Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance *

Michael L. Anderson, Tim Oates, Waiyian Chong, and Don Perlis

Abstract

In this study, we investigated the perturbation tolerance of reinforcement learning—that is, its ability to recover from unexpected changes. We found that the perturbation tolerance of reinforcement learners was negatively correlated with the degree of the perturbation, and that using relatively simple metacognitive monitoring and control components could significantly improve its perturbation tolerance.

1 Background and Approach

Maintaining adequate performance in dynamic and uncertain settings has been a perennial stumbling-block for intelligent systems, and an ongoing challenge to AI. A dynamic and uncertain environment means that conflict between what is currently believed (or tacitly assumed) and what is currently true is inevitable, and therefore an intelligent system's beliefs, assumptions, rules for thinking—in short, any aspect of the system which models or otherwise carries information about the environment or how to act in it—should ideally be open to alteration. Any system intended for real-world deployment should be able to accommodate such conflicts, both expected and unexpected; it must be, as we say, *perturbation tolerant*. The term is meant as an extension and generalization of John McCarthy's notion of "elaboration tolerance"—a measure of the ease with which a reasoning agent can add and delete axioms from its knowledge base [McCarthy, 1998]. Our term is more general than McCarthy's because his is explicitly limited to formal, symbolic systems, and an elaboration is defined as an action taken to change such a system [Amir, 2000]. But a cognitive agent may well consist of more than just a formal reasoning system, and flexibly coping with a changing world may therefore involve altering components in addition to, or instead of, its formal reasoner. Thus, we define a perturbation as any change, whether in the world or in the system itself, that impacts the performance of the agent, and *perturbation tolerance* as the ability of an agent to quickly re-establish desired performance levels after a perturbation.

Our approach to this very general problem has been to equip artificial agents with the ability to *notice* when something is amiss, *assess* the anomaly, and *guide* a solution into place. Because this basic strategy involves monitoring, reasoning about, and perhaps even altering one's own decision-making components, it is a *metacognitive* strategy, and we call the basic Note-Assess-Guide process the *metacognitive loop* (MCL). In our ongoing work, we have found that including an MCL component can enhance the performance of many different types of systems, including natural language human-computer interfaces, commonsense reasoners and deadline-coupled planning systems. In this study, we investigated the perturbation tolerance of reinforcement learning, and whether it could be improved with MCL.

*This research is supported in part by the AFOSR.

2 Q-learning

One particularly popular reinforcement learning algorithm is Q-learning. As is well known, the basic idea behind Q-learning is to try to determine which actions, taken from which states, lead (eventually) to rewards for the agent (however these are defined). The value of each action which could be taken in each state—its Q-value—is a time-discounted measure of the maximum reward available to the agent by following a path through state space of which the action in question is a part. In 1-step Q-learning, the Q-value of each state-action pair is constantly updated, as the agent acts in the world, according to the following equation:

$$(1)\quad Q(s,a) = Q(s,a) + \alpha * (r + (\gamma * maxQ(s',b)) - Q(s,a))$$

In this equation, $Q(s,a)$ is the Q-value of taking action a from state s, α is a learning rate, which controls the speed with which new experience changes the existing Q-value, r is the reward (if any) received for taking a from s, and $maxQ(s',b)$ is the Q-value of the highest-value action (b) from state (s'), which is the state resulting from taking a from s. $maxQ(s',b)$ is multiplied by the discount factor γ to satisfy the intuition that later rewards are less valuable than current ones. α and γ are assigned values between 0 and 1.

Because the Q-values are constantly adjusting according to equation 1, any changes in the reward structure of the world will eventually be reflected in the agent's *policy*, its assessment of what it should do in each state to achieve maximum utility. In addition, most implementations of Q-learning build in an exploration factor ϵ. A Q-learner will take the action recommended by its policy with probability $(1 - \epsilon)$, and will take a random action with probability ϵ. This helps ensure

that the agent comprehensively and continuously explores its world, rather than sticking to what it already knows will bring rewards.

The Q-learning algorithm is guaranteed, in a static world, to eventually converge on an optimal policy [Watkins, 1989; Watkins and Dayan, 1992], regardless of the initial state of the Q-learning policy and the reward structure of the world. Moreover, if the world changes slowly, Q-learning is guaranteed to converge on near-optimal policies [Szita *et al.*, 2002]. This is to say that Q-learners are already somewhat perturbation tolerant. However, we found that the actual performance of a Q-learner in the face of perturbation varies considerably, and, indeed, that post-perturbation performance is negatively correlated to the degree of perturbation.

2.1 The perturbation tolerance of standard Q-learning

To determine the perturbation tolerance of Q-learning, we built a standard table-based 1-step Q-learner, with settings $\alpha = 0.5$, $\gamma = 0.9$, and $\epsilon = 0.05$. Starting with all Q-values = 0, the Q-learner was placed in an 8x8 grid-world—the possible states being locations in the grid—with reward 1 (r1) in square (1,1) and reward 2 (r2) in square (8,8). Before continuing with a description of the experimental design, allow us to acknowledge that this is a relatively simple domain. However, two points need to be made in this regard. First, although the 8x8 2-reward grid-world domain *is* simple, given 4 possible actions from 64 distinct states and 248 possible state to state transitions (only four of which yield rewards), it is actually more complicated than the way it is typically visualized, as a small, flat plain with a couple of fruit trees. Second, we chose such a simple domain not to make it easier to find a problem that we could then fix, but to make it *harder*. Our working assumption is that recovering from perturbations will in fact become harder as the domain becomes more complex; thus, the simpler the domain in which perturbation tolerance is an issue, the more common the problem is likely to be.

Moving on, the initial reward structure [r1,r2] of the world was one of the following: [10,-10]; [25,5]; [35,15]; [19,21]; [15,35]; [5,25]. The Q-learner was allowed to take 10,000 actions in this initial world, which was enough in all cases to establish a very good albeit non-optimal policy. After receiving a reward, the Q-learner was randomly assigned to one of the non-reward-bearing squares in the grid. In turn 10,001, the reward structure was abruptly switched to one of the following: [25,5]; [35,15]; [21,19]; [15,35]; [5,25], [-10,10]. Taking all possible cases of perturbation so described, and omitting duplicates, gives 22 possible perturbations. Each case was run 20 times, and the results averaged over these 20 runs.

We defined the performance of the Q-learner as the ratio of actual average reward per action taken (henceforth, per turn) to the ideal average reward per turn, i.e., the average reward per turn theoretically available to a Q-learner following an optimal policy in the given environment. Actual average reward per turn, for each turn t, was calculated using a 200 turn window ranging from t to $t + 200$. Calculating this way, the performance of the Q-learner can be plotted over time (turn number) to give a graph like that shown in Figure 1.

Figure 1 shows the performance of a standard Q-learner,

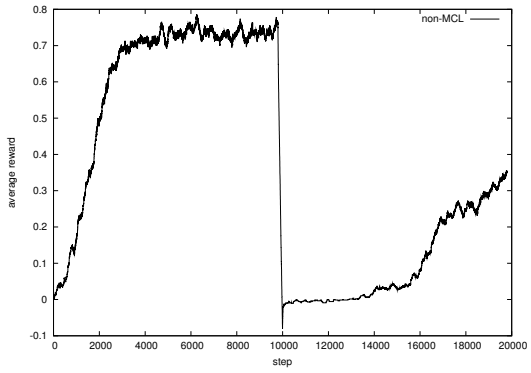

Figure 1: Performance of a standard Q-learner, with perturbation at step 10,001

over 20,000 turns, with an initial reward structure of [-10,10] and a post-perturbation reward structure of [10,-10].

For the current study we were interested primarily in the *post-perturbation* performance of the Q-learner, and in comparing this to the degree of the perturbation. To estimate the degree of perturbation, we first imagined the learned policy as a topographic overlay on the grid world, where positive rewards are attractors, and negative rewards are repulsors, and the grade of the topography at each point corresponds to the degree of attraction to a given reward. For simplicity, we considered only a topography-following line drawn between the two rewards, split into 4 equal segments. Each of these segments will slope toward one of the rewards, that is, it will slope either left or right. If we assign a value -1 to a leftward slope, and 1 to a rightward slope, then the abstract policy resulting from each reward structure can be represented by an ordered set ranging from $(-1,-1,-1,-1)$ to $(1,1,1,1)$. Considering the value of the ordered set which would have to be added to the pre-perturbation policy to make it equal to the post-perturbation policy, gives one element of our estimate of the degree of the perturbation. Thus, for instance, to go from the abstract policy for reward structure [10,-10] $(-1,-1,-1,-1)$ (everything slopes to the left) to the abstract policy for reward structure [-10,10] $(1,1,1,1)$ (everything slopes to the right) involves a large topographical change (T) of value 8, i.e. $(2,2,2,2)$, but going from [19,21] to [21,19] involves essentially no topographical change.

Another factor in measuring the degree of perturbation we considered was any valence change in the rewards. A valence change makes the perturbation greater for two reasons. First, a negative reward which becomes positive (V^+) is masked from the agent because the policy is strongly biased against visiting that state. In fact, the chance of the agent discovering the reward (C_r), given a policy biased against visiting the relevant state, is equal to the chance (c) of it being in an adjacent state (s), times ϵ, times the inverse of the number of actions the agent can take (A_n):

(2) $C_r = c * \epsilon * (1/A_n)$

The chance c of it being in an adjacent state s varies somewhat with the implementation, but is generally equal to the inverse of the number of states in the world (S_n), minus the

number of states containing rewards (R_n) (that is, the chance of the agent being assigned to s randomly after it receives a reward), plus (once again) the chance (c') of its having been in an adjacent state (s') to s, and having randomly chosen to go to state s. Each of these factors must be multiplied by the adjacency (j) of the state space, that is, the number of possible states that are adjacent to the reward state, and to s:

(3) $c = j * (1/(S_n - R_n) + (c' * \epsilon * (1/A_n)))$

To solve this equation, it must modified by an appropriate substitution for c' (itself based on equation 3), and so on, until all the possibilities in the state space have been covered. It takes us too far afield to resolve this series of equations in the current case, but it should be clear that they add up to a pretty low value.

Second, a positive reward which becomes negative (V^-) creates a situation in which, although the agent is locally repulsed by the reward, it is still globally attracted to it. Until the negative reward value propagates fully through its policy, the policy will contain non-rewarding transitional attractors, that is, states to which the agent is attracted, despite the fact that they contain no reward.

In light of the above considerations, we devised the following equation to estimate the degree of perturbation (D_p) in each of the 22 cases tested:

(4) $D_p = T/2 + 3V^+ + V^-$

The post-perturbation performance of the standard Q-learner, as a function of degree of perturbation, is summarized in Figure 2. As can be seen from the graph, not only did the post-perturbation performance vary considerably, but it was negatively correlated with the degree of perturbation ($R = -0.85, p < 0.01$).

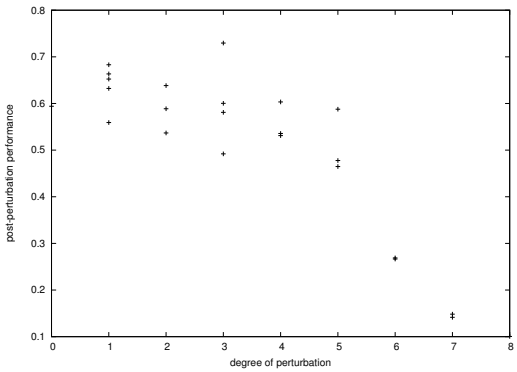

Figure 2: Post-perturbation performance of a standard Q-learner, as a function of degree of perturbation

2.2 The perturbation-tolerance of enhanced Q-learning

We hypothesized that the perturbation-tolerance of Q-learners could be improved by enhancing them with MCL. In order to make the test results as clear as possible, we first tested the simplest MCL algorithm we could think of: if the expected reward is not received three times in a row, throw out the policy and start over. Thus, we built a simple-MCL enhanced Q-learner, with exactly the same settings as the standard Q-learner used in the previous case, and ran it through exactly the same protocol as described above.

The simple MCL-enhanced Q-learner, although not entirely unaffected by certain perturbations, performs about equally well in all cases. This is not particularly surprising if one reflects that the abstract policies for all the reward structures considered are topographically equi-distant from $(0, 0, 0, 0)$, which is the abstract policy to which the MCL-enhanced Q-learner reverts when it detects a perturbation. Thus, in the ideal case, the post-perturbation performance graph should look exactly like the initial performance. However, the simple-MCL enhanced Q-learner, like the standard Q-learner, has a certain amount of trouble with valence changes, and for similar reasons. It won't visit a positive reward turned negative more than twice, except by chance, nor will it visit the negative reward turned positive, except by chance. As the probability here is low—governed by the series of equations defined by equations 2 and 3—it can take some time for it to receive the required 3 unexpected rewards. This explains why, in Figure 3, the post-perturbation performance graph deviates from the initial performance. More sophisticated perturbation-detection mechanisms (see below) improve performance in this case.

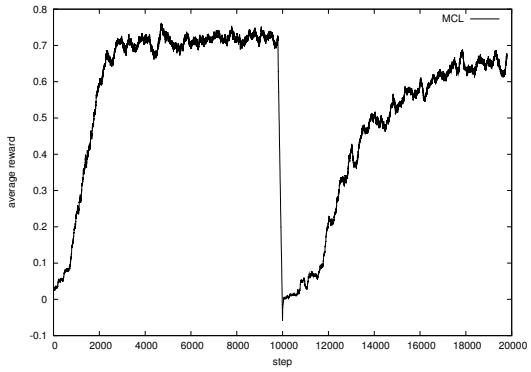

Figure 3: Performance of an MCL-enhanced Q-learner, with perturbation at step 10,001

Like Figure 1, Figure 3 shows the performance of the enhanced Q-learner over 20,000 turns, with an initial reward structure of [-10,10] and a post-perturbation reward structure of [10,-10].

The [10,-10] to [-10,10] perturbation is a high-degree perturbation (value 8), and shows a dramatic difference in post-perturbation performance. The results are not so dramatic in all cases, nor do they always favor MCL enhanced Q-learning. Figure 4, for instance, shows a low-degree perturbation (value 0), from [19,21] to [21,19]. Note that because simple MCL throws out its policy and begins again, there is a period where it significantly under-performs standard Q-learning.

Thus, comparing the *overall* performance of simple-MCL enhanced Q-learning to standard Q-learning reveals a slight under-performance when the degree of perturbation is small, but significant performance gains as the degree of perturba-

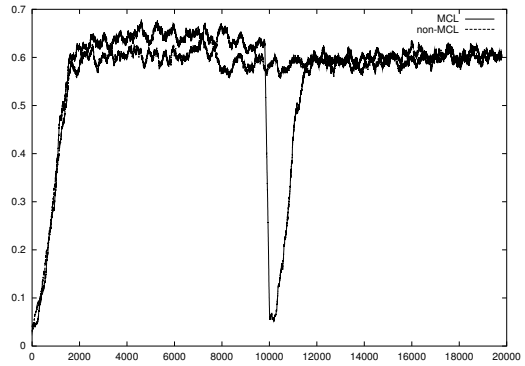

Figure 4: Performance of standard and simple MCL-enhanced Q-learning, with 0-degree perturbation at step 10,001.

tion increases. (See the non-MCL and simple-MCL lines in Figure 5.)

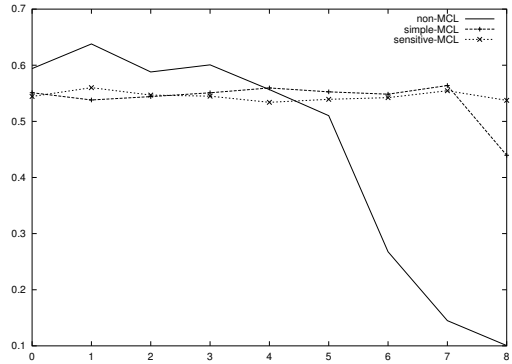

Figure 5: Post-perturbation performance of standard Q-learning, simple-MCL and sensitive-MCL, as a function of degree of perturbation.

These results are fairly striking, especially given the simplicity of the MCL enhancement. However, there are two areas where improvement is called for. First, as noted above, simple-MCL has a difficult time detecting certain perturbations, and so is slow to react in some cases. Second, simple-MCL enhanced Q-learning underperforms standard Q-learning in response to low-degree perturbations. We addressed the first of these problems by improving MCL's ability to detect perturbations, and the second of these by devising methods of *responding* to perturbations somewhat more subtle than simply throwing out an established policy and starting over. These two improvements are discussed below.

Sensitive MCL

In order to address the problem that simple-MCL had in detecting certain kinds of perturbations, we developed an MCL component that was able to track not just expected reward values, but expected time to reward and expected average reward/turn, and was thus much more sensitive to perturbations than simple-MCL. For "sensitive-MCL" a perturbation could be any of the following: (1) an unexpected reward, (2) the number of turns between rewards was three times the expected value, or (3) the average reward per turn dropped to eighty-five percent of the expected value. This version of MCL did indeed out-perform simple-MCL on high-degree perturbations, as it reacted more quickly, but otherwise performed the same as, or perhaps very slightly under-performed, simple-MCL, since its increased sensitivity caused it to react to problems other than the experimentally caused perturbation. (See Figure 5.) This caused it to throw out its policy too readily; in some cases the learner went through three or four policies in a single run.

Epsilon variations

The other outstanding issue with simple-MCL was the fact that it under-performed standard Q-learning after low-degree perturbations. This is clearly a result of the single, drastic response available to simple-MCL of throwing out its policy and starting over. One possible alternative to this is to temporarily raise ϵ, the exploration factor. Using sensitive-MCL as the starting point we tested two versions of this strategy, one in which (after three perturbation detections) ϵ was fixed at 0.8 for 2,000 turns, and then returned to 0.05, and another in which ϵ was initially set to 0.8, and then allowed to decay linearly to 0.05 over 2,000 turns.

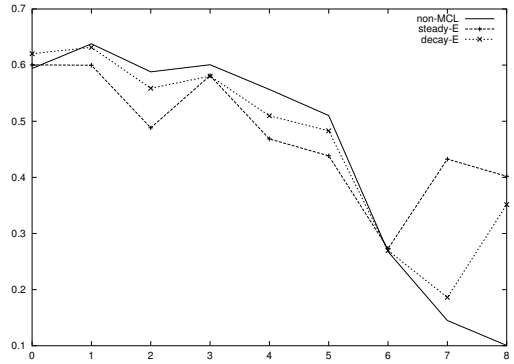

Figure 6: Post-perturbation performance of Q-learning, steady-ϵ and decaying-ϵ, as a function of degree of perturbation.

The results are interesting in a number of ways. First of all, the runs where epsilon is changed tracks standard Q-learning pretty closely, which is somewhat surprising, as one might expect *some* improvement from the increased exploration. However, a closer look shows that the near equality is the result of the tradeoff between exploration and exploitation: naturally, when the balance is toward exploration, average reward per turn suffers, and so Q-learners with high ϵ values will initially under-perform those with low ϵ values. However, one would expect later exploitation to be more efficient. This is indeed the case here. If one focuses on the last 2000 turns of each run (see Table 1), it is quite clear that the Q-learners that took more time to explore significantly outperform those that did little exploration. Thus it appears that over a longer run, the value of a high initial ϵ would eventually emerge.

MCL Type	Performance
Standard Q-learning	0.60
Sensitive-MCL	0.64
Decaying-ϵ	0.66
Steady-ϵ	0.70

Table 1: Post-perturbation performance, averaged over last 2000 turns for all runs.

Sophisticated MCL

Given the results above, it should be possible, with a good assessment of the perturbation, to perform well in response to high-degree perturbations by throwing out one's policy and starting over, and in response to low-degree perturbations by temporarily raising ϵ. We tried to address this by developing a "sophisticated-MCL" component that took a more nuanced approach to evaluating and dealing with perturbations. Like the other systems, it waited to react until it detected three perturbations, but unlike those systems, it chose its reaction based on an evaluation of the seriousness of those perturbations. As with sensitive-MCL there were three types of perturbations: increased time to reward; decreased average performance; and an unexpected reward value. The first two types of perturbation were always assigned a degree of two; however, in the case of an unexpected reward value, sophisticated MCL assessed its seriousness as follows:

Is there a valence change?
  Yes: Is it - to +?
    Yes: degree 4
    No: Is the expected reward $>$ average reward?
      Yes: degree 4
      No: degree 2
  No: Is the expected reward $>$ average reward
    and $>$ actual reward?
    Yes: Is actual reward / expected reward $< .75$?
      Yes: degree 3
      No: degree 1
    No: degree 1

After three perturbations had been noted and assessed, if the combined degree of perturbation exceeded seven, the policy was thrown out; otherwise, ϵ was raised in the amount of the estimated degree of the perturbation divided by ten, and allowed to decay linearly over 2,000 turns. In addition, two different damping mechanisms were used to counteract the volatility of sensitive MCL. First, in the case of detections of increased time to reward and decreased performance, other detections of these types of perturbations occurring within 50 turns were ignored. Second, after a response to the perturbation had been implemented (after three detections), sophisticated MCL waited for 2,000 turns before responding to further perturbations, in order to give its solution time to work.

The results show that sophisticated-MCL does as well or better than standard Q-learning in all cases (see Figure 7). However, in response to high-degree perturbations, especially those involving valence changes, sophisticated-MCL can under-perform sensitive-MCL. This is because in these cases, for the same set of reasons discussed above, it can be difficult for a Q-learner, relying on its own experience, to get a good assessment of how the world has changed, for instance if it is biased by its current policy against visiting a certain region of its world. Thus, having only partial information, it can assess a given perturbation as being less serious than it actually is, and, rather than throwing out its policy and starting again (which it probably should do) it instead responds by temporarily raising ϵ. Possibly it is the case that this particular issue could be overcome by implementing a special directed exploration strategy, which MCL could trigger in the case of a perturbation, that would *force* the agent to visit particular states (or perhaps all states), regardless of the bias of its policy.

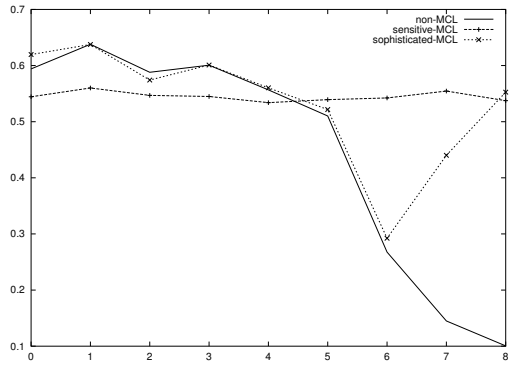

Figure 7: Post-perturbation performance of standard Q-learning, sensitive-MCL and sophisticated-MCL, as a function of degree of perturbation.

Still, even given this issue, averaging post-perturbation performance over *all* runs shows that sophisticated-MCL outperforms all the other systems overall (see Table 2).

MCL Type	Performance
Standard Q-learning	0.530
Simple-MCL	0.545
Sensitive-MCL	0.546
Steady-ϵ	0.510
Decaying-ϵ	0.526
Sophisticated-MCL	0.567

Table 2: Post-perturbation performance, averaged over all turns for all runs.

Before moving on to the next sections, it is worth noting that although the rewards in these cases were deterministic, there is no reason in principle why these techniques would not work in a non-deterministic world. For instance, while the expected reward values in the above case were simple numbers, they could just as easily have been *models* of expected values over time. As with the case of monitoring expected time to reward, if experience deviated sufficiently from the expected *model* for reward, this would count as a perturbation.

3 Other RL Techniques

Although we worked most extensively with Q-learning, we have also run similar experiments with both SARSA [Sutton and Barto, 1995] and Proritized Sweeping [Moore and Atkeson, 1993]. We have found that the post-perturbation

performance of these algorithms is very similar to that of Q-learning, and we therefore conclude that agents using these algorithms would benefit equally from MCL enhancement.

4 Comparison to Related Research

Many techniques under the general topic of reinforcement learning [Sutton and Barto, 1995], such as neuro-dynamic programming [Bertsekas and Tsitsiklis, 1996], have been developed for acting under uncertainty. However, these techniques are very different from the proposed work in that they focus on action selection itself, not monitoring or reasoning about the action selection process or its performance, and they adapt to non-stationarity only by continually training. The latter requires continual exploration, or deviation from the optimal action policy, whereas MCL systems can act optimally until they notice that something is wrong and then take remedial actions focused on the problem at hand. Alternately, a different purely stochastic technique might be to factor uncertainty about rewards directly into the model; if it were known ahead of time that the world might undergo a random perturbation in the future, this knowledge could be accounted for in the form of a probability model and included in the decision-making optimization. Here again, it seems that insofar as some non-actual possible future model of the world was taken account of in deriving an action policy, the result of this would be a policy that was *sub*-optimal for the current state of the world. In contrast, the MCL approach is to allow the policy to optimize for current circumstances, but to monitor performance and make on-line adjustment in the case of problems. Furthermore, as noted, the stochastic technique would depend on having advance knowledge (and even perhaps a model) of the possibility for change. The MCL approach can respond equally to anticipated and unanticipated events.

Tsumori and Ozawa [Tsumori and Ozawa, 2003] showed that in *cyclical* environments, reinforcement learning performance could be enhanced with a long-term memory, and a "change detector", which would recall stored policies when a given known environment reappeared. This work is in the spirit of MCL, although we think it is important to monitor one's own performance and not only the environment.

Another approach to the problem of using reinforcement learning in dynamic environments is to explicitly include information about the state of all dynamic objects in the domain, monitor those states, and change the relevant information and learn a new policy when these states change [Wiering, 2001]. This can be a useful expedient when one knows up front which objects, and which states of which objects, are likely to change, but the technique would likely be difficult to generalize to poorly-known domains, or unexpected changes. Here again, monitoring one's own performance is a more general expedient, which can help one detect when something *unexpected* has occurred. Finally, model-based approaches can be computationally expensive, whereas the version of MCL tested here requires virtually no additional overhead.

5 Conclusion

We have shown not only that the perturbation tolerance of reinforcement learning varies inversely with the degree of the perturbation, but have devised and tested some simple enhancements that significantly improve its perturbation tolerance. Although there is little doubt that response strategies can be developed that are more effective than the ones we tested, we would like to emphasize the *general* finding that noting, assessing, and differentially responding to anomalies gives better performance than either ignoring them, or always doing the same thing in response to them.

References

[Amir, 2000] Eyal Amir. Toward a formalization of elaboration tolerance: Adding and deleting axioms. In M. Williams and H. Rott, editors, *Frontiers of Belief Revision*. Kluwer, 2000.

[Bertsekas and Tsitsiklis, 1996] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[McCarthy, 1998] John McCarthy. Elaboration tolerance. In *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, 1998.

[Moore and Atkeson, 1993] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[Sutton and Barto, 1995] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1995.

[Szita *et al.*, 2002] István Szita, Bálint Takács, and András Lörincz. ϵ-MDPs: Learning in varying environments. *Journal of Machine Learning Research*, 3:145–174, 2002.

[Tsumori and Ozawa, 2003] K. Tsumori and S. Ozawa. Incremental learning in dynamic environments using neural network with long-term memory. In *Proceedings of the Int. Conf. on Neural Networks*, 2003.

[Watkins and Dayan, 1992] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[Watkins, 1989] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

[Wiering, 2001] M. A. Wiering. Reinforcement learning in dynamic environments using instantiated information. In *Proceedings of the Eighth International Conference on Machine Learning*, 2001.