Actual return reinforcement learning versus Temporal Differences: Some theoretical and experimental results

Mark D. Pendrith
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 Australia
pendrith@cse.unsw.edu.au

Malcolm R. K. Ryan
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 Australia
malcolmr@cse.unsw.edu.au

Abstract

This paper argues that for many domains, we can expect credit-assignment methods that use *actual returns* to be more effective for reinforcement learning than the more commonly used temporal difference methods. We present analysis and empirical evidence from three sets of experiments in different domains to support this claim. A new algorithm we call C-Trace, a variant of the P-Trace RL algorithm is introduced, and some possible advantages of using algorithms of this type are discussed.

1 Introduction

Whether it is better to learn on the basis of actual outcomes (also known as *rollouts* or *actual returns*), as in Monte Carlo methods, or to learn on the basis of interim estimates, as in temporal difference (TD) methods, has been identified as a key question in reinforcement learning (RL) [17]. The evidence to date has been mixed. In [17], Sutton points out the former enjoy a number of theoretical advantages when certain classes of function approximators are used [7, 3]; Boyan and Moore [4] have demonstrated such effects particularly convincingly in simulated domains. On the other hand, in empirical studies the latter have been shown to enjoy better learning rates when $TD(\lambda)$-based implementations have been used [16, 17]. Jaakkola et al. [8] have pointed to the advantages of actual return learners over TD methods when learning in a non-Markovian domain where the optimal policy is non-stationary.

In this paper, we extend previous work [11] which looked at the possible advantages of using actual return-based methods for credit-assignment in noisy and non-Markovian domains.

We present results from three different sets of experiments. The first set, using a simulated pole-and-cart domain, looks particularly at the effect of noise and disturbance on learning rate, and on the robustness of the learnt control policy. Five reinforcement learning algorithms have been studied, three of which (BOXES, AHC and 1-step Q-learning) have been written about extensively; the others, P-Trace and Q-Trace, have been described in [11].

Motivated directly by issues arising from the results of the first experiments, the second experiment compares the performance of four of these algorithms in an artificial non-Markovian decision problem.

In the third experiment, we scale up from simulations to test our hypotheses on a real-world control application. The setting is a real robotic domain which is both noisy and non-Markovian, thereby providing a test-bed that combines the key elements from the earlier experiments. This also makes it representative of an important class of dynamical system control problems. In this experiment, we directly compare a TD learner (Watkins' 1-step Q-learning) to a variant of the actual return learner P-Trace algorithm we call C-Trace.

2 Algorithms

2.1 BOXES, AHC and Q-learning

The BOXES algorithm is Sammut and Law's [13] modern variant of Michie and Chambers' original reinforcement learning algorithm, which was first applied to the benchmark pole-and-cart control problem [10]. It uses actual returns rather than a TD approach to learning expected payoffs from state/action pairs.

The Adaptive Heuristic Critic (AHC) algorithm [2] was also originally studied in the context of the pole-and-cart problem. This algorithm was developed very much in the spirit of a 1-step TD design proposed by

Witten [19]; the main conceptual advance is from the 1-step to a multiple step TD mechanism, using an "eligibility trace" with an exponentially decayed recency parameter λ.

1-step Q-learning was introduced by Watkins [18]. The key advance here was to represent explicitly the learnt value function as state/action pair payoffs rather than state payoffs, which meant the algorithm gained the key property of *experimentation insensitivity* [18, 12] in Markov domains, and led to proofs of convergence and optimality in these domains.

More recently, variants which incorporate both multiple step TD and state/action pair Q-value representation have been proposed; these include the Q(λ)-learning algorithm [12]. This is discussed briefly in the section 3.

2.2 P-Trace

This subsection briefly describes the P-Trace algorithm. More detail can be found in [11]. P-Trace is an algorithm which, like Q-learning, learns a mapping of ⟨state,action⟩ to expected total future rewards (or payoff). It learns these state/action pair Q-values using direct rather than TD methods. By "direct", we mean a Q-value estimate $Q(s,a)$ is modified on the basis of the actual payoff outcomes experienced after action a is executed from state s. In this way, it is essentially a Monte Carlo approach. In contrast, TD methods modify Q-values on the basis of differences between temporally successive estimates.

We recall the update rule of 1-step Q-learning:

$$Q(s_t,a) \leftarrow (1-\beta)Q(s_t,a) + \beta y \tag{1}$$

where

$$y = r_t + \gamma \max_b Q(s_{t+1},b) \tag{2}$$

where s_{t+1} is the successor state of s_t after executing action a, $0 \leq \gamma < 1$ is the discount factor and r_t is the immediate reward following the action a from state s_t.

P-Trace uses a structurally similar update rule to Q-learning, but with a different y-value, that of the *actual return*, i.e.:

$$y = \mathbf{r}(s_t) \stackrel{\triangle}{=} \sum_{i=0}^{n-1} \gamma^i r_{t+i} \tag{3}$$

where $t+i$ ranges from the current time-step t up to termination at time-step $t+n$, and r_{t+i} is the reward received at time-step $t+i$. The update rule is also applied less often in the case of P-Trace. Specifically, it is only applied when a terminal state has been reached, rather than after every time-step, and then only to the state/action pairs visited for which there were no non-policy actions executed subsequently. This conditional resetting of the eligibility trace, or "trace-zeroing" as we will subsequently refer to it, keeps the return "pure" (hence P-Trace.)

If a state/action pair is visited n times before termination, then conceptually the update rule is applied n times, once for the actual return associated with each visit. In practice the P-Trace algorithm performs the corresponding update in one step using the update rule

$$Q(s,a) \leftarrow (1-\beta)^n Q(s,a) + z \tag{4}$$

where z is

$$z = \sum_{i=1}^{n} (1-\beta)^{n-i} \beta y_i \tag{5}$$

and where y_i is the discounted return calculated from the ith visit. For the purposes of an efficient algorithmic implementation, we derive[1]

$$\sum_{i=1}^{n} (1-\beta)^{n-i} \beta y_i = \sum_{i=1}^{n} \rho_i x_i \tag{6}$$

where ρ_i is the discounted return received between the ith visit and the $i+1th$ visit (or termination if $i=n$), and x_i is defined as

$$x_i = \sum_{j=1}^{i} (1-\beta)^{n-j} \beta \gamma^{\tau_{ji}} \tag{7}$$

where τ_{ji} is the number of timesteps between the jth visit and the ith visit. As n increases, the sum $\sum_{i=1}^{n} \rho_i x_i$ can be incrementally recalculated as

$$Sum \leftarrow (1-\beta)Sum + \rho_n x \tag{8}$$

whereupon x is then incrementally recalculated according to the rule

$$x \leftarrow (1-\beta)x\gamma^{t_n} + \beta \tag{9}$$

with $Sum = 0$ and $x = \beta$ as the initial values. (t_n is the number of timesteps between the $n-1th$ and the nth visit.) These incremental recalculation rules translate directly into the pseudo-code in Figure 1

2.3 P-Trace variants: Q-Trace and C-Trace

Q-Trace is a TD/actual return hybrid that incorporates the P-Trace update mechanism with the 1-step Q-learning TD method. It was designed primarily to empirically investigate possible advantages of a mixed mode credit-assignment method. It is described more fully in [11].

[1] Refer to [11] for more details, if required.

```
while not terminal state do

   get current state s
   select action a ( stochastic action selection )

   if non-policy action selected then
      for all (i, j) such that VisitCount_ij > 0 do
#if C-Trace
         c ← max_m Q_im
         k ← GlobalClock − StepCount_ij
         ρ_ij ← ρ_ij + cγ^k ( truncated return correction )
         n ← VisitCount_ij
         Sum_ij ← (1 − β)Sum_ij + x_ij ρ_ij
         Q_ij ← (1 − β)^n Q_ij + Sum_ij ( CTR update )
#endif
         VisitCount_ij ← 0 ( zero traces )
      endfor
   endif

   if VisitCount_sa = 0 then ( first visit )
      Sum_sa ← 0
      x_sa ← β
   else ( subsequent visits )
      Sum_sa ← (1 − β)Sum_sa + x_sa ρ_sa
      k ← GlobalClock − StepCount_sa
      x_sa ← (1 − β)x_sa γ^k + β
   endif

   ρ_sa ← 0 ( reset inter-visit reward total )
   VisitCount_sa ← VisitCount_sa + 1
   StepCount_sa ← GlobalClock ( "time-stamp" visit )

   take action a

   if reinforcement signal r received then
      for all (i, j) such that VisitCount_ij > 0 do
         k ← GlobalClock − StepCount_ij
         ρ_ij ← ρ_ij + rγ^k
      endfor
   endif

   GlobalClock ← GlobalClock + 1

endwhile

for all (s, a) such that VisitCount_sa > 0 do
   n ← VisitCount_sa
   Sum_sa ← (1 − β)Sum_sa + x_sa ρ_sa
   Q_sa ← (1 − β)^n Q_sa + Sum_sa ( terminal update )
endfor
```

Figure 1: Pseudo-code for C-Trace, the new P-Trace variant suitable for both continuous and trial-based learning environments. Between **#if C-Trace** and **#endif** are the added steps that change P-Trace to C-Trace. The variables Sum, x, ρ, $StepCount$ and $VisitCount$ are kept separately for each state/action pair; the subscripts identify to which state/action pair the variable belongs.

In this paper we introduce the newer C-Trace algorithm (see Figure 1). C-Trace is used in the "learning to walk" robot experiment. C-Trace has been designed to be suitable for learning in both continual (i.e. non-terminating) and trial-based learning environments.

C-Trace performs updates more frequently than P-Trace. In addition to updating when a terminal state is reached according to the P-Trace rule, it will perform an update using a *corrected truncated return* (CTR) [18] whenever a non-policy action has been selected for execution. That is, for C-Trace

$$y = \mathbf{r}^{(n)}(s_t) \stackrel{\triangle}{=} \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n U(s_{t+n}) \qquad (10)$$

where $t+n$ is the time-step a non-policy action is selected, and s_{t+n} is the state from which it is selected, and $\gamma^n U_{t+n}$ is the *truncated return correction*, where

$$U(s_{t+n}) = \max_b Q(s_{t+n}, b) \qquad (11)$$

which is the estimated expected return (or Q-value estimate) associated with the current policy action for state s_{t+n}.

C-Trace uses of a mixture of n-step CTRs, where the number of steps n in the return horizon may vary from one return to the next. This distinguishes C-Trace from other RL algorithms.

Varying Watkins' [18] notation slightly, we use $V^\pi(x)$ for the expected value of the actual return from state x following policy π from that point on (assuming a Markovian domain). Watkins points out that the reason CTRs are useful estimators is that the expected value of the CTR tends to be closer to V^π than to U. This is sometimes called the *error-reduction property* of CTRs.[2]

Although not discussed explicitly in Watkins' thesis, it follows naturally from his treatment in Chapter 7 that as the CTR horizon n is increased, the error-reduction property becomes more vigourous. In the limiting case of $n = \infty$, the CTR becomes the actual return.

This observation motivates the use of actual returns in P-Trace. However, as in Monte Carlo methods, the use of actual returns alone limit P-Trace to learning in trial-based learning domains. C-Trace overcomes this limitation by using CTRs as well as actual returns.

[2]Following Watkins, let K be defined as the maximum absolute error of U, that is

$$K = \max_x |U(x) - V^\pi(x)|$$

Then, since $V^\pi(x) = E[\mathbf{r}(x)]$, from (3) and (10) it follows that

$$\max_x |E[\mathbf{r}^{(n)}(x)] - V^\pi(x)| \leq \gamma^n K$$

Note that $E[r^{(n)}(x)]$ is not necessarily closer to V^π than to $U(x)$ for all x, but the maximum error of $E[r^{(n)}(x)]$ is less than the maximum error of U. Hence, the error-reduction property.

By design, C-Trace uses CTRs in such a way as to maximise the mean step-size horizon n, while preserving the key property of experimentation insensitivity in the presence of active exploration. This is accomplished by delaying the application of the truncation correction to the return for as long as possible, only applying it at the point when a non-policy action is selected. The property of experimentation insensitivity [12] means that, like 1-step Q-learning, it can learn the optimal policy even if, for the sake of active exploration, it does not always following the optimal policy. This is discussed further in section 3.

3 Related work

It is instructive to compare the P-Trace algorithm and its variants to both Q(λ)-learning [12], and to Monte Carlo methods (see, for example, [1]).

As we have already mentioned, the P-Trace mechanism is conceptually related to Monte Carlo methods, and indeed can be viewed a Monte Carlo method specialised for sampling $Q^{\pi}(s,a)$ values, where π represents the current policy.

This specialisation is achieved via the "trace zeroing" mechanism; a simple Monte Carlo method that did not employ this method would be sampling returns reflecting $\pi + e$, where e represents the current exploration strategy. Thus, a simple Monte Carlo method will be experimentation sensitive, which should be noted as the key difference with respect to the P-Trace method. In the case of C-Trace, applying the CTR before trace-zeroing does not affect experimentation insensitivity.

Secondly, since the $TD(\lambda)$ return is equal to the actual return when $\lambda = 1$ [16, 18], it is reasonable to ask whether P-Trace is equivalent to Q(λ)-learning with $\lambda = 1$. Perhaps surprisingly, the answer is no. In Q(λ)-learning, the sum of differences in estimates of eventual actual return from state to state will not cancel each other out properly to provide an accurate return with respect to the greedy policy if a non-policy action is chosen somewhere along the way; Q(λ)-learning is experimentation sensitive for $\lambda = 1$ (and, in general, for all $\lambda > 0$) if active exploration is to occur [12]. As previously mentioned, the P-Trace mechanism for calculating actual returns is experimentation insensitive because the trace is "zeroed" after a non-policy action is selected. Further, P-Trace also avoids a difficulty with the Q(λ)-learning approach described by both Peng & Williams [12] and Cichosz [6] to do with an inaccuracy proportional to the square of the learning factor β in the calculated return.[3]

Additionally, both P-Trace and C-Trace enjoy better space and time complexity characteristics. In environments with sparse rewards the average case time complexity for P-Trace is only $1/n$ that of $TD(\lambda)$-based methods, where a non-zero reward is received on average every n time-steps. (This is a consequence of the eligibility list having to be processed only after a non-zero reward is received, rather than after every time-step, as in the case of TD methods.) When using standard "accumulating traces" [17, 15], the worst-case complexity (space and time) is $\mathbf{O}(S \times A)$ for $TD(\lambda)$ based implementations (where S is the number of discrete states in the system and A is the number of allowable actions from within each state), as opposed to $\mathbf{O}(S)$ for P-Trace.

Finally, there is some evidence the P-Trace mechanism leads to greater learning stability due to reduced perturbation of Q-value estimates during learning, which can be particularly problematic for higher values of λ.[4]

4 Experiment 1: Noisy pole-and-cart

The noise experiments with the pole-and-cart domain were designed to investigate the effects of systematically increased noise and disturbance on learning rate and on the quality of the learnt policy.

Two separate noise models were investigated, both yielding similar results. These could be described as a "proportional noise model" (model 1), where the white noise perturbation applied to each state variable is proportional to the current absolute value of the state variable, and a "normalised noise model" (model 2), where the white noise perturbation applied to each state variable is scaled by the previously measured standard deviation of that state variable during balancing trials. As noise and disturbance were increased, the learning times of 1-step Q-learning (QL) were most severely impacted; by the time the noise level reached 8%, it was three orders of magnitude slower than BOXES, and two orders of magnitude slower than the other algorithms studied (see plots in Figure 2. Note that the learning times in these plots

[3] In some preliminary experiments with Q(λ)-learning algorithm on the pole-and-cart problem with λ close to 1, Q(λ)-learning would not learn the task even in the absence of noise with the learning factor set at 0.5 (as was used in the other algorithms in the experiments reported here). However, it *would* learn if this factor was reduced to 0.2, the suggested setting in [12]. This seems to suggest that this systemic inaccuracy should not be treated as insignificant.

[4] It is well documented that $TD(\lambda)$ performs relatively poorly with $\lambda = 1$ [16, 17]; it has been suggested this is empirical evidence for the superiority of TD over actual return methods. This interpretation does not fit well with our results, however. We suggest the real culprit in the case of $TD(1)$ learning may be instability arising from the large perturbation of the estimators as the traces build up, rather than a TD versus actual return issues *per se*.

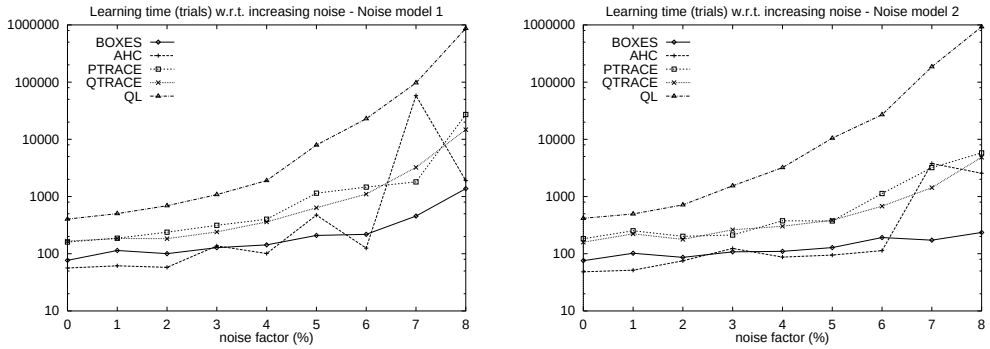


Figure 2: Noisy pole-and-cart: Learning times

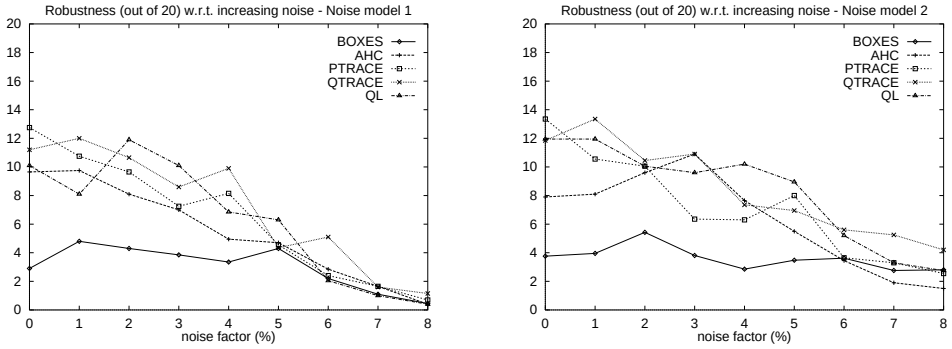


Figure 3: Noisy pole-and-cart: Robustness scores

are in logarithmic scale.) This is compared to learning times at the zero noise level, where learning times for all algorithms studied were far more comparable. The algorithms employing actual returns (BOXES, P-Trace and Q-Trace) showed greater learning stability and predictability overall as the noise level increased. Although AHC demonstrated good median learning times, better than the mean times might suggest, the algorithm exhibited a high sensitivity to small differences in initial conditions with respect to learning times, an effect which became more pronounced as the noise level increased.

With regard to learning quality, the robustness of the learnt policies after the learning criterion of balancing the pole for 10,000 time steps (each time step 0.02s) was tested by restarting the simulation a further 20 times from slightly different randomly selected initial conditions. By counting the number of times the learning criterion of balancing the pole for 10,000 steps could be repeated, a robustness score out of 20 was generated for each run (a run being set of learning trials). 20 such runs were conducted at each noise level for each noise model and for each algorithm, leading to 1800 runs performed in all. The average values resulting from these runs are plotted in Figure 3.

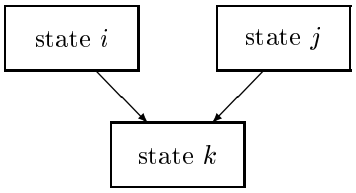
The results for noise model 1 and noise model 2 share many consistent features. For both models, we observe that QL, Q-Trace and P-Trace share similar robustness characteristics, with mean scores starting off in the range 10.1–11.9 at the low noise levels, and generally steadily decreasing to values in the range 0.4–1.1 (noise model 1) and 2.7–4.2 (noise model 2) by the time the 8% noise level is reached. In general, the control actions learnt by BOXES were shown to be relatively the least robust, particularly at the lower noise levels less than 5%. Overall AHC fell between BOXES and the others in robustness performance. At the higher noise levels, however, there is a general convergence towards zero with respect to robustness scores over all algorithms.

There are several possible reasons that may help to explain the differences in learning performance between QL and the other algorithms. Firstly, and most obviously, a trace mechanism provides for more immediate propagation of the reinforcement signal information, assigning credit in various degrees to possibly many more than the one state/action pair that is immediately affected in QL. The slow backwards propagation of credit that is associated with QL is thus potentially accelerated.

There may be another possible contributing factor to the observed effects. QL is designed to work in a Markovian environment, where the expected value of the payoff from choosing a particular action from a

particular state is independent of which states were antecedent. In the case of the pole-and-cart system with the particular input representation under consideration, a "state" is a discretised chunk of the state-space, and the payoff from an action selected from one of these states could conceivably be influenced by which immediately preceding states have been visited, particularly if the chunks are large. That is to say, selecting action n from box k being fed from box i might exhibit quite different payoff characteristics than selection the same action from box k being fed from box j.

This is the behaviour of a non-Markovian domain. We might now consider why P-Trace and Q-Trace (and actual return learning algorithms generally) might be expected to cope better with credit assignment in such a domain. Consider the simple case of a hypothetical state in a non-Markovian system we will call state k, which has two possible antecedent states, states i and j. The policy for state k is currently action n.



Now, because our domain is non-Markovian, the expected payoff for action n from state k may be different if the immediately preceding state is i or j. If the antecedent state is i, let the expected payoff for action n from state k be 0.95; if the antecedent state is j, let the expected payoff be -0.95.

Let us also suppose that the state transition $i \rightarrow k$ occurs more frequently than the state transition $j \rightarrow k$; say a ratio of about 10:1.

So, the Q-value of action n from state k would be expected to converge to something in the order of 0.85. State j might well be artificially "encouraged" whenever it made a state transition $j \rightarrow k$, thinking it was doing well, when in fact this would on average lead to a relatively poor outcome if action n is policy for state k.[5]

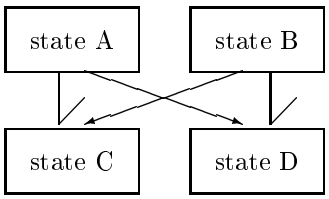
If credit assignment was done via a direct trace mechanism, however, state j would soon learn that a transition to state k is generally less rewarding than a TD derived Q-value would suggest. Thus, the important general advantage the actual return learner has in non-Markovian domains should be clear from this example: it can take into account biasing effects of preceding states when assigning credit if necessary.[6]

The next experiment provides some direct empirical support for this analysis.

5 Experiment 2: A non-Markovian decision problem

To empirically test some of the theoretical results regarding differences between actual return and TD learners in non-Markovian domains, a simple experiment was performed involving an artificially constructed non-Markovian decision problem.

Consider the following four state non-Markovian domain:



States A and B are are the two possible starting states. There are two possible actions available to choose from in each state.

The state transition rules are as follows:

- Action 0 from state A or state B always leads to state C
- Action 1 from state A or state B always leads to state D
- Any action taken from state C or D immediately terminates with a reinforcement signal.

The reinforcement received from an action made in state C or D may depend not only upon the current state, but also upon what the previous states were (that is, we cannot assume the Markov property.)

The reinforcement schedule is set as follows:

[5] A similar construction is used for didactic purposes in Singh, Jaakkola and Jordan [14]. Their analysis of the limitations of TD(0) learning is essentially the same.

[6] Also, while QL has been shown to be "experimentation insensitive" [12] in Markovian domains, it may not be widely appreciated that this property does *not* hold for QL in non-Markovian domains. To see why this is the case, consider the case where an alternate exploration strategy reverses the relative frequencies of the $i \rightarrow k$ and $j \rightarrow k$ state transitions in our example above. In this case, the Q-value for action n from state k would converge to quite a different value to the one originally calculated, closer to -0.85 than to 0.85. Thus QL is experimentation sensitive in non-Markovian domains.

Action 0 from state C
  if the previous state was A: 1.0
  if the previous state was B: –2.0

Action 1 from state C
  if the previous state was A: 0.0
  if the previous state was B: 0.0

Action 0 from state D
  if the previous state was A: –2.0
  if the previous state was B: 1.0

Action 1 from state D
  if the previous state was A: –0.1
  if the previous state was B: –0.1

The goal of a learning agent in this domain is to learn a policy that maximises its rewards over time. For a trial, the agent is started randomly in state A or B. Learning takes place over 10,000 trials.

We note the domain was so constructed that the actual optimal policy of A (action 0), B (action 1), C (action 0), D (action 0) is immediately apparent by inspection; also that the average per-trial payoff for an agent that strictly adhered to this policy would be 1.0. It was also constructed so that any agent that would be making an either/or overall judgement about the potential payoff from state C or D regardless of the preceding state (i.e. makes the Markov assumption, as does a TD learner) would prefer the mediocre but relatively safe policy of selecting action 1 from states C and D.

If we add the additional constraint that state A and state B must agree on their action (and hence next state) choice, then the best policy that could be settled upon would be A (action 0), B (action 0), C (action 1), D (don't care), which has a per-trial payoff of 0. This is the best a pure TD learner could be expected to do. The next best policy of A (action 0), B (action 0), C (don't care), D (action 1), has a slightly worse per-trial payoff of –0.1.

Of course, as the learning agents have less than perfect information about the domain, and exploration is necessary, we would expect the actual average per-trial payoff to be less than for strategies based on perfect information.[7] This is observed in the tabled results.

The average payoff per trial for five algorithms[8] is shown Table 1 (each row represents a separate run of 10,000 trials for each algorithm using a different seed for the pseudo-random number generator).

[7]For QL, Q-Trace and P-Trace a simple "Boltzmann distribution" stochastic action selector (SAS) as described in Lin [9] was used with temperature T set to 1.0.

[8]BOXES was not included in the experiment because in its present form it does not learn from mixed sign reinforcement signals.

	Average Payoff/Trial			
Seed	P-Trace	Q-Trace	AHC	QL
0	0.50485	–0.06154	0.45525	–0.18734
1	0.54315	–0.06873	–0.05042	–0.18507
2	0.50122	–0.07032	0.49490	–0.19266
3	0.47765	–0.06264	–0.05010	–0.20189
4	0.47525	–0.10331	0.99979	–0.18651
5	0.47968	–0.04491	0.50139	–0.18248
6	0.49644	–0.05691	0.49479	–0.18019
7	0.47288	–0.02905	–0.04943	–0.20071
8	0.51468	–0.06949	0.50419	–0.18603
9	0.49213	–0.06065	–0.05110	–0.18373

Table 1: Results for non-Markov decision problem

Overall, the results are consistent with the analysis of the previous section. Of the algorithms trialled, P-Trace, the actual return learner, was able to cope with this non-Markovian domain best, while QL, the pure TD learner fared worst.

Q-Trace and AHC in different ways represent compromises between pure TD and actual return learners. It is therefore not surprising that overall their performances fall between that of the pure actual return (P-Trace) and the pure TD learner (QL). The AHC results exhibited the greatest variance, which was also a notable feature in the noisy pole-and-cart experiment. In this experiment, AHC's performance was close to P-Trace 60% of the time, while for the other 40% the performance closely resembled that of Q-Trace. Also note the anomalous result for AHC with the random seed set at 4. The number 0.99979 indicates that AHC did not try any other than the theoretical optimal policy more than three times out of 10,000 trials. This would indicate that little exploration was occurring. Why it settled so quickly into the optimal policy in this one case is unclear.

The performance of all the algorithms might be contrasted with the expected outcome for a purely random agent as a baseline result. All eight reinforcement outcomes are equally likely with random action selection, so the expected per-trial outcome for a purely random agent is simply the average value of all possible reinforcement outcomes, which is –0.725. All do significantly better than a random agent would.

This simple experiment effectively highlights the advantages an actual return learner has over a pure TD learner in a non-Markovian environment even when the optimal policy is a stationary one[9]; an actual return learner does not suffer what might be termed the

[9]In complementary work [8], the advantage of actual return learning for non-Markovian decision problems (NMDPs) when the optimal policy is non-stationary has been discussed; in this experiment we consider the case of learning for NMDPs with stationary optimal policies.

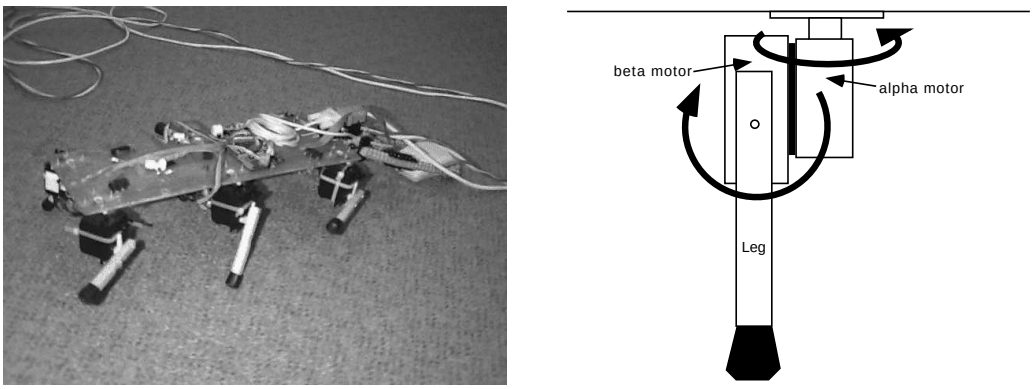


Figure 4: The robot learning to walk, and the leg servo-motor arrangement in detail.

"history aliasing" effect a TD method introduces.

6 Experiment 3: Walking robot

The final study involved a real-world (i.e. not simulated) 6-legged insectoid robot faced with the problem of learning to walk with a forward motion, minimising backward and lateral movements. In this domain, 1-step Q-learning was compared to the C-Trace variant of the P-Trace algorithm.

The robot is, by robotics standards, built to an inexpensive design. Each leg has two degrees of freedom and is powered by two hobbyist servo motors (see Figure 4). A 68HC11 miniboard converts instructions from a 486 PC sent via a RS-232-C serial port connection into the pulses that fire the servo motors. The main sensor the robot has for detecting movement is a standard PC mouse being dragged behind it across the floor in a cradle designed to keep the mouse as flat to the surface as possible.

The robot was given a set of primitive "reflexes" in the spirit of Rodney Brooks' "subsumption" architecture [5]. A leg that is triggered will incrementally move to lift up if on the ground and forward if already lifted. A leg that does not receive activation will tend to drop down if lifted and backwards if already on the ground. In this way a basic stepping motion was encoded in the robots "reflexes".

The legs were grouped to move in two groups of three to form two tripods. The learning problem was to discover an efficient walking policy by triggering or not triggering each of the tripod groups from each state. Thus the action set to choose from in each discretised state consisted of four possibilities: 0) Trigger both groups of legs 1) Trigger group A only 2) Trigger group B only 3) Do not trigger either group. Although quite a restricted learning problem, interesting non-trivial behaviours and strategies were seen to emerge.

This domain was very noisy as well as being non-Markovian by virtue of the compact but coarse discretised state-space representation. The compact representation[10] meant learning was fast[11] when using C-Trace; it was much less successful using Q-learning for the same representation.

The robot received positive reinforcement for forward motion as detected by the PC mouse, and negative reinforcement for backward and lateral movements. The average nett forward movement over the first hour of learning for each algorithm is presented in the plots in figure 5. Clearly, C-Trace was more effective in this domain than 1-step Q-learning, converging relatively quickly to a policy with mean performance comparable to a hand-crafted gait, which was used as a benchmark.

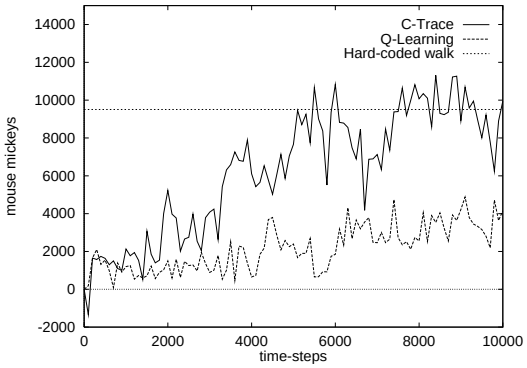


Figure 5: Learning performance for the robot. *Mouse mickeys* indicate the forward speed of the robot, as measured in units of mouse movement per 100 time-steps.

[10] 1024 "boxes" in 6 dimensions: alpha and beta motor positions for each leg group (4 continuous variables each discretised into 4 ranges), plus 2 boolean variables indicating the triggered or untriggered state of each leg group at the last control action.

[11] The robot would learn to walk from scratch in a manner comparable to a hand-crafted gait in about 5000 time-steps, which corresponds to about 30 minutes real time.

7 Conclusions and Future Work

There is still much work to be done on this basic question of actual returns versus TD methods, both theoretically and empirically. The work presented here can be extended in several directions. For example, it would be useful to see a comparison across a number of domains of actual return methods like P-Trace and variants against $TD(\lambda)$-based methods with λ tuned for optimal performance within each domain. Indeed, we can report briefly on preliminary results from such a study currently in progress.

Comparing a slightly newer version of C-Trace to a $TD(\lambda)$-based algorithm (Sutton's variant of Rummery & Niranjan's *sarsa*, see [17]) in the domains *puddle-world, mountain-car, pole-and-cart* and the 21-state Markov chain prediction problem [4, 17, 15], we found that the C-Trace variant performed very nearly equally with respect to learning rate to *sarsa* optimised for λ, with generally much better parameter insensitivity characteristics for the choice of the step-size β. This was true for all the domains with the notable exception of the 21-state Markov chain prediction problem, for which the $TD(\lambda)$-based method was markedly superior in learning rate.

At this stage we can only provide plausible speculation as to why this is the case, but it is our intuition that the answer to the general question "which is better: TD or Monte Carlo" will eventually turn out to be domain dependent, with the critical domain characteristics that indicate the use of one method over the other yet to be fully identified.

For some domain characteristics, however, the picture is clarifying. If the domain in question displays either noisy and/or non-Markov characteristics, as will be the case in many real-world control applications, then there is some empirical evidence and a reasonable body of theoretical results that indicate that in these instances actual returns may be preferable to TD methods.

References

[1] A.G. Barto and M. Duff. Monte Carlo matrix inversion and reinforcement learning. In D.S.Touretsky, ed., *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994.

[2] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[3] D. P. Bertsekas. A counterexample to temporal differences learning. *Neural Computation*, 7:270–279, 1995.

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

[5] Rodney Brooks. Intelligence without reason. In *Proc. of the* 12^{th} *Int. Joint Conf. on Art. Intell.*, pages 569–595, 1991.

[6] P. Cichosz. Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *JAIR*, 2:287–318, 1995.

[7] P. Dayan. The convergence of TD(λ) for general λ. *Machine Learning*, 8:341–362.

[8] T. Jaakkola, S.P. Singh, and M.I. Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*. Morgan Kaufmann, 1995.

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

[10] D. Michie and R.A. Chambers. BOXES: An experiment in adaptive control. In E.Dale and D.Michie, eds., *Machine Intelligence 2*, pages 137–152. Edinburgh: Edinburgh Univ. Press, 1968.

[11] M.D. Pendrith. On reinforcement learning of control actions in noisy and non-Markovian domains. Tech. report, UNSW-CSE-TR-9410, School of Comp. Sci. and Eng., Uni. of NSW, Australia, 1994.

[12] J. Peng and R.J. Williams. Incremental multi-step Q-learning. In W.Cohen and H.Hirsh, eds., *Proc.* 11^{th} *Int. Conf. on Machine Learning*. Morgan Kaufmann, 1994.

[13] C.A. Sammut. Recent progress with BOXES. In K.Furakawa, S.Muggleton, and D.Michie, eds., *Machine Intelligence 13*. The Clarendon Press, OUP, 1994.

[14] S.P. Singh, T. Jaakkola, and M.I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In W.Cohen and H.Hirsh, eds., *Proc.* 11^{th} *Int. Conf. on Machine Learning*. Morgan Kaufmann, 1994.

[15] S.P. Singh and R.S. Sutton. Reinforcement learning with replacing eligibility traces. To Appear In: *Machine Learning*, 1996.

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

[17] R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. To Appear in: *Advances in Neural Information Processing Systems 8*. MIT Press, 1996.

[18] C.J.C.H. Watkins. *Learning from Delayed Rewards*. Ph.D. Thesis, King's College, Cambridge, 1989.

[19] I.H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34:286–295, 1977.