Journal of Experimental and Theoretical Artificial Intelligence
Vol. 18, No. 3, September 2006, 387–411

Taylor & Francis
Taylor & Francis Group

The metacognitive loop I: Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance||

MICHAEL L. ANDERSON*†¶, TIM OATES†§,
WAIYIAN CHONG‡ and DON PERLIS†‡

†Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742, USA
‡Department of Computer Science, University of Maryland,
College Park, MD 20742, USA
§Department of Computer Science, University of Maryland,
Baltimore County, Baltimore, MD 21250, USA
¶Department of Psychology, Franklin & Marshall College,
Lancaster, PA 17604, USA

(*Received 9 January 2006; in final form 27 June 2006*)

Maintaining adequate performance in dynamic and uncertain settings has been a perennial stumbling block for intelligent systems. Nevertheless, any system intended for real-world deployment must be able to accommodate unexpected change—that is, it must be *perturbation tolerant*. We have found that metacognitive monitoring and control—the ability of a system to self-monitor its own decision-making processes and ongoing performance, and to make targeted changes to its beliefs and action-determining components—can play an important role in helping intelligent systems cope with the perturbations that are the inevitable result of real-world deployment. In this article we present the results of several experiments demonstrating the efficacy of metacognition in improving the perturbation tolerance of reinforcement learners, and discuss a general theory of metacognitive monitoring and control, in a form we call the *metacognitive loop*.

Keywords: Learning; Commonsense reasoning; Autonomous agents

*Corresponding author. Email: michael.anderson@fandm.edu
||This research is supported in part by the AFOSR and ONR.

Journal of Experimental and Theoretical Artificial Intelligence
ISSN 0952–813X print/ISSN 1362–3079 online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/09528130600926066

1. Introduction

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. And the problem is even worse than this, for not just the deployment environment, but the system itself can change: sensor calibration can drift, parts can wear or break, and the knowledge base can get large and unwieldy, or even be partially deleted or corrupted. Any system intended for real-world deployment should be able to accommodate such changes, 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. Performance is meant to be construed broadly to encompass any measurable aspect of the agent's operation. *Perturbation tolerance*, then, is the ability of an agent to quickly recover—that is, to re-establish desired/expected performance levels—after a perturbation.

People tend to do this well. Indeed, the ability to ''get along'' in a wide range of changing environments can be said to be something of a hallmark of human cognition, and the cognitive ability that has been most difficult to reproduce in artificial systems. This is in stark contrast to formal domains where special intellectual cleverness or skill is involved, for instance, solving math problems or playing chess; here, artificial systems have nearly equaled if not surpassed human performance. It should be noted, however—and this is an important point—that the fact that humans are *good* at adjusting to perturbations does not imply that it is necessarily *easy* for us. Indeed, it can be jarring when something goes wrong, or we are confronted with something unexpected. This subjective sense of having to shift gears, of being forced to *re-think* something in these situations, has long suggested to us that noting and recovering from perturbation involves specialized components that may not typically be involved in everyday decision-making. We have hypothesized that in these situations a metacognitive loop (MCL) is brought to bear, wherein an individual (i) notes there is a difficulty to be addressed; (ii) assesses the options for dealing with the difficulty; and (iii) guides one of the options into action (Anderson and Perlis 2005).

In fact, there is some empirical evidence for the importance of metacognition in dealing with the unexpected or unfamiliar. In studies of human learning strategies, for instance, an individual studying for a test will make judgments of the relative difficulty of the material, using these to frame study strategies (Nelson *et al.* 1994,

Nelson and Dunlosky 1994). Not surprisingly, in these cases, accuracy of metacognitive judgments correlates with academic performance. Moreover, neurophysiological findings indicate that the frontal lobe has specialized responsibility for metacognitive behavior (Nelson 1996). For instance, patients with frontal lobe damage have trouble handling a ''reversal shift'', which involves (1) recognizing that a word or concept one has learned to apply to, say, big things, is now being used by others to refer to small things, and (2) making the appropriate adjustment (Kendler and Kendler, 1962, 1969).

Thus, our approach to the very general problem of perturbation tolerance has been to equip artificial agents with MCL: the ability to *notice* when something is amiss, *assess* the anomaly†, and *guide* a solution into place (Anderson and Perlis 2005). In our view, this is largely what perturbation-tolerant commonsense reasoning consists of, rather than finding special clever solutions to thorny problems. After all, performance in the face of unexpected perturbations can be enhanced even when one cannot figure out exactly what is wrong, or what to do about it, so long as one is able to realize that *something* is wrong, and ask for help, or use trial-and-error, or even give up and work on something else. In our ongoing work, we have found that including an MCL component can enhance the performance of many different types of systems, including commonsense reasoners (Elgot-Drapkin and Perlis 1990, Purang 2001), deadline-coupled planning systems (Kraus *et al.* 1990, Nirkhe 1994, Nirkhe *et al.* 1997), natural language human–computer interfaces (HCI) (Traum *et al.* 1999, Anderson *et al.* 2003, Josyula *et al.* 2003), robotic navigation (Hennacy *et al.* 2003), and reinforcement learners (see subsequently).

2. What is MCL?

In the next sections, we will detail specific examples of systems where MCL-enhanced versions perform better than non-MCL systems. But before moving on to the examples, it is perhaps worthwhile to get a bit clearer on what MCL is, what it means for the design of intelligent systems, and when it is likely to prove beneficial.

2.1 *The nature of MCL*

What, then, is MCL? MCL is two closely related things. *First*, it is an approach to the design of intelligent systems that makes *self-monitoring*, *self-control*, and *self-improvement* a central, fundamental part of the system's architecture. The overall impact of the MCL approach—and, more generally, of the intensive use of expectations generated and monitored by the system itself—on system design is an important issue that deserves wider attention. General design principles and practices are needed. In our view, systems should have expectations attached to (generated for) every action they take, both locally (at the level of system components), and globally (for actions taken by the system), as well as more general expectations for the sorts of things that should happen when they are operating properly—that is, they should

† In accordance with the given approach to perturbations, we define an anomaly as a deviation from expectations for performance or outcomes.

have general expectations for their performance. Ideally, the system would have specific solutions for specific violations of these expectations, as well as more generic strategies for dealing with novel problems. But it is important to note (as we will emphasize throughout) that having specific, targeted solutions for every problem is not a necessary condition for improving performance. In many cases, a system that, noticing a problem it cannot address, could move itself into some "safe" state, where it will neither harm itself or anything else, and call for help would be a significant improvement over the state of the art.

Second, MCL is not just a design philosophy, it is also a specific hypothesis about *how* to achieve perturbation tolerance, and what form metacognitive monitoring and control should take. This hypothesis has two parts. One is the note–assess–guide structure of the metacognitive loop itself, and the other is the notion that a limited number of metacognitive strategies will suffice for the vast majority of problems. Thus, we expect that the components that implement the MCL portion of a larger intelligent system can be kept fairly fast and lightweight.

Our more general claim is that the practice of self-monitoring for anomalies, assessing, and responding to those anomalies is a better, more efficient, and ultimately more effective approach to perturbation tolerance than are any of: (1) ignoring the problem, (2) building everything in at the outset or (3) trying to continually monitor and model a sizable portion of the world.

Perhaps, the claim that ignoring the problem is a poor response can be allowed to stand without argument. But why cannot the "MCL" part be handled at design time by the system designer, who will consider all the likely problems a system will encounter, and build in the solutions? The simple answer is that this very common strategy has repeatedly, and often spectacularly, failed. A recent entry in the DARPA Grand Challenge—a robotic maze-running/road-race contest—ran into a fence that it was unable to see. As it could not see the fence, it did not register it as an obstacle, and it continued to try to move forward. Eventually, the system had to be shut down lest it burn out its motors. Likewise, consider the case of a satellite that was given the command to turn away from Earth and examine a distant star. The maneuver pointed the satellite's communications dish away from Earth, and mission control had neglected to include the command to eventually turn *back* toward Earth in the original command sequence—and, since the satellite was pointing away from Earth, it could no longer receive communications from Earth. The result was the loss of the satellite, as mission control waited for months for the satellite's orbit to put it in position to receive more commands. Systems that had some built-in expectations—either for what to expect as a result of certain actions (e.g., forward progress), or more general expectations for its own operation, based on the sort of thing it was (e.g., frequent contact with Earth)—might have noticed the violation of these expectations, and set about trying to figure out what was going on.

Furthermore, in support of the notion that self-monitoring is more effective for this purpose than is world-monitoring, consider that the world is vast, indefinitely detailed, and dynamic. It is impossible to model *all* of it; but it is likewise impossible to always know in advance which changes in, and which aspects of, the world are important to monitor. In contrast, the system is relatively small, its performance variables are known, and accurate modeling poses much less of a challenge. If the world changes in ways that affect performance, monitoring

for anomalies will indicate such changes; and when the world changes in ways that do not affect performance, why waste resources noting the fact? Furthermore, although it will always be the case that world monitoring is an important aspect of ensuring the overall performance of a system, allowing the monitoring to be directed by the known current needs of the system can help ensure its tractability. Thus, for instance, when the system has undertaken an action, it will monitor the outcomes of that action, and compare them with expectations, making appropriate adjustments in light of environmental feedback. Likewise, in the case of an anomaly, it may well be that solving (or even assessing) the anomaly requires world monitoring—for instance, if there arises a contradiction between beliefs that might be adjudicated by gathering further evidence—but in such a case the monitoring task is limited and tractable, amounting to a kind of directed search.

2.2 *The value of MCL*

This brings us to the question of when—under what circumstances—MCL is likely to be beneficial. One way to get a sense of this has to do with the imagined *degree* of a perturbation. In general, the bigger, more serious or more extensive a change, the larger impact it would be expected to have on performance. Intuitively, then, the more serious the perturbation, the more important it is to be able to specifically assess and respond to it, thus bringing about a faster recovery. This intuition was borne out in the reinforcement learning experiments detailed below: the greater the degree of the perturbation, the greater was the relative impact of MCL on the post-perturbation performance. On the other side of the coin, the opposite intuition was also borne out: it is often better to do little or nothing in response to very small perturbations. Taking drastic corrective actions in these cases can actually *decrease* relative post-perturbation performance. What this means is that MCL must be made sophisticated enough to in fact do little or nothing in response to small degree perturbations, thereby not harming relative post-perturbation performance. Still, from the standpoint of a system that would have done nothing *anyway*, time and resources spent in *deciding* to do nothing are wasted. As in all choices about implementation, the overall benefits to be gained in the expected environments must be weighed against overall costs.

Another class of situations in which MCL can be helpful is in the face of what might be called *barrier perturbations*. There are some perturbations where, although it may be difficult or inappropriate to measure their *degree*, they nevertheless represent an impediment to further progress on a task. For instance, an incorrect or missing fact can prevent further reasoning on a topic; and the emergence of a direct contradiction in the system's knowledge base likewise threatens further valid inference. Whatever the *degree* of such perturbations may be, explicitly addressing them is nevertheless *necessary* for the system to move forward on its relevant goals. One instance of a barrier perturbation, discussed below, is the case where a natural language HCI system does not know one of the main words uttered by a user, and therefore cannot complete the user's request. In such a case, an MCL system can recognize this fact, and take steps to learn the word, or, if that proves too difficult, to get the user to rephrase the utterance. In such cases, it is clear that doing nothing is not a valid option.

2.3 *Measuring the value of MCL*

Based on the above considerations, we expect that MCL will prove a good general expedient for enhancing the perturbation tolerance of intelligent systems. But, again, this does not mean that MCL is helpful in every case, nor that anomaly detection and assessment-guided response is an infallible method of dealing with perturbations. For, as described, MCL rests on two fundamental events, both of which can be in error: *detections* and *interventions*. The system can fail to detect perturbations that matter, or falsely detect perturbations; likewise, the system can fail to intervene properly when an intervention is called for, or intervene when doing nothing would have been better. MCL has been shown to be effective in cases where a change in the world negatively impacts the measured performance of the system (see subsequently). However, it will obviously be less effective in cases where changes increase the *opportunity* for high performance without impacting actual performance. For instance, in the case where a change occurs in a region of the environment that the system has reasons to avoid, and is thus outside of the awareness of the system, naturally the system will not, in general, be able to take advantage of such a change, even if such a change represents an increased potential reward. This would be an example of a failure in detection.†

Thus, one important issue in determining the costs, benefits, and applicability of MCL involves the likelihood that environments with noise (stochastic processes) will generate false anomaly detections, and result in unnecessary interventions. Naturally, the chance of a false positive is a function of the discernment/sophistication of the MCL detection mechanism, and the amount of variation in the environment. However, we can roll these into one variable, C_{FP}, the chance of a false positive. On the other hand, on the assumption that the environment does sometimes undergo actual performance impacting changes (perturbations), there is a probability of *true* detections. As above, the chance of a true positive is a function of the discernment/sophistication of the MCL detection mechanism, and the amount of perturbation in the environment. We can likewise roll this into the variable C_{TP}. In both cases, over any specified length of time, these two variables yield a single number of false positives F and true detections D.

For each false positive F_n there is a cost, which is a measure of the area between the (hypothetical) performance curve (performance over time) had no unnecessary intervention occurred, and the actual performance of the system, which, for any given intervention, will temporarily degrade. Of course, different interventions will have different costs; in the case of reinforcement learning, throwing out an established policy and starting over is more costly than simply raising ϵ for a time. However, if each such intervention has a fixed cost, and a probability of being implemented, then a weighted average can be calculated to yield a constant average cost per incident I.

Likewise, for each true positive there is a value, which is a measure of the area between the hypothetical performance curve had the given intervention not occurred, and the actual performance of the MCL system. That is, the value of an intervention depends on such things as the amount of impact on performance and the amount

† Still, it should be noted that the problem of imperfect knowledge, however important to MCL, is not important *only* or even *especially* to MCL.

of time it takes, in the absence of an intervention, to bring performance up to previous values. However, here too it seems that, if there is a definite value associated with each kind or degree of perturbation, and these occur with a given frequency, then a weighted average can likewise be generated to give a constant average value per incident V.

Finally, there is the cost of MCL itself, measured as the amount of resources (time, memory, clock cycles) required to achieve a given level of performance. One way to think of this is if the non-MCL system requires R_1 clock cycles to achieve performance P_1, and the MCL system R_2 clock cycles to achieve performance P_2, then the relative efficiency of the two systems at getting performance is (P_1/R_1) *vs* (P_2/R_2). However, one also has to consider that there is some relative value between a unit of performance and a unit of resource. If resources are valuable compared with performance, then efficiency is valuable; if however, resources are cheap compared with performance, then efficiency is less of a concern. For instance, if each unit of performance is worth \$1.00, and each unit of resource costs \$0.10, then even if (P_1/R_1) *vs* $(P_2/R_2) = (1/1)$ *vs* $(2/10)$, because a unit of performance is 10 times as valuable as a unit of resource, the relative value of the two systems is not 5:1, but 1:2. Let us call the relative value of performance and resources Q.

The overall relative cost or benefit of MCL, then, can be calculated as the total performance P of a non-MCL system, over a given interval t, minus the number of false positives F in that interval times their cost I, plus the number of true positives D in that interval times their value V, divided by the resources required R, and multiplied by the relative value of performance and resources Q: $(P - (F * I) + (D * V)) * (Q/R)$.

Focusing initially just on those cases where MCL systems outperform non-MCL systems, and putting aside for the moment whether this increased performance is worth the cost, it is clear that MCL enhances performance whenever $(D * V) > (F * I)$.

3. Background work

The considerations above provide a general idea of what MCL is, and when to expect MCL systems to outperform their non-MCL counterparts. In the next sections, we will detail specific examples of MCL-enhanced systems that outperform their non-MCL counterparts. First, we discuss some past work in the areas of automated, non-monotonic reasoning, and natural language HCI. Then, in section 4, we will report on new work demonstrating the performance improvements that can be gained by enhancing reinforcement learning with MCL.

3.1 *Reasoning with contradictions*

The challenge of dealing with a dynamic world is especially acute for symbolic reasoners, for as the world changes, the reasoner will inevitably encounter conflicts—often in the form of direct contradictions†—between the information it

† "Direct contradiction" here means a conflict between P and $\neg P$, as opposed to more general inconsistencies which can be very hard to detect.

gathers from the world, and the beliefs resident in its knowledge base (KB). One simple approach to this problem might be to always trust the sensors, and automatically update the relevant portions of the KB with sensory information, thus side-stepping any contradictions between sensory information and the KB. But, in addition to suffering from a certain naïveté about the reliability of sensors, such an approach ignores the fact that sensory information might not only directly contradict a given proposition in the KB, but might *entail* conflicts with *other* elements of the KB. And the trouble is that for any sufficiently complex KB which was not produced by logical rules from a database known to be consistent, and/or to which non-entailed facts are to be added (e.g., from sensory information), it is not generally possible to know that it is consistent, nor to use principled methods to maintain consistency (Perlis 1986). This is the *consistency check problem*. Contradictions are in this sense practically inevitable.

This brings us to another problem, the *swamping problem*: for, in addition to the obvious reasons for wanting to maintain consistency (if we are to query our KB, we would generally prefer to get an answer, and not an answer and its negation), there is a more theoretical issue: from a contradiction, everything follows. More technically, given a contradiction, all well-formed formulas (wffs) are entailed as theorems.† It perhaps goes without saying that a system that will eventually come to believe every possible sentence in its language is unlikely to be a very effective agent.

Active logic is a formalism that has been developed with such challenges in mind (Elgot-Drapkin 1988, Elgot-Drapkin and Perlis 1990, Elgot-Drapkin *et al.* 1993). Motivating its design is the thought that one of the factors that supports the flexibility of human reasoning is that it takes place step-wise, in time. This allows the agent to maintain control over, and track, its own reasoning processes. Moreover, agents will sometimes need to be able to examine, and have beliefs about their own beliefs—for instance, about whether, and why, they are warranted, and whether they should still be trusted. As will be detailed below, active logic provides a way to monitor the ongoing reasoning process, note anomalies in the form of contradictions, and take specific steps to adjudicate those contradictions, or, at the very least, to stop reasoning with the contradictory pair.

Each "step" in an active logic proof itself takes one active logic time-step; thus, inference always moves into the future at least one step and this fact can be recorded in the logic. In active logic, beliefs are held at times, and the KB is therefore considered to be a temporally embedded and evolving set of formulas. Thus, the meaning of an inference rule such as equation (1) (an active logic analogue to *modus ponens*), is that if A and $A \rightarrow B$ are in the KB at time (step number) $\mathtt{i}$, then B will be added to the KB at time $\mathtt{i}+1$.

$$\begin{array}{ll} \mathtt{i} & : \dfrac{\mathtt{A}, \mathtt{A} \rightarrow \mathtt{B}}{} \\ \mathtt{i}+1 & : \quad \mathtt{B} \end{array} \tag{1}$$

† Paraconsistent logics offer one approach to the swamping problem (also known as the explosion problem, or *ex contradictione quodlibet* (ECQ) (Priest *et al.* 1989, Priest 2002)). However, most paraconsistent logics tend to sidestep, rather than note and deal with, contradictions. We believe, in contrast, that it can be useful to confront and reason about contradictions as they arise. See subsequently.

In active logic, since the notion of inference is time-dependent, it follows that at any given time only those inferences that have actually been carried out so far can affect the present state of the agent's knowledge. As a result, even if directly contradictory wffs, $\mathtt{P}$ and $\neg\,\mathtt{P}$, are in the agent's KB at time $\mathtt{t}$, it need not be the case that those wffs have been used by time $\mathtt{t}$ to derive any other wff, $\mathtt{Q}$. Indeed, it may be that $\mathtt{t}$ is the first moment at which both $\mathtt{P}$ and $\neg\,\mathtt{P}$ have simultaneously been in the KB.

By endowing an active logic with a ''conflict-recognition'' inference rule such as that in (2), *direct* contradictions can be recognized as soon as they occur, and further reasoning can be initiated to repair the contradiction, or at least to adopt a strategy with respect to it, such as simply avoiding the use of either of the contradictands for the time being. The `Contra` predicate is a metapredicate: it is about the course of reasoning itself (and yet, is also part of that same evolving history).

$$\begin{array}{ll} \mathtt{i} & : \quad \dfrac{\mathtt{P}, \neg\,\mathtt{P}}{} \\ \mathtt{i}+1 & : \quad \mathtt{Contra}(\mathtt{i}, \mathtt{P}, \neg\,\mathtt{P}) \end{array} \tag{2}$$

The idea then is that, although an indirect contradiction may lurk undetected in the KB, it may be sufficient for many purposes to deal only with direct contradictions. Sooner or later, if an indirect contradiction causes trouble, it may reveal itself in the form of a direct contradiction. After all, a real agent has no choice but to reason only with whatever it has been able to come up with *so far*, rather than with implicit but not yet performed inferences. Moreover, since consistency (i.e., the lack of direct or indirect contradictions) is, in general, undecidable, all agents with sufficiently expressive languages will be forced to make do with a hit-or-miss approach to contradiction detection. The best that can be hoped for, then, seems to be an ability to reason effectively in the presence of contradictions, taking action with respect to them only when they become revealed in the course of inference (which itself might be directed toward finding contradictions, to be sure).

These temporal and metacognitive aspects make active logic systems more flexible—more perturbation tolerant—than traditional AI systems and therefore more suitable for reasoning in noisy, dynamic and inconsistent environments. Active logic systems have been developed which can reason in the presence of, and in some cases automatically resolve, contradictory information (Elgot-Drapkin and Perlis 1990, Elgot-Drapkin *et al.* 1993, Purang 2001), and have also been applied to such related areas as deadline-coupled planning (Miller and Perlis 1993).

3.2 *Noting and fixing errors in dialog*

One of the most important application areas for active logic has been natural language HCI. Natural language is complex and ambiguous, and communication for this reason always contains an element of uncertainty. To manage this uncertainty, human dialog partners continually monitor the conversation, their own comprehension, and the apparent comprehension of their interlocutor. Both partners elicit and provide feedback as the conversation continues, and make conversational adjustments as necessary. The feedback might be as simple as ''Got it?'', eliciting a simple ''yes'', or as complex as ''Wait. I don't think I understand the concept

of hidden variables'', which could result in a long digression. We contend that the ability to engage in such metalanguage, and to use the results of metadialogic interactions to help understand otherwise problematic utterances, is the source of much of the flexibility displayed by human conversation (Perlis *et al.* 1998). Although there are other ways of managing uncertainty (and other types of uncertainty to be managed), we have demonstrated that improved performance can be achieved by enhancing existing HCI systems with the ability to self-monitor, note anomalies such as contradictions, misinterpretations or unknown words, and fix the problems by, among other things, engaging in meta-dialog with the user.

One achievement was the design and implementation of a model of action-directive exchanges (task-oriented requests) based on an the active logic model of inference. Our model works via a step-wise transformation of the literal request made by a user (e.g., ''Send the Boston train to New York'') into a specific request for an action that can be performed by the system or domain. In the case of 'the Boston train', the system we have implemented is able to interpret this as 'the train in Boston', and then further disambiguate this into a specific train currently at Boston station, which it will send to New York. Information about each step in the transformation is maintained, to accommodate any repairs that might be required in the case of negative feedback (if for instance, the system picks the wrong train, and the user says ''No'' in response to the action). This implementation represents an advance not just in its ability to reason initially about the user's intention (e.g., by 'the Boston train' the user means . . .) but in its ability to respond in a context-sensitive way to post-action user feedback, and use that feedback to aid in the interpretation of the user's original and future intentions. For instance, in one specific case tested, the user says ''Send the Boston train to New York'' and then, after the system choses and moves a train, says ''No, send the *Boston* train to New York''. Such an exchange might occur if there is more than one train at Boston station, and the system chose a train other than the one the user meant. Whereas the original TRAINS-96 dialog system (Allen *et al.* 1995) would respond to this apparently contradictory sequence of commands by sending the very same train, our enhanced HCI system *notes* the contradiction, and, by *assessing* the problem, identifies a possible mistake in its choice of referent for 'the Boston train'. Thus, the enhanced system will choose a different train the second time around, or if there are no other trains in Boston, it will ask the user to specify the train by name (Traum *et al.* 1999, Traum and Andersen 1999).

More recently we have made further progress along these lines, by enhancing the ability of our HCI system to more accurately assess the nature of dialog problems, and to engage in metadialog with the user to help resolve the problem. For instance, if the user says ''Send the Metro to Boston'', the original system would have responded with the unhelpful fact that it was unable to process this request. Our system, in contrast, notices that it does not know the word 'Metro', and will instead request specific help from the user, saying: ''I don't know the word 'Metro'. What does 'Metro' mean?'' Once the user tells the system that 'Metro' is another word for 'Metroliner', it is able to correctly implement the user's request (Anderson *et al.* 2003, Josyula *et al.* 2003).

The two key rules that help note such perturbations in the reasoning are a rule for detecting contradictions (which is a standard feature of active logic), and the ability to set and monitor expectations. As with the derivation of a contradiction,

if an agent *notices* that an expectation has not been achieved, this causes the agent to recognize a potential problem, and thus provides an opportunity for the agent to *assess* the different options it has to fix the problem. For instance, if the user does not respond to a system query within the expected time limit, then the system recognizes that there is a problem and tries different methods for fixing it (and, not incidentally, it also monitors the effectiveness of the solutions, and will change strategies if they appear to be ineffective).

In the same spirit as the above work, we have turned our attention to the possibility of enhancing traditional machine learning with metacognitive monitoring and control. In the next sections, we describe the particulars of our approach in that case, and the results so far.

4. Perturbation tolerance and reinforcement learning

Recently, we have been investigating approaches to the problem of dealing with a dynamic world that centrally involve various methods of reinforcement learning. Using learning techniques to address the problem of change is sensible for two closely related reasons. First, the current state of the world may be unknown, or may differ from one's initial representations or expectations, and it therefore makes sense to equip an agent with the ability to adjust its representations, expectations, and policies for acting, so these can better fit the world as it actually is. Second, if it is possible for the world to change even after a period of initial learning and adjustment, the agent ought to be able to learn about these later changes, too. The continuous learning approach is to equip agents with learning algorithms that use feedback to constantly adjust to the world, so as to keep one's representations and policies for acting as current as possible.

4.1 *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 to rewards for the agent (however these are defined), and which actions, from which states, lead to the states from which the said rewards are available, and so on. 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 the simplest form of Q-learning, 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:

$$Q(s, a) = Q(s, a) + \alpha * (r + (\gamma * \max Q(s', b)) - Q(s, a)) \quad (3)$$

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 $\max Q(s', b)$ is the Q-value of the highest-value action (b) from state (s'), which is the state resulting from taking a from s. $\max Q(s', b)$ 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.

In table-based Q-learning, the end of this iterative updating is a list of all state–action pairs, and their Q-values. This table is used to determine the agent's *policy*, its assessment of what it should do in each state to achieve maximum utility. Because the Q-values are constantly adjusting according to Equation 3, any changes in the reward structure of the world will eventually be reflected in the agent's policy. In addition, most implementations of Q-learning build in an exploration factor ϵ. A Q-learner will take the action recommended by its policy (the so-called "greedy action") 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, learning the effect of all the actions it can take from all possible states, 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 perturbations varies considerably, and, indeed, that post-perturbation performance is negatively correlated to the degree of the perturbation. In addition, we showed that enhancing Q-learning with even simple forms of MCL could improve its perturbation tolerance.

4.1.1 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 the settings $\alpha = 0.5$, $\gamma = 0.9$, and $\epsilon = 0.05$. Starting with all Q-values $= 0$, the Q-learner was placed in an 8×8 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 it is typical to describe such a domain as if it were a model of a small, flat plain with a couple of fruit trees, such a description is in fact an oversimplification. The 8×8 2-reward grid-world domain *is* simple, but given four possible actions from 64 distinct states and 248 possible state-to-state transitions (only four of which yield rewards), it will support a much more complicated description than "open plain with two fruit trees"—it would be just as appropriate to offer the more complex-sounding story that the agent, to find a treasure, must: climb the waterfall, open the red door, pick up the gold key, turn the lock, open the green door, climb the ladder, pick up the silver key, turn the lock, and open the chest. In any event, whenever one is applying reinforcement learning (RL) to a domain, what one is interested in discovering and modeling are the states and the state-to-state transitions; a grid-world is, in essence, just a simple way of defining a world consisting of states and transitions; however, those states and transitions map onto the "real" world. 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.

Table 1. Ideal average reward/turn for different reward structures.

Structure	Ideal average
[10, −10]	1.43
[25, 5]	3.57
[35, 15]	5.69
[19, 21]	4.46

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]; [19, 21]†; [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 $t + 200$.

The ideal average reward per turn was calculated by finding the set of squares in the grid from which it would be preferable to go to each reward, determining the average distance (number of turns) d to the reward from that set of squares, and dividing the value of the reward by d. Note that in some cases, it is always preferable to go to one reward over another, no matter where in the world the agent is. Table 1 shows the ideal average reward per turn available, for each reward structure, in an 8×8 world.

Given these values, the performance of the Q-learner can be plotted against turn number, to give a graph like that shown in figure 1.

Figure 1 shows the performance of a standard Q-learner, over 20,000 turns, with an initial reward structure of [−10, 10] and a post-perturbation reward structure of [10, −10]. The case was run 20 times, and the results were averaged.

For the current study, we were interested primarily in the *post-perturbation* performance of the Q-learner, and in comparing this with the degree of the perturbation. For this comparison we defined the post-perturbation performance as the average performance/turn, for all turns taken after the perturbation.

To estimate the degree of perturbation, we first considered that for each square in the grid (state in the policy), the policy instructs the learner to go toward one reward (up or left) or the other (down or right). One element of the difference

† Except when the initial structure was [19, 21], in which case the post-perturbation structure was [21, 19].

‡ Note that so long as $\epsilon > 0$ it is not possible to achieve the theoretical ideal average reward per turn.

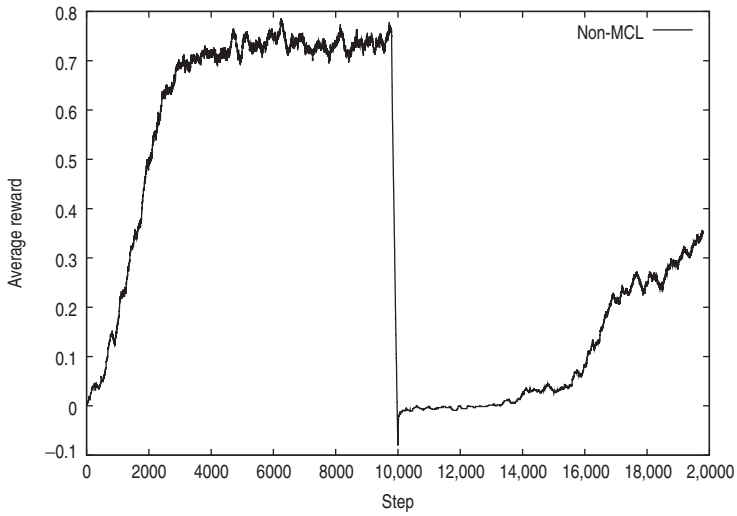

Figure 1. Performance of a standard Q-learner, with perturbation at step 10,001.

between worlds before and after a perturbation is the number of states where this instruction will need to change. Thus, for instance, to go from the policy for reward structure $[10, -10]$ (go up or to the left in every state) to the policy for reward structure $[-10, 10]$ (go down or to the right in every state) involves 64† changes of instruction, but going from [19,21] to [21,19] requires no such 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):

$$C_r = c * \epsilon * (1/A_n) \tag{4}$$

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:‡

$$c = j * (1/(S_n - R_n) + (c' * \epsilon * (1/A_n))) \tag{5}$$

† We count all 64 states for simplicity.

‡ Depending on the shape of the state space, the adjacency value may be the same for all states, or vary depending on the state in question. For simplicity of expression, we are assuming the former.

To solve this equation, it must be modified by an appropriate substitution for c' (itself based on Equation 5), 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:

$$D_p = T/16 + 3V^+ + V^- \tag{6}$$

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 ($R = -0.85, p < 0.01$).

4.1.2 The perturbation tolerance of enhanced Q-learning. Given our past work on MCL mentioned above, we hypothesized that the perturbation tolerance of Q-learners could be improved by enhancing them with metacognitive monitoring and control. In order to make the results as clear as possible, and to keep the computational overhead involved in MCL at a minimum, 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 earlier.

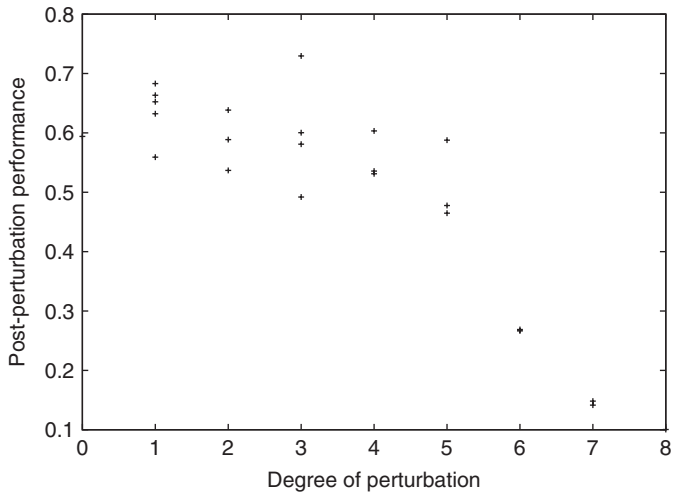

Figure 2. Post-perturbation performance of a standard Q-learner, as a function of degree of perturbation.

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 simple MCL-enhanced Q-learner starts from the scratch 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 will not 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 4 and 5—it can take some time for it to receive the required three unexpected rewards. This explains why, in figure 3, the post-perturbation performance graph deviates from the initial performance. More sophisticated perturbation-detection mechanisms (see subsequently) improve performance in this case.

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 case was run 20 times, and the results were averaged.

Figure 4 shows figure 1 and figure 3 superimposed, for easy comparison.

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 simple-MCL-enhanced Q-learning. Figure 5, 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

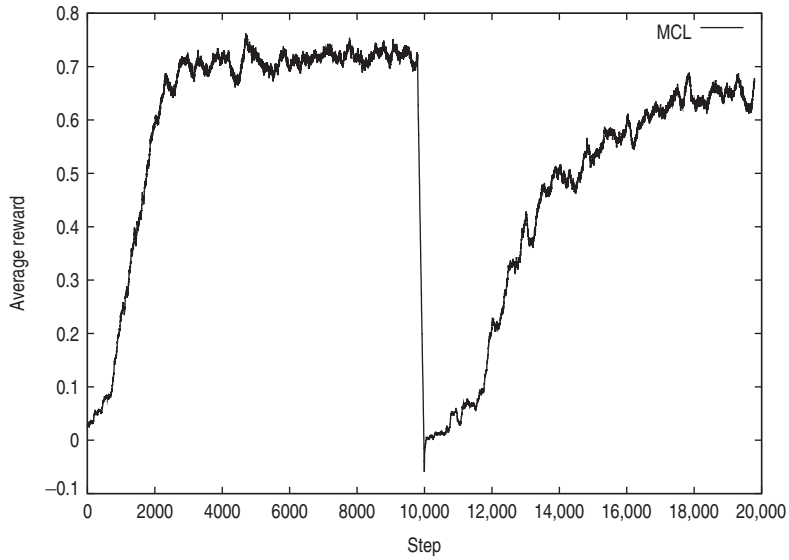

Figure 3. Performance of an MCL-enhanced Q-learner, with perturbation at step 10,001.

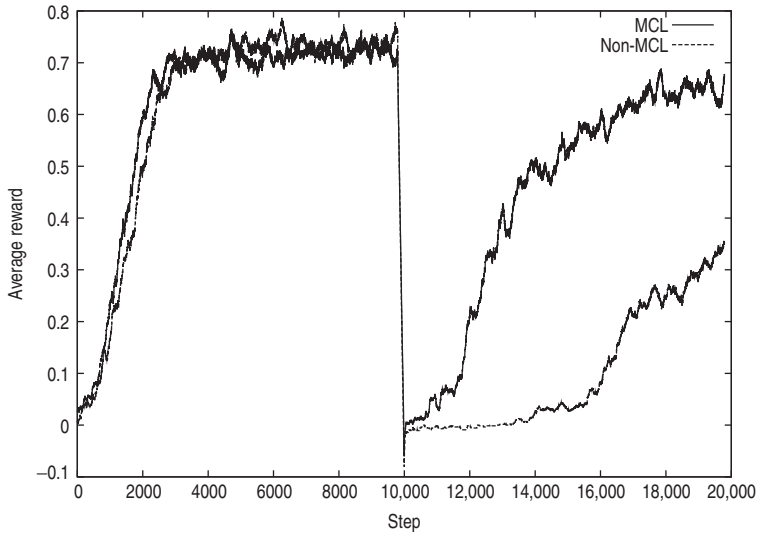

Figure 4. Superimposition of figures 1 and 3, to compare the performance of a standard and an MCL-enhanced Q-learner, with perturbation at step 10,001.

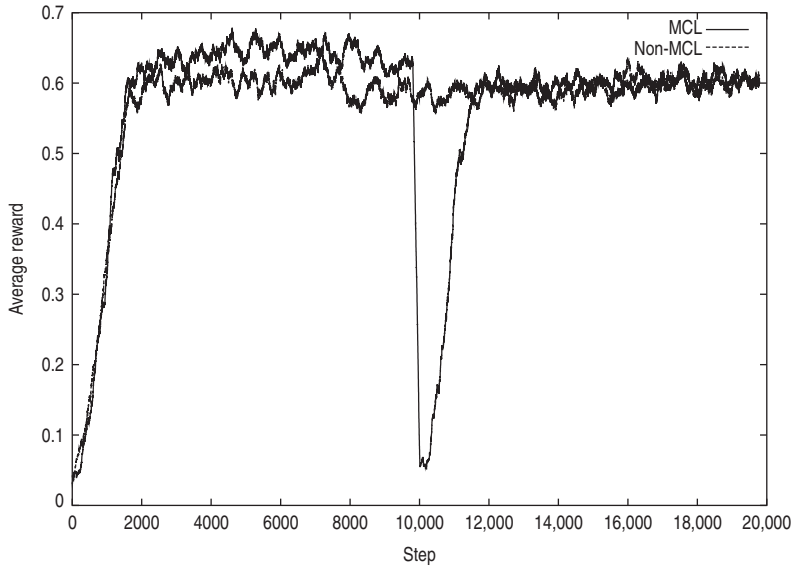

Figure 5. Performance of standard and simple MCL-enhanced Q-learning, with 0-degree perturbation at step 10,001.

perturbation is small, but significant performance gains as the degree of perturbation increases. The comparative performance results are summarized in figure 6.

The results show a high degree of correlation between the degree of the perturbation and the ratio of MCL to non-MCL performance ($R = 0.79, p < 0.01$). These results are fairly striking, especially given the simplicity of the MCL enhancement. However, there are two areas where improvement is called for.

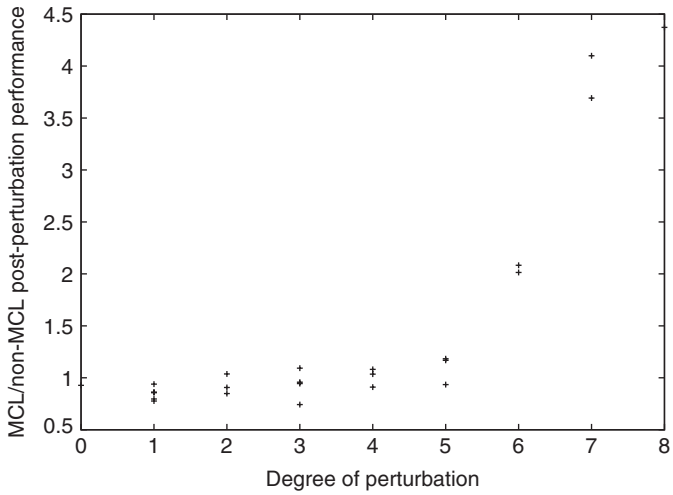

Figure 6. Ratio of MCL/non-MCL post-perturbation performance, as a function of the degree of perturbation.

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 under-performs 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. Results are given subsequently.

4.1.3 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. The ''three strikes'' and throw out the policy approach was retained, but 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 85% of the expected value. This ''sensitive-MCL'' component did indeed outperform 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 over-react to minor problems, and not just the experimentally caused perturbation. 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 (all experiments were run according to the same protocol described earlier). Figure 7 compares the performance of standard Q-learning with simple MCL and sensitive MCL-enhanced Q-learning, as a function of the degree of perturbation.

4.1.4 ϵ variations. The other outstanding issue with simple MCL was the fact that it under-performed standard Q-learning after low-degree perturbations. This is

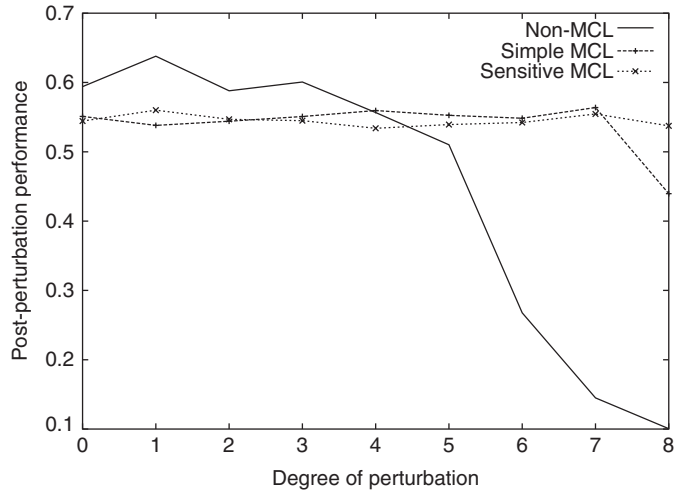

Figure 7. Post-perturbation performance of standard Q-learning, simple-MCL and sensitive-MCL, as a function of degree of perturbation.

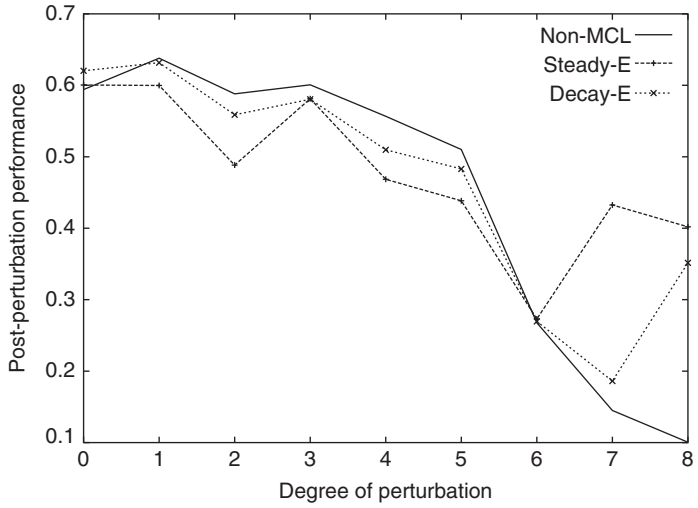

Figure 8. Post-perturbation performance of Q-learning, steady-ϵ and decaying-ϵ, as a function of the degree of perturbation.

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 and the ''three-strikes'' approach, we tested two versions of this strategy, one in which (after three perturbation detections) ϵ was fixed at 0.8 for 2000 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 2000 turns (figure 8).

The results are interesting in a number of ways. First of all, the runs where ϵ is changed tracks standard Q-learning pretty closely, which is somewhat surprising,

Table 2. Post-perturbation performance, averaged over last 2000 turns for all runs.

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

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, the importance of exploration is that one can have a more accurate picture of the environment, and thus 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 (table 2), it is quite clear that the Q-learners that took more time to explore significantly outperform those that did little exploration. In the 10,000 turns after the perturbation during which performance was measured, the cost and advantages of an initially high ϵ generally balanced out. However, it appears that over a longer run, the value of a high initial ϵ would eventually emerge.

4.1.5 Sophisticated MCL. From the results above, it looks as if 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. For instance, a perturbation where a high-value reward (a reward expected to be higher than the average reward received) decreased significantly would be considered more serious than a large but positive change in the expected reward, or a significantly increased time to reward. If its estimation of the seriousness of the perturbations reached a given threshold, it would throw out its policy and start again; otherwise it would take the less drastic step of temporarily increasing ϵ, and allowing it to decay linearly over 2000 turns.

Sophisticated MCL utilized a decision tree to make its determination of the degree of each of the three detected perturbations, and added the totals to get an estimate of the overall seriousness of the problem. As with sensitive MCL there were three types of perturbations: increased time to reward (in the case tested, three times the average time to reward); decreased average performance ($<80\%$ of the expected performance); and an unexpected reward value. The first two types of perturbation

were always assigned a degree of two. 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 10, and allowed to decay linearly over 2000 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 2000 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 (figure 9). However, in response to high-degree perturbations, especially those involving valence changes, sophisticated MCL can under-perform

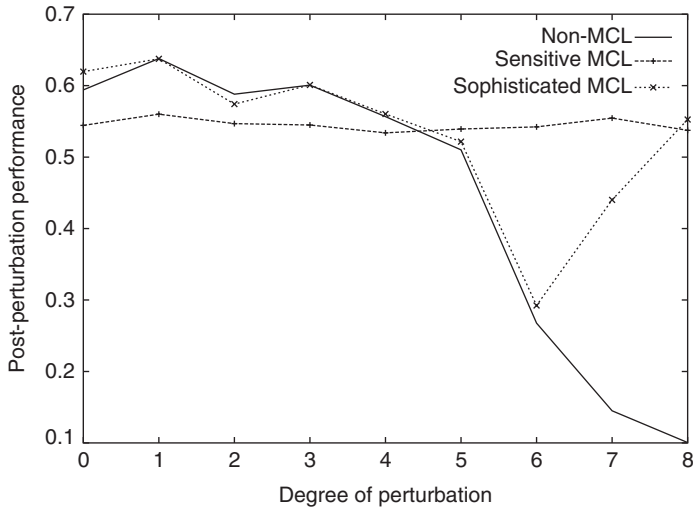

Figure 9. Post-perturbation performance of standard Q-learning, sensitive-MCL and sophisticated-MCL, as a function of degree of perturbation.

Table 3. Post-perturbation performance, averaged over all turns for all runs.

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

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. Something similar to this has been suggested by Leslie Kaelbling, whereby, for exploration, the chance of taking a given action from a given state was inversely proportional to the number of times that action had been taken from that state in the past.

Still, even given this issue, averaging post-perturbation performance over *all* runs shows that sophisticated MCL outperforms all the other systems overall (table 3).

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.

Furthermore, 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. Although there is little doubt that response strategies can be developed that are more effective than the ones we tested (for instance, the directed exploration strategy suggested above, or a method for zeroing out selected portions of a policy, or some combination of these and others), we expect the effectiveness of the note–assess–guide strategy to stand.

4.2 *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.

Note that in the reinforcement learning studies, none of the perturbations were what we earlier defined as barrier perturbations. This is in part because the systems tested were all continuously learning and exploring, and so changes in the world will all eventually be reflected in their action policies. Although MCL can be *helpful* in these cases, it is not *necessary*. However—and this brings up an important class of cases where MCL can be helpful—it is not always feasible to deploy a system that is continuously learning and exploring. In some cases, continuous learning could cause unwanted drift (in pattern recognition or concept learning, for instance, exposure to a long string of marginal cases can change the definitions in undesirable ways). Likewise, taking random, exploratory actions can have disastrous consequences in some environments. Hence, does National Aeronautics and Space Administration (NASA), for instance, discourage random innovation and deviation from established protocols. In such cases, where systems are equipped with fixed action-guiding components, it can be especially important to monitor actual performance, because, insofar as the system will not automatically adjust, perturbations will *continually* impact or impede performance unless and until specific remedial action is taken.

5. Comparison to related research

In general, what is distinctive about our approach is its insistence on the general applicability of the note–assess–guide strategy embodied in MCL, and, in particular, on the value of the first step: noting that there is a problem in the first place. After all, performance in the face of unexpected perturbations can be enhanced even when one cannot figure out exactly what is wrong, or what to do about it, so long as one is able to realize that something is wrong, and ask for help, or use trial-and-error, or even give up and work on something else. And in those cases where it *is* possible to determine precisely what is wrong, and take specific remedial action (as in the case of realizing that one does not know a given word, and taking steps to learn it), still the necessary first step is crucial: admitting there is a problem to be addressed.

For instance, 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 (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.

6. Conclusion

In this article we described a general approach to the problem of perturbation tolerance in intelligent systems. Our approach centrally involves system self-monitoring and self-alteration, and, more specifically, rests on the notion that systems should have specific expectations for their own performance, and the outcomes of their actions, and should watch for violations of these expectations. We call the practice of *noting* failed expectations (anomalies), *assessing* the options for responding, and *guiding* a response into effect the metacognitive loop (MCL). In addition, we discussed several ongoing projects, presenting both past findings and new results that demonstrate the value of metacognitive monitoring and control in improving the performance of intelligent systems. We believe that the variety of the systems enhanced suggests the wide applicability of metacognitive approaches in general, and MCL in particular.

References

J.F. Allen, L.K. Schubert, G. Ferguson, P. Heeman, C.H. Hwang, T. Kato, M. Light, N. Martin, B. Miller, M. Poesio and D.R. Traum, "The TRAINS project: a case study in building a conversational planning agent", *J. Exp. Theor. Artifi. Int.*, 8, pp. 7–48, 1995.

E. Amir, "Toward a formalization of elaboration tolerance: Adding and deleting axioms", in *Frontiers of Belief Revision*, M. Williams and H. Rott, Eds, Dordrecht: Kluwer, 2000, pp. 147–162.

M.L. Anderson, D. Josyula and D. Perlis, "Talking to computers", in *Proceedings of the Workshop on Mixed Initiative Intelligent Systems*, Acapulco, Mexico: IJCAI-03, 2003, pp. 1–8.

M.L. Anderson and D. Perlis, ''Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness'', *J. Logic Comput*., 15, pp. 21–40, 2005.
D.P. Bertsekas and J.N. Tsitsiklis, *Neuro-Dynamic Programming*, Nashua, NH: Athena Scientific, 1996.
J. Elgot-Drapkin and D. Perlis, ''Reasoning situated in time I: Basic concepts'', *J. Exp. Theor. Artif. Int*., 2, pp. 75–98, 1990.
J. Elgot-Drapkin, S. Kraus, M. Miller, M. Nirkhe and D. Perlis, ''Active logics: A unified formal approach to episodic reasoning'', *Technical Report UMIACS-TR # 99-65, CS-TR # 4072*, College Park, MD: University of Maryland, College Park, 1993.
J. Elgot-Drapkin, ''Step-logic: Reasoning Situated in Time''. PhD thesis, Department of Computer Science, University of Maryland, College Park, 1988.
K. Hennacy, N. Swamy and D. Perlis, ''RGL study in a hybrid real-time system'', in *Proceedings of IASTED NCI 2003*, Mexico, Cancun, 2003, pp. 203–208.
D. Josyula, M.L. Anderson and D. Perlis, ''Towards domain-independent, task-oriented, conversational adequacy'', *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003, pp. 1637–1638.
H.H. Kendler and T.S. Kendler, ''Vertical and horizontal processes in problem solving'', *Psychol. Rev*., 69, pp. 1–16, 1962.
H.H. Kendler and T.S. Kendler, ''Reversal-shift behavior: Some basic issues'', *Psychol. Bull*., 72, pp. 229–232, 1969.
S. Kraus, M. Nirkhe and D. Perlis, ''Deadline-coupled real-time planning'', *Proceedings of 1990 DARPA workshop on Innovative Approaches to Planning, Scheduling and Control*, 1990, pp. 100–108.
J. McCarthy, ''Elaboration tolerance'', in *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, 1998.
M. Miller and D. Perlis, ''Presentations and this and that: Logic in action'', in *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, 1993.
A.W. Moore and C.G. Atkeson, ''Prioritized sweeping: Reinforcement learning with less data and less time'', *Mach. Learn*., 13, pp. 103–130, 1993.
T.O. Nelson and J. Dunlosky, ''Norms of paired-associate recall during multitrial learning of Swahili-english translation equivalents'', *Memory*, 2, pp. 325–335, 1994.
T.O. Nelson, J. Dunlosky, A. Graf and L. Narens, ''Utilization of metacognitive judgments in the allocation of study during multitrial learning'', *Psychol. Sci*., 4, pp. 207–213, 1994.
T.O. Nelson, ''Consciousness and metacognition'', *Am. Psychol*., 51, pp. 102–116, 1996.
M. Nirkhe, S. Kraus, M. Miller and D. Perlis, ''How to (plan to) meet a deadline between *now* and *then*'', *J. Logic Comput*., 7, pp. 109–156, 1997.
M. Nirkhe, ''Time-situated reasoning within tight deadlines and realistic space and computation bounds'', PhD thesis, Department of Computer Science, University of Maryland, College Park, 1994.
D. Perlis, K. Purang and C. Andersen, ''Conversational adequacy: mistakes are the essence'', *International Journal of Human-Computer Studies*, 48, pp. 553–575, 1998.
D. Perlis, ''On the consistency of commonsense reasoning'', *Comput. Intelli*., 2, pp. 180–190, 1986.
G. Priest, R. Routley, J. Norman, ''Paraconsistent Logic: Essays on the Inconsistent'', Philosophia Verlag, Munchen, 1989.
G. Priest, ''Paraconsistent logic'', in *Handbook of Philosophical Logic*, 2nd ed., D. Gabbay and F. Guenther, Eds, Dordrecht: Kluwer Academic Publishers, 2002, pp. 287–393.
K. Purang, ''Systems that detect and repair their own mistakes'', PhD thesis, Department of Computer Science, University of Maryland, College Park, 2001.
R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1995.
I. Szita, B. Takács and A. Lörincz, ''ϵ-MDPs: Learning in varying environments'', *Journal of Machine Learning Research*, 3, pp. 145–174, 2002.
D. Traum and C. Andersen, ''Representations of dialogue state for domain and task independent meta-dialogue'', *Proceedings of the IJCAI99 workshop: Knowledge And Reasoning in Practical Dialogue Systems*, pp. 113–120, 1999.
D. Traum, C. Andersen, Y. Chong, D. Josyula, M. O'Donovan-Anderson, Y. Okamoto, K. Purang and D. Perlis, ''Representations of dialogue state for domain and task independent meta-dialogue'', *Electronic Transactions on Artificial Intelligence*, 3, pp. 125–152, 1999.
K. Tsumori and S. Ozawa, ''Incremental learning in dynamic environments using neural network with long-term memory'', *Proceedings of the International Conference on Neural Networks*, pp. 2583–2588, 2003.
C.J.C.H. Watkins and P. Dayan, ''Q-learning'', *Mach. Learn*., 8, pp. 279–292, 1992.
C.J.C.H. Watkins, ''Learning from Delayed Rewards'', PhD thesis, Cambridge University, Cambridge, England, 1989.
M.A. Wiering, ''Reinforcement learning in dynamic environments using instantiated information'', *Proceedings of the Eighth International Conference on Machine Learning*, pp. 585–592, 2001.