Proceedings of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, 2006.

ReGiKAT: (Meta-)Reason-Guided Knowledge Acquisition and Transfer
– or –
Why Deep Blue Can't Play Checkers and Why Today's Smart Systems Aren't Smart

Mike Anderson
University of Maryland
College Park
anderson@cs.umd.edu

Tim Oates
University of Maryland
Baltimore County
oates@cs.umbc.edu

Don Perlis
University of Maryland
College Park
perlis@cs.umd.edu

Abstract

We propose a methodology for the design and implementation of systems that will learn to succeed where in the past they have failed. Such systems will know what they are supposed to be doing, know when they are succeeding and when they are not, and be able make targeted changes to their own modules to correct their failures. That is, they will be self-monitoring, self-correcting, and self-guiding learning-and-reasoning systems, advanced active learners able to decide what, when, and how to learn. We also describe some of our pilot studies utilizing this methodology.

Keywords: Metareasoning, learning.

Acknowledgements

We would like to thank DARPA, AFOSR and ONR for partial support of the work reported here.

I Introduction

Imagine yourself at work in your office. You expertly apply yourself to the many tasks at hand, occasionally puzzling over a few unfamiliar matters and quickly dispatching them: one you decide to discard as unimportant; another you recognize as similar to one you dealt with yesterday; a third you hand off to a colleague; a fourth you decide can wait until tomorrow. Later, on the way home, your usual route is clogged with unusually heavy traffic, and you are unsure how to proceed, so you call your spouse for advice. After dinner your spouse suggests a game of cards, and in particular a new game; after some explanations of the rules, and a few questions, you are ready to play, and soon become quite at home with the new game, learning quickly from your mistakes, determining in the process which techniques from other card games still apply, which need to be modified, and which scrapped. Muddling through seems to be a human trademark: we get the job done by hook or crook, we ask for help, or we gracefully bow out. Yet computers are notoriously bad at this.

We propose a methodology for the design and implementation of systems that muddle through, and that can learn to succeed where in the past they have failed. Such systems will know what they are supposed to be doing, know when they are succeeding and when they are not, and be able make targeted changes to their own modules to correct their failures. That is, they will be self-monitoring, self-correcting, and self-guiding learning-and-reasoning systems, advanced active learners able to decide what, when, and how to learn.

We believe this approach can help us move forward on an issue that has plagued research in artificial intelligence since its inception: although we know how to build systems that perform at a human (or near-human) level in certain highly constrained domains (such

as chess, or medical diagnosis), we have not discovered how to endow automated systems with human-level common sense, i.e., the ability to muddle through when things do not fit into neatly prescribed patterns. Yet this is not due to lack of attempts to address this problem. What then is missing? We propose that a large part of the problem is the historical separation of AI into distinct subfields, and in particular the subfields of machine learning and of automated reasoning, which have carried on in almost complete isolation from each other.

In contrast, human agents routinely use learning and reasoning in concert, and especially when faced with the unexpected or unfamiliar. When facing an anomalous situation, a human (i) notices the anomaly, (ii) assesses it in terms of type and importance, and (iii) adopts a course of action with respect to it. We call this process the metacognitive loop (MCL), and it involves both learning and reasoning, in various ways.

It is reasonable, then, to suppose that an automated form of MCL might endow automated systems with considerable robustness. Let us dub such a hypothesized system ReGiKAT, as it would employ REason-GuIded Knowledge Acquisition and Transfer. Our long-range aim is to investigate how far the MCL approach can go toward building a viable ReGiKAT (or Regi for short), and where its limits may be. MCL will allow systems the needed autonomy to function in domains where human supervision cannot be constantly supplied, in part by facilitating the transfer of knowledge learned in one domain to other domains. In fact, this is a prime example of MCL at work, since what works in one domain may—or may not—work in another, and failures in the second domain are anomalies *until they are corrected* by further learning and/or reasoning. The second domain can even be a version of the first domain but at a later time, or in a different context.

II The most important problem in AI

Brittleness is arguably the single most important problem in AI, and perhaps in (computer) systems overall: a system designed for specific tasks fails utterly when faced with an unanticipated perturbation that takes it even slightly outside those task specifications. Yet humans perform admirably under such perturbations, easily adjusting to most minor changes as well as to many major ones.

We define a perturbation as any change, whether in the world or in the system itself, that impacts performance. *Perturbation tolerance*, then, is the ability of a system to quickly recover—that is, to re-establish desired/expected performance levels—after a perturbation. To achieve this, a perturbation-tolerant system should not only notice when it isn't behaving how it ought or achieving what it should, but be able to use this knowledge to make targeted alterations to its own modules. Such changes can be as simple as recalibrating its sensors, or as complex as training new (or retraining old) behaviors, changing its rules of inference, learning new words and concepts, even adopting different basic ontologies in different circumstances.

While it may often be possible to anticipate the kinds of problems a system will face over its lifetime, and build in specific mechanisms to handle these issues, we doubt this will prove, in the long run, to be the most effective strategy. We believe, in contrast, that efforts should be aimed at implementing mechanisms that help systems help themselves. The goal should be to *increase* their agency and freedom of action in responding to problems, instead of limiting it and hoping that circumstances do not stray from the anticipations of the system designer. We should be creating self-aware, self-guided learners. Indeed, we believe that such metacognitive skills are the key to achieving near-human-level (or, indeed, any useful kind of) perturbation tolerance. Metacognitive learners would be advanced active learners, able to decide what, when, and how to learn.

An MCL-based system will know what it is attempting to do, so that it can determine when things are not going well, instead of blindly following its programming over the proverbial cliff—as did one of the DARPA Grand Challenge entries which kept trying to drive through a fence it could not see. If that system had known it was supposed to make forward progress and noticed that it was not doing so, this would have been the first step to overcoming the problem. Or consider the case of the satellite given the command to turn and look at some object away from Earth, but not told to turn back to Earth when finished. Once the satellite turned, there was no way to feed it further commands, and the satellite was nearly lost. In contrast, a system that had general expectations for its operation (frequent communication from Earth), based on the sort of system it was, might have been able to use this knowledge to recover from such mistakes. More generally, MCL will allow systems the needed autonomy to function in domains where human supervision cannot be constantly supplied.

Our general strategy, then, in working toward this goal has been to equip artificial agents with the ability to *notice* when something is amiss, *assess* the anomaly, and *guide* a solution into place (the NAG cycle). This basic strategy of self-guided learning (i.e., the metacognitive loop) involves the system monitoring, reasoning about, and, when necessary, altering its own decision-making components. Indeed, in our view this is largely what perturbation-tolerant commonsense reasoning consists in, rather than in finding special clever solutions to thorny problems.

Although it is of course the case that MCL will be most effective when specific solutions to problems can be implemented, it is worth noting that 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—and speed learning in—different types of systems, including reinforcement learners, natural language human-computer interfaces, commonsense reasoners, deadline-coupled planning systems, robot navigation, and, more generally, repairing arbitrary direct contradictions in a knowledge base.

III A sketch of the general MCL algorithm: the key role of expectations

The primary end result of the MCL approach is systems with continually evolving—and improving—decision-making components. To achieve this, the knowledge base (KB) of the system continually evolves as new beliefs are asserted (whether due to inference, perception, or other processes) and old beliefs are retracted (or rather, simply not promoted to the next time-step—we speak of these as being disinherited). In addition, MCL systems can retrain or otherwise adjust their non-symbolic components, or even create new components and bring them online.

Here is a high-level sketch of how MCL plays out in algorithmic terms. As noted above, the core of MCL involves three major processes: noting anomalies, assessing them, and guiding solutions into place. Anomalies arise from expectations: an anomaly is an expectation that is not met. This can include, however, tacit, subsymbolic, or implicit expectations. That is, a given situation might trigger the response "X unexpected" even if there was no explicit representation of the expected counterpart, $\neg X$, beforehand.

An expectation can be represented in the form Exp(E), or as a simple belief E. The following default rule relates simple beliefs and expectations. It says simply that if you *expect* E, you can generally come to *believe* E, unless it is contradicted by other beliefs or observations.

$$\begin{array}{ll} \text{t} & : \text{Exp(E)} \\ \text{t+1} & : \text{E} \end{array}$$

Condition for above rule: None of the fol-

lowing formulas belong to the KB at time t: $\neg E$, $Obs(\neg E)$, $Obs(F)$ where E is $\neg F$.

IV How MCL helps

MCL can enhance performance for two related reasons. First, it can monitor and influence on-line performance even without making any basic changes or improvements to action-producing or decision-making components. An example of this would be noticing that progress on a task has stopped (i.e. that the system is "stuck") and directing specific efforts to getting "un-stuck", or simply moving on to a different task. Second, and more powerfully, MCL can direct the system to actively learn something that it (apparently) doesn't know, or has gotten wrong. Since there is evidently a great deal that can be learned, depending on the system and the scenario, MCL in this guise is best understood as a principled method of *organizing* and *controlling* learning—deciding whether to learn, what to learn, with what methods, and (importantly) when to stop. An example of this latter ability would be noticing that a problem in processing a given user command appears to be caused by ignorance of a certain word, and taking steps to learn the unknown word.

Both of these abilities are crucial to improving the perturbation tolerance of a given system, and they generally work in concert. Thus, for instance, we have shown that an MCL-enhanced reinforcement learner can—by choosing when to ignore anomalies, when to make minor on-line adjustments, and when to order re-learning of its action policy—always perform *at least as well as*, and in many cases *significantly out-perform*, a standard reinforcement learner when operating in a changing world [4].

V Implemented MCL systems

MCL-enhanced reinforcement learning. In a simple demonstration of this idea, we built a standard reinforcement learner (we tested Q-learning [19, 20], SARSA [18] and Prioritized Sweeping [13]), and placed it in an 8x8 world with two rewards—reward 1 (r1) in square (1,1) and reward 2 (r2) in square (8,8). The 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. In turn 10,001, the values of the rewards were abruptly changed. See [4] for a complete account of the experimental design and results.

We found that the perturbation tolerance (i.e. the post-perturbation performance) of standard reinforcement learners varied considerably, and was negatively correlated to the degree of the perturbation—the bigger the change, the worse they did. Roughly speaking, the problem was that such learners in effect have to unlearn what they have learned in order to then learn something new; they cannot make the decision that what they have learned is no longer working and that they should jettison it and start from scratch. Thus they go through a potentially slow unlearning curve before they can begin learning afresh.

However, even a very simple MCL enhancement to reinforcement learning, that simply discards what it has learned after noticing a few discrepancies, and starts over, perform significantly better than the non-MCL versions. Moreover, a more sophisticated version of MCL, that generates and monitors expectations for performance (average reward per turn, average time between rewards, and amount of reward in each state), and chooses between the available methods (of doing nothing, making an on-line adjustment, and relearning its policy, in light of its assessment of the anomalies) performed best overall.

MCL-enhanced navigation. Another agent that we have been developing uses a neural net for navigation; however it also has a monitoring component that notices when navigational failures (such as collisions) take place, and records these and their circumstances. It is then able to use this information to assess the failures and make targeted changes to the neural net, including starting with a different set of weights, or re-training on a specific set

of inputs. The agent exhibits better behavior while training, and also learns to navigate effectively more quickly [11].

Although both the above systems are relatively simple, they do illustrate the ways in which self-monitoring and control can help systems maintain performance in the face of changes, and, more particularly, they demonstrate cooperation between the ability to initiate new actions, and the ability to initiate new learning. Still, one has to expect that as the scenarios, and the systems themselves, become more complex, more sophisticated and expressive reasoning mechanisms will be required to usefully assess and appropriately respond to anomalies. And these reasoning mechanisms must not only be able to be used in the *service* of perturbation tolerance, but must *themselves* be perturbation tolerant. It is with such considerations in mind that we have been working on perturbation-tolerant logical reasoners, based on active logic [8, 9, 10].

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[1]—between the information it gathers from the world, and the beliefs resident in its knowledge base (KB).

As argued in [14], contradictions of some sort are practically inevitable; but only direct contradictions need be "visible" to a reasoning system. This is the thrust of much of our past work on time-situated commonsense reasoning. And it lends itself nicely to noting anomalies: when a direct contradiction (e.g., between an expectation and an observation) appears in the KB, something is amiss.

Our formal approach to effective reasoning in the presence of contradictions—active logic—is motivated in part by the observation that all reasoning takes place step-wise, in time. This allows an agent to maintain control over, and track, its own reasoning processes.

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

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.

By endowing an active logic with a "conflict-recognition" inference rule such as that in (V), *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 meta-predicate: it is about the course of reasoning itself (and yet is also part of that same evolving history).

$$\begin{array}{ll} \texttt{i} & : \quad \texttt{P, } \neg\texttt{P} \\ \texttt{i+1} & : \quad \overline{\texttt{Contra(P, } \neg\texttt{P, i)}} \end{array}$$

The idea then is that, although an indirect contradiction may lurk undetected in the knowledge base, 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.

These temporal and metacognitive aspects make active logic systems more flexible than traditional AI systems and therefore more suitable for reasoning in noisy, dynamic and inconsistent environments. Together with negative introspection (the ability to know what is currently *not* in the KB), a robust ability to continue to reason normally as wffs are added, changed or deleted from its knowledge base, and the ability to initiate external actions (see [16] for details), these features make it a good candidate to provide the symbolic world-reasoning and meta-reasoning required for a real world system [6, 17].

MCL-enhanced human-computer dialog. One of the most important application areas for active logic has been natural language human-computer interaction (HCI). Natural language is complex and ambiguous, and communication for this reason always contains an element of uncertainty. To manage this un-

certainty, 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.

We contend that the ability to engage in such meta-language (asking for help with or advice about language), and to use the results of meta-dialogic interactions to help understand otherwise problematic utterances, is the source of much of the flexibility displayed by human conversation [15]. 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 engage in meta-reasoning and meta-dialog [3, 12].

A recent advance we have made along these lines was to enhance the ability of our HCI system to more accurately assess the nature of dialog problems, and to engage in meta-dialog 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 doesn't 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 [3, 12]. It can use these same methods to learn new commands, so long as the new command can be explained in terms of (including being compounded from) commands it already knows. The system is currently integrated with a simulated home management domain, where it can process—and interactively resolve mistakes in—user commands, thereby allowing the user to control lights, a digital video recorder, a pool heater; and to play chess.

VI New domain

As noted above, we have built and tested simple pilot versions of MCL in many specialized domains: reinforcement learning, natural language dialog, simbot navigation, and commonsense (nonmonotonic) reasoning However, what we propose now is to amplify the domains we have been using so that they are more complex and more dynamic; require multiple learning-modalities (not just RL, but also neural nets and other inductive learners, etc.); and include real-time performance requirements. This will allow a more thorough testing of the MCL approach.

The domain we have chosen for our initial, more sophisticated version of Regi is the multi-player tank game Bolo [7]. The basic idea behind Bolo is fairly straightforward: each player has a tank in a 2-dimensional environment, based on an island (surrounded by water, and only accessible by boat), with the following terrain types: roads, grass, swamp, rubble, craters, water and walls. The terrain can be changed at run-time by exploding mines to create craters, by shooting at walls to create rubble, and by using trees to build roads, bridges, or walls. There are bases for refueling, and pillboxes that shoot at unfriendly tanks. Pillboxes can be captured and made "friendly", and friendly pillboxes can be picked up and moved around. The object of the game is to capture all the pillboxes and bases, before the other players do.

Bolo has several characteristics that make it a good choice for our research. The first is that one has complete control over the starting terrain configuration (the "map"), so that it can be made quite simple, or quite complex, and of course an agent can be trained in one scenario and tested in another. The second is that, although it is designed to be a multi-player game, it can be played in single-player, or "practice" mode. Furthermore, it has both real-time constraints, as well as the need for higher-order planning. Thus, Bolo can be adjusted to be very easy, or very complex, with many gradations in between.

Performance metrics for perturbation-tolerant

systems. In previous work, we developed a method of measuring the complexity of an environment, including its variability and volatility—the degree to which it changes from place to place and over time—that is directly applicable to the Bolo domain [1, 2]. This measure of complexity can be easily combined with task-based performance measures (e.g. task completion over time) to measure the ability of the agent to deal with increasing degrees of environmental complexity and change.

VII Comparison with Current Technology

Many techniques under the general topic of reinforcement learning [18], such as neuro-dynamic programming [5], have been developed for acting under uncertainty. However, these techniques are very different from the proposed work in that they focus on action selection, not symbolic reasoning, and adapt to non-stationarity only by continually training. The latter requires continual exploration, or deviation from the optimal action policy, whereas MCL systems act optimally until they notice that something is wrong and then take remedial actions focused on the problem at hand.

What is distinctive about MCL is that it makes it possible to adapt learning and reasoning over time to improve tolerance to perturbations. Consider Regi, who might learn to navigate optimally in his environment using neuro-dynamic programming. If the environment changes, Regi's value function may no longer suggest optimal actions. However, this is precisely the kind of thing the oversight module can *notice* by, for example, keeping empirical data on discounted rewards obtained from various states. If the empirical data begin to diverge from the learned value function, there is either a problem (when the actual values are lower than expected) or an opportunity (when the actual values are higher than expected).

In either case, the system can *assess* the discrepancies, perhaps to identify regions of the state space where they occur, and *guide* a solution into place. The solution might be to take exploratory actions in those regions of the state space where the value function is imprecise, or to simply increase the exploration probability globally if no coherent region of the state space is identifiable. In the former situation, Regi continues to behave optimally in those states for which the value function is accurate, limiting exploration, and thus short-term loss of reward, to those states corresponding to aspects of the environment that have changed. If all else fails, Regi can ask for advice, which might take the form of a list of states paired either with their optimal actions ("you should turn left in that state") or constraints on their values ("the value of that state is between 10 and 12").

References

[1] Michael L. Anderson. A flexible approach to quantifying various dimensions of environmental complexity. In *Proceedings of the Performance Metrics for Intelligent Systems Workshop*, 2004.

[2] Michael L. Anderson. Specification of a test environment and performance measures for perturbation-tolerant cognitive agents. In *Proceedings of the AAAI Workshop on Intelligent Agent Architectures*, 2004.

[3] Michael L. Anderson, Darsana Josyula, and Don Perlis. Talking to computers. In *Proceedings of the Workshop on Mixed Initiative Intelligent Systems, IJCAI-03*, 2003.

[4] Michael L. Anderson, Tim Oates, Waiyian Chong, and Don Perlis. Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance, submitted.

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

[6] Manjit Bhatia, Paul Chi, Waiyian Chong, Darsana P. Josyula, Michael O'Donovan-Anderson, Yoshi Okamoto, Don Perlis, and K. Purang. Handling uncertainty with active logic. In *Proceedings, AAAI Fall Symposium on Uncertainty in Computation*, 2001.

[7] Stuart Cheshire. An experiment in real-time networking. Master's thesis, Cambridge University, Cambridge, UK, 1989.

[8] J. Elgot-Drapkin. *Step-logic: Reasoning Situated in Time.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 1988.

[9] 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, Univ of Maryland, UMIACS and CSD, 1993.

[10] J. Elgot-Drapkin and D. Perlis. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98, 1990.

[11] Ken Hennacy, Nikhil Swamy, and Don Perlis. RGL study in a hybrid real-time system. In *Proceedings of IASTED NCI 2003, Cancun, Mexico.*

[12] Darsana Josyula, Michael L. Anderson, and Don Perlis. Towards domain-independent, task-oriented, conversational adequacy. In *Proceedings of the Eighteenth international Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1637–8, 2003.

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

[14] D. Perlis. On the consistency of commonsense reasoning. *Computational Intelligence*, 2:180–190, 1986.

[15] D. Perlis, K. Purang, and C. Andersen. Conversational adequacy: mistakes are the essence. *Int. J. Human-Computer Studies*, 48:553–575, 1998.

[16] K. Purang. *Systems that detect and repair their own mistakes.* PhD thesis, Department of Computer Science, University of Maryland, College Park, Maryland, 2001.

[17] K. Purang, D Purushothaman, D. Traum, C. Andersen, and D. Perlis. Practical reasoning and plan execution with active logic. In *IJCAI-99 Workshop on Practical Reasoning and Rationality*, 1999.

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

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

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