Are Inductive Learning Algorithms Safe in a Continuous Learning Environment?

Mike Barley and **Michael Goebel** and **Hans Guesgen** and **Pat Riddle**
Department of Computer Science
University of Auckland
{barley, mgoebel, hans, pat}@cs.auckland.ac.nz

Introduction

As intelligent software agents are pushed out into the real world and increasingly take over responsibility for important tasks, it becomes important that there are ways of guaranteeing that the agents are safe. But what does "safe" mean in this context. Certainly if the agents are interacting with people then to a certain extent, "safe" will mean that people can trust their model of the agent to be reliable and that the future behavior of the agent is predictable. Specifically, if the agent exhibits learning then the user needs to be able to predict how that learning affects the agent's future behavior. Right now, the main learning entity that most people interact with are other people. At least initially, people will assume that learning agents' behavior will have many of the same properties as shown by human learning behavior.

In this paper we will look at one such property, learning's preservation of competence. People assume that learning improves performance, rather than degrades it. In particular, people assume that if someone knows how to solve a particular problem and then learns more about that domain, that they will still be able to solve that problem. Occasionally, learning interferes with problem solving performance and this might result in the learner taking longer to solve the problem than before. However, we normally expect that the learner will still be able to solve the problem[1].

This places an important constraint on agent learning behavior. People will be unwilling to work with learning agents if the agent's learning makes its behavior unpredictable. Specifically, learning should not decrease the range of problems that an agent can solve.

[1]This paper ignores learning style and ability differences between different age groups and different cultures. For example, primary school age children's competence in one skill can degrade while they learn an allied skill (e.g., competence in addition temporarily degrades while they learn subtraction skills). While these differences will eventually have an impact on agents, they are beyond the scope of this paper.

Continuous Learning

We're interested in how this can be guaranteed in environments where the agent is continuously learning and comes up with deliberative plans but has limited resources. *Continuous learning* means both that the agent will continue to learn throughout its lifetime and that its learning should immediately affect its deliberative behavior. It is this latter meaning that we focus on in this paper. Limited problem-solving resources mean that the agent must learn reasonably quickly from its experience. We believe that these constraints are reasonable in many situations (e.g., a command agent for a spacecraft).

Continuous learning is important in environments where mistakes can be costly and lessons learned from mistakes need to be applied as soon as possible. A trivial example is playing with fire, a system should not need to "burn" its fingers more than once to learn not to touch burning logs. Continuous learning is also important in situations where learning is accumulative (i.e., builds upon early learning), without the immediate application of the new knowledge, learning opportunities are lost. An example of this is where the system only learns from successes and problems are getting progressively harder. If the search control knowledge extracted from each example is not immediately made available, the system will be unable to solve subsequent problems and therefore unable to learn from them.

Our conjecture is that inductive learning approaches will not be conservative enough for such agents and that deductive learning approaches will be needed. However, before trying to use current deductive approaches we want to make sure that the current inductive approaches to learning search control knowledge for planners are indeed not safe and have some idea of how unsafe they can be.

This paper takes a preliminary look at the current state of affairs for one learning agent: the **blackbox**(Kautz & Selman 1999) planner with a specific learning component(Huang, Selman, & Kautz 2000). **blackbox** was one of the top competitors at the AIPS-98 planning competition(McDermott 2000). It is also one of the few modern planner for which a learning component has been created. The paper by Huang,

Selman, and Kautz(Huang, Selman, & Kautz 2000) describes experiments with the learner that show that the learned search control rules (SCRs) can dramatically improve blackbox's performance (speeding up the planner's performance on some problems by over two orders of magnitude). What is missing, from our perspective, is how well the learner preserves blackbox's coverage of past problem solved.

In the next section we very briefly describe **blackbox** and its learner. In the section following that, we describe **blackbox**'s current approach (using a sandbox for the learned rules) to making its learning safer and why it's not appropriate for continuous learning environments. In the penultimate section, we describe our experiment and in the last two sections we discuss future research and conclude.

blackbox and its Learner

blackbox allows one to specify a number of different planners to be used in solving a problem. In our experiment, we use the **graphplan**(Blum & Furst 1997) planner. **graphplan** can detect that certain goals are mutually exclusive (*mutex*-ed) based on the problem's constraints. This enables it to avoid working on problems which it can show are impossible for it to solve. However, its algorithm for detecting *mutex*es is incomplete.

blackbox's learner is a variant of Quinlan's **FOIL**(Quinlan 1996) that learns relational search control rules. These search control rules are expressed in a temporal logic language which only references world state predicates and final goal predicates. This avoids the problem of SCRs interfering with each other at the meta-level(Barley & Guesgen 2001).

Sandbox Safety

One common protocol for providing safe learning, which we call the *sandbox* learning protocol, is to quarantine the learned SCRs from the performance engine. Even if the learning is going on in real-time, the new rules are kept isolated from the performance part of the system. The learner proposes new candidate sets of SCRs and these candidate sets are screened by a quality assurance process. Only when a candidate set passes the quality assurance process can it become the new SCR rule set.

blackbox has a number of runtime switches that enable **blackbox** to select from a number of different learning protocols. One combination allows **blackbox** to follow a variation of the sandbox protocol. The variation[2] described next is the one used in the experiments described by Huang, Selman, and Kautz(Huang, Selman, & Kautz 2000) which we mentioned in the section. We first provide an overview of **blackbox**'s variation and then we look at it in slightly more detail.

[2]What we provide here is a simplified description of the variation. This simplified version is functionally equivalent to the actual protocol used.

The **blackbox** variation of the sandbox learning protocol uses a training problem set to both learn the candidate rule set and to run the quality assurance test. As in the standard sandbox learning protocol, the **blackbox** variation keeps the learned SCRs separate from those actually used by the planner. We call the set of SCRs that are being learned, the *candidate* rule set. We call the set of SCRs used by the planner, the *performance* rule set. The **blackbox** sandbox protocol does two runs over the problem set. The first run creates the candidate search control rule set and the second run verifies that none of the candidate SCRs reject any solution found by **blackbox** during the first run. Any rule in the candidate set that rejects a solution is pruned away.

In the first run, **blackbox** runs sequentially on the problems in the problem set. For each problem in the problem set, if **blackbox** solves that problem then it uses the solution as a collection of training examples from which it creates new SCRs, which are added to the candidate rule set.

In the second run, the learner verifies the candidate rule set against the solution found in the first run for each problem in the training problem set. Any rule that rejects that solution is pruned away. The search control rule set produced by this second run is guaranteed not to reject any solution that the planner would find with the original performance rule set for the training problem set. However, there still may be other problems outside of that training set where the new performance rule set rejects all the solutions but where the original performance rule set does not.

The sandbox protocol for safe learning is reasonable for many situations. In fact, it is the standard protocol one uses for software releases. However, it doesn't satisfy our goals for a continuously learning agent. Its learning is not immediately available to affect the agent's behavior, instead the newly learned rules are placed in a candidate rule set isolated from the planner. Additionally, when learning continuously, there are questions about how large to make the sandbox. The smaller the sandbox is, the larger the chance will be of adding a new SCR to the performance engine that will prevent it from solving some future problem. The larger the sandbox is, the longer the delay will be between learning the rule and deploying it in the performance engine. It's not obvious how to automatically balance these concerns.

This paper examines what happens to the safeness of the learned SCRs if we adopt an incremental protocol for learning. In this approach, the search controls are added to the current rule set as they are learned.

The Experiment

Our experiment is to explore how safe **blackbox**'s learning algorithm is when using an incremental learning protocol. In this protocol, search control rules learned from solving a problem are added to the current search control rule set which will be used in attempting

to solve the next problem. This incremental learning protocol is the approach we expect to be used by agents that can continuously learn from their experience.

For this experiment, we took one of the domains, **mprime**, that **blackbox** attempted in the AIPS-98 competition. In this experiment, we did two runs. The first run (which we call the **base** run) is without any search control rules. This partitions the problems into three parts: problems **blackbox** solved, problems **blackbox** can definitely not solve (mutex-ed), and problems where **blackbox** ran out of resources before solving the problem or finding out that it was unsolvable. Problems where **blackbox** ran out of resources, we call *killed* problems. We ran **blackbox** without any time limits but there was a 256 MB memory limit. All killed problems ran out of memory. **blackbox** can à priori detect some of the problems that it definitely cannot solve from its SAT encoding[3] of the problem. If it cannot find *mutex*-free goals then it won't be able to solve the problem. We call these problems, *mutex*-ed problems.

We then did a hundred more runs, each one using an incremental learning protocol. For each of these learning runs, the problems were randomly ordered. Our incremental learning protocol consisted of starting with an empty search control rule set and trying to solve the first problem in the selected random ordering. If the problem is not solved then **blackbox** does not attempt to learn any search control rules. If the problem is solved then **blackbox** attempts to learn new rules[4]. If any rules are learned, these are added to the performance rule set. The next problem in the selected ordering is then attempted and this incremental learning process continues until we have run out of problems.

Figure 1 shows the effects of incremental learning on the planner's competence. Specifically, it shows what happens to the planner's ability to solve problems as it incrementally learns from its experience. As we see on the rightmost column, with no SCRs (and no learning) **blackbox** was able to solve 15 of the 30 problems and found it was definitely unable to solve one problem. For the remaining 14 problems it ran out of resources. The first row is the most interesting and informative row. From the first row, we see that if we use incremental learning that, on average, 4.1 of those 15 problems remained solved, while 10.2 become definitely unsolvable (i.e., mutex) and for 0.7 of them, the planner will run out of resources. Unfortunately, the information in the second row is ambiguous. We have no idea how many of those 5.1 problems (if any) would originally have been solved if the memory limit had been sufficiently high. We also know nothing about which of the 8.1 problems would have been originally solvable if there had been enough memory and how many of those would have remained solvable (given enough memory). The third row is uninteresting.

The ambiguity of the second row's information affects our ability to predict how much incremental learning would have affected **blackbox**'s performance if it had had more memory. We will only look at the best case for incremental learning, i.e., where the lowest percentage of solved problems became mutex-ed problems. The best case would be if the 5.1 problems that went from killed to mutex (on average) would not have been solvable even with unlimited memory and that the 8.1 problems (on average) that went from killed to killed would have been solved both with out and with incremental learning if there had been enough memory. If this had been the case then we would have had 23.1[5] problems that were solved without incremental learning and 13.1 problems that were still solved with incremental learning. So, in the best case, approximately 40% of the problems that were solvable without learning would become mutex using incremental learning. From this it is fairly clear that **blackbox**'s learning algorithm is not safe for this incremental learning protocol.

Future Research

In our past work, we showed that using certain types of meta-level predicates in the preconditions of SCRs can lead to learned rules having counter-intuitive effects upon the performance engine(Barley & Guesgen 2001). These counter-intuitive effects could cause unsafe behavior.

In this paper, we showed that avoiding using these types of meta-level predicates in SCR preconditions is not sufficient to guarantee that learned rules will not cause unsafe planner behavior. We showed that using the incremental learning protocol with an inductive learning algorithm can lead to unsafe behavior, namely, loss of domain coverage.

Our goal is to develop a learning algorithm which is guaranteed to be safe to use with the incremental learning protocol. Our next step is to look at the current state of deductive SCR learners. In the past, deductive learners used EBL to derive approximately sufficient conditions for search control actions. They were approximations[6] because they normally left out portions of the full explanations. We will be looking at current deductive SCR learners to see if that is still true and also to explore sound ways of generalizing the explanation beyond the example it explains.

Conclusions

In this paper, we attempted to accomplish four things:

[3] **blackbox** initially transforms a planning problem into a boolean SAT encoding of a constraint satisfaction problem, we will not discuss this further in this paper. Blum and Furst describe this in (Blum & Furst 1997).

[4] Learning from failure has not been implemented in **blackbox**

[5] The 15 problems that were solved with the current memory limit and the 8.1 killed problems we are assuming would have been solved if given enough memory.

[6] EBL-derived SCRs were usually both under-generalized and over-generalized.

/ with SCRs no SCRs	solved	killed	mutex	# totals #(no SCRs)
solved	4.1(2.1)	0.7(0.6)	10.2(1.9)	# 15
killed	0.8(0.8)	8.1(1.7)	5.1(2.2)	# 14
mutex	0(0)	0(0)	1(0)	# 1
totals (with SCRs)	4.9(2.9)	8.8(2.3)	16.3(4.1)	# 30

The entries in the boxes are ‘‘mean(standard deviation)’’.

Figure 1: Problem-Solving Performance with and without SCRs

1. Argue that the safeness of an agent, to some extent, depends upon how well the system's behavior mirrors the user's expectations of its behavior. In this case, the expectation is that learning should not decrease the range of problems the system can solve.
2. Identify an important class of situations where the current approach to safe learning (the sandbox learning protocol) is not appropriate. Namely, situations where continuous learning is important when it is desirable to be able to immediately deploy the learned rules.
3. Show that inductive learning is not always safe for learning SCRs using the incremental learning protocol.
4. Suggest where we should go next in our search for a learning algorithm that will be safe (with respect to preservation of domain coverage) using the incremental learning protocol.

Acknowledgments

We thank Yi-Cheng Huang and the members of the **blackbox** group for making their planner/learner code available to us and for taking the time to answer our many questions. We also thank Ella Atkins and Diana Spears for their many constructive comments and suggestions.

References

Barley, M., and Guesgen, H. 2001. Meta-level preconditions: An obstacle to safe SCR learning. In *IC-AI 2001 Conference Proceedings*. LasVegas, Nevada,USA: 2001 International Conference on Artificial Intelligence.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence*.

Huang, Y.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *Proceedings of the Seventeenth International Conference on Machine Learning*.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 318–325.

McDermott, D. 2000. The 1998 ai planning systems competition. *AI Magazine* 21(2):35–55.

Quinlan, J. 1996. Learning first-order definitions of functions. *Journal of Artificial Intelligence Research* 5:139–161.