On Reinforcement Learning of Control Actions in Noisy and Non-Markovian Domains.

Mark Pendrith
School of Computer Science and Engineering
The University of New South Wales
Sydney 2052 Australia

E-mail: pendrith@cse.unsw.edu.au

UNSW-CSE-TR-9410 — 30 AUGUST 1994

Communicated by Claude Sammut

Abstract

If reinforcement learning (RL) techniques are to be used for "real world" dynamic system control, the problems of noise and plant disturbance will have to be addressed. This study investigates the effects of noise/disturbance on five different RL algorithms: Watkins' Q-Learning (QL); Barto, Sutton and Anderson's Adaptive Heuristic Critic (AHC); Sammut and Law's modern variant of Michie and Chamber's BOXES algorithm; and two new algorithms developed during the course of this study. Both these new algorithms are conceptually related to QL; both algorithms, called P-Trace and Q-Trace respectively, provide for substantially faster learning than straight QL overall, and for dramatically faster learning (by up to a factor of 200) in the special case of learning in a noisy environment for the dynamic system studied here (a pole-and-cart simulation).

As well as speeding learning, both the P-Trace and Q-Trace algorithms have been designed to preserve the "convergence with probability 1" formal properties of standard QL, i.e. that they be provably "correct" algorithms for Markovian domains for the same conditions that QL is guaranteed to be correct. We present both arguments and experimental evidence that "trace" methods may prove to be both faster and more powerful in general than TD (Temporal Difference) methods. The potential performance improvements using trace over pure TD methods may turn out to be particularly important when learning is to occur in noisy or stochastic environments, and in the case where the domain is not well-modelled by Markovian processes.

A surprising result to emerge from this study is evidence for hitherto unsuspected chaotic behaviour with respect to learning rates exhibited by the well-studied AHC algorithm. The effect becomes more pronounced as noise increases.

1 Introduction

Reinforcement learning techniques are currently being investigated for suitability of use in a wide variety of environments. These range from game playing environments such as backgammon, where these machine learning techniques have been successfully applied to develop systems capable of Master-level play (Tesauro 1992), to noisy and possibly non-Markovian robotic environments (Mahadevan 1994, Mataric 1994, Tham & Prager 1994).

The central thesis presented in this report is that for many domains, we can expect that a non-TD method we call "trace learning" to be much more effective a method of credit assignment for reinforcement learning than pure TD methods. Analytical and experimental results suggest that "trace" learners may prove to be both faster and more powerful learners in general than their TD counterparts.

The experimental evidence is primarily based on a series of studies involving the well-known benchmark pole-and-cart (or inverted pendulum) control problem. The pole-and-cart learning task is set out in Michie and Chambers (1968), and Barto, Sutton and Anderson (1983). For those readers unfamiliar with this control problem, Appendix A provides a detailed description. The only new dimension that has been added for the purposes of the experiments described here has been to make the model less deterministic by simulating a degree of noise or disturbance in the system. At each recalculation of the simulated systems next state, the four state variables were "fuzzed" by the addition of a degree of white noise.

The motivation for this study was to investigate the performance of various credit-assignment algorithms in noisy domains, as a preliminary study evaluating candidates for a reinforcement learning controller for a flight simulator and possibly other up-scale systems. In any realistic dynamic system's control environment, noise and disturbance will be inevitable; only the degree to which they will be encountered will vary. There has been no such comparative study of this aspect of reinforcement learning algorithms published previously, to the best knowledge of the author.

Two different noise models were used in the trials. These are detailed in section 3.2.

It should be pointed out that neither noise model is probably very realistic with respect to the actual noise/disturbance effects that might be expected to be encountered with a real pole-and-cart rig. However, they serve sufficiently well for a first study investigating the effects of moving progressively from a deterministic to increasingly stochastic environment on the learning rates and quality of learning for the five different algorithms studied. The use of two different noise models was motivated by the concern that any effects observed could conceivably be specific to the noise model chosen. However, the results do not suggest any such dependencies. The results for the two different noise models tell essentially the same story with respect to learning rates and learning quality for each of the algorithms studied.

Five reinforcement learning algorithms have been studied, three of which (BOXES, AHC and QL) have been written about extensively; the others, P-

Trace and Q-Trace, are new.

It turns out that these new algorithms, which use "trace learning", learn much faster than the pure TD algorithm studied (Q-learning) in this domain. The relative difference in learning rate increases as the noise level of the learning environment is raised. Analysis suggests the potentially important result that trace learning is a strictly more powerful technique than single or multi-step TD learning in non-Markovian domains. It is proposed that trace learning could be used as a replacement for TD methods for better learning in non-Markovian domains, and could be used (possibly in conjunction with TD methods) to speed-up learning in Markovian domains.

2 Methods

The methods and terminology used borrow heavily from Sammut (1988). A complete series of independent experiments as described below was performed for each noise model.

Basic terminology: A "trial" consists of starting the pole-and-cart system and letting the controller run until either the system fails (the pole falls over or the cart hits an end of the track) or the success criterion of n control steps without failure is reached. A "run" consists of many trials and concludes with a successful trial (i.e. the control task has been successfully learnt). Learning is cumulative over trials within a run; learning for the controller at the start of the $k+1th$ trial within a run picks up where it finished at the end of the kth trial. The "policy" action for the controller is the action that is currently most-favoured for a particular state; this may change over the course of learning.

An experimental series is as follows: For each of the five algorithms, and for nine different noise levels (0–8% in 1% intervals), we do 20 different runs learning to balance the pole. Each run within a noise level for a learning algorithm starts with a different seed for the random number generator. The differences between the runs are related to the initial conditions of the pole-and-cart system at the beginning of each trial (from randomly set but recoverable starting positions), and any other allowably random factors if applicable (e.g. the initial "policy" actions for BOXES, or the action of the "stochastic action selectors" within AHC, QL, Q-Trace and P-Trace; refer Lin (1992) and the next section for discussion on the style of stochastic action selectors used.)

The number of trials it takes to successfully balance the pole for each run is recorded; balancing the pole within a trial for 10,000 sequential control actions without failure (corresponding to 3m 20s in real time) is considered to be "successfully balanced". The average number of trials required to learn the task for an algorithm at a certain noise level is interpreted as a measure of learning rate for that noise level.

At the end of each run, once the pole has been successfully balanced within a trial for the requisite number of time-steps, a further 20 trials with the policy control actions fixed from the end of the last learning run are performed to try to assess the quality or "robustness" of the learnt control actions. Each of these trials are judged successful if from a randomised initial starting position

the control rules can again keep the pole balanced for 10,000 time steps, and unsuccessful otherwise. A "robustness" or "quality of learning" score (out of 20) is thus generated for each algorithm at each noise level.

So, for each run, two numbers are extracted: The number of trials required to learn the control task for this particular run as a measure of learning rate, and a score (out of 20) measuring the quality or robustness of the learnt control policy. Twenty of these number pairs are generated for each noise level to achieve a degree of smoothing through averaging, and to get a measure of the variance of these values within a noise level for a particular algorithm.

The results of these experiments are presented in section 5. Sections 3 and 4 describe the learning algorithms and environment. Section 6 provides discussion of these results, and section 7 presents the results of a simple experiment motivated by some of the theoretical issues raised in this discussion.

3 The algorithms

3.1 Pole-and-cart simulation

For the sake of consistency, exactly the same pole-and-cart simulation code was used for all algorithms. The code is based on code from Claude Sammut's implementation of BOXES, which was in turn based on equations originally derived by Charles Anderson and used in Barto, Sutton and Anderson (1983) and Anderson (1989). The equations of motion along with other details of the simulation can be found in Appendix A.

3.2 The noise models

The first noise model simply scaled the range of the random noise value to the absolute value of the state variable to be fuzzed. So, if the state values at time-step n were calculated to be $x, \dot{x}, \theta, \dot{\theta}$ by the pole-and-cart model's "noise-free" calculations, and the noise level was set to 5%, then the actual state values the simulation would see at the next time step would be $x + random(-x * 0.05, x * 0.05)$, etc. In addition to this, a small absolute value of random disturbance in the range ± 0.0001 was added to ensure that the system did not become noise-free even for zero state variable values.

The second noise model applies a more even level of noise over the state-space than does the first. Instead of scaling the noise levels to the current state variable values, which has the effect of the noise increasing away from the zero states, the noise is scaled to the measured standard deviations of each state variable as measured over several trials. In this model, a 5% noise level means that the random additive fuzz value for each state variable is generated in the range of $\pm 5\%$ of two standard deviations for the state variable.

The measured values for the respective standard deviations were as follows:

x : 0.287055
$\dot{x}$: 0.436060

$\theta : 0.041905$
$\dot{\theta} : 0.575032$

3.3 BOXES

The version of BOXES used is the Sammut and Law variant as described in Sammut (in press). The parameter settings were left unchanged from those that had been selected after tuning for the noise free version of the pole-and-cart balancer.[1] Claude Sammut provided his implementation in C of the algorithm for the experiments that were conducted.

3.4 AHC

The version of AHC is the one described in Barto, Sutton and Anderson (1983). The C source code for AHC used for these experiments was supplied by Richard Sutton of GTE, which he has generously made available by anonymous ftp. The parameter settings were left unchanged from the values set in the anon ftp source distribution.[2]

3.5 Q-Learning

The one-step QL algorithm was implemented from descriptions in Watkins (1989), Watkins & Dayan (1992) and Sutton, Barto & Williams (1992). For these experiments, the discount factor was set to 0.99 and the learning rate series set to a constant 0.5. The "stochastic action selector" (SAS) used is based on one in Lin (1992). One difference in implementing the SAS is that for every second trial in a run, the SAS is turned off, and the action selected is always policy (according to the current Q-value). This allowed for an interleaving of "exploration" and "exploitation" trials, which was found to improve performance for QL on this task. (In fact, it was moderately difficult to find a version of Lin's simple action selector to learn to balance the pole reliably at all, even in the absence of noise.)

As the formal convergence properties of the QL algorithm in Markovian domains is independent of the action selection strategy (as long as it is guaranteed that every action will potentially be tried an infinite number of times)[3] (Watkins & Dayan 1992, Sutton, Barto & Williams 1992), this action selection strategy is entirely compatible with QL, and fits quite naturally with the structure of the learning task. Indeed, it is a generalisable approach to the exploration problem that should fit well with any learning task where the goal is failure avoidance, and failure results in a new learning trial being initiated. For more detail refer to section 3.7 "Stochastic Action Selection (QL, Q-Trace and P-Trace)".

[1] DK = 0.98, K = 1.7, C0 = 0.0, C1 =1.0

[2] ALPHA = 1000, BETA = 0.5, GAMMA = 0.95, LAMBDAw = 0.9, LAMBDAv = 0.8

[3] Peng & Williams (1994) refer to this property of QL as "experimentation insensitivity".

3.6 The new algorithms: P-Trace and Q-Trace

The new "P-Trace" algorithm reintroduces a direct credit assignment "memory trace"[4] that is used to assign credit more heavily for the actions most recently tried prior to a reinforcement signal being received. It is like BOXES and AHC in this respect, but different from QL and other pure TD credit assignment algorithms.

Also newly developed for the purposes of this study is the "Q-Trace" algorithm. Like AHC, it is a hybrid that also incorporates TD learning. The TD method is that of QL, hence the "Q" in the name. The trace mechanism is that of P-Trace; the trace mechanism of P-Trace has been specially designed to be compatible with QL in this hybrid form.[5]

Both algorithms have been introduced into this study for comparative purposes as examples of other "pure" and hybrid trace learners, to help settle some speculation raised in the analysis of the performance of the other algorithms tested. They have also been designed to be "clean" and mathematicially tractable algorithms to facilitate reasoning about the underlying principles involved. Pseudo-code for these algorithms is in Appendix B; section 4 describes the new algorithms in detail.

3.7 Stochastic Action Selection (QL, Q-Trace and P-Trace)

The "stochastic action selector" (SAS) used in in conjunction with the QL, Q-Trace and P-Trace algorithms in these experiments is based on that used in Lin (1992). To actively explore different actions in similar situations, action are chosen randomly according to a Boltzmann probability distribution:

$$Prob(a_i) = e^{q_i/T} / \sum_{k \in actions} e^{q_k/T}$$

where q_i is the current estimate of the Q-value of action a_i, and the temperature T adjusts the randomness of action selection.

In practice, the action selector was turned off every second learning trial within a run to exploit fully the learning that had occurred up to that point; if the trial ended in failure, the next trial would invoke the stochastic action selector to reinitiate exploration. This was done because QL seemed to have much more trouble learning the task without "interleaving" exploration and exploitation trials in this way.

[4] The term "trace" is adopted from Barto, Sutton & Anderson (1983).

[5] At first glance, these new algorithms might seem conceptually similar to Peng and Williams' $Q(\lambda)$-learning algorithm (Peng & Williams 1994) because of the presence of an "activity trace". However, the resemblance is superficial. The purpose and use of the trace in $Q(\lambda)$-learning and P-Trace are very different.

In $Q(\lambda)$-learning, the purpose of the trace is to generalise from one-step to multi-step temporal differencing as a means of credit assignment, directly analogous to the approach taken by Sutton's $TD(\lambda)$ (Sutton 1988). In P-Trace, the purpose of introducing the trace is not to *generalise* but rather to *replace* the TD credit assignment mechanism with a non-TD method. There is no TD learning occuring in P-Trace; further, the TD learning which is occurring in the TD/trace hybrid Q-Trace (which shares the same trace mechanism as P-Trace) is one-step TD, and is operating entirely without recourse to the trace mechanism.

It is worth noting that the performance of both P-Trace and Q-Trace may well have been assisted by this method as well. Reflecting upon how the trace mechanism is implemented (see section 4 and Appendix B for details), a "policy only" trial should allow for the greatest credit assignment, as the traces are effectively reset to zero upon a non-policy action being selected.

For the trials where the SAS was active, the temperature T was selected randomly from the range 0.0–0.1 each time the SAS was invoked. This meant that the randomness or "boldness" of exploration varied within a range at each step. Again, this was motivated by trial and error tuning to get QL to learn the task reliably, rather than by theoretical concerns.

For the sake of consistency, QL, Q-Trace and P-Trace all used the same SAS set-up throughout these experiments.

3.8 P-Trace

Conceptually, P-Trace is very simple; like QL it is trying to estimate the expected future discounted reward for an action a from state s, with respect to the current policy actions of the controller. Unlike QL, it does not rely on improving the estimate for $Q(s,a)$ by comparing it to the sum of the immediate reward received and the maximal estimated Q-value of all actions in the next state; rather, it waits for a terminal state to be reached, and then uses the sum of all time discounted reinforcement signals received since state/action pair (s,a) was tried to update the estimate for $Q(s,a)$.

The one trick involved is that all active "traces" are zeroed whenever a non-policy action is selected for the sake of active exploration. This means that a state/action pair only receives payoff information if policy has been followed all the way thereafter up to the terminal state being reached; otherwise, it would be learning the expected future discounted reward for a state/action with respect to a mix of the current policy and non-policy actions of the controller, which is not what is wanted.

In the simple case where no state/action pair is applied more than once prior to a terminal state being reached, we could apply the QL-like update rule

$$Q(s,a) := (1-\beta)Q(s,a) + \beta y \tag{1}$$

where, for P-Trace

$$y = \sum_{i=1}^{n} \gamma^{\kappa_i} r_i \tag{2}$$

where β is the learning factor, $\gamma < 1$ is the discount factor, κ_i is the number of time-steps between the time state/action pair (s,a) was visited and reinforcement signal r_i was received, and n is the total number of reinforcement signals received.

In the case where state/action pair (s,a) might have been visited several times before a terminal state is reached, things are slightly more complicated.

What we do conceptually is to keep a separate "trace" going for each time state/action pair (s,a) is visited. When the terminal state is reached, we update the $Q(s,a)$ estimate in a way that is equivalent to applying the update rule (1) n times, where n is the number of times state/action pair (s,a) was visited. For each application of the update rule to $Q(s,a)$ the value of y reflects the time since state/action pair (s,a) was visited for that trace.

In practice, keeping track of each separate trace in this way would lead to an unwieldy and inefficient implementation. We can simplify the update of the $Q(s,a)$ estimate to a one step application of rule (3)

$$Q(s,a) := (1-\beta)^n Q(s,a) + y \tag{3}$$

if y is defined as

$$y = \sum_{i=1}^{n} (1-\beta)^{n-i} \beta \sum_{j=i}^{n} \gamma^{k_{ij}} \rho_j \tag{4}$$

where n is the number of times the state/action pair has been visited prior to the terminal state being reached, and k_{ij} is defined as the number of time-steps between the i^{th} visit and the j^{th} visit. ρ_j is defined as the total discounted reward received between the j^{th} visit and the $(j+1)^{th}$ visit (or termination if $j = n$).

The sum y can be calculated incrementally each time the state/action pair is visited. Applying the "visit rules" in the pseudo-code in fig. 1 will do the job.

To show that (3) is equivalent to applying (1) n times for a separate trace value kept for each visit, we suppose $Q(s,a)_0$ is the value before the update procedure, and $Q(s,a)_n$ is the value after the n^{th} application of (1):

$$\begin{aligned}
Q(s,a)_1 &= (1-\beta)Q(s,a)_0 + \beta y_1 \\
\\
Q(s,a)_2 &= (1-\beta)Q(s,a)_1 + \beta y_2 \\
Q(s,a)_2 &= (1-\beta)((1-\beta)Q(s,a)_0 + \beta y_1) + \beta y_2 \\
Q(s,a)_2 &= (1-\beta)^2 Q(s,a)_0 + (1-\beta)\beta y_1 + \beta y_2 \\
\\
Q(s,a)_3 &= (1-\beta)Q(s,a)_2 + \beta y_3 \\
Q(s,a)_3 &= (1-\beta)^3 Q(s,a)_0 + (1-\beta)^2 \beta y_1 + (1-\beta)\beta y_2 + \beta y_3
\end{aligned}$$

and so on. The above expansion generalises to

$$Q(s,a)_n = (1-\beta)^n Q(s,a)_0 + \sum_{i=1}^{n} (1-\beta)^{n-i} \beta y_i$$

```
while not terminal state do

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

    if non-policy action selected then               (* zero traces *)
        for all (i,j) such that VisitCount_ij > 0 do
            VisitCount_ij := 0
        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              (* steps since last visit *)
        x_sa := (1 − β) x_sa γ^k + β
    endif

    ρ_sa := 0
    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 + r γ^k
        endfor
    endif

    GlobalClock := GlobalClock + 1

endwhile

for all (s,a) such that VisitCount_sa > 0 do         (* terminal state reached *)
    n := VisitCount_sa
    Sum_sa := (1 − β)Sum_sa + x_sa ρ_sa
    Q_sa := (1 − β)^n Q_sa + Sum_sa                  (* update rule (3) *)
endfor
```

Fig 1: Pseudo-code for the "visit rules" to calculate (3) incrementally.[6]

[6]The variables *Sum*, *x*, *ρ*, *StepCount* and *VisitCount* are kept separately for each state/action pair, as they are associated with a particular Q-value. The subscripts identify to which state/action pair the variable belongs. Also note that the pseudo-code in fig.1 is mainly for illustrative purposes. For the basis of an implementation the more extensive pseudo-code in Appendix B may be more useful.

Now, since y_i is the total discounted reward received between the i^{th} visit and termination, we can write the expression

$$y_i = \sum_{j=i}^{n} \gamma^{k_{ij}} \rho_j$$

for all y_i in a series of updates. k_{ij} is defined as the number of time-steps between the i^{th} visit and the j^{th} visit. ρ_j is defined as the total discounted reward received between the j^{th} visit and the $(j+1)^{th}$ visit (or termination if $j = n$).

We can recast the above as

$$Q(s,a)_n = (1-\beta)^n Q(s,a)_0 + y$$

where y is defined as

$$y = \sum_{i=1}^{n} (1-\beta)^{n-i} \beta \sum_{j=i}^{n} \gamma^{k_{ij}} \rho_j$$

which is (3) and (4) as required.

Next, we show that the "visit rules" above do indeed calculate the value of y in equation (4). Since k_{ij} is defined as the number of time-steps between the i^{th} visit and the j^{th} visit, we can write the expression

$$k_{ij} = \sum_{m=i+1}^{j} t_m$$

where t_m is the number of time-steps between the $(m-1)^{th}$ visit and the m^{th} visit. This is useful because it allows us to expand (4) to

$$\begin{array}{lllllllllll}
(1-\beta)^{n-1} & (\beta\rho_1 & + & \beta\gamma^{t_2}\rho_2 & + & \beta\gamma^{(t_2+t_3)}\rho_3 & + & \cdots & + & \beta\gamma^{(t_2+\cdots+t_n)}\rho_n) & + \\
(1-\beta)^{n-2} & (& & \beta\rho_2 & + & \beta\gamma^{t_3}\rho_3 & + & \cdots & + & \beta\gamma^{(t_3+\cdots+t_n)}\rho_n) & + \\
(1-\beta)^{n-3} & (& & & & \beta\rho_3 & + & \cdots & + & \beta\gamma^{(t_4+\cdots+t_n)}\rho_n) & + \\
 & & & & & & & & & \vdots & \\
(1-\beta) & (& & & & & & \beta\rho_{n-1} & + & \beta\gamma^{t_n}\rho_n) & + \\
 & (& & & & & & & & \beta\rho_n) &
\end{array}$$

Rearranging the above sum by column rather than by row, we get

$$\begin{aligned}
\sum_{i=1}^{n} (i=\beta)^{n-i}\beta y_i \quad = \quad & \rho_1((1-\beta)^{n-1}\beta) + \\
& \rho_2((1-\beta)^{n-1}\beta\gamma^{t_2} + (1-\beta)^{n-2}\beta) + \\
& \rho_3((1-\beta)^{n-1}\beta\gamma^{(t_2+t_3)} + (1-\beta)^{n-2}\beta\gamma^{t_3} + (1-\beta)^{n-3}\beta) + \\
& \vdots \\
& \rho_n((1-\beta)^{n-1}\beta\gamma^{(t_2+\cdots+t_n)} + \cdots + \beta)
\end{aligned}$$

which can be compactly written as

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

where x_i is defined as

$$x_i = \sum_{j=1}^{i}(1-\beta)^{n-j}\beta\gamma^{\sum_{m=j+1}^{i} t_m}$$

As n increases, we note that the sum $\sum_{i=1}^{n}\rho_i x_i$ can be incrementally recalculated as

$$Sum := (1-\beta)Sum + \rho_n x$$

whereupon x is then incrementally recalculated according to the rule

$$x := (1-\beta)x\gamma^{t_n} + \beta$$

with $Sum = 0$ and $x = \beta$ as the initial values. These incremental recalculation rules translate directly to the "visit rules" specified in Fig 1.

Finally, to address an implementation consideration: With the update rules in their simplest form as described above, the worst-case size of a list to keep track of all the visited state/action pairs up until a terminal state is reached is the entire number of states in the system. If this is implementationally unfeasable or otherwise unacceptable, the worst-case size can be set to a smaller fixed value n, where $\epsilon = \gamma^n$ is predecided to be small enough that the total discounted reward for any state/action pair last visited more than n time-steps ago will be so close to zero that the difference doesn't matter.[7]

In this case, the algorithm need only to be modified slightly; a list is kept of state/action pairs visited up until a terminal state is reached. If a state/action pair upon being visited is not currently on the list, it is put on the end of the list. If a state/action pair upon being visited is already on the list, it is moved to the end of the list. If the list grows to the length $n+1$, the state/action pair at the head of the list is removed, and update rule (3) is applied to it with y equal to the partial discounted reward accumulated so far, without waiting for a terminal state to be reached. In this way the list will be maintained to a maximum preset fixed length, and as long as $\epsilon = \gamma^n$ is chosen to be sufficiently small, there should be no significant effect on the convergence properties of the trace method.

3.9 Q-Trace

Q-Trace is the same as P-Trace except that it also performs a TD update to the Q-value of the most recently tried state/action pair using the standard QL update procedure

$$Q(s,a) := (1-\beta)Q(s,a) + y \tag{5}$$

[7] Peng & Williams (1994) propose an analogous modification for their $Q(\lambda)$-learning algorithm.

where, for QL

$$y = \beta(r + \gamma \max_b Q(s', b)) \tag{6}$$

where s' is the state immediately following action a from state s, β is the learning factor, $\gamma < 1$ is the discount factor and r is the reinforcement signal received (if any) immediately following action a from state s.

Q-Trace is therefore supplied with two separate channels of information to provide updates for its Q-table: A series of direct values from the trace mechanism, and a series of indirectly derived values via the QL TD method. It should also be clearer why the trace mechanism described above has been carefully set-up to only allow "policy action" feedback; in the case of Q-Trace, it is important the two channels of information "agree" in the sense that they are actually estimates of the same thing.

To help clarify the relationship between the algorithms: Q-Trace with the TD described above disabled becomes P-Trace; Q-Trace with the trace mechanism disabled is QL. The source code in Appendix B may be helpful in making the above more explicit.

4 Results

The results of the experiments comparing the performance of five learning algorithms in the noisy pole-and-cart domain are presented in both tabular and graphical formats.

4.1 Noise model 1:

Below are the tables of results for noise model 1. The left-most column indicates the noise level as a percentage value; each row holds the results for a noise level within a noise model. "BX" denotes BOXES, "AH" stands for AHC, "QL" is Q-Learning and "QT" is for Q-Trace. "PT" is for P-Trace.

The columns in table 1a hold the mean robustness scores for each algorithm averaged over 20 learning runs conducted at each noise level. It should be interpreted as a score out of 20, as there are twenty "robustness trials" conducted at the end of each run to test the quality of the learnt control rules.

The remaining tables 1b – 1d show the median, mean and standard deviation for learning rate for each algorithm at each noise level. We include the median as well as the arithmetic mean because of the skewing effects of a relative handful of runs, particularly in the case of AHC. It is a good idea to compare the mean and the median in conjunction with the standard deviation to build up a reasonable picture of what is going on with respect to a measure of central tendency; comparing mean learning times alone can be misleading.

Using the methodology and terminology of Sammut (1988), learning rate is measured by the number of trials required to learn the control task for a run.[8]

[8] This makes more sense in the case of BOXES, where learning only occurs at the end of each trial, whereas for a TD learner it might make more sense to measure learning rate in terms of the total number of time steps, since learning is a more incremental process. However, we use this measure for the sake of consistency with earlier studies.

A trial commences with the pole and cart in a randomised (but recoverable) position in the state space, and continues until either the system is deemed to have failed (state values exceed certain predefined limits) or the system has been successfully controlled without failure for a specified amount of (simulated) time. A trial that ends in success marks the end of the learning phase of a run; the control rules are "frozen" at this point and the robustness trials mentioned above commence. A "robustness trial" is scored 0 if it ends in system failure (as described above), or 1 if it ends with the system having avoided failure for the specified amount of time.

Tabular results for noise model 1:

Noise	Mean Robustness Score				
(%)	BX	AH	PT	QT	QL
0	2.9	9.6	12.7	11.2	10.1
1	4.8	9.7	10.7	12.0	8.1
2	4.3	8.1	9.6	10.6	11.9
3	3.8	7.0	7.2	8.6	10.1
4	3.3	4.9	8.1	9.9	6.8
5	4.3	4.7	4.5	4.3	6.3
6	2.2	2.8	2.4	5.1	2.0
7	1.1	1.6	1.6	1.6	1.0
8	0.4	0.4	0.7	1.1	0.4

Table 1a

Noise	Median Trials				
(%)	BX	AH	PT	QT	QL
0	75	49	100	143	329
1	99	54	101	155	415
2	77	57	161	149	513
3	95	69	153	235	735
4	126	73	178	348	1549
5	183	92	437	443	2697
6	179	81	631	1039	12535
7	351	236	1002	2843	51509
8	834	360	16630	6131	436857

Table 1b

Noise	Mean Trials				
(%)	BX	AH	PT	QT	QL
0	76.6	56.3	160.1	166.5	403.6
1	114.2	61.5	186.8	185.1	503.0
2	100.2	57.9	238.0	183.6	690.9
3	127.8	137.2	314.3	240.5	1080.6
4	142.9	100.9	400.6	360.0	1905.0
5	209.1	473.3	1143.6	636.2	7949.3
6	218.1	125.5	1462.0	1101.5	22937.5
7	456.2	58001.6	1802.7	3228.4	97301.6
8	1374.0	1900.9	27161.2	14757.1	867567.4

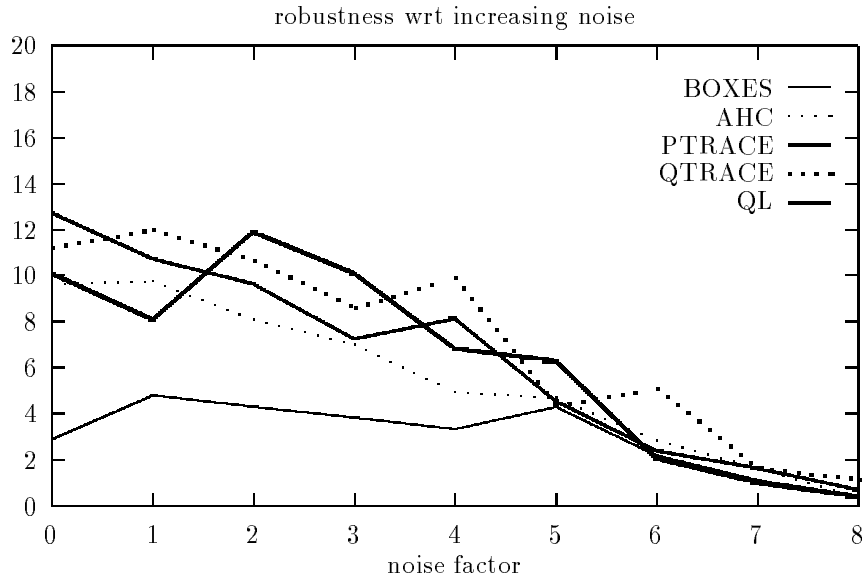
Table 1c

Noise (%)	Std dev Trials BX	AH	PT	QT	QL
0	24.2	20.6	123.9	82.7	184.3
1	59.4	36.4	152.8	83.5	267.3
2	55.7	16.3	163.8	76.0	504.6
3	88.4	252.0	394.7	117.8	688.9
4	63.0	79.7	469.1	133.8	1484.3
5	103.7	857.5	1400.4	377.6	8576.8
6	126.2	111.4	1479.0	548.0	18037.4
7	285.8	251354.0	1562.9	1762.3	95836.6
8	1114.0	5970.7	28100.1	15792.5	845976.0

Table 1d

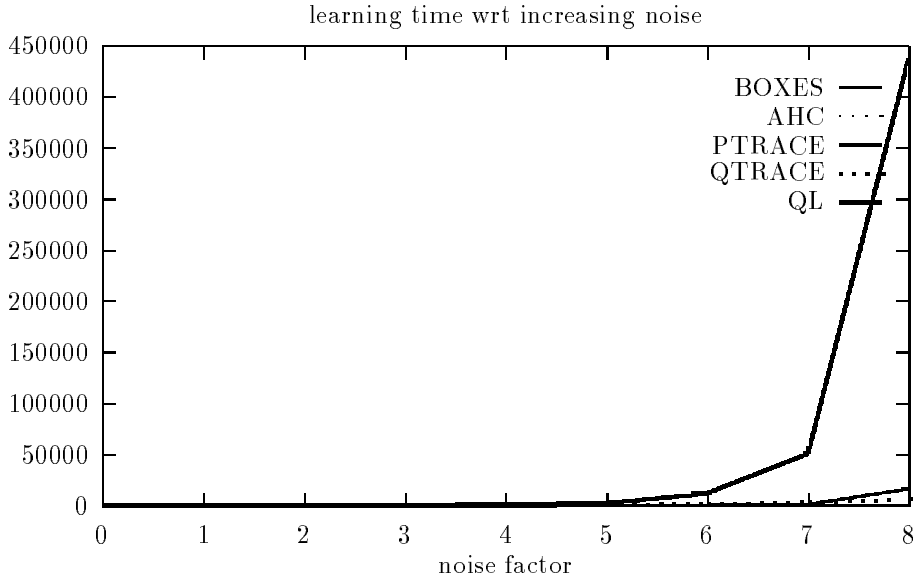
Next, we present the values in this table in a graphical format as a series of plots. (For some of these values, we have included a second plot involving only four algorithms, omitting the values for QL where problems of differing orders of magnitude saw the QL learning rates "swamp" the plot, obscuring the relationships between the other algorithms.)

Noise model 1: Mean robustness (out of 20) wrt to increasing noise factor (%):



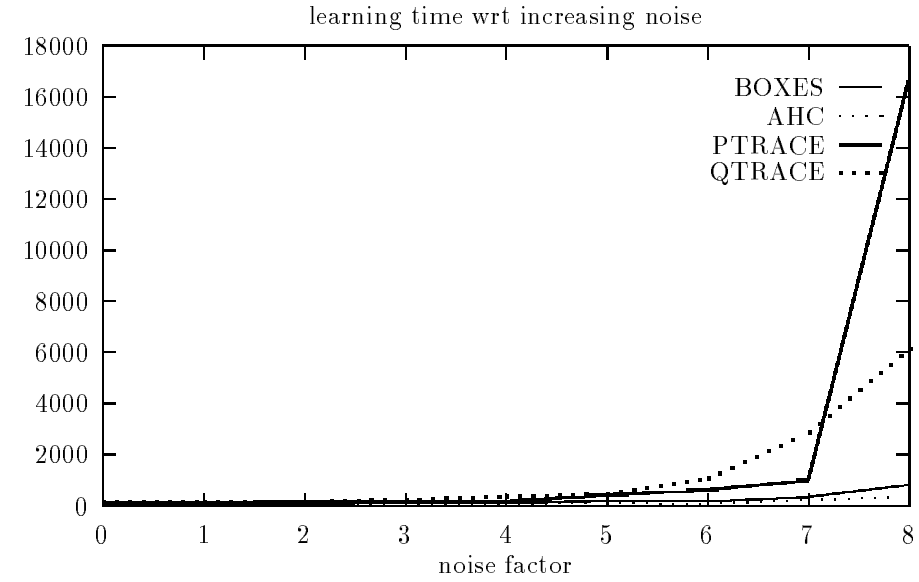
Plot 1.1

Noise model 1: Median learning time (trials) wrt to increasing noise factor (%):



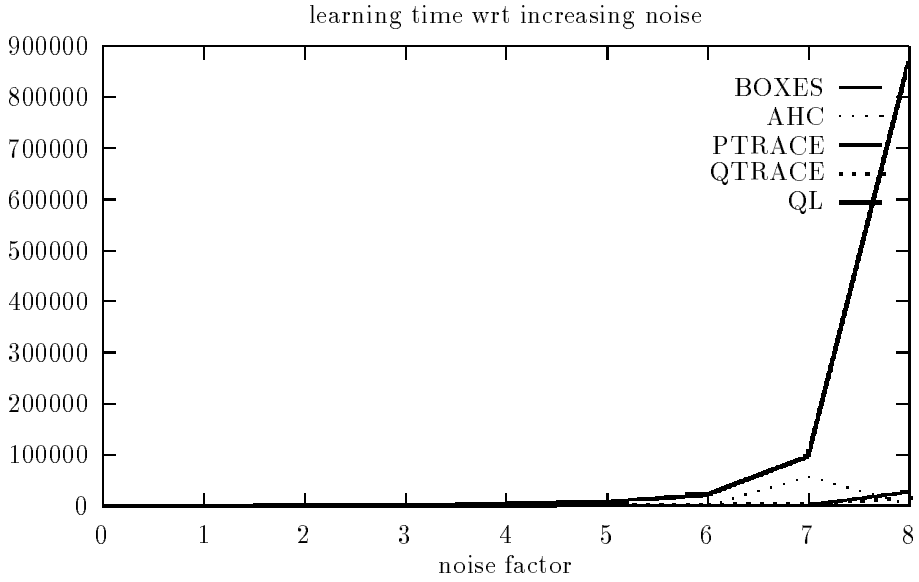
Plot 1.2

Noise model 1: Median learning time (trials) wrt to increasing noise factor (%):



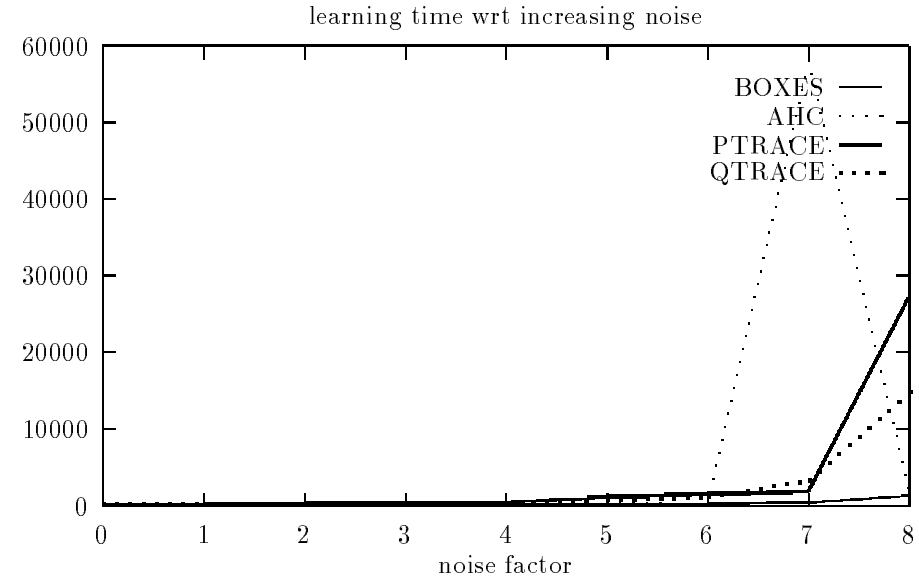
Plot 1.3 (same as plot 1.2 above, but without showing QL. Note different orders of magnitude on y-scale.)

Noise model 1: Mean learning time (trials) wrt to increasing noise factor (%):



Plot 1.4

Noise model 1: Mean learning time (trials) wrt to increasing noise factor (%):



Plot 1.5 (same as plot 1.4 above, but without showing QL. Note different orders of magnitude on y-scale.)

4.2 Noise model 2:

The format of the presentation of results for noise model 2 is the same as for noise model 1 (see above).

Tabular results for noise model 2:

Noise	Mean Robustness Score				
(%)	BX	AH	PT	QT	QL
0	3.7	7.9	13.3	11.8	11.9
1	3.9	8.1	10.5	13.3	11.9
2	5.4	9.6	10.0	10.4	10.0
3	3.8	10.9	6.3	10.9	9.6
4	2.8	7.6	6.3	7.3	10.2
5	3.4	5.5	8.0	6.9	8.9
6	3.6	3.4	3.6	5.6	5.2
7	2.7	1.9	3.3	5.2	3.3
8	2.8	1.5	2.5	4.2	2.7

Table 2a

Noise	Median Trials				
(%)	BX	AH	PT	QT	QL
0	74	47	99	129	359
1	77	48	107	180	395
2	85	65	118	180	743
3	100	83	169	246	1135
4	108	70	180	254	1837
5	111	91	308	340	5713
6	149	103	565	525	13651
7	183	149	1763	1016	102683
8	187	255	3447	3709	753183

Table 2b

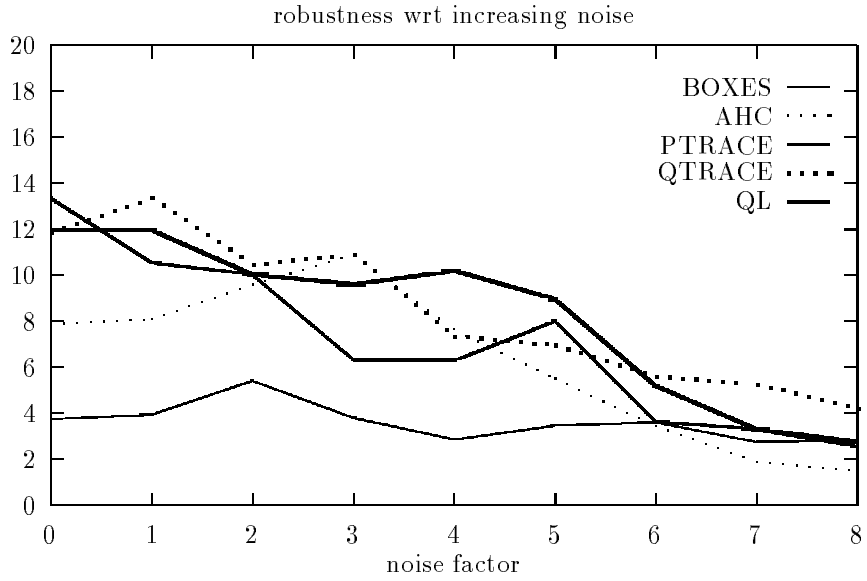
Noise	Mean Trials				
(%)	BX	AH	PT	QT	QL
0	75.6	48.3	182.9	159.1	416.9
1	101.9	51.6	252.2	222.7	495.3
2	86.3	75.3	201.7	178.7	715.0
3	107.9	123.6	212.4	262.9	1537.8
4	110.1	87.3	377.1	302.5	3205.5
5	128.1	94.7	373.5	385.4	10484.5
6	191.9	113.4	1129.9	677.4	26978.5
7	172.9	3781.0	3209.7	1432.1	185714.3
8	235.5	2550.8	5800.7	4860.8	927381.4

Table 2c

Noise (%)	Std dev Trials BX	AH	PT	QT	QL
0	28.0	17.1	198.8	110.7	297.5
1	75.2	13.3	331.8	104.5	298.1
2	48.2	40.4	178.6	83.9	287.6
3	51.2	139.1	141.1	101.7	1636.3
4	39.9	52.7	364.2	176.1	2634.4
5	77.3	32.1	335.4	230.4	12121.8
6	134.3	55.5	1103.8	446.4	27129.3
7	81.8	15551.4	2961.2	1353.0	144506.9
8	157.5	9372.7	5268.8	3615.1	747734.5

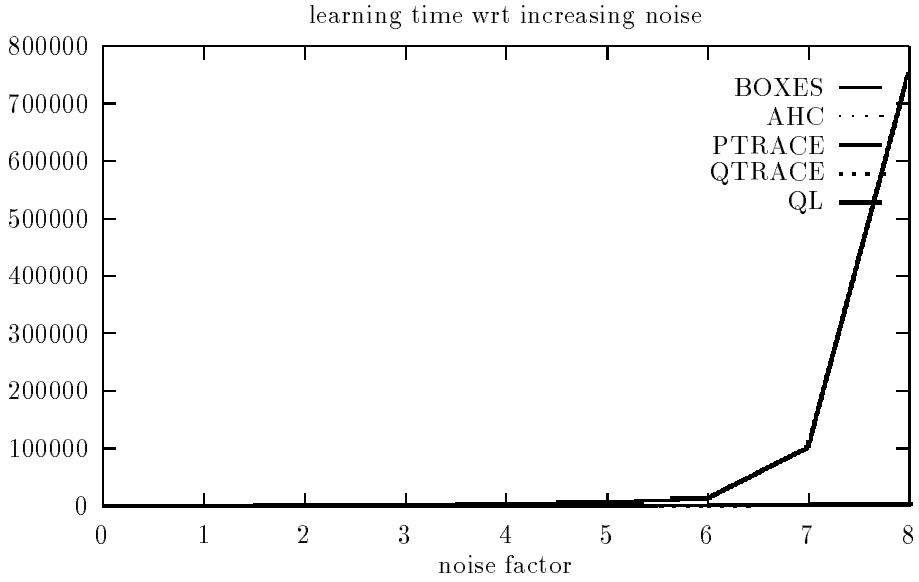
Table 2d

Noise model 2: Mean robustness (out of 20) wrt to increasing noise factor (%):



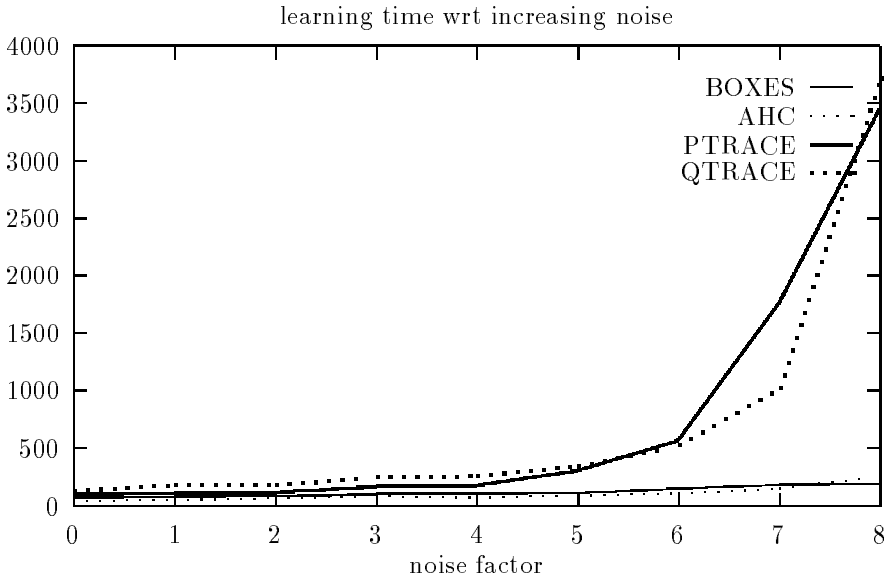
Plot 2.1

Noise model 2: Median learning time (trials) wrt to increasing noise factor (%):



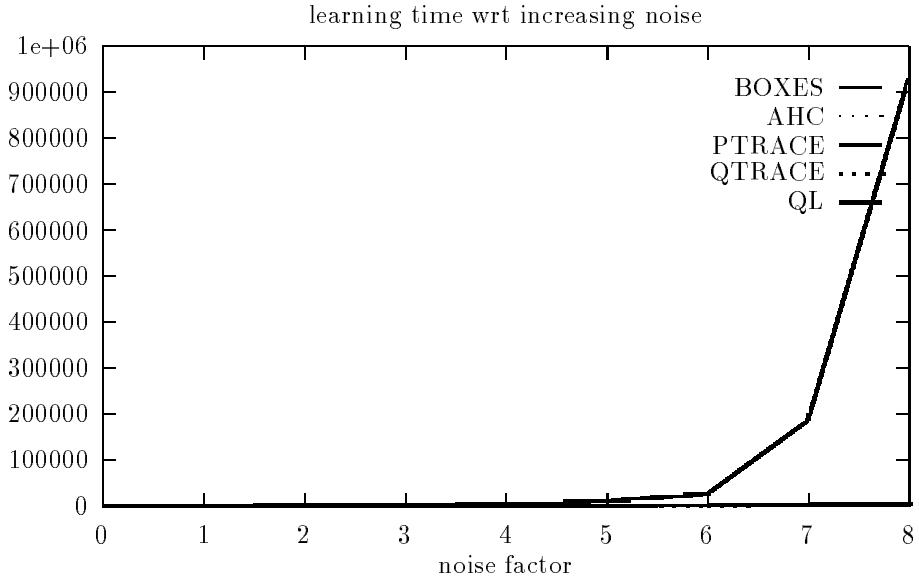
Plot 2.2

Noise model 2: Median learning time (trials) wrt to increasing noise factor (%):



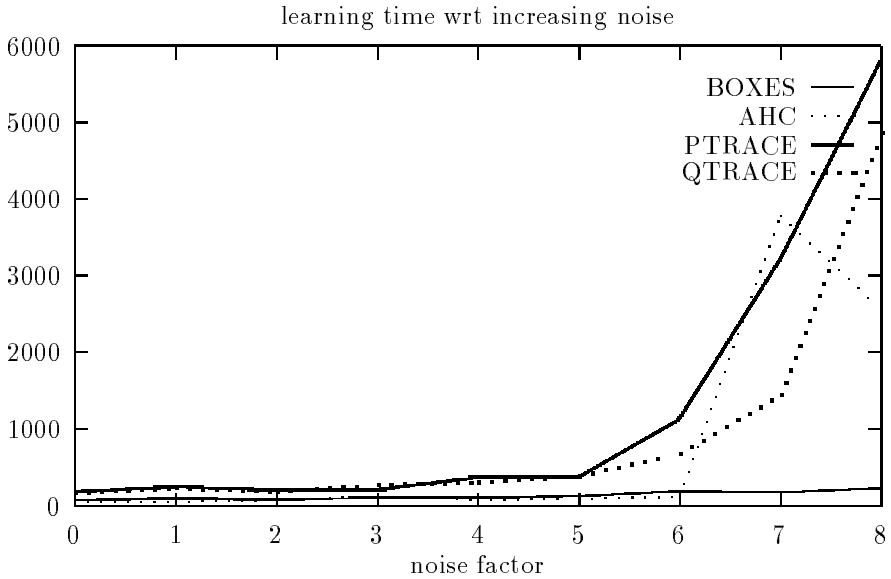
Plot 2.3 (same as plot 2.2 above, but without showing QL. Note different orders of magnitude on y-scale.)

Noise model 2: Mean learning time (trials) wrt to increasing noise factor (%):



Plot 2.4

Noise model 2: Mean learning time (trials) wrt to increasing noise factor (%):



Plot 2.5 (same as plot 2.4 above, but without showing QL. Note different orders of magnitude on y-scale.)

5 Discussion

5.1 Experimental Results

First, it is noteworthy that for all 1800 runs performed (20 runs x 9 noise levels x 5 algorithms x 2 noise models) the balancing task was learnt eventually in every case; that is, there are no "missing values" in the data above. It was certainly far from obvious while running the experiments that this would turn out to be the case; in the case of QL and some runs of AHC, a single run would take millions of trials to learn the task, corresponding to days of CPU time on a workstation class machine. In some respects, the fact that the runs resulted finally in learning the task is more puzzling than if they had showed signs of getting stuck in local minima. It would appear that Minsky's "mesa-effect" (Minsky 1961) is perhaps a better explanation than that of local minima, to touch on an old debate.

The results for noise model 1 and noise model 2 share many consistent features. For both models, we observe that QL and Q-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 $< 5\%$. Overall AHC and P-Trace fell between BOXES and the Q-Learners in robustness performance, although it is noteworthy that P-Trace did return the highest scores at the 0 noise level for both noise models, and was closer in general to the performance of the Q-Learners than to that of AHC. Notice that at the higher noise levels, however, there is a general convergence towards zero with respect to robustness scores over all algorithms.

It should be commented upon that runs past 8% were not practical because of the very long learning times for some runs of QL and AHC as mentioned above; although some runs in this range completed relatively quickly (taking perhaps only a few minutes of computing time), other runs would take days to complete on a fast workstation class machine, and both AHC and QL seemed to "hit a brick wall" so to speak on some runs after the 8% level. In contrast, BOXES, P-Trace and Q-Trace were much more consistent and predictable learners in this respect, as can be seen by looking at the respective variances in the tables above.

This extreme sensitivity to initial conditions with respect to learning times, particularly in the case of AHC, is quite surprising; and, as far as the author knows has been hitherto largely unsuspected.[9] Whether this is genuinely chaotic behaviour, or there is a more mundane explanation, deserves further investigation. It would seem fair to suggest that this behaviour could be a serious potential problem if predictability and reliability of learning times were considered to be at all important factors in choosing either of these algorithms for noisy domains.

As regards learning rates, QL is clearly the overall big loser, and increasingly

[9] The only hint of this sort of behaviour might be found in the results of Sammut (1988).

so as noise level increased. At the 8% noise level, QL was about 30x slower on average than P-Trace and 60x slower than Q-Trace for noise model 1, and about 200x slower than both for noise model 2. QL was in general three orders of magnitude slower than BOXES at this noise level. BOXES was shown to be a very consistent learner as noise levels increased. AHC, although its median learning rates were very comparable to BOXES as noise level increased, had large differences in mean learning rates, as the means were wildly skewed by a handful of runs that took inordinate learning times. From a descriptive statistics point of view, this could almost be viewed as a "two population" effect, although there would seem to be no justifiable methodological reason to treat these as outliers and not include them in the averages.

Of the algorithms above, the most similar to QL is Q-Trace, and yet their performance, particularly under noisy conditions, are quite distinct. Q-Trace exhibits both faster and more predictable learning rates, and equal or perhaps marginally better quality of learning judging from the results of the "robustness" trials. Why?

Before we attempt to answer this, some insight into the motivation for the design of the P-Trace and Q-Trace algorithms may be helpful. In the first version of these experiments, only the BOXES, AHC and QL algorithms were trialled. What quickly became apparent were different weaknesses for each of these algorithms, resulting in no clear winner:

a) The extreme slowness in learning and sensitivity with respect to noise QL exhibited in contrast to the other algorithms.

b) The relatively poor quality of learning or "robustness" scores exhibited by BOXES.

c) The wildly varying learning rates exhibited by AHC for different initial conditions, particularly as noise levels increased.

Additionally, formal convergence results for either AHC or BOXES to date have not been forthcoming, in contrast to the situation with QL. This made QL's poor practical showing even more disappointing, particularly as the conventional wisdom is that QL should be particularly well suited to stochastic or noisy environments.

Being both TD algorithms, *prima facie* it would appear that AHC and QL are closer cousins than AHC and BOXES, yet the performance of AHC and BOXES with respect to learning rates are much closer. The one thing that AHC and BOXES do have in common that is absent in QL is a direct trace mechanism for credit assignment. AHC is a hybrid in this respect; it incorporates both the direct trace mechanism of a "pure" trace learner, such as BOXES, and a TD mechanism of a "pure" TD learner, such as QL. We hypothesised that the absence of a direct trace mechanism in QL was part of the reason why QL was performing so badly with respect to learning rate, particularly under noisy conditions.

The quality of learning or "robustness" scores for BOXES left much to be desired when compared to its TD rivals, however. The design of a new algorithm was motivated by the idea that a direct trace mechanism be reintroduced to QL to possibly improve learning rate, particularly under noisy conditions, but to preserve the relatively good quality of learning that could be associated with

the TD learners. An additional design goal was to introduce the trace in such a way that will also preserve the "provably correct" properties associated with pure QL, if possible. Three plausible strategies were considered: a) adding TD to BOXES, b) simplifying the already hybrid AHC in such a way to make it mathematically more tractable and at the same time perhaps sorting out some of its unpredictability, c) adding a trace mechanism to QL. Upon consideration, this last approach was deemed the most straightforward, and the result is Q-Trace.

Two things became immediately apparent with Q-Trace; firstly, that learning times were vastly improved over "pure" QL, and secondly that this did not seem to be at the expense of learning quality (if anything learning quality might be slightly better.) However, the learning rates were still significantly slower for Q-Trace than for BOXES or (median) AHC rates, particularly at the higher noise levels. This led to the speculation that perhaps in general a "pure" trace learner might learn faster than a hybrid or pure TD learner, but with reduced quality of learning (analogous to a too fast annealing process). P-Trace (which is simply Q-Trace with the TD learning disabled, making it a "pure" trace learner) was introduced into the experiments to help investigate this possibility.

Interestingly, the speculation above appears to be only half right, at best. The prediction was that P-Trace would show faster learning times at the expense of a decrease in learning quality, with respect to Q-Trace. In practice, the learning quality did decrease slightly (although P-Trace was still exhibiting far superior robustness scores over BOXES), but the learning rates overall remained very similar to those of Q-Trace.

So it would seem that to explain the differences in learning rates and robustness scores of BOXES in comparison to the other algorithms, an explanation besides TD learning in itself must be proposed. So what is left? Scrutiny of P-Trace and BOXES might suggest that the exploration/exploitation action selection heuristics, which are quite different for both algorithms, might possibly be the critical factor. This remains speculation for the moment, however.[10]

Returning now to the question of why should there be such a discrepancy between QL and Q-Trace in their learning rates, we can see why it might make more sense to phrase this in a more general way: Why is there such a discrepancy between the learning rates of the trace learners as opposed to a pure TD learner in this domain? Why does it appear that trace learning is so much more effective a method of credit assignment? We attempt to at least partially answer this question in the discussion in the following subsection.

5.2 Theoretical Issues

There are several possible reasons that may help to explain the discrepancy in learning rates between the pure TD learner (QL) and the algorithms employing trace mechanisms for credit assignment. Firstly, and most obviously, a trace

[10]Lin (1992) makes the comment that in his experience the "Boltzmann distribution" stochastic action selectors we have been using here for QL, Q-Trace and P-Trace are not particularly good performers in terms of effectively trading off exploration and exploitation.

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 one-step QL. The slow backwards propagation of credit that is associated with QL is thus potentially accelerated. It is worth remembering that in TD learning in general, there is potentially much "churning" of estimate values with little or zero information content in the earlier stages of learning. Trace learning gets some information to all visited states in one step.[11]

Other, subtler reasons may also help to explain these results. One possibility is that we are witnessing what might be called a TD "chinese whisper" effect.[12] Since information is potentially propagated through a long series of intermediaries to a state/action pair that is temporally distant from direct reinforcement signals, we may be seeing a TD analogue of this phenomenon. Information that is passed through many intermediaries in a noisy environment may be prone to a cumulative noise effect. Eventually, the signal may be swamped by noise. Although there has not been a rigorous mathematical investigation of this possibility at this stage, it would seem to be an idea worth following up. If this speculation has any merit, the implications for TD learning in a noisy environment could be quite important.

There may be yet 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 quite influenced by which immediately preceding states have been visited, particularly if the chunks are large.[13] That is to say, selecting action n from box k being fed from box i might

[11] It is worth noting that Lin's technique of "experience replay" (Lin 1992) is significantly less effective in speeding learning for QL if the reinforcement signals are negative in sign. This, of course, is the case for the pole-and-cart problem, but the reason applies quite generally; the back-propagation of the effect of a reinforcement signal must stop once the Q-max value for a state is unchanged, and negative values will in general have less influence in causing change to the Q-max values.

To illustrate the effect, consider the following special case: If all Q-values are initially set to zero, as is the case in QL (Watkins 1989), what is the effect of the first reinforcement signal if Lin's replay method is applied? If the reinforcement is positive in sign, the effect of the signal will be propagated in some degree all the way back down the line to every state/action pair that led up to the reinforcement signal. This is fine; Lin's method in this case has the equivalent action of a trace mechanism. But if the reinforcement signal is negative in sign, "experience replay" accomplishes nothing; the Q-max value will remain unchanged at zero for the penultimate state, and the back-propagation stalls at this point.

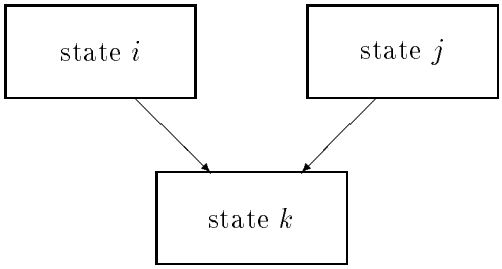
Overall, for the purposes of speeding QL, we should expect Lin's "experience replay" method be less effective than a trace mechanism. Certainly this is not contradicted by trying his method with the pole-and-cart problem, which not surprisingly (in hindsight) proved to be almost totally ineffective.

[12] The (almost certainly apocryphal) WWI story of the message "send reinforcements; we are going to advance" being garbled to "send three and fourpence; we're going to a dance" by the time the message was relayed to headquarters is commonly used to exemplify the effect.

[13] Watkins (1989) agrees with this analysis of the pole-and-cart problem.

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 trace 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.[14]

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 estimate 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.

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, it should be clear from this example the important general advantage the trace learner has in non-Markovian domains: it can take into account biasing effects of preceding states in assigning credit if necessary.

QL and other pure TD methods cannot do this without resorting to explicitly re-representing the original states and their preceding state histories in a combinatorial explosion of new states.[15] This would be potentially disastrous both in terms of representation space requirements and the resulting slow-down in learning due to the many more states that have to be explored for effective credit assignment to occur. Realistically, although theoretically possible, this

[14] A similar construction is used for didactic purposes in Singh, Jaakkola and Jordan (1994). Their analysis of the limitations of TD(0) learning is essentially the same.

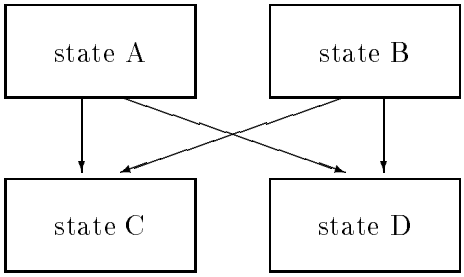
[15] Although TD algorithms that learn from more than one subsequent estimate such as $TD(\lambda)$ (Sutton 1988) with $\lambda > 0$ would be expected to cope with non-Markovian domains progressively better as $\lambda \rightarrow 1$, it should be clear that for all $\lambda < 1$, $TD(\lambda)$ should not be expected to perform as well as a pure trace learner in a non-Markovian domain. The above argument generalises fairly straightforwardly to include these cases.

approach would be out of the question for almost all conceivable non-trivial practical applications. We can conclude that if the domain is not at least reasonably well-modelled by Markovian processes, we should have little reason to expect QL and related pure TD learners to perform well.[16] The next section introduces some empirical support for this claim.

6 Some experimental results comparing four RL algorithms in a non-Markovian domain.

To empirically test some of the theoretical ideas introduced in the previous section, we perform a simple experiment involving an artificially constructed non-Markovian domain.

Consider the following four state non-Markovian domain:



States A and B are are the two possible starting states, which are non-terminating, in the sense that actions taken from A or B will always lead to a state transition to another state rather than to termination. States C and D are terminating states, as the actions from these states will always lead to immediate termination. There is always a reinforcement signal associated with termination in this domain.

There are two possible actions available to choose from in each state; the state transition rules as follows:

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

Because the domain is non-Markovian, the reinforcement received from an action made in state C or D may depend not only upon what is 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:

[16] Watkins (1989) mentions that a Markov learner devised by Barto, Sutton & Anderson (1983) to compare with AHC returned "disappointing results" when applied to the cart-and-pole problem.

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 in state A or B (0.5 probability of starting in one over the other). Learning takes place over 10,000 trials.

Results. For QL, Q-Trace and P-Trace a simple "Boltzmann distribution" SAS as described in Lin (1992) was used with temperature T set to 1.0. The average payoff per trial for four algorithms[17] is shown below (each row represents a separate run of 10,000 trials for each algorithm using a different seed for the pseudo-random number generator).

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

Discussion. 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

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

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. 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, the actual average per-trial payoff we would expect to be less than for strategies based on perfect information. This is observed in the data tabled above.

Consistent with the analysis in the discussion section 6.2 of this paper, of the algorithms trialled, P-Trace, the pure trace learner, was able to cope with this non-Markovian domain best, while QL, the pure TD learner fared worst.

Q-Trace and AHC, as hybrid trace/TD learners, were expected to fall between the pure trace and pure TD learners in their performance, which they did; but interestingly 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. Again, a "two population" effect emerges. Also note the anomalous result for AHC with the random seed set at 4. The number 0.999790 indicates that AHC did not try any other than the theoretical optimal policy more than 21 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. At the very least we can conclude that AHC shows itself again to be the least predictable in terms of its performance.

The performance of all the algorithms might be contrasted with the expected outcome for a purely random agent as a baseline result; since all eight reinforcement outcomes are equally likely with random action selection, the expected per-trial outcome for a purely random agent should be simply the average value of all possible reinforcement outcomes, which is –0.725. All do significantly better than a random agent would. The worst-case per-trial average reward for a policy in this domain is –2.0, which provides a theoretical lower bound for the results.

7 Conclusions

Several important issues have been brought to light in the course of these experiments, not the least of which is the fact that some of the best-known credit-assignment algorithms currently in use in reinforcement learning research have exhibited potentially serious problems in coping with noise. QL in particular becomes a disastrously slow learner when faced with noise.

While BOXES is a fast learner and proved to be relatively resilient to noise at the levels tested, its learning quality was shown to be generally inferior to

that of the other algorithms tested.

AHC exhibits a strange, almost chaotic unpredictability with respect to learning rate that seems to be more pronounced in the presence of noise. So while the median learning rates for this algorithm are very good, better even than BOXES, the mean learning rates tell a quite different story. Whether further tuning of the various algorithmic parameters could settle this behaviour down remains speculation at this point. This highlights a practical problem generally associated with a complex algorithm that is heavily dependent upon heuristic methods; if such an algorithm is difficult to understand even when it appears to be working well, it can be difficult to know where to start to fix it when it appears to be not working so well.

This issue has been the driving force behind the work into "provably correct" learning algorithms, of which QL is one of the most intensively studied at the moment. It is suggested that Q-Trace and P-Trace provide a general improvement on this algorithm by reintroducing the old concept of the "learning trace", a concept that became unfashionable as TD learning became the principal focus of research of the RL community in general. In the light of the foregoing results, it is argued that it is entirely premature to abandon the concept just yet.

What distinguishes Q-Trace and P-Trace from other trace learners is that the trace is designed in such a way as to preserve the convergence results that apply to QL for the special case of learning in a pure Markovian environment. Thus, in this special sense, "correctness" should be preserved, which of course is one of the major appeals of QL over both AHC and BOXES, for which no known equivalent theoretical results exist.

Q-Trace and P-Trace also achieve their significant performance increases in learning rate without having to trade-off quality of learning. In these experiments, the quality of Q-Trace's learning is not significantly different to that of QL's, perhaps it is even marginally superior, whereas its average learning rate ranges from 2-3x faster on average in the absence of noise to approximately 200x faster in the case of 8% noise for noise model 2. P-Trace has similar learning rates to Q-Trace, and only marginally poorer learning quality for this domain overall.

Analysis also suggests that there are advantages inherent in P-Trace and Q-Trace (and in "memory trace" algorithms for credit assignment generally) over pure TD algorithms when learning is to take place in environments that are not well-modelled by Markovian processes. This last point is potentially the most important of all. If this is true, and we also now have some preliminary experimental work which strongly supports this analysis, then we have the situation where the trace algorithms are not simply better performers within the class of problems that TD learners can learn, but they are *strictly more powerful* learners than the TD learners, in that they can learn a class of problems which is a strict superset of the class of problems TD learners can learn.

In summary, key issues relating to reinforcement learning in noisy environments remain open questions, and extensive further investigation in this area is required. However, we now have arguments and evidence that "trace" learners may prove to be both faster and more powerful learners in general than their

TD counterparts. The potential performance improvements using trace over pure TD methods may turn out to be particularly important when learning is to occur in noisy or stochastic environments, and in the case where the domain is not well-modelled by Markovian processes.

8 Acknowledgements

Thanks to Claude Sammut, my supervisor, for suggesting I undertake these experiments in the first place, and for making his BOXES code available. Thanks to Richard Sutton and Charles Anderson for making available their code for the AHC algorithm. Thanks to Michael McGarity for helping to devise an "especially pleasing" form of the update rule for the P-Trace and Q-Trace algorithms. Thanks to Michael Harries and Linda Milne for feedback on an early draft of this report. Special thanks to Michael Bain, whose encouragement and advice over many months has been invaluable.

9 References

Anderson,C.W.(1989). "Learning to control an inverted pendulum using neural networks" IEEE Control Systems Magazine, vol 9, no 3, pp 31-37.

Barto,A.G.; Sutton,R.S.; Anderson,C.W.(1983). "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems" IEEE Transactions on Systems, Man, and Cybernetics, vol SMC-13 no 5, pp 834-846 (Sept/Oct 1983),

Lin,L-J.(1992) "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching" Machine Learning 8: pp 293-321.

Mahadevan,S.(1994) "To discount or not to Discount in Reinforcement Learning: A Case Study Comparing R Learning and Q Learning" Proc. of Eleventh Int. Conf. on Machine Learning, W.Cohen & H.Hirsh (Eds.), New Brunswick, New Jersey: Morgan Kaufmann.

Mataric,M.J.(1994) "Reward Functions for Accelerated Learning" Proc. of Eleventh Int. Conf. on Machine Learning, W.Cohen & H.Hirsh (Eds.), New Brunswick, New Jersey: Morgan Kaufmann.

Michie,D.; Chambers,R.A.(1968) "BOXES: An experiment in adaptive control." in "Machine Intelligence 2", E.Dale and D.Michie, Eds. Edinburgh: Edinburgh Univ. Press, 1968, pp 137-152.

Minsky,M.L.(1961) "Steps towards artificial intelligence" Proc. IRE, vol 49, pp 8-30.

Peng,J.; Williams,R.J.(1994) "Incremental Multi-Step Q-Learning" Proc. of Eleventh Int. Conf. on Machine Learning, W.Cohen & H.Hirsh (Eds.), New Brunswick, New Jersey: Morgan Kaufmann.

Sammut,C.A.(1988) "Experimental results from an evaluation of algorithms that learn to control dynamic systems" Proc. of the Fifth Int. Conf. on Machine Learning, John Laird (Ed.), Ann Arbor, Michigan: Morgan Kaufmann.

Sammut,C.A.(in press). "Recent progress with BOXES" in "Machine Intelligence 13", K.Furakawa, S.Muggleton and D.Michie, Eds. Oxford: The Clarendon Press, OUP.

Singh,S.P.; Jaakkola,T; Jordan,M.I.(1994) "Learning without State-Estimation in Partially Observable Markovian Decision Processes" Proc. of Eleventh Int. Conf. on Machine Learning, W.Cohen & H.Hirsh (Eds.), New Brunswick, New Jersey: Morgan Kaufmann.

Sutton,R.S.(1988) "Learning to Predict by the Methods Of Temporal Difference" Machine Learning 3: pp 9-44.

Sutton,R.S.; Barto,A.G.; Williams,R.J.(1992). "Reinforcement Learning is Direct Adaptive Optimal Control" IEEE Control Systems Magazine, vol 12, no 2, pp 19-22.

Tesauro,G.(1992). "Practical Issues in Temporal Difference Learning" Machine Learning 8: pp 257-277.

Tham,C.K.; Prager,R.W.(1994) "A Modular Q-Learning Architecture for Manipulator Task Decomposition" Proc. of Eleventh Int. Conf. on Machine Learning, W.Cohen & H.Hirsh (Eds.), New Brunswick, New Jersey: Morgan Kaufmann.

Watkins,C.J.C.H.(1989). "Learning from Delayed Rewards" Ph.D. Thesis, King's College, Cambridge.

Watkins,C.J.C.H.; Dayan,P.(1992). "Technical note: Q-Learning" Machine Learning 8: pp 279-292.

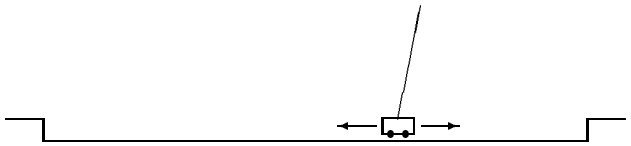
10 Appendices

A The pole-and-cart problem.

A four dimensional state space is partitioned into 162 discretised states or "boxes" (from which the Michie & Chambers algorithm derives its name). The state variables are comprised of the position of a cart on a fixed length track,

the cart's velocity, the angle (from the vertical) of a pole hinged to the cart, and the pole's angular velocity.

For each of these discretised state regions the pole-and-cart system has to decide upon a policy control action, which is to apply an impulse force of 10N to the cart in one of the two possible directions in the direction of the track. The goal is to learn a control strategy to avoid system failure; failure is defined by the cart hitting the ends of the track or the angle of the pole exceeding 12 degrees from the vertical. A fixed reinforcement signal (-1.0) is made available to the system upon a failure state being reached; no other information is available directly from the problem environment.



The specifications for the simulated pole-and-cart system are as follows:

length of track: 4.8m
mass of cart: 1.0 kg
mass of cart+pole: 1.1 kg
length of pole: 1m
distance of pole's centre of mass from pivot: 0.5 m

The partitioning points for the state space are as follows:

position on track (m): ± 0.8 (3 partitions)
velocity of cart on track (m/s): ± 0.5 (3 partitions)
angle of pole from vertical (degrees): $0, \pm 1.0, \pm 6.0$ (6 partitions)
angular velocity of pole (degrees/s): ± 50.0 (3 partitions)

The equations of motion used are those derived by Anderson (1988).

$$\ddot{\theta} = \frac{mg \sin\theta - \cos\theta(F + m_p l \dot{\theta}^2 \sin\theta)}{(4/3)ml - m_p l \cos^2\theta}$$

$$\ddot{x} = [F + m_p l(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta)]/m$$

where θ is the pole angle in radians, x is the cart's position on the track ($-2.4 < x < 2.4$), m is the mass of the cart and pole (1.1kg), g is the acceleration due to gravity (9.8m/s/s), F is the applied control force ($\pm 10.0N$), m_p is the mass of the pole (0.1kg), l is the distance from the pivot to the pole's centre of mass (0.5m).

The simulation uses the Euler method of integration. The sampling rate of the pole-and-cart's state and the rate at which control actions are applied are the same as the basic simulation rate, 50Hz.

In these experiments, the criterion for a successful control strategy was that the system could perform 10,000 consecutive control actions starting from a randomised (but not unrecoverable) initial state without encountering a failure state. This translates to the equivalent of keeping the pole balanced for three minutes and twenty seconds in real time.

B P-Trace and Q-Trace: Pseudo-code.

```
/* P-Trace and Q-Trace C-like pseudo-code.        Mark Pendrith, May 1994.

   For each state/action pair we keep a set of the following
   variables: Q, x, Sum, Rho, StepCount and VisitCount, all
   initialised to zero. The indexing form x[i][j] identifies
   the x variable belonging to state i and action j, etc.
*/

list StateActionsList = nil;
long GlobalClock = 0;

StartSystem();  /* reset dynamic system to initial state */

while (not halting criteria) {

  s = get_state();
  a = get_action(s); /* Stochastic Action Selector */

  if (!policy(s,a))
    zero_traces(StateActionsList);

  if (!inlist(StateActionsList,s,a))
    add2list(StateActionsList,s,a);
    /* add state/action pair s,a to end of list if not already in list */

  if (VisitCount[s][a] == 0) {  /* first visit */
    Sum[s][a] = 0;
    x[s][a] = BETA;
  }
  else {                        /* subsequent visit */
    Sum[s][a] *= (1-BETA);
    Sum[s][a] += x[s][a] * Rho[s][a];
    k = GlobalClock - StepCount[s][a];  /* steps since last visit */
    x[s][a] *= pow(GAMMA,k);
    x[s][a] *= (1-BETA);
    x[s][a] += BETA;
  }
```

```
  Rho[s][a] = 0;
  VisitCount[s][a]++;
  StepCount[s][a] = GlobalClock;

  r = TakeAction(a); /* take action; return immediate reinforcement */

  if (r != 0) {
    for all (i,j) in StateActionsList {
      k = GlobalClock - StepCount[i][j];
      Rho[i][j] += r * pow(GAMMA,k);
    }
  }

  if (failed) {  /* terminal state reached */
    for all (i,j) in StateActionsList {
      n = VisitCount[i][j];
      Sum[i][j] *= (1-BETA);
      Sum[i][j] += x[i][j] * Rho[i][j];
      Q[i][j] = pow((1-BETA),n)*Q[i][j] + Sum[i][j];
    }
    zero_traces(StateActionsList);
    StartSystem();
  }
#if QTRACE       /* do QL TD update for Q-Trace, but not for P-Trace */
  else {
    s' = get_state();
    Q[s][a] = (1-BETA)*Q[s][a] + BETA*(r + GAMMA*Q[s'][max]);
  }
  /* note that ``trace'' has already effectively performed this update if */
  /* a terminal state has just been reached, so no need to duplicate. */
#endif

  GlobalClock++;
}

/* procedure to ``zero'' traces */

void zero_traces(StateActionsList)

list StateActionsList;

{
  for all (i,j) in StateActionsList
    VisitCount[i][j] = 0;

  StateActionsList = nil;
}
```