Learning While Exploring: Bridging the Gaps in the Eligibility Traces

Fredrik A. Dahl and Ole Martin Halck

Norwegian Defence Research Establishment (FFI)
P.O. Box 25, NO-2027 Kjeller, Norway
{Fredrik-A.Dahl, Ole-Martin.Halck}@ffi.no

Abstract. The reinforcement learning algorithm TD(λ) applied to Markov decision processes is known to need added exploration in many cases. With the usual implementations of exploration in TD-learning, the feedback signals are either distorted or discarded, so that the exploration hurts the algorithm's learning. The present article gives a modification of the TD-learning algorithm that allows exploration without cost to the accuracy or speed of learning. The idea is that when the learning agent performs an action it perceives as inferior, it is compensated for its loss, that is, it is given an additional reward equal to its estimated cost of making the exploring move. This modification is compatible with existing exploration strategies, and is seen to work well when applied to a simple grid-world problem, even when always exploring completely at random.

1 Introduction

Reinforcement learning is a machine learning paradigm where agents learn "by themselves" from experience in a real or synthetic environment. Temporal difference learning (TD-learning) is a reinforcement learning algorithm that works by estimating the long-term utility of model states. It was originally designed for predicting the expected infinite-horizon return from Markov chains [1]. TD-learning tunes some evaluation function that assigns values to process states. It has been shown that TD-learning using lookup-tables for evaluating states converges toward the solution for states that are visited infinitely many times.

TD-learning can also be applied to Markov decision processes (MDPs), which are Markov chains where an agent has partial control over the process. This approach, due to Tesauro [2], transforms TD-learning from "a model-free algorithm for learning to predict, into a model-based algorithm for learning to control" [3]. An agent may use a state evaluation function as a *policy* for controlling the MDP by always choosing states with the highest possible evaluation (together with some strategy for breaking ties). If the evaluation function is correct, the derived policy is optimal. However, in a learning phase, there is a trade-off between exploiting the knowledge of the agent by following the derived policy, and exploring other options that may turn out to be better. One can easily see that an agent that never explores may fail to discover the best available policy. On the other hand, an agent that explores too often pays a price when the evaluation function is close to the optimal one. In addition –

L. De Raedt and P. Flach (Eds.): ECML 2001, LNAI 2167, pp. 73-84, 2001.

and this is the problem this paper addresses – the TD-learning signals resulting from exploration will be distorted, and are therefore usually discarded.

The present article gives an algorithm that allows exploration without the downside of either discarding or distorting the learning signals. Our algorithm is based on the idea that the evaluation function can generate estimates of the cost of not following the greedy policy. We treat these estimates as rewards that the agent receives as compensation for making what it evaluates to be inferior actions.

The article is laid out as follows: Section 2 gives formal definitions of MDPs and TD-learning, with our preferred notation. In Section 3 we define our algorithm. Section 4 gives some experimental results in a simple grid world, Section 5 discusses some additional aspects of the algorithm, and Section 6 concludes the article.

2 Temporal Difference Learning and Markov Decision Processes

In this section we give some background on finite Markov decision processes and the temporal difference learning algorithm TD(λ) as applied to such processes. Over the years, a large body of literature on this and related subjects has appeared; we refer the reader to [4], [5], [6] and the references therein for more information on this field and reinforcement learning in general; most of this section is adapted from the former two works. A theoretical treatment of methods using value functions can be found in [7].

2.1 Markov Decision Processes

Informally, a (finite) *Markov decision process* (MDP) is a Markov process where in each state, an agent chooses an action, which influences the next step of the process and the reward the agent receives. We formalise an MDP as consisting of the following elements:

- A set S of *states*, with a subset $\Sigma \subseteq S$ of *terminal* states.
- A set A of *actions*, along with a function $A : S \to 2^{\mathsf{A}}$ mapping each state to the set of admissible actions that can be taken when the process is in that state.
- A *transition* probability function $p_T(s' \mid s, a)$, giving the probability that a process in state s will move to state s' given action a.
- A *reward* probability function $p_R(r \mid s, a, s')$, giving the probability of receiving reward r upon moving from state s to state s' through action a.
- A discounting factor $\gamma \in (0,1]$.

An *episode* of an MDP proceeds as follows:

1. A starting state s_0 is given; $t \leftarrow 0$.
2. If $s_t \in \Sigma$, then $T \leftarrow t$ and end.
3. Choose an action $a_{t+1} \in A(s_t)$ according to some policy.
4. A new state s_{t+1} is drawn according to $p_T(\cdot \mid s_t, a_{t+1})$.

5. Receive a reward r_{t+1} drawn according to $p_R(\cdot \mid s_t, a_{t+1}, s_{t+1})$.
6. $t \leftarrow t+1$; go to step 2.

In practice, the reward r_{t+1} is often deterministic, depending for instance on (s_t, a_{t+1}) or on s_{t+1} alone.

The agent's total *return* from the episode is defined as the discounted sum of rewards $\sum_{t=0}^{T-1} \gamma^t r_{t+1}$ if the episode terminates, and $\sum_{t=0}^{\infty} \gamma^t r_{t+1}$ otherwise. In the latter case, $\gamma < 1$ is required to ensure that the return is finite.

For Markov (non-decision) processes, every state s has an associated *value* $V(s)$, giving the expected return from the process when starting at s. If, in an MDP, an agent employs a fixed policy π, that is, if in a given state each action has a fixed probability of being chosen by the agent, the MDP reduces to such a Markov process. In this case, we denote the value function of this process as V^π.

The *optimal value function* $V^*: S \to \mathbf{R}$ maps each state to the greatest expected return that can be achieved from that state, and is consequently defined by

$$\forall s \in S : V^*(s) = \max_\pi V^\pi(s). \tag{1}$$

By following a policy π^* achieving this maximum (there is always a deterministic policy of this kind), an agent maximises its expected return from the process. The optimal value function can in principle be found by using dynamic programming to solve the Bellman equations

$$V^*(s_t) = \max_{a_{t+1} \in A(s_t)} E[r_{t+1} + \gamma V^*(s_{t+1})], \tag{2}$$

where the expectation is taken with respect to p_T and p_R.[1] In practice, though, this set of equations is often too large to solve explicitly even if the MDP is known exactly, and this is where machine learning, and temporal difference learning in particular, comes in useful.

2.2 Temporal Difference Learning in MDPs

Sutton's original exposition of the TD(λ) algorithm mainly concentrates on the problem of predicting the outcome of Markov processes [1]. There, the algorithm is used for estimating the value function of the process. Clearly, the algorithm can then also be used for estimating the value function V^π of an MDP given a fixed policy π. On the other hand, if we have estimated such a value function, we may use this estimate to change the policy so that the expected return increases.

These two processes may be combined in a way that hopefully yields a good estimate of the optimal value function V^* and the optimal policy π^*. Exactly how value estimation and policy optimisation are interleaved may vary – for example, value estimates may be updated after each action, after each episode or after a batch

[1] For clarity we abuse the notation slightly, by suppressing the dependency of the new state on the old state and the action, and also the dependency of the reward on the action and the two states.

of episodes. For simplicity and ease of implementation, we consider a version of TD(λ) where the value estimates are updated after each episode, starting from the end of the episode. Similar formulations can be found in [3] and (less explicitly) in [2]. Using this formulation, we do not need explicit representations of the so-called eligibility traces (defined in e.g. [4]), since these are implicitly calculated along the way. Furthermore, we assume that the current estimate $\hat{V}$ of the value function is represented as a lookup-table. For a given $\lambda \in [0,1]$ and small, positive *step size* α, the training procedure we will refer to as *naive* TD(λ) then goes as follows:

1. Initialise $\hat{V}(s)$ to 0 for each terminal s and (e.g.) randomly for all other $s \in S$.
2. Run an episode of the MDP (as above).
3. $\delta \leftarrow 0$.
4. For t from $T-1$ down to 0 do:
   4a. $\delta \leftarrow r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) + \gamma\lambda\delta$.
   4b. $\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha\delta$.
5. If termination criterion is met, then end, else go to step 2.

2.3 Exploitation and Exploration in Learning

So far, we have not said explicitly *how* the agent is to choose its actions (step 3 in the MDP procedure). The seemingly obvious choice is to use a *greedy* policy, that is, to pick the action that it sees as best according to its current value estimates. Using this policy, the agent *exploits* its current beliefs in order to maximise its return. If we assume that the agent has full knowledge of the model (an assumption that will be kept in the following), the agent then chooses the action

$$a_{t+1} = \arg\max_{a \in A(s_t)} \hat{Q}(s_t, a) \equiv \arg\max_{a \in A(s_t)} E[r_{t+1} + \gamma \hat{V}(s_{t+1}) \mid a_{t+1} = a]. \tag{3}$$

The problem with this strategy is that the agent may have inaccurate value estimates, hindering it from discovering actions that may be more valuable than the one it believes to be the best. Therefore, it is necessary occasionally to *explore* the state space in order to find out whether there are better ways to act. The *exploitation–exploration dilemma* (see e.g. [8]) deals with the fact that exploring in order to discover new knowledge will usually also reduce the return given to the agent. A number of different strategies for handling the exploration–exploitation trade-off have been devised [8].

There is another fundamental problem with exploring, however, no matter which action selection policy is used. If the TD(λ) algorithm is used in the naive form above, the feedback signals will be negatively influenced by the exploring actions. Thus, even if the agent were to possess the optimal value function at some time, the feedback from the exploring actions will be biased and push the evaluations away from the correct ones. In fact, the only value function receiving unbiased learning signals in this process is the one that is optimal *under the given exploration policy*. (A parallel to this observation was made by John [9], who noted that the optimal value function and policy are no longer optimal under forced exploration.)

The usual way of dealing with this problem in TD-learning for control is *not* carrying the feedback across the exploring action – see for instance the tic-tac-toe example in the introduction to [4]. In our formulation of TD(λ) above, this means that step 4a is changed to:

4a'. If a_{t+1} was exploring, then $\delta \leftarrow 0$, else $\delta \leftarrow r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) + \gamma\lambda\delta$.

In the context of Q-learning, this approach corresponds to what [4] refers to as *Watkins's Q(λ)* [10]. We shall refer to this version as *trace-breaking* TD(λ), since it is equivalent to setting the eligibility traces to zero when an exploratory action is performed.

In this algorithm, having the optimal value function will lead to unbiased learning signals; thus, this function is stable if the process is deterministic. On the other hand, the algorithm has the disadvantage that it breaks the chain of feedback from later states to earlier ones when exploration occurs. This loss of information means that a greater number of episodes are needed for learning, especially if the agent explores frequently.

3 The Loss-Compensating Algorithm

The motivation of our algorithm is the observation that by breaking the signal chain at an exploring action, valuable information is lost. We want to remedy this loss by devising a way of carrying the learning signals across the exploring step, without at the same time distorting the learning signal as in the naive algorithm. The idea is to exploit the knowledge already present in the evaluation function in order to achieve this goal. Of course, this kind of bootstrapping – using the present value estimates for the purpose of finding better ones – is the principle behind all TD-learning; here it is taken one step further.

Our algorithm works in the following way: Whenever the agent makes an exploratory action, it gives itself a reward equal to its own evaluation of how inferior the action is. If the agent has a correct evaluation of every state, this will produce a TD-learning signal that is unbiased, because it then evaluates the downside of the "unwanted" exploration correctly. (If the process is deterministic, this means that the evaluations will remain correct.) Referring to the procedures given in Section 2, we can implement this algorithm by changing two steps. First, step 3 of the MDP is replaced by:

3'. Act by performing the following steps:

3'a. Find the greedy action: $\hat{a}_{t+1} \leftarrow \arg\max_a \hat{Q}(s_t, a)$.

3'b. Choose an action a_{t+1} according to some policy.

3'c. Calculate the perceived loss: $l_{t+1} \leftarrow \hat{Q}(s_t, \hat{a}_{t+1}) - \hat{Q}(s_t, a_{t+1})$.

Second, step 4a in the TD(λ) algorithm is changed to:

4a''. $\delta \leftarrow r_{t+1} + l_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) + \gamma\lambda\delta$.

Preliminary experiments with this algorithm revealed one flaw: for certain parameter values – notably for high values of λ – the value estimates diverged by increasing exponentially. This is bad news, and can be explained informally as follows in the special case of $\lambda = 1$: The compensating rewards l are always non-negative, so for a given evaluation function, the expected value of the feedback may be higher than the actual expected return, but not lower. The feedback signals therefore have a positive bias, unless the evaluation function is correct. If the greedy use of the evaluation function recommends an inferior action at a given state, the compensating reward for exploring the actually correct action is the source of the bias. For a given state visited by the process, the bias in the feedback signal is equal to the expected sum of these future false rewards. Therefore, the feedback bias tends to be proportional to the average error of the evaluation function. This means that the feedback error tends to be proportional to the current error, which is a standard recipe for exponential explosion.

A simple fix for this instability – apart from tuning down λ – is implemented by truncating the evaluations at a maximum value M. After each training step, we set $\hat{V}(s_t) \leftarrow \min(M, \hat{V}(s_t))$. With this modification, there is of course no risk of exponential explosions beyond the limit M. In most cases, a good upper bound on the state values is available, either from prior knowledge of the problem, or from the actual payoffs achieved by the agent during training. Still, it may seem naive to hope that this simple trick will make an unstable algorithm convergent. However, the experiments that we present in the next section indicate that it actually works, at least in our grid-world example.

We are now able to write our full *loss-compensating* TD(λ) algorithm as follows: Given an MDP, λ, α, M, and an action selection policy:

1. Initialise $\hat{V}(s)$ to 0 for each terminal s and (e.g.) randomly for all other $s \in S$.
2. Run an episode of the MDP:
   - M1. Initialise starting state s_0, $t \leftarrow 0$.
   - M2. If $s_t \in \Sigma$, then $T \leftarrow t$ and go to step 3.
   - M3. Act by performing the following steps:
     - M3a. Find the greedy action: $\hat{a}_{t+1} \leftarrow \arg\max_a \hat{Q}(s_t, a)$.
     - M3b. Choose an action a_{t+1} according to the action selection policy.
     - M3c. $l_{t+1} \leftarrow \hat{Q}(s_t, \hat{a}_{t+1}) - \hat{Q}(s_t, a_{t+1})$.
   - M4. A new state s_{t+1} is drawn according to $p_T(\cdot \mid s_t, a_{t+1})$.
   - M5. Receive a reward r_{t+1} drawn according to $p_R(\cdot \mid s_t, a_{t+1}, s_{t+1})$.
   - M6. $t \leftarrow t+1$; go to step M2.
3. Learn from the episode:
   - L1. $\delta \leftarrow 0$.
   - L2. For t from $T-1$ down to 0 do:
     - L2a. $\delta \leftarrow r_{t+1} + l_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t) + \gamma\lambda\delta$.
     - L2b. $\hat{V}(s_t) \leftarrow \min(M, \hat{V}(s_t) + \alpha\delta)$.
4. If termination criterion is met, then end, else go to step 2.

4 Experiments

The state space of our experimental MDP problem is the grid world of a fully connected rectangular 5×5 grid, with terminal states $\{(5,5),(2,4),(3,4),(4,4),(5,4)\}$. The node $(5,5)$ is the goal, and reaching this goal gives a reward of 1. The other four terminal nodes are "traps", which give a reward of –10. We set the discount factor γ to 0.9. On each time step, the agent can move along an arc in the network or stay put, and the starting point of the process is the lower left-hand corner $(1,1)$. Figure 1 shows the grid world. Clearly, the optimal policy is to move four times upwards and then four times to the right, which takes the process to the goal node in eight turns, for a return of γ^8.

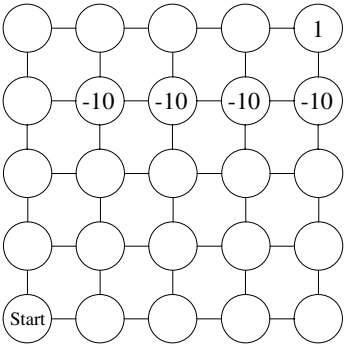

Fig. 1. The grid world

With the given formulation, the process may take a very long time to terminate, giving problems for our episode-based algorithm. To overcome this problem, we use the common technique of transforming the discount factor into a "survival rate" of the process: Instead of discounting future rewards directly, we apply a stopping probability of $1-\gamma$ at each time step. If this random draw stops the process, we let the process jump to an additional terminal node $(0,0)$ with payoff 0. In the MDP formalism above, this means that the transition probabilities from a state given an action are γ for entering the next node and $1-\gamma$ for going to $(0,0)$. (An exception occurs when the agent moves to a trap node, in which case the process never moves to $(0,0)$). For any given policy, this stochastic formulation gives the same expected payoff as the one using infinite horizons and discounted payoffs. Future payoffs are discounted indirectly, because they are less likely to occur. Note that this trick is applicable to any MDP with discounted rewards.

We apply "blind" exploration, so that for each decision there is a fixed probability $p_{explore}$ of the agent making a random action, with equal probability assigned to each possible action. We have tested a wide range of parameter settings for α, $p_{explore}$ and λ, and compared our loss-compensating algorithm to the trace-breaking algorithm.

We use two different error measures, designed to measure how quickly the algorithms learn to approximate the true value function $V*$ by their estimates $\hat{V}$. The error measures are defined as the root-mean-square difference between the true values and the estimates, taken along the optimal path and over all states, respectively. Denoting the shortest distance from a state s to the goal node by $d(s)$, we write the mathematical definitions of these error measures as

$$e_{rms}^{P} = \left(\frac{1}{8} \sum_{s \in P} \left(V*(s) - \hat{V}(s) \right)^2 \right)^{1/2} = \left(\frac{1}{8} \sum_{s \in P} \left(\gamma^{d(s)} - \hat{V}(s) \right)^2 \right)^{1/2} \tag{4}$$

– where $P = \{(1,1),(1,2),(1,3),(1,4),(1,5),(2,5),(3,5),(4,5)\}$ is the subset of S belonging to the optimal path – and

$$e_{rms}^{S} = \left(\frac{1}{20} \sum_{s \in S} \left(V*(s) - \hat{V}(s) \right)^2 \right)^{1/2} = \left(\frac{1}{20} \sum_{s \in S} \left(\gamma^{d(s)} - \hat{V}(s) \right)^2 \right)^{1/2}. \tag{5}$$

Note that it is not necessary to include the terminal states in the sums, since the algorithms guarantee that these are evaluated to the correct value of 0.

For each algorithm and parameter setting, we ran ten trials with different random initialisations of the value functions; the same set of initialisations was used for every algorithm and parameter setting. Our algorithm worked better in all cases, and Figures 2 and 3 give a typical example. In these figures, the parameters settings are $\alpha = 0.05$, $\lambda = 0.75$, and $p_{explore} = 0.5$. In the loss-compensating case, a maximum learning signal of $M = 1$ was used.

From the figures we see that our algorithm learns the task significantly faster than the benchmark algorithm does, although the long-term average performance is similar. This is compatible with the fact that our algorithm makes more use of the data available.

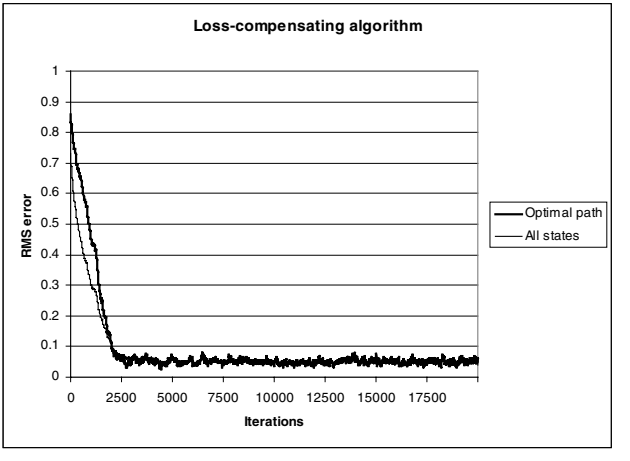

Fig. 2. RMS error as a function of iterations using the loss-compensating algorithm. Parameter settings: $\alpha = 0.05$, $\lambda = 0.75$, $p_{explore} = 0.5$, $M = 1$

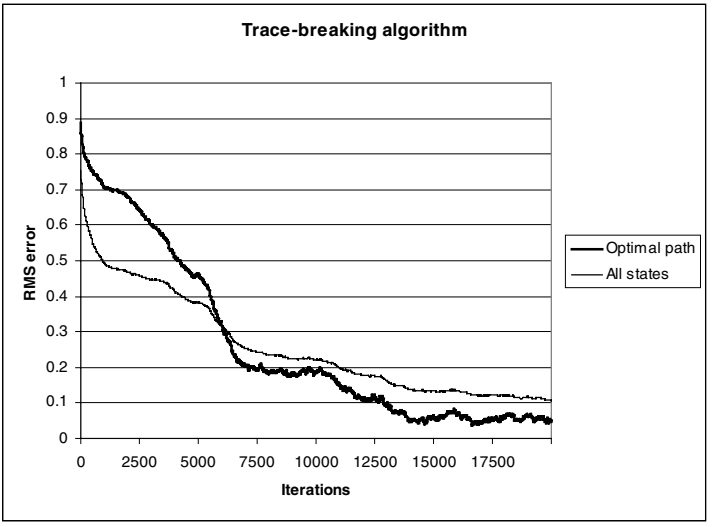

Fig. 3. RMS error as a function of iterations using the trace-breaking algorithm. Parameter settings: $\alpha = 0.05,\ \lambda = 0.75,\ p_{explore} = 0.5$

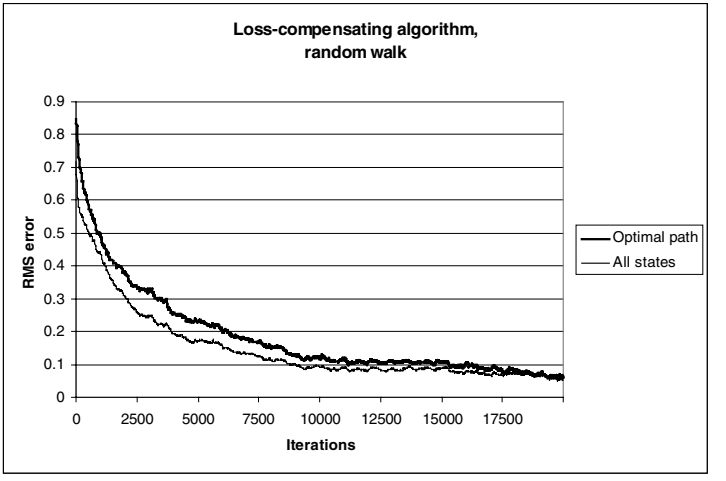

Fig. 4. RMS error of the loss-compensating algorithm learning from a random walk. Parameter settings: $\alpha = 0.05,\ \lambda = 0.75,\ p_{explore} = 1,\ M = 1$

Figure 4 shows the error curves of our algorithm with parameters $\alpha = 0.05,$ $\lambda = 0.75,$ and $p_{explore} = 1.$ We see that the algorithm works even with a random walk exploration of the state space, so the agent learns the optimal value function simply by observing a completely random policy in action. The trace-breaking algorithm would not learn anything with $p_{explore} = 1,$ as all feedback signals would be discarded.

The graph in Figure 2 shows that the root-mean-square error curves do not actually converge toward zero, but rather fluctuate at a given error level. This is due to the randomness of the MDP, and is typical for TD(λ) applied to such processes. As usual with TD(λ), the random fluctuations can be controlled by setting a sufficiently low step size α. We ran ten trials under the same conditions that produced the results in Figure 2, except that for each iteration n we set $\alpha = 0.1 \times (1 + n/500)^{-1}$. The rather successful results are shown in Figure 5.

For the sake of comparison we have also tested the naive algorithm, without ever observing discovery of the optimal policy. The naive algorithm tends to learn to move towards the starting node, and stay there. Unless $p_{explore}$ is very low (approximately 0.05 or less), this is actually the logical result, as the risk of being forced into a trap state by an exploratory action outweighs the potential reward from reaching the goal node. Even when $p_{explore}$ is low, exploration is likely to make the process hit a trap node once in a while, and it is therefore reasonable that the agent learns to stay as far away from the trap nodes as possible. Once this policy is established, it is virtually impossible for the process ever to reach the goal node, given the low $p_{explore}$.

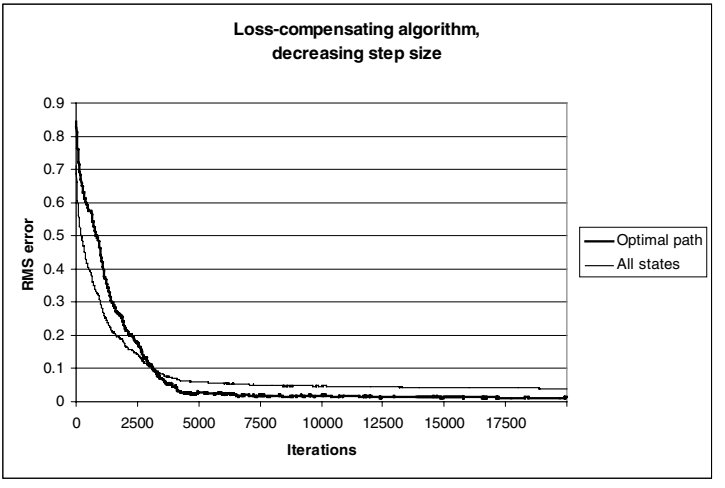

Fig. 5. RMS error of the loss-compensating algorithm using decreasing step size α. Other parameter settings: $\lambda = 0.75$, $p_{explore} = 0.5$, $M = 1$

5 Discussion

In the experiments, we used a very simple exploration policy, with a constant probability for exploring at each step, and uniform probabilities for choosing the exploring actions. However, our algorithm does not impose any constraint on how exploration is to be done. Consequently, it can be used regardless of what kind of exploration policy one might choose to employ.

Our trick of using the evaluation function for extracting rewards that compensate for exploration can be used beyond the context that we consider in the present article. It is not restricted to the episodic formulation of the algorithm, since its only additional requirement is the calculation of the loss l in step M3 above. It also generalises easily to model-based TD-learning where the agent must estimate the model – i.e. the transition and reward probabilities – in addition to the value function, and to Q(λ). In fact, the idea of compensating the agent for the loss incurred by performing an inferior action has a parallel in simple one-step Q-learning [10] – that is, in Q(0) – where the value estimate of a state–action pair is pushed towards the estimate associated with the *optimal* action in the subsequent state. Seen in this light, the relation between loss-compensating TD(0) and naive TD(0) is similar to that between Q(0) and Sarsa(0).

This property of Q(0) does not seem to have been extended in a consistent way to positive λ, although the algorithm that in [4] is called *Peng's Q(λ)* does make some use of the idea [11]. In Peng's Q(λ), the feedback signal is a sum of components, each of which comes from the actual experience of the agent for a number of steps along the process, except for the signal from the last step, which is maximised over possible actions. Unlike our method as applied to Q(λ), however, this algorithm does not produce unbiased learning signals if the optimal value function has been reached.

Our experiments indicate that our modification to the TD(λ) algorithm for learning control may significantly accelerate learning. However, the theoretical properties of the new algorithm are an important subject for future research.

6 Conclusion

We have presented a simple and intuitive modification of TD(λ) learning, based on the idea that the evaluation function can generate estimates of the cost of not following the greedy policy. We treat these estimates as rewards that the agent receives as compensations for making what it evaluates to be inferior actions. We have observed that the compensating rewards are all positive, so that the learning signal will be biased. This may result in exponential explosion of the value estimates, but simply truncating the value estimates at a given level appears to fix this problem, at least in our grid world application. In our experiments, the algorithm outperformed the version of the TD(λ) algorithm that works by truncating feedback signals over exploratory actions. The algorithm is compatible with arbitrary exploration policies. Furthermore, we observe that our trick is also applicable to Q(λ), although this has not yet been tested.

References

1. Sutton, R.S.: Learning to predict by the methods of temporal differences. *Machine Learning* 3 (1988) 9–44.
2. Tesauro, G.J.: Practical issues in temporal difference learning. *Machine Learning* 8 (1992) 257–277.

3. Boyan, J.A., Moore, A.W.: Learning evaluation functions for large acyclic domains. In: Saitta, L. (ed.): *Proceedings of the Thirteenth International Conference on Machine Learning,* Morgan Kaufmann (1996) 63–70.
4. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: an Introduction*. MIT Press (1998). URL: `http://www-anw.cs.umass.edu/~rich/book/the-book.html`
5. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. *Journal of Artificial Intelligence Research* 4 (1996) 237–285.
6. Bertsekas, D.P., Tsitsiklis, J.N.: *Neuro-Dynamic Programming*. Athena Scientific (1996).
7. Szepesvari, C., Littman, M.L.: A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation* 11 (1999) 2017–2060.
8. Thrun, S.B.: The role of exploration in learning control. In: White, D.A., Sofge, D.A. (eds.): *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, Van Nostrand Reinhold, New York (1992).
9. John, G.H.: When the best move isn't optimal: Q-learning with exploration. In: *Proceedings, 10th National Conference on Artificial Intelligence*, AAAI Press (1994) 1464.
10. Watkins, C.J.C.H.: *Learning from Delayed Rewards.* PhD thesis, University of Cambridge, UK (1989).
11. Peng, J.: *Efficient Dynamic Programming-Based Learning for Control.* PhD thesis, Northeastern University, Boston (1993).