j(s,a) - \breve{Q}_j(s,a)\Big) = 0 \quad \Rightarrow \quad \lim_{j\to\infty}\Big(Q'_j(s,a) - Q_j(s,a)\Big) = 0\,, \qquad \text{if } \mathcal{P}^{\mathcal{M}}_{sa} = 0\,. \tag{41}$$

Note, $\|\breve{Q}'_j - \breve{Q}_j\| \leq \|\Delta_j\|$. However, $\|Q'_j - Q_j\| \leq \|\Delta_j\|$ does not hold in general, because of the exclusion of (s,a,'bm') triples with $\mathcal{P}^{\mathcal{M}}_{sa} = 0$ from the domain of Δ_j. Instead, the following holds:

$$\begin{aligned}
\|Q'_j - Q_j\| &= \max(\|Q'_j - Q_j\|_{P^{\mathcal{M}}_{sa}>0}, \|Q'_j - Q_j\|_{P^{\mathcal{M}}_{sa}=0}) \\
&\leq \max(\|Q'_j - Q_j\|_{P^{\mathcal{M}}_{sa}>0}, \|\breve{Q}'_j - \breve{Q}_j\|_{P^{\mathcal{M}}_{sa}=0} + \|c_j\| + \|c'_j\|) \\
&\leq \max(\|U'_j - U_j\|, \|U'_j - U^*\| + \|c_j\| + \|c'_j\|) \\
&= \|U'_j - U_j\| + \|c_j\| + \|c'_j\| \\
&= \|\Delta'_j\| + c''_j\,,
\end{aligned}$$

with $c''_j = \|c_j\| + \|c'_j\|$ converging to 0 w.p.1.

For $\mathcal{P}^{\mathcal{M}}_{sa} > 0$ we can rewrite (35) as

$$\begin{aligned}
Q'_{j+1}(s_j,a_j) &= (1-\beta^{s_j a_j})Q'_j(s_j,a_j) + \beta^{s_j a_j} Y'_j(s_j,a_j) \\
&= Y'_j(s_j,a_j) + (1-\beta^{s_j a_j})[Q'_j(s_j,a_j) - Y'_j(s_j,a_j)]\,.
\end{aligned}$$

In Section E.2 we proved that $\Delta'_j(s,a,\text{'bm'}) = Q'(s,a) - Q^*(s,a)_j$ converges to 0 w.p.1. On the other hand, it follows from (40), that $F'_j(s_j,a_j,\text{'bm'})$, which is equal to $Y'_j(s_j,a_j) - Q^*(s_j,a_j)$, also converges to 0 w.p.1. Therefore, both $Q'_j(s_j,a_j)$ and $Y'_j(s_j,a_j)$ converge to the same value, so we can write

$$Q'_{j+1}(s_j,a_j) = Y'_j(s_j,a_j) + e_j(s_j,a_j)\,, \qquad \text{if } \mathcal{P}^{\mathcal{M}}_{s_j a_j} > 0\,,$$

with $e_j(s_j,a_j)$ converging to 0 w.p.1.

Δ_j is updated according to

$$\Delta_{j+1}(s,a,b) = (1-\zeta_j(s,a,b))\Delta_j(s,a,b) + \zeta_j(s,a,b)\,F_j(s,a,b)\,.$$

For $(s,a,b) \neq (s_j,a_j,b_j)$, $\zeta_j(s,a,b) = 0$ and $F_j(s,a,b) = 0$. While for (s_j,a_j,b_j) the following holds:

$$\zeta_j(s_j,a_j,b_j) = \begin{cases} \alpha_j^{s_j a_j} & \text{if } b_j = \text{'mf'} \\ 1 & \text{if } b_j = \text{'bm'}\,, \end{cases}$$

and

$$F_j(s_j,a_j,b_j) = \begin{cases} \gamma \max_{a'} Q'_j(s'_{j+1},a') - \gamma \max_{a'} Q_j(s'_{j+1},a') & \text{if } b_j = \text{'mf'} \\ Y'_j(s_j,a_j) - Y_j(s_j,a_j,b_j) + e_j(s_j,a_j) & \text{if } b_j = \text{'bm'}\,. \end{cases}$$

We now check the three conditions of Lemma 20. Conditions 1 and 2 from the lemma follow from conditions 1 and 2 of Theorem 12. Condition 3, we prove below.

For b_j = 'mf', the following holds:

$$\begin{aligned} |F_j(s_j,a_j,\text{'mf'})| &= \gamma\left|\max_{a'} Q'_j(s'_{j+1},a') - \max_{a'} Q_j(s'_{j+1},a')\right| \\ &\leq \gamma\|Q'_j - Q_j\| \\ &\leq \gamma\|\Delta_j\| + \gamma c''_j\,, \end{aligned} \tag{42}$$

while for b_j = 'bm', we can write

$$\begin{aligned} |F_j(s_j,a_j,\text{'bm'})| &= |Y'_j(s_j,a_j) - Y_j(s_j,a_j) + e_j(s_j,a_j,b_j)| \\ &\leq w_0^{s_j,a_j}\,|\breve{Q}'_j(s_j,a_j) - \breve{Q}_j(s_j,a_j)| + |e_j(s_j,a_j)| + \\ &\qquad \gamma(1-w_0^{s_ja_j})\sum_{s'}\hat{\mathcal{P}}^{s'}_{s_ja_j}\left|\max_{a'} Q'_j(s',a') - \max_{a'} Q_j(s',a')\right| \\ &\leq w_0^{s_j,a_j}\|\Delta_j\| + \gamma(1-w_0^{s_ja_j})\|\Delta_j\| + |e_j(s_j,a_j)| + \gamma(1-w_0^{s_ja_j})\,c''_j \\ &= \left((1-\mathcal{P}^{\mathcal{M}}_{s_ja_j}) + \gamma\mathcal{P}^{\mathcal{M}}_{s_ja_j}\right)\|\Delta_j\| + |e_j(s_j,a_j)| + \gamma(1-w_0^{s_ja_j})\,c''_j + \\ &\qquad \left(w_0^{s_j,a_j} + \gamma(1-w_0^{s_ja_j}) - (1-\mathcal{P}^{\mathcal{M}}_{s_ja_j}) - \gamma\mathcal{P}^{\mathcal{M}}_{s_ja_j}\right)\|\Delta_j\|\,. \end{aligned}$$

We define

$$\begin{aligned} f_j(s_j,a_j) &= \left(w_0^{s_j,a_j} + \gamma(1-w_0^{s_ja_j}) - (1-\mathcal{P}^{\mathcal{M}}_{s_ja_j}) - \gamma\mathcal{P}^{\mathcal{M}}_{s_ja_j}\right)\|\Delta_j\| \\ &\qquad + |e_j(s_j,a_j)| + \gamma(1-w_0^{s_ja_j})\,c''_j\,. \end{aligned}$$

Note that $\lim_{j\to\infty} f_j = 0$, since e_j and c''_j converge to 0 and $w_0^{s_j,a_j}$ converges to $1-\mathcal{P}^{\mathcal{M}}_{s_ja_j}$. Using this definition and (36), we can write

$$|F_j(s_j,a_j,\text{'bm'})| \leq \kappa^{s_ja_j}\|\Delta_j\| + f_j(s_j,a_j)\,. \tag{43}$$

Note that $\kappa^{s_ja_j} < 1$. From (42) and (43) it follows that the third condition of Lemma 20 is also satisfied. Hence, all conditions hold and Δ_j converges to 0 w.p.1.Combining this with (41), proves Lemma 19. ■

E.4 Proof of Theorem 12

Because U'_j converges to U^* (Lemma 18) and U_j converges to U'_j (Lemma 19), it follows that also U_j converges to U^*. From this it follows that Q converges to Q^*, proving Theorem 12.

Appendix F. Lemma 20

Lemma 20 *Consider a stochastic process (α_t,Δ_t,F_t), $t\geq 0$, where $\alpha_t,\Delta_t,F_t : X\to\mathbb{R}$ satisfy the equations:*

$$\Delta_{t+1}(x) = (1-\alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x)\,,$$

where $x\in X$ and $t = 0,1,2,\ldots$. Assume that the following conditions hold:

1. *The set X is finite.*

2. $\alpha_t(x) = [0,1]$, $\sum_t \alpha_t(x) = \infty$.

3. $\|F_t\| \leq \kappa\|\Delta_t\| + c_t$, *where* $\kappa \in [0,1)$ *and* c_t *converges to zero w.p. 1 ,*

where $\|\cdot\|$ *denotes a maximum norm. Then* Δ_t *converges to zero with probability one.*

Note that this lemma is similar to Lemma 17, but the conditions for the learning rates are less strict ($\sum_t(\alpha_t(x_t))^2 < \infty$ is missing), while the condition for F_t is more strict (condition 3 uses the value of F_t instead of its expected value).

Proof The outline of this proof is that we define a related process Δ'_t that converges to 0 and show that $\|\Delta_t\| \leq \|\Delta'_t\|$ for all t. We will ignore c_t in this proof. This can be safely done, since c_t converges to zero, $\kappa < 1$ and $\sum_t \alpha_t(x) = \infty$ for all x. Therefore, this term is asymptotically unimportant.

We define $\Delta'_0(x) = \|\Delta_0\|$ for all x. For $t > 0$, $\Delta'_t(x)$ is defined as

$$\Delta'_{t+1}(x) = (1 - \beta_t(x))\Delta'_t(x) + \beta_t(x)\kappa\|\Delta'_t\|\,, \tag{44}$$

with $\beta_t(x) \leq \alpha_t(x)$ and $\beta_t(x) \in [0,1]$, $\sum_t \beta_t(x) = \infty$, $\sum_t(\beta_t(x))^2 < \infty$ w.p.1. It follows from (44) that $\|\Delta'_{t+1}\| \leq \|\Delta'_t\|$. It also follows that if $\Delta'_t(x) \geq \kappa\|\Delta'_t\|$ then $\Delta'_{t+1}(x) \geq \kappa\|\Delta'_t\| \geq \kappa\|\Delta'_{t+1}\|$. And since $\Delta'_0(x) \geq \kappa\|\Delta'_0\|$ it follows that

$$\Delta'_t(x) \geq \kappa\|\Delta'_t\|\,, \qquad \text{for all } t\,. \tag{45}$$

Using Lemma 17, it can easily be shown that Δ' converges in the limit to 0 w.p.1.

We now prove that $\|\Delta_t\| \leq \|\Delta'_t\|$ for all t. We start by proving

$$|\Delta_t(x)| \leq \Delta'_t(x) \quad \text{for all } x \quad \Rightarrow \quad |\Delta_{t+1}(x)| \leq \Delta'_{t+1}(x) \quad \text{for all } x\,. \tag{46}$$

Assuming the left part of (46), for $|\Delta_{t+1}(x)|$ the following holds:

$$\begin{aligned} |\Delta_{t+1}(x)| &\leq (1 - \alpha_t(x))|\Delta_t(x)| + \alpha_t(x)\,\kappa\|\Delta_t\| \\ &\leq (1 - \alpha_t(x))\Delta'_t(x) + \alpha_t(x)\,\kappa\|\Delta'_t\|\,. \end{aligned}$$

Since (45) and $\beta_t(x) \leq \alpha_t(x)$, we can continue as

$$\begin{aligned} |\Delta_{t+1}(x)| &\leq (1 - \beta_t(x))\Delta'_t(x) + \beta_t(x)\,\kappa\|\Delta'_t\| \\ &\leq \Delta'_{t+1}(x)\,. \end{aligned}$$

This proves (46). And since $|\Delta_0(x)| \leq \Delta'_0(x)$, it follows that $|\Delta_t(x)| \leq \Delta'_t(x)$ holds for all t, and hence, $\|\Delta_t\| \leq \|\Delta'_t\|$ proving the lemma. ■

References

C.G. Atkeson, A.W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1):11–73, 1997.

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

J. Boyan and A.W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, 1995.

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.

C. Diuk, L. Li, and B.R. Leffler. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503–556, 2005.

T. Jaakkola, M.I. Jordan, and S. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.

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

M. Kearns and S. Singh. Finite-sample convergence rates for Q-learning and indirect algorithms. *Advances in Neural Information Processing Systems*, 11:996–1002, 1999. ISSN 1049-5258.

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

M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1149, 2003.

L.J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, 1992.

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

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

G.A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical report, Tech. rep. CUED/F-INENG/TR166, Cambridge University, 1994.

S. Singh, T. Jaakkola, M.L. Littman, and C. Szepesvari. Convergence results for single-step on-policy reinforecement-learning algorithms. *Machine Learning*, 38:287–308, 2000.

A.L. Strehl and M.L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22th International Conference on Machine Learning*, pages 856–863, 2005.

A.L. Strehl, L. Li, E. Wiewiora, J. Langford, and M.L. Littman. PAC model-free reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 881–888, 2006.

R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1): 9–44, 1988.

R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*, pages 216–224, 1990.

R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1045, 1996.

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

R.S. Sutton and S.P. Singh. On step-size and bias in temporal-difference learning. In *Proceedings of the 8th Yale Workshop on Adaptive and Learning Systems*, 1994.

C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

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

M. Wiering and J. Schmidhuber. Fast online Q(λ). *Machine Learning*, 33:105–115, 1998.