In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, pp.759–766. Morgan Kaufman.

Eligibility Traces for Off-Policy Policy Evaluation

Doina Precup DPRECUP@CS.UMASS.EDU
Department of Computer Science, University of Massachusetts, Amherst, MA 01003-4610, USA

Richard S. Sutton SUTTON@RESEARCH.ATT.COM
Satinder Singh BAVEJA@RESEARCH.ATT.COM
AT&T Labs—Research, 180 Park Avenue, Florham Park, NJ 07932, USA

Abstract

Eligibility traces have been shown to speed reinforcement learning, to make it more robust to hidden states, and to provide a link between Monte Carlo and temporal-difference methods. Here we generalize eligibility traces to *off-policy learning*, in which one learns about a policy different from the policy that generates the data. Off-policy methods can greatly multiply learning, as many policies can be learned about from the same data stream, and have been identified as particularly useful for learning about subgoals and temporally extended macro-actions. In this paper we consider the off-policy version of the policy evaluation problem, for which only one eligibility trace algorithm is known, a Monte Carlo method. We analyze and compare this and four new eligibility trace algorithms, emphasizing their relationships to the classical statistical technique known as *importance sampling*. Our main results are 1) to establish the consistency and bias properties of the new methods and 2) to empirically rank the new methods, showing improvement over one-step and Monte Carlo methods. Our results are restricted to model-free, table-lookup methods and to offline updating (at the end of each episode) although several of the algorithms could be applied more generally.

1. Introduction

In reinforcement learning, we generally learn from experience, that is, from the sequence of states, actions, and rewards generated by the agent interacting with its environment. This data is affected by the decision-making policy used by the agent to select its actions, and thus we often end up learning something that is a function of the agent's policy. For example, the common subproblem of *policy evaluation* is to learn the value function for the agent's policy (the function giving the expected future reward available from each state–action pair). In general, however, we might want to learn about policies other than that currently followed by the agent, a process known as *off-policy learning*. For example, 1-step Q-learning is often used in an off-policy manner, learning about the greedy policy while the data is generated by a slightly randomized policy that ensures exploration.

Off-policy learning is especially important for research on the use of temporally extended actions in reinforcement learning (Kaelbling, 1993; Singh, 1992; Parr, 1998; Dietterich, 1998; Sutton, Precup & Singh, 1999). In this case, we are interested in learning about many different policies, each corresponding to a different macro-action, subgoal, or option. Off-policy learning enables the agent to use its experience to learn about the values and models of all the policies in parallel, even though it can follow only one policy at a time (Sutton, Precup & Singh, 1998).

In this paper we consider the natural generalization of the policy evaluation problem to the off-policy case. That is, we consider two stationary Markov policies, one used to generate the data, called the *behavior policy*, and one whose value function we seek to learn, called the *target policy*. The two policies are completely arbitrary except that the behavior policy must be *soft*, meaning that it must have a non-zero probability of selecting every action in each state. (The last method we consider has weaker requirements, not even requiring that the behavior policy be stationary, only non-starving.) This policy evaluation problem is a particularly clear and pure case of off-policy learning. Whatever we learn about it we expect to elucidate, if not directly transfer to, the problem of learning value functions and models of temporally extended macro-actions.

There are few existing model-free algorithms[1] that apply

[1] In this paper we restrict attention to methods that learn directly from experience rather than form an explicit model of the environment. Such model-free methods have been emphasized in reinforcement learning because of their simplicity and robustness to modeling errors and assumptions.

to off-policy policy evaluation. There is a natural one-step method, TD(0), but the more general TD(λ), for $\lambda > 0$, fails because it includes some effect of multi-step transitions, which are contaminated by the behavior policy and not compensated for in any way. The only prior method we know of that uses multi-step transitions appropriately is the weighted Monte Carlo method described briefly by Sutton and Barto (1998). There are at least three variations of Q-learning which use eligibility traces, Watkins's (1989) Q(λ), Peng and Williams' (1996) Q(λ), and naive Q(λ) (Sutton & Barto, 1998). Like 1-step Q-learning, these are all off-policy methods, but they apply only to the special case in which the target policy is deterministic and changing (to always be greedy with respect to the current value function estimate). These methods cannot be applied directly to our simpler but more general policy evaluation problem, although two of our four new methods reduce to Watkins's Q(λ) in the special case in which the target policy is deterministic.

2. Reinforcement Learning (MDP) Notation

In this paper we consider the episodic framework, in which the agent interacts with its environment in a sequence of episodes, numbered $m = 1, 2, 3, \ldots$, each of which consists of a finite number of time steps, $t = 0, 1, 2, \ldots, T_m$. The first state of each episode, $s_0 \in S$ is chosen according to some fixed distribution. Then, at each step t, the agent perceives the state of the environment, $s_t \in S$, and on that basis chooses an action, $a_t \in A$. In response to a_t, the environment produces, one step later, a numerical reward, $r_{t+1} \in \Re$, and a next state, s_{t+1}. If the next state is the special *terminal state*, then the episode terminates at time $T_m = t+1$. We assume here that S and A are finite and that the environment is completely characterized by one-step state-transition probabilities, $p^a_{ss'}$, and one-step expected rewards, r^a_s, for all $s, s' \in S$ and $a \in A$.

A stationary way in which the agent might behave, or *policy*, is specified by a mapping from states to probabilities of taking each action: $\pi : S \times A \to [0,1]$. The value of taking action a in state s under policy π, denoted $Q^\pi(s,a)$, is the expected discounted future reward starting in s, taking a, and henceforth following π:

$$Q^\pi(s,a) \stackrel{\text{def}}{=} E_\pi\Big\{ r_1 + \gamma r_2 + \cdots + \gamma^{T-1} r_T \;\Big|\; s_0 = s, a_0 = a \Big\}.$$

where $0 \le \gamma \le 1$ is a discount-rate parameter and T is the time of termination. The function $Q^\pi : S \times A \to \Re$ is known as the *action-value function* for policy π. The problem we consider in this paper is that of estimating Q^π for an arbitrary *target policy* π, given that all data is generated by a different *behavior policy* b, where b is soft, meaning $b(s,a) > 0, \forall s \in S, a \in A$.

3. Importance Sampling Algorithms

One way of viewing the special difficulty of off-policy learning is that it is a mismatch of distributions—we would like data drawn from the distribution of the target policy but all we have is data drawn from the distribution of the behavior policy. *Importance sampling* (e.g., see Rubinstein, 1981) is a classical technique for handling just this kind of mismatch. In particular, it is for estimating the expected value of a random variable x with distribution d from samples, when the samples are drawn from another distribution d'. For example, the target distribution d could be normal, while the sampling distribution d' is uniform, as below.

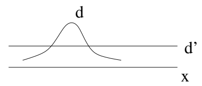

Figure 1. Different target and sampling distributions

In its classical form, importance sampling is based on the following simple observation:

$$\begin{aligned} E_d\{x\} &= \int_x x\, d(x)\, dx = \int_x x \frac{d(x)}{d'(x)} d'(x) dx \\ &= E_{d'}\left\{ x \frac{d(x)}{d'(x)} \right\}, \end{aligned}$$

which leads to the importance sampling estimator,

$$\approx \frac{1}{n} \sum_{i=1}^{n} x_i \frac{d(x_i)}{d'(x_i)} \tag{1}$$

where the x_i are samples selected according to d'. This estimator computes the average of the sample values, where each sample is weighted differently based on the ratio of its likelihood of occurring under the two distributions. This weighting gives more importance to samples that occur rarely under the sampling distribution d' but occur frequently under the target distribution d. If d and d' are the same, then all the samples have a weight of 1, and the estimator becomes the usual arithmetic average of the samples. The importance sampling estimator (1) is *consistent*, meaning it converges with probability one to $E_d\{x\}$ as the number of samples goes to infinity, and *unbiased*, meaning its expected value after any number of examples is also $E_d\{x\}$ (Rubinstein, 1981).

A less well known variant of this technique is *weighted importance sampling*, which performs a weighted average of the samples, with weights $\frac{d(x_i)}{d'(x_i)}$. The weighted importance sampling estimator is:

$$\frac{\sum_{i=1}^{n} x_i \frac{d(x_i)}{d'(x_i)}}{\sum_{i=1}^{n} \frac{d(x_i)}{d'(x_i)}}.$$

This estimator is a consistent but biased estimator of $E_d\{x\}$ (Rubinstein, 1981). Nevertheless, this estimator is often faster and more stable in practice than (1). Intuitively, this is due to the fact that, if an unlikely event occurs, its weight will be very large and will cause a large variation in the conventional estimator; but in the weighted estimator, the large weight appears in the denominator as well, which smoothes the variation.

Now consider applying importance sampling to off-policy policy evaluation in MDPs. The samples come in the form of episodes, which are complete sequences of states, actions and rewards, ending in a terminal state. We want to estimate the action value $Q^\pi(s,a)$ for an arbitrary state s and action a. Let M be the number of episodes containing state-action pair (s,a), and let t_m be the first time when $(s_t,a_t)=(s,a)$ in the mth of these episodes. Then we define the *first-visit importance sampling estimator* of $Q^\pi(s,a)$ as

$$Q^{IS}(s,a) \stackrel{\text{def}}{=} \frac{1}{M}\sum_{m=1}^{M} R_m w_m, \tag{2}$$

where R_m is the return following (s,a) in episode m,

$$R_m \stackrel{\text{def}}{=} r_{t_m+1} + \gamma r_{t_m+2} + \ldots + \gamma^{T_m - t_m - 1} r_{T_m},$$

and w_m is the importance sampling weight assigned to episode m,

$$w_m \stackrel{\text{def}}{=} \frac{\pi_{t_m+1}}{b_{t_m+1}} \frac{\pi_{t_m+2}}{b_{t_m+2}} \cdots \frac{\pi_{T_m-1}}{b_{T_m-1}},$$

where π_t and b_t are short for $\pi(s_t,a_t)$ and $b(s_t,a_t)$ respectively. Similarly, we define the *weighted importance sampling estimator* (Sutton & Barto, 1998) as

$$Q^{ISW}(s,a) \stackrel{\text{def}}{=} \frac{\sum_{m=1}^{M} R_m w_m}{\sum_{m=1}^{M} w_m}. \tag{3}$$

4. Per-Decision Algorithms

The estimators defined above all consider complete returns R_m without breaking down into their constituent rewards; this is the property that leads to their being called Monte Carlo methods. An estimator that used the way returns break down into rewards could potentially be more efficient than these, or more easily implemented on an incremental, step-by-step basis. In this section we present a new algorithm that performs importance sampling weightings for each decision step along the way.

Let us examine the term $R_m w_m$ from equations (2) and (3):

$$R_m w_m = \sum_{i=t_m+1}^{T_m} \gamma^{i-t_m-1} r_i \frac{\pi_{t_m+1}}{b_{t_m+1}} \cdots \frac{\pi_{i-1}}{b_{i-1}} \frac{\pi_i}{b_i} \cdots \frac{\pi_{T_m-1}}{b_{T_m-1}}.$$

The terms of the sum can be naturally separated into two parts, one containing the $\frac{\pi}{b}$ ratios from t_{m+1} to $i-1$, and one containing the ratios from i to T_{m-1}. Intuitively, the weight on reward r_i should not depend on the future after time i, only on the history to that point. This is the idea behind the *per-decision importance sampling estimator*:

$$Q^{PD}(s,a) \stackrel{\text{def}}{=} \frac{1}{M}\sum_{m=1}^{M}\sum_{k=1}^{T_m-t_m} \gamma^{k-1} r_{t_m+k} \prod_{i=t_m+1}^{t_m+k-1} \frac{\pi_i}{b_i}.$$

This estimator weights each reward along a trajectory according to the likelihood of the trajectory up to that point, under the target and the behavior policy. If the target and the behavior policy are the same, the estimator is simply the average of the returns from each episode.

Theorem 1 *The per-decision importance sampling estimator Q^{PD} is a consistent unbiased estimator of Q^π.*

The main idea of the proof (in the appendix) is to show that the expected value of the per-decision estimator Q^{PD} is the same as the expected value of the classical importance sampling estimator Q^{IS}, which is known to be consistent and unbiased.

We can also devise a weighted version of the per-decision importance sampling algorithm, similar to the weighted version of classical importance sampling. The idea is simply to divide the estimator by the sum of the weights during each episode:

$$Q^{PDW}(s,a) \stackrel{\text{def}}{=} \frac{\sum_{m=1}^{M}\sum_{k=1}^{T_m-t_m} \gamma^{k-1} r_{t_m+k} \prod_{i=t_m+1}^{t_m+k-1} \frac{\pi_i}{b_i}}{\sum_{m=1}^{M}\sum_{k=1}^{T_m-t_m} \gamma^{k-1} \prod_{i=t_m+1}^{t_m+k-1} \frac{\pi_i}{b_i}}.$$

This *weighted per-decision importance sampling estimator* is consistent but biased, just like the weighted importance sampling estimator Q^{ISW} (Precup, 2000).

An eligibility-trace version of per-decision importance sampling is shown in Algorithm 1. The algorithm maintains eligibility traces for each state-action pair in the usual manner of temporal-difference (TD) algorithms. The only difference is that here the eligibility trace is multiplied on each step not just by a decay-rate λ, but also by an importance sampling factor $\frac{\pi(s_t,a_t)}{b(s_t,a_t)}$. This factor corrects for the effect of the behavior policy. The algorithm shown uses *online updating*, meaning that the value estimates are updated on every time step. The *offline* version would make the same increments and decrements as shown, but only at the end of each episode. The changes are accumulated "on the side" until the end of the episode, the value estimates remaining unchanged until then. Under offline updating the algorithm can be made to exactly implement the per-decision importance sampling estimator Q^{PD} by choosing $\lambda = 1$ and $\alpha(s,a) = 1/n(s,a)$, where $n(s,a)$ is the number of times state-action pair s,a has previously been updated. Another choice for α causes the algorithm to exactly implement the corresponding weighted estimator, Q^{PDW}. The

Algorithm 1 Online, Eligibility-Trace Version of Per-Decision Importance Sampling

1. Update the eligibility traces for all states:

$$e_t(s,a) = e_{t-1}(s,a)\gamma\lambda\frac{\pi(s_t,a_t)}{b(s_t,a_t)}, \qquad \forall s,a$$
$$e_t(s,a) = 1, \text{iff } t = t_m(s,a),$$

where $\lambda \in [0,1]$ is an eligibility trace decay factor.

2. Compute the TD error:

$$\delta_t = r_{t+1} + \gamma\frac{\pi(s_{t+1},a_{t+1})}{b(s_{t+1},a_{t+1})}Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)$$

3. Update the action-value function:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha e_t(s,a)\delta_t, \qquad \forall s,a$$

algorithm remains consistent under general λ and general decreasing α:

Theorem 2 *For any soft, stationary behavior policy b, and any $\lambda \in [0,1]$ that does not depend on the action a_t, Algorithm 1 with offline updating converges w.p.1 to Q^π, under the usual step-size conditions on α.*

The proof of the theorem (see appendix) is an application of the general convergence theorem of Jaakkola, Jordan, and Singh (1994).

5. Tree Backup Algorithm

The methods we have discussed so far all use the behavior policy in their updates; they require that it be known, Markov (purely a function of the current state), and explicitly represented as action probabilities. For complex agents, however, none of these may be true. In this section we consider a method that requires nothing of the behavior policy other than that it be non-starving, i.e., that it never reaches a time when some state-action pair is never visited again. The behavior policy can be nonstationary, non-Markov, and completely unknown; it does not appear anywhere in the definition of the estimator or its algorithm.

The main idea of the new method, called *tree backup*, is illustrated in Figure 2. At each step along a trajectory, there are several possible choices of action according to the target policy. The one-step target combines the value estimates for these actions according to their probabilities of being taken under the target policy. At each step, the behavior policy chooses one of the actions, and for that action, one time step later, there is a new estimate of its value, based

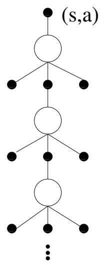

Figure 2. Backup diagram for the tree backup algorithm

Algorithm 2 Online, Eligibility-Traces Version of Tree Backup

1. Update the eligibility traces for all states:

$$e_t(s,a) = e_{t-1}(s,a)\gamma\lambda\pi(s_t,a_t), \qquad \forall s,a$$
$$e_t(s,a) = 1 \text{ iff } t = t_m(s,a)$$

where $\lambda \in [0,1]$ is an eligibility trace decay parameter.

2. Compute the TD error:

$$\delta_t = r_{t+1} + \gamma\sum_{a\in A}\pi(s_{t+1},a)Q(s_{t+1},a) - Q(s_t,a_t)$$

3. Update the action-value function:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha e_t(s,a)\delta_t, \qquad \forall s,a$$

on the reward received and the estimated value of the next state. The tree backup algorithm then forms a new target, using the old value estimates for the actions that were not taken, and the new estimated value for the action that was taken. This process can be iterated over many steps. If we iterate it over n steps, we obtain the *n-step tree-backup estimator*:

$$Q_n^{TB}(s,a) \stackrel{\text{def}}{=} \frac{1}{M}\sum_{m=1}^{M}\gamma^n Q(s_{t_m+n},a_{t_m+n})\prod_{i=t_m+1}^{t_m+n}\pi_i + \sum_{k=t_m+1}^{t_m+n}\gamma^{k-t_m+1}\prod_{i=t_m+1}^{k-1}\pi_i\left(r_k + \gamma\sum_{a\neq a_k}\pi(s_k,a)Q(s_k,a)\right)$$

For $n = 1$, the tree backup estimator reduces to the familiar one-step TD estimator, TD(0).

The tree backup estimator also has a simple incremental implementation using eligibility traces. An online version of this implementation is given by Algorithm 2. In general, λ can be chosen as a function of the state s_t, but cannot depend on the action a_t. A choice of λ that is dependent on the state can have empirical advantages. For example, in the experiments reported in the next section, the eligibil-

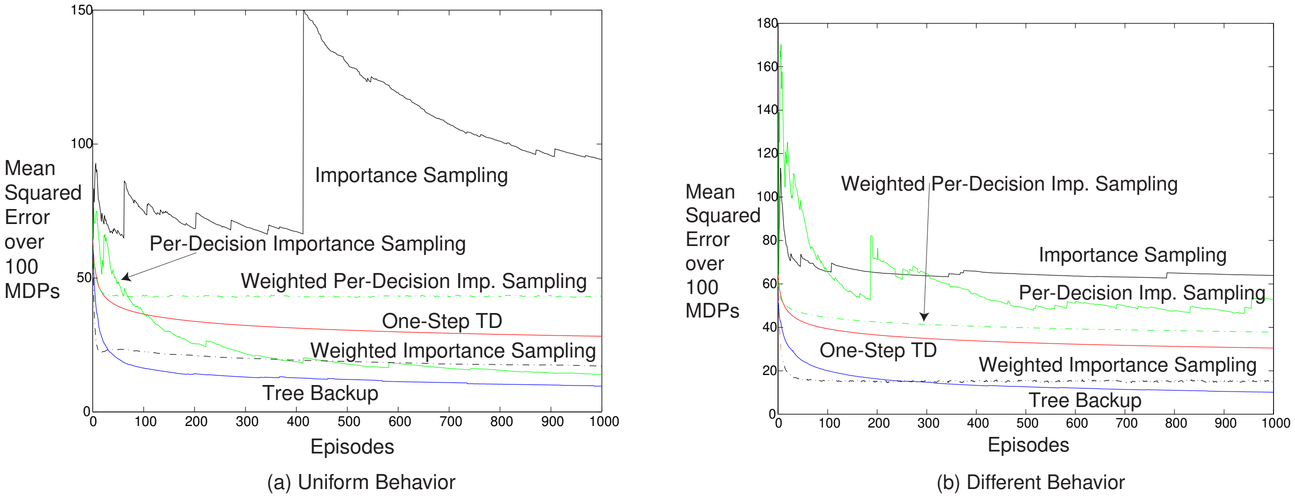

Figure 3. Average performance of all algorithms. On the left the behavior policy chose 50-50 from the two actions. On the right the behavior policy chose with 80-20 probabilities exactly opposite those of the target policy.

ity traces were divided at each step by $\max_a \pi(s_t, a)$. This prevents the traces from decaying too quickly.

Theorem 3 *For any non-starving behavior policy, for any choice of* $\lambda \in [0,1]$ *that does not depend on the actions chosen at each state, the offline version of Algorithm 2 converges w.p.1 to* Q^π*, under the usual conditions on* α*.*

The proof of this result (see appendix) relies on showing a contraction property of Q_n^{TB}, for any n, and on applying again the convergence theorem of Jaakkola et al. (1994).

6. Empirical Comparison

We obtained empirical results with the explicit estimators, Q^{IS}, Q^{ISW}, Q^{PD}, and Q^{PDW}, the one-step method, TD(0), and a tree backup method using the normalization by $\max_a \pi(s_t, a)$ as discussed above. Except for the two importance sampling methods, all were implemented using the offline versions of Algorithms 1 and 2, using appropriate settings for α and λ as discussed earlier. Not all estimators have online versions (which are potentially more efficient), so we used offline versions in all cases to facilitate direct comparison of the underlying ideas. This choice is also convenient because it results in entirely parameter-free algorithms.

We compared the estimators on a suite of 100 randomly constructed MDPs. Each MDP had 100 nonterminal states and one terminal state. In each nonterminal state, there were two actions available, and each action branched to 4 randomly selected next states with random probabilities (the partition of unity was selected by picking three random split points uniformly randomly from $[0,1]$). The target policy was to select the first action with 80% probability and the second with 20%. The immediate rewards were chosen uniformly randomly from $[0,1]$. Two different behavior policies were used. In the *uniform behavior* case, both actions were equally likely, whereas in the *different behavior* case, the first action was selected with 20% probability and the second with 80% probability, resulting in a policy very different from the target policy. The initial state of each episode was always the same. All the MDPs we constructed in this way terminated with probability one; we used $\gamma = 1$. As a performance measure of each estimator after each number of episodes, we used the mean square error between the estimator and the true action values, averaged over the 100 tasks and over the 200 state-action pairs within each task (state-action pairs that had never been visited were excluded from the averages). This performance measure is shown for up to 1000 episodes in the two panels of Figure 3.

The importance sampling estimator was generally quite slow and had high variance. The weighted version performed much better. Per-decision importance sampling was relatively efficient in the long run in the uniform behavior case, but relatively slow in the different behavior case. Surprisingly, the weighted version of per-decision importance sampling performed fairly poorly in both cases, although still managing to beat the unweighted version in the different behavior case. The tree backup estimator was uniformly the most efficient of all methods in the medium and long term, beaten only by weighted importance sampling by small amounts for small numbers of episodes.

In summary, our results strongly favor the tree backup algorithm, because of its superior overall performance and because of its weaker requirements of the behavior policy.

7. Unifying Tree Backup and Per-Decision

In order to understand better the two multi-step TD algorithms (tree backup and per-decision importance sampling), consider the full trajectory tree presented in Figure 4. The root of the tree is a state-action pair, and the tree contains all the possible states and actions at each point. States are represented by hollow circles, and actions are represented by filled circles. One trajectory through the tree can be obtained by sampling the states at each ramification according to the environment's transition probabilities, and by sampling actions according to the behavior policy.

Both multi-step algorithms do backups along such trajectories. The per-decision importance sampling algorithm uses the actual rewards obtained during the trajectory. Because the sampling at each action ramification is done according to the behavior policy probabilities, the importance sampling correction is necessary to ensure correct estimates. The tree backup algorithm considers all possible actions at each step, not just the one taken. It backs up values according to a cut like the one represented with the dotted line in Figure 4. Because all action choices are considered in the backup, the convergence is guaranteed for any behavior policy that is non-starving.

This interpretation suggests that the two algorithms can be combined, without losing the convergence guarantees. If at a given state, the behavior policy is Markov and it is known, we can use per-decision importance sampling. If the behavior is unknown and/or non-Markov, we can use the tree backup algorithm. This mixture could ensure faster and more stable error reduction than either algorithm alone. We have not yet explored this idea empirically.

The two multi-step TD algorithms also have an interesting relationship to multi-step Q-learning. In their control version, both algorithms cut the eligibility trace whenever an exploratory action is taken. Their updates are equivalent to Watkins's $Q(\lambda)$ algorithm.

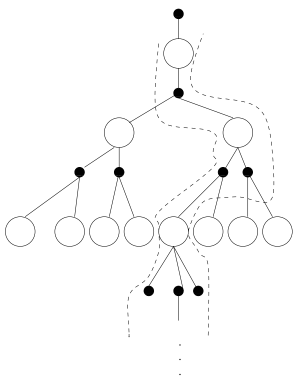

Figure 4. Full trajectory tree for an MDP

8. Conclusions

In this paper we presented four novel algorithms for using eligibility traces in off-policy learning. We proved that these algorithms converge to correct action values under offline updating in the tabular case. These appear to be the first convergence results for multi-step off-policy learning. We also showed that the tree backup algorithm converges correctly for non-stationary and non-Markov behavior policies, as long as they are non-starving. This promising algorithm also performed best in our empirical results. Preliminary results have shown that it can speed learning about macro-actions compared to one-step methods (Precup, 2000).

References

Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann.

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

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 167–173). Morgan Kaufmann.

Parr, R. (1998). *Hierarchical control and learning for Markov Decision Processes*. Doctoral dissertation, Computer Science Division, University of California, Berkeley, USA.

Peng, J., & Williams, J. (1996). Incremental multi-step Q-learning. *Machine Learning*, *22*, 283–290.

Precup, D. (2000). *Temporal abstraction in reinforcement learning*. Doctoral dissertation, Department of Computer Science, University of Massachusetts, Amherst, USA.

Rubinstein, R. (1981). *Simulation and the Monte Carlo method*. NewYork: Wiley.

Singh, S. P. (1992). Scaling reinforcement learning by learning variable temporal resolution models. *Proceedings of the Ninth International Conference on Machine Learning* (pp. 406–415). Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S., Precup, D., & Singh, S. (1998). Intra-option learning about temporally abstract actions. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 556–564). Morgan Kaufman.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211.

Watkins, C. J. C. H. (1989). *Learning with delayed rewards*. Doctoral dissertation, Psychology Department, Cambridge University, Cambridge, UK.

Appendix

Proof of Theorem 1:
We know that the classical importance sampling estimator Q^{IS} is consistent and unbiased:

$$E\left\{\left(\sum_{k=1}^{T-t}\gamma^{k-1}r_{t+k}\right)\prod_{i=t+1}^{T-1}\frac{\pi_i}{b_i} \mid s_t=s,a_t=a,b\right\}=Q^\pi(s,a).$$

We will show that the per-decision importance sampling estimator Q^{PD} has the same expected value as Q^{IS}. Let us move the importance sampling correction inside the sum, and examine the expectation for the k-th term:

$$E\left\{\gamma^{k-1}r_{t+k}\prod_{i=t+1}^{T-1}\frac{\pi_i}{b_i} \mid s_t=s,a_t=a,b\right\}=$$
$$=E\left\{\gamma^{k-1}r_{t+k}\frac{\pi_{t+1}}{b_{t+1}}\cdots\frac{\pi_{t+k-1}}{b_{t+k-1}} \mid s_t,a_t,\ldots s_{t+k-1},a_{t+k-1}\right\}$$
$$\cdot E\left\{\frac{\pi_{t+k}}{b_{t+k}}\cdots\frac{\pi_{T-1}}{b_{T-1}} \mid s_t,a_t,\ldots s_{t+k},a_{t+k},b\right\}.$$

Since the underlying environment is an MDP, the second factor can be re-written as:

$$E\left\{\frac{\pi_{t+k}}{b_{t+k}}\cdots\frac{\pi_{T-1}}{b_{T-1}} \mid s_{t+k},a_{t+k},b\right\}.$$

The expected value of this term is 1. Therefore,

$$E\left\{\left(\sum_{k=1}^{T-t}\gamma^{k-1}r_{t+k}\right)\prod_{i=t+1}^{T-1}\frac{\pi_i}{b_i} \mid s_t=s,a_t=a,b\right\}=$$
$$E\left\{\sum_{k=1}^{T-t}\gamma^{k-1}r_{t+k}\prod_{i=t+1}^{t+k-1}\frac{\pi_i}{b_i} \mid s_t=s,a_t=a,b\right\}$$

which concludes the proof. $\diamond$

Proof of Theorem 2:
The proof is structured in two stages. First, we consider the corrected truncated return corresponding to Q^{PD}. The corrected truncated return sums the rewards obtained from the environment for only n steps, then uses the current estimate of the value function to approximate the remaining value:

$$R_t^{(n)}=\sum_{k=1}^{n}\gamma^{k-1}r_{t+k}\prod_{l=t+1}^{t+k-1}\frac{\pi_l}{b_l}+\gamma^nQ(s_{t+n},a_{t+n})\prod_{l=t+1}^{t+n-1}\frac{\pi_l}{b_l}$$

We need to show that $R_t^{(n)}-Q^\pi$ is a contraction mapping in the max norm. If this is true for any n, then by applying the general convergence theorem, the n-step return converges to Q^π. Then any convex combination will also converge to Q^π. For example, any combination using a λ parameter in the style of eligibility traces will converge to Q^π.

Let $\Omega(s,a,k)$ denote the set of all possible trajectories of k state-action pairs starting with (s,a):

$$\Omega(s,a,k)=\{\langle s_0,a_0,s_1,a_1,\ldots s_{k-1},a_{k-1}\rangle | s_0=s,a_0=a\},$$

and let ω denote any such trajectory: $\omega=\langle s_0,a_0,s_1,a_1,\ldots s_{k-1},a_{k-1}\rangle$. Then the expected value of the corrected truncated return for state-action (s,a) can be expressed as follows:

$$E\left\{R_t^{(n)} \mid s_t=s,a_t=a,b\right\}=$$
$$\sum_{k=1}^{n}\sum_{\omega\in\Omega(s,a,k)}\Pr\{\omega \mid s_0=s,a_0=a,b\}\gamma^{k-1}r_k\prod_{l=1}^{k-1}\frac{\pi_l}{b_l}$$
$$+\sum_{\omega\rangle\in\Omega(s,a,n)}\Pr\{\omega \mid s_0=s,a_0=a,b\}\gamma^nQ(s_n,a_n)\prod_{l=1}^{n-1}\frac{\pi_l}{b_l}$$
$$\sum_{k=1}^{n}\sum_{\omega\in\Omega(s,a,k)}\left(\prod_{l=1}^{k-1}p_{s_{l-1}s_l}^{a_{l-1}}b(s_l,a_l)\right)\gamma^{k-1}r_k\prod_{l=1}^{k-1}\frac{\pi_l}{b_l}$$
$$+\sum_{\omega\in\Omega(s,a,n)}\left(\prod_{l=1}^{n-1}p_{s_{l-1}s_l}^{a_{l-1}}b(s_l,a_l)\right)\gamma^nQ(s_n,a_n)\prod_{l=1}^{n-1}\frac{\pi_l}{b_l}$$
$$=\sum_{k=1}^{n}\gamma^{k-1}\sum_{\omega\in\Omega(s,a,k)}r_{s_{k-1}}^{a_{k-1}}\prod_{l=1}^{k-1}p_{s_{l-1}s_l}^{a_{l-1}}\pi(s_l,a_l)$$
$$+\gamma^n\sum_{\omega\in\Omega(s,a,k)}Q(s_n,a_n)\prod_{l=1}^{k-1}p_{s_{l-1}s_l}^{a_{l-1}}\pi(s_l,a_l)$$

By applying the Bellman equation for Q^π iteratively n times, we obtain:

$$Q^\pi(s,a)=\sum_{k=1}^{n}\sum_{\omega\in\Omega(s,a,k)}\gamma^{k-1}r_{s_{k-1}}^{a_{k-1}}\prod_{l=1}^{k-1}p_{s_{l-1}s_l}^{a_{l-1}}\pi(s_l,a_l)$$
$$+\gamma^n\sum_{\omega\in\Omega(s,a,k)}Q^\pi(s_{t+n},a_{t+n})\prod_{l=1}^{k}p_{s_{l-1}s_l}^{a_{l-1}}\pi(s_l,a_l).$$

Therefore,

$$\max_{(s,a)}|E\left\{R_t^{(n)} \mid b\right\}-Q^\pi(s,a)|\leq\gamma^n\max_{(s,a)}|Q(s,a)-Q^\pi(s,a)|.$$

This means that any n-step return is a contraction in the max norm, and therefore, by applying theorem 1 of Jaakkola, Jordan and Singh (1994), it converges to Q^π.

In the second stage, we show that by applying the updates of Algorithm 1 for n successive steps, we perform the same update as by using the n-step return $R_t^{(n)}$. The eligibility trace for state-action pair (s,a) can be re-written as:

$$e_t(s,a) = \gamma^{t-t_m} \prod_{l=t_m+1}^{t} \frac{\pi_l}{b_l}.$$

We have:

$$\sum_{k=1}^{n} e_{t+k-1}(s,a)\delta_{t+k-1}(s,a) =$$
$$\sum_{k=1}^{n} \gamma^{k-1} \left(\prod_{l=t+1}^{t+k-1} \frac{\pi_l}{b_l} \right) (r_{t+k} + \gamma \frac{\pi(s_{t+k},a_{t+k})}{b(s_{t+k},a_{t+k})} Q(s_{t+k},a_{t+k})$$
$$-Q(s_{t+k-1},a_{t+k-1}))$$
$$= \sum_{k=1}^{n} \gamma^{k-1} r_{t+k} \prod_{l=t+1}^{t+k-1} \frac{\pi_l}{b_l} + \gamma^n Q(s_{t+n},a_{t+n}) \prod_{l=t+1}^{t+n-1} \frac{\pi_l}{b_l}$$
$$-Q(s_t,a_t)$$
$$= R_t^{(n)} - Q(s_t,a_t).$$

Since our algorithm is equivalent to applying a convex mixture of n-step updates, and each update converges to correct Q-values, algorithm 1 converges to correct Q-values as well. $\diamond$

Proof of Theorem 3:
The proof is again in two stages. First we show that $E\{Q_n^{TB}(s,a) \mid b\} - Q^\pi$ is a contraction, in order to apply again theorem 1 of Jaakkola, Jordan and Singh (1994). We use a proof by induction.

Let Q be the current estimate of the value function. For $n=1$:

$$\max_{(s,a)} |E\{Q_1^{TB}(s,a) \mid b\} - Q^\pi(s,a)| =$$
$$\max_{(s,a)} |r_s^a + \gamma \sum_{s',a'} p_{ss'}^a \pi(s',a') Q(s',a')$$
$$-r_s^a - \gamma \sum_{s',a'} p_{ss'}^a \pi(s',a') Q^\pi(s',a')|$$
$$\leq \gamma \max_{(s,a)} |Q(s,a) - Q^\pi(s,a)|.$$

For the induction step, we assume that

$$\max_{(s,a)} |E\{Q_n^{TB}(s,a) \mid b\} - Q^\pi(s,a)| \leq$$
$$\leq \gamma \max_{(s,a)} |Q_n^{TB}(s,a) - Q^\pi(s,a)|,$$

and we show the same holds for $Q_{n+1}^{TB}(s,a)$. We can rewrite $Q_{n+1}^{TB}(s,a)$ as follows:

$$Q_{n+1}^{TB}(s,a) = r_{t+1} + \gamma \sum_{a' \in A} \pi(s_{t+1},a')$$
$$\left(Q(s_{t+1},a')(1 - I_{a'a_{t+1}}) + I_{a'a_{t+1}} Q_n^{TB}(s_{t+1},a') \right),$$

where $I_{a'a_{t+1}}$ is an indicator variable equal to 1 if $a' = a_{t+1}$ and 0 otherwise. Then we have:

$$\max_{(s,a)} |E\{Q_{n+1}^{TB}(s,a) \mid b\} - Q^\pi(s,a)| =$$
$$\max_{(s,a)} |r_s^a + \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a')$$
$$E\{(1 - I_{a'a_{t+1}})Q(s',a') + I_{a'a_{t+1}} Q_n^{TB}(s',a') \mid b\}$$
$$-r_s^a - \gamma \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a') Q^\pi(s',a')|$$
$$= \gamma \max_{(s,a)} | \sum_{s'} p_{ss'}^a \sum_{a'} \pi(s',a')$$
$$E\left\{ (1 - I_{a'a_{t+1}})(Q(s',a') - Q^\pi(s',a')) + \right.$$
$$\left. +I_{a'a_{t+1}}(Q_n^{TB}(s',a') - Q^\pi(s',a')) | b \right\} |$$
$$\leq \gamma \max_{(s,a)} |Q(s,a) - Q^\pi(s,a)|.$$

By applying now theorem 1 of Jaakkola, Jordan and Singh (1994), we can conclude that any n-step return converges to the correct action value.

Since all the n-step returns converge to Q^π, any convex linear combination of n-step returns also converges to Q^π. In particular, we can use a fixed λ parameter, as is usually done in TD(λ), or even a λ parameter dependent on the state.

For the second part of the proof, we show that applying Algorithm 2 (with $\lambda = 1$) for n steps is equivalent to using $Q_n^{TB}(s,a)$. The eligibility trace for state-action pair (s,a) can be re-written as:

$$e_{t+k}(s,a) = \gamma^k \prod_{l=t+1}^{t+k} \pi(s_l,a_l)$$

By adding and subtracting the weighted action value, $\pi(s_{t+k},a_{t+k})Q(s_{t+k},a_{t+k})$ for the action taken on each step from the return, and regrouping, we have:

$$Q(s_t,a_t) + \sum_{k=1}^{n} \gamma^{k-1} \prod_{l=t+1}^{t+k-1} \pi(s_l,a_l)(r_{t+k}$$
$$+\gamma \sum_{a \in A} \pi(s_{t+k},a)Q(s_{t+k},a) - Q(s_{t+k-1},a_{t+k-1}))$$
$$= Q(s_t,a_t) + \sum_{k=1}^{n} e_{t+k}(s_t,a_t)\delta_{t+k},$$

which concludes the proof. $\diamond$