Journal of Intelligent Information Systems, 9, 261–275 (1997)

Knowledge-directed Adaptation in Multi-level Agents

JOHN E. LAIRD, DOUGLAS J. PEARSON laird@umich.edu
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2110

SCOTT B. HUFFMAN huffman@tc.pw.com
Price Waterhouse Technology Center, 68 Willow Road, Menlo Park, CA 94025

Abstract. Most work on adaptive agents have a simple, single layer architecture. However, most agent architectures support three levels of knowledge and control: a reflex level for reactive responses, a deliberate level for goal-driven behavior, and a reflective layer for deliberate planning and problem decomposition. In this paper we explore agents implemented in Soar that behave and learn at the deliberate and reflective levels. These levels enhance not only behavior, but also adaptation. The agents use a combination of analytic and empirical learning, drawing from a variety of sources of knowledge to adapt to their environment. We hypothesize that complete, adaptive agents must be able to learn across all three levels.

Keywords: Learning, adaptation, Instruction, Error Correction, EBL, Multi-level agents, Soar

1. Introduction

Over the last ten years, there has been a convergence in the design of architectures for constructing intelligent autonomous agents that must behave in complex environments. Many architectures support three levels of knowledge with corresponding levels of control: a reflex level made up of independent rules or a finite state machine, a deliberate level that provides sequencing, simple decision making, and the ability to pursue multiple goals, and a reflective layer for deliberate planning. In theory, each of these levels can draw from a variety of sources of knowledge to adapt to their environment. However, the vast majority of work on adaptive agents has emphasized only a single source of knowledge, reinforcement, with improvement only at a single reflex layer. The advantage of these approaches is that they can make use of simple feedback that is available in many environments; however, in return, agents using these techniques are limited in the complexity of behavior that they can generate and learn.

In order to extend adaptivity to more complex agents — agents with multiple levels of knowledge and control — we have developed a more deliberate approach which has been instantiated within two systems built within the Soar architecture (Laird, Newell and Rosenbloom, 1997, Lehman, Laird and Rosenbloom, 1996, Laird and Newell, 1993, Rosenbloom, et al., 1996): Instructo-Soar (Huffman and Laird, 1995) and IMPROV (Pearson, 1996, Pearson and Laird, 1996). We find that agents with multiple levels of knowledge and control are not only more able to achieve complex goals, they also can use these layers to adapt to their environment. These agents use analytic and empirical learning techniques to draw on multiple sources of knowledge, including external instruction, internal domain theories, and past behavior.

We have focused on systems that learn at the two higher levels, deliberate and reflective, within agents which employ all three levels of control. This research complements the work that has been done to combine empirical and analytic learning methods (such as EBNN (Mitchell and Thrun, 1993), EBC (DeJong, 1995) and EBRL (Dietterich and Flann, 1995)) that typically focus on agents that use only a single level of control.

Although our approach is deliberate in many ways, it is distinguished from deliberate approaches that treat the knowledge of the agent as a declarative structure that can be examined and modified at will. Such first-order declarative access to the agent's complete knowledge base poses significant computational complexity problems as the agent's knowledge grows to cover complex environments. In contrast to our approach, these declarative/deliberate methods are typically restricted to noise-free domains with instantaneous action (Gil, 1991, Ourston and Mooney, 1990) which require relatively small domain theories.

In the remainder of this paper, we present our general approach to agent design and adaptation within the context of Soar. We then illustrate this approach by examining two integrated systems; Instructo-Soar, which learns to extend its knowledge through situated instruction; and IMPROV, which learns to correct errors in its knowledge through interaction with its environment. Instructo-Soar uses analytic learning over problem solving at the reflective level to acquire new task knowledge at the deliberate level. IMPROV uses largely empirical learning at the reflective and deliberate levels to identify and correct knowledge learned from either incorrect instructions or incorrect generalizations of correct instructions. IMPROV's recovery method can in turn be directed and guided by knowledge gained from further instructions processed by Instructo-Soar. We have combined IMPROV and Instructo-Soar to produce a single integrated system, so that instructions can speed up empirical learning and empirical learning can correct errors learned from instruction.

2. Multi-Level Agent Structure

Architectures that support reactive and deliberate reasoning in complex environments are typically composed of multiple levels of computational systems. The reason for this is that reactive systems naturally have an emphasis on fast but limited responses which can be realized by computational processes that do not lend themselves to the more deliberate and open-ended computation of planning or reflective systems. Our discussion assumes that an agent consists of the three levels as shown in Figure 1. Many architectures have structures similar to this; however we will concentrate on Soar in our discussion because it is the basis for both of our learning systems and it has some important properties that support the types of learning we wish to demonstrate.

1. **The reflex level:** This is where knowledge is directly and automatically retrieved based on the current situation. The processing is often limited to simple stimulus-response activity, although sometimes this level is used to parse input into more abstract descriptions that are used at the deliberate level. Most adaptive agents that learn have only this level with the knowledge encoded in rules, finite-state automata, or similar fixed network structures. These types of computational frameworks usually have bounded computational time for responding to changes in the world, which make them suitable

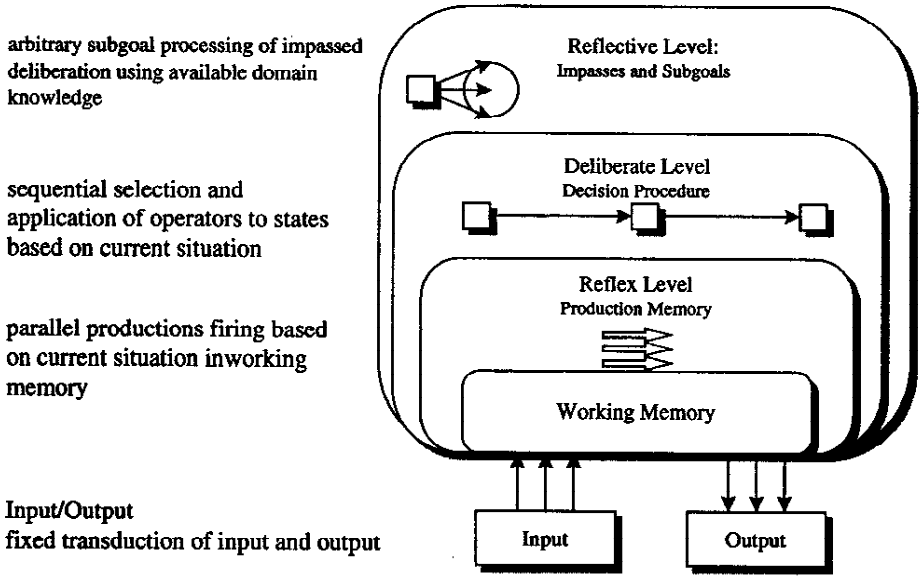


Figure 1. Levels of Processing

for quick, compiled responses. A common approach is to create sets of modules that perform specific low-level functions that require tight interaction with the world, such as moving forward in a mobile robot. These modules are then dynamically strung together by the deliberate level to generate more complex behavior that responds to multiple, dynamic goals.

Soar's knowledge for this level is encoded in rules. The conditions of these rules are dynamically compiled into a discrimination network using the RETE match algorithm (Doorehbos, 1993, Forgy, 1982). Most rule-based systems use a fixed conflict resolution scheme to select and fire a single rule on each cycle. In contrast, Soar fires all of its newly matched rules in parallel. Thus, Soar uses its rules as a parallel associative memory, and postpones all decision making to the next, higher level. This separation of parallel memory access and decision making into two levels is important in supporting learning because once a new rule is learned, there does not have to be any additional learning as to when to use it in conjunction with other knowledge. The knowledge of when to use a rule is already embedded in its conditions.

2. **The deliberate level:** This is where knowledge is used to decide what to do next based on the current situation and goal, and is often called the sequencing level. This level allows for the explicit comparisons of alternatives and for the integration of multiple chains of reasoning, both of which are not possible at the reflex level. While the reflex level will usually be built out of parallel, independent computational elements, the deliberate level usually has sequential behaviors where there are data dependencies

between the computational elements. The component behaviors that this level selects are then the modules of the reflex level.

In Soar, this level consists of the selection, application, and termination of deliberate operators. Instead of having a monolithic data structure representing the pre-conditions, actions, and post-conditions of an operator, Soar uses its reflex level for these functions. Thus, the pre-conditions of an operator are encoded in rules that match against the current situation and propose the operator when it is appropriate. Additional rules can compare proposed operators and test the current situation and goals, and create *preferences* which rate the alternative operators. A fixed decision procedure interprets these preferences and decides on the best alternative. Once an operator is selected, additional rules perform the actions of the operator. Finally a rule tests that all actions have been completed and terminates the operator. Rules provide a rich language for supporting deliberative reasoning. It is trivial to encode complex disjunctive and conjunctive conditions for selecting, comparing, applying and terminating operators. Thus, conditional execution of operators is supported, as well as operators that require multiple steps over time.

So in contrast to most rule-based systems that select a single rule to fire using conflict resolution, Soar uses its rules to select a single operator, and then additional rules to apply it. Soar's approach allows task-dependent knowledge to select its deliberative acts, while other rule-based systems rely on a fixed conflict resolution scheme.

The deliberate level has some advantages for learning that are listed below:

(A) Selection knowledge (when and why to do something) can be learned and corrected independently of implementation knowledge (what to do). This is possible because the knowledge for selecting an operator is represented, and can be learned, independently of the knowledge about how to perform the operator.

(B) Learning can be incremental, so that existing knowledge does not have to be modified (Laird, 1988). This greatly simplifies learning so that it is not necessary to identify which piece of knowledge is incorrect, only which decision was incorrect.

(C) Conflicts or gaps in the agent's knowledge can be detected when there is insufficient knowledge to decide which operator to select, or insufficient knowledge on how to apply the current operator.

3. **The reflective level:** This is where arbitrary knowledge is used to either select the next deliberate action, or decompose a complex action into smaller parts. In general, this is where planning, analogical reasoning, internal explorations, and other meta-level activities are performed. Many systems use a completely separate computational system for the reflective level. This level may generate plans that are then dispatched to the lower levels for execution.

   In Soar, the reflective level is not a separate module. This level is invoked automatically when the deliberate level reaches an *impasse* and cannot make progress, either because there is insufficient knowledge to select or apply an operator. In response to an impasse, Soar generates a new sub-problem, whose goal is to resolve the impasse. Instead of using a different computational scheme, Soar once again uses rules and operators;

however, instead of working on a task, they are now being applied to the meta-level problem of resolving the impasse.

The reflective level has some advantages for learning that are listed below:

(A) The agent can create hypothetical situations distinct from its current sensing that it can reason about and learn from. These hypothetical situations may be combinations of previously encountered situations where the combination itself has never been encountered. This is important in learning from instruction where an instructor may describe potentially dangerous situations that the agent should avoid.

(B) The agent can decompose a new problem into subproblems that have commonalities with other, previously solved problems.

(C) The agent can incorporate more declarative, and expensive to interpret, forms of knowledge, such as instruction.

(D) Complex, time-consuming behavior at the reflective level can be compiled into rules for real-time processing at the deliberate and reflex levels.

The Soar architecture incorporates three levels in generating behavior. For a problem in which it has extensive experience, or where extensive knowledge has been pre-programmed, it will use only the reflex and deliberate levels to generate behavior. This approach has been useful in many domains. One of the first applications was the control of a Hero mobile robot with an arm (Laird and Rosenbloom, 1990). In Hero-Soar, the Soar program received sensing information such as the position of all the joint angles and distance information from a sonar sensor. Hero-Soar had reactive components to quickly stop it if it was about to hit an obstacle. It relied mostly on deliberate selection and applications of operators, such as opening and closing its gripper, moving forward, and turning. These operators were implemented as rules, which gave it the ability to have conditional and robust execution of the operators. It also had more complex operators such as clean the floor, pick-up a cup, find a basket, and drop a cup. Each of these was dynamically decomposed into its primitive operators. To perform new tasks, Hero-Soar would dynamically perform look-ahead searches to generate plans. Learning would compile the planning process, so that with experience, planning was needed less and less.

More recently, Soar has been used for a variety of domains including stick-level flying of the SGI flight simulator in a program called Air-Soar (Perason, et al., 1993). It has also been used for large-scale tactical aircraft control (Tambe, et al., 1995), simulated ground vehicle control, and natural language understanding and generation (Lehman, et al., 1993). In addition, this three-level approach has proven useful in modeling a variety of human behaviors (Newell, 1990).

3. Adaptation

In our approach, adaptation is based on a cycle in which the agent attempts to first correct or extend its reflective knowledge (which is essentially a multi-level domain theory used for internal planning), and then use the reflective knowledge as a basis for correcting its

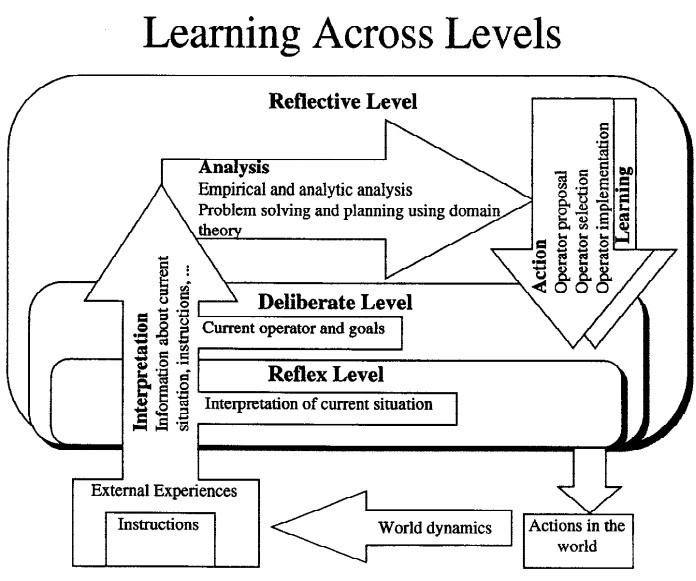


Figure 2. Adaptation of Multi-Level Soar Agent

deliberate and reflex knowledge. Correction to its deliberate knowledge influences which goals it can achieve in the world. The cycle continues because as the agent becomes more capable, it may be given new problems and tasks, which in turn require extensions to its reflective knowledge.

Figure 2 shows a simplified view of this cycle. In the first phase, information from the environment flows into the reflex level where it is parsed and interpreted. In parallel, the deliberate level is trying to make decisions so that the agent can pursue its goals. If the agent is unable to make a decision, its deliberate and reflex knowledge must be incomplete or incorrect in some way. When this happens, the reflective level is automatically invoked. Automatic invocation is possible in Soar because of the well-defined processing of selection, application, and termination of operators.

At the reflective level, the agent attempts to determine what part of the domain theory is either missing or incorrect. We have focused on knowledge errors at the deliberate or reflective levels. If knowledge is simply lacking then instruction might be appropriate. However, if its experiences in the world suggest that its reflective knowledge is incorrect, then it uses both analytic and empirical means to determine how to correct its reflective domain theory (planning knowledge). For some errors, this can be a complex process that involves multiple interactions with the environment that are not depicted in the figure. Once the domain theory knowledge has been corrected, it then uses it to replan its behavior. The results it produces are then compiled into knowledge that can be directly applied at the deliberate level in similar situations in the future.

Throughout the rest of the paper, we use a running example to describe the integrated instruction/recovery agent's performance. For these simple examples, we have provided the integrated agent with only enough domain knowledge to attempt the problems, so that we can demonstrate the contributions both methods can make, individually and in combination. First, we describe an instruction scenario using Instructo-Soar that introduces errors into the agent's knowledge at the deliberate and reflective levels. Next, we discuss IMPROV's recovery technique applied to the error, first with no instruction available, and then with instruction directing each stage of the recovery process.

4. Instruction-based Adaptation: Instructo-Soar

Instructo-Soar was developed to study learning by instruction. Early work on learning by instruction, such as SHRDLU (Winograd, 1972), did not lead to long-term learning of complex procedures, and systems of this type were limited to very simple commands. Instructo-Soar supports learner-driven instruction where the agent requests instruction only when it needs it. As it learns from instruction, less and less interaction is required. During instruction, the instructor can provide many different types of instruction, including direct commands, to hypotheticals, to negatives. Instructo-Soar uses a specialization of explanation-based learning (EBL) (Mitchell, Keller and Kedar-Cabelli, 1986) called *situated explanation* to extend its domain theory based on instruction. Instructo-Soar learns new procedures and extensions of procedures for novel situations, and other domain knowledge such as control knowledge, knowledge of operators' effects, and state inferences.

Instruction involves the following stages: First, Instructo-Soar requests instruction whenever its deliberate knowledge is insufficient to determine which action to take next. The request for instruction is situated within its current task, and all Instructo-Soar asks for is what step should it take next. Once it receives an instruction, Instructo-Soar determines what situation (state and goal) the instruction applies to – either the current situation or a hypothetical one specified in the language of the instruction. For example, if Instructo-Soar requests help in stacking red blocks, and the instructor says:

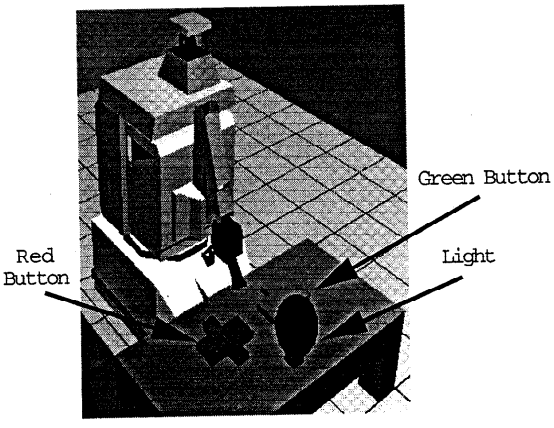
> "Pick up the red block."

Instructo-Soar assumes that this instruction refers to the current situation and to the current goal of stacking red blocks. If the instructor says,

> "To stack the green blocks, pick up the biggest green block."

Instructo-Soar assumes that the instruction is for a hypothetical goal of stacking green blocks.

Once Instructo-Soar has parsed the instruction, Instructo-Soar attempts to generate a plan (the explanation) at the reflective level, that is consistent with the instruction and leads to the goal. If an explanation is found, the agent can learn general knowledge from the instruction (as in standard EBL). If the explanation fails, it means the agent is missing some knowledge required to complete the explanation. The missing knowledge can be acquired either through further instruction, or in some cases through simple induction over the "gap" in the incomplete explanation. However, either instructions or the agent's inductions can be incorrect and lead to learning errorful knowledge.



"Push the green button."
How do I do that?
"Move to the grey table."
"Move your hand above the green button."
"Move down."
"The operator is finished."

Figure 3. (a) The example domain and (b) Instructions on how to push a button

Instructo-Soar's domain includes a table with red and green buttons and a light on it (see Figure 3 (a)). The red button turns the light on and off, but the agent does not know this. Consider an example in which the agent is first taught a general procedure for how to push buttons, using the instructions in Figure 3 (b). Some time later, the instructor says:

"To turn on the light, push the red button."

To perform a situated explanation, the agent first determines that this instruction applies to a situation where the goal is to turn on the light. Then, using its existing domain theory, it forward projects the action of pushing the red button in that situation. However, since it does not know that the button affects the light, this projection does not explain how pushing the button causes the goal of turning on the light to be reached.

In this case, the agent makes a simple inductive leap to complete the explanation. It guesses that since there is a single action specified to reach a goal, that action directly causes the goal to be reached – e.g., pushing the button has the effect of turning on the light. Its inductive bias specifies that in this type of induction, the types of objects involved and their relationship to one another are important but other features, like the button's color, are not. Thus, the agent learns a rule that says "pushing any button on the same table as a light causes the light to turn on." This rule allows the agent to complete its explanation of the original instruction, producing a control rule that applies at the deliberate level that says "if the goal is to turn on a light on a table, choose the action of pushing some button on that table."

Previous versions of Instructo-Soar would ask the instructor to verify the inductive leap – Instructo-Soar was a "paranoid" agent, afraid to learn any wrong knowledge. However, as we will see, the addition of IMPROV allows this verification step to be skipped, with the result being that errors in future performance will be corrected by the agent itself through interaction with the environment.

5. Recovery from incorrect knowledge: IMPROV

In developing IMPROV, we studied how to use the reflective level to deliberately correct domain knowledge based on experiences in a dynamic external environment where there is noise. IMPROV corrects errors in planning knowledge (at the reflective level), and then uses the corrected planning knowledge to correct errors in behavior at the deliberate level. IMPROV can correct the conditions and actions of existing hierarchically organized planning and execution knowledge; however, it cannot learn completely new operators. Thus, IMPROV refines a system that has the potential to be correct. IMPROV has been demonstrated in a dynamic blocks world, and a simulated driving domain.

Although previous systems have deliberately corrected their domain knowledge, they have been limited to simple domains, where there was neither noise in sensing nor dynamics in the external environment. They assume that actions are atomic and instantaneous and that there is no uncertainty in sensing the environment. Both of these conditions greatly increase the difficulty in determining that an error occurred, the reason for the error, and the correction required for eliminating the error. Theory revision systems apply empirical techniques directly to the domain theory, and although these approaches can work in environments with noise, they are unable to use other sources of knowledge (such as instructions or other domain knowledge) to improve the correction. In addition, these approaches only address the correction of knowledge and must be embedded in a more complete system that uses the knowledge, detects errors in behavior, localizes the error, and then corrects it. IMPROV does all phases of knowledge correction and can learn in domains with noise and dynamics, using many sources of knowledge if they are available.

IMPROV uses four principal stages to error recovery: (1) detecting an error, (2) finding a plan from the current situation to the goal, (3) identifying the operator which led to the error and (4) correcting the operator knowledge. IMPROV achieves these stages by: (1) recognizing that the agent is no longer making progress in its problem solving, (2) searching at the reflective level for a successful plan biased by knowledge acquired from previous external interactions, (3) comparing successful and incorrect plans to identify the incorrect operators and (4) training an inductive learner on the results of executing the plans, which corrects the planning knowledge.

In our example, when the agent is asked to turn on the light, its overgeneral knowledge ("pushing any button causes the light to turn on") may lead it to push the wrong (green) button. When it does, IMPROV detects the failure (the light doesn't come on), searches for the correct plan (pushing the red button) and then learns that to turn on the light, the agent must push the red button, not the green one. Alternatively, at any (or all) stages of the recovery process, the instructor can provide simple guidance that speeds up the process by avoiding search.

5.1. Detecting an Error

IMPROV has two methods for detecting errors. One is to have domain-specific knowledge that detects that the agent is in an invalid situation, one that the agent should never be in if it is correctly performing the task. For example, in the driving domain, if the agent ever goes off the road, there could be domain-dependent knowledge that signals that an error has

occurred. More generally, IMPROV detects errors when executing a plan by recognizing that it is unable to make progress towards its goals. This occurs when either it does not know how to achieve a goal from the current situation, or because it has conflicting suggestions as to what to do next. These conditions signify that an error has occurred because IMPROV will only attempt to execute a task once it has internally planned out a solution, which in turn leads it to have sufficient knowledge to make a decision at each step during execution. In our example, IMPROV detects an error when no operator is proposed after the agent pushes the wrong button and the light doesn't come on. Note that detection may not be possible until significantly after the incorrect knowledge was used, which complicates the problem of discovering what changes must be made to correct the error.

5.2. Finding a plan that reaches the goal

After detecting an error in its deliberate knowledge, IMPROV temporally stops attempting to directly perform the task. It shifts to the reflective level where it searches for a path from the current situation to the goal. This search is controlled using all available knowledge, be it acquired from prior experience, instruction, or from explicit programming. In our example, plans that include the push-button operator are preferred over plans that do not include operators associated with turning on lights because of the earlier instruction. If the agent has no previous knowledge and no other guidance, this search defaults to exhaustive iterative deepening.[1]

Once a plan is found that appears to solve the problem, the agent attempts to carry it out. It is possible errors in planning will allow invalid plans to be generated. Therefore, the agent successively generates alternative plans when errors arise, executing each in turn, to find one that satisfies the current goal.

5.3. Identifying the cause of the error

Once a solution has been found, the agent attempts to identify which operator is incorrect. As we will see in the next section, the agent is able to correct its behavior without identifying which rule(s) in the operator's proposal or application knowledge is incorrect.

IMPROV identifies which operator(s) have incorrect knowledge by comparing the successful plan to the original (incorrect) plan. Ordering and instantiation differences between the plans indicate potentially incorrect operators. In our simple light example, both the correct and incorrect plans contain a single operator (push-button) with different instantiations, so it is identified as the erroneous operator. This is a trivial example used for illustration. IMPROV can compare arbitrarily long plans, with arbitrary differences between the correct and original plans. If the correct plan has an operator that is missing from the original plan, then it is assumed that the operator's preconditions are overly specific. If the original plan has an operator that is missing from the correct plan, then it is assumed that the operator's preconditions are overly general. Out of order operators are treated as a combination of overly specific and overly general conditions.

	Weak Recovery Method	Assistance from Instructor	Example Instructions
Error Detection	Detect inability to make progress toward the goal.	Instructor indicates a failure is about to occur.	"Stop!"
Finding the correction path	Case–guided search	Instruction–guided search	"Push the red button." "Think about the red button."
Identifying the cause of the failure.	Induction to identify differences between instances.	Instructor indicates important differences between instances.	"Think about color."

Figure 4. The stages of a recovery and how instruction can help.

5.4. Correcting the error

Once the operators with incorrect knowledge have been identified, IMPROV uses an extension of the incremental category learner, SCA (Miller, 1993, Miller and Laird, 1996), to correct the operator. The category being learned is which operator to select for a given state and goal. To avoid the error in future, the agent must decide which features of the state or goal caused the operator to succeed or fail. In our example, the agent must learn that the reason pushing the red button works is because the button is red, not because it is the button on the right, or because the robot's hand is open or closed etc.[2] In its weakest form, IMPROV simply relies on the empirical induction made by SCA to determine the features that are responsible for the failure. If additional knowledge is available, IMPROV will use it to focus on the most relevant features. IMPROV revises the reflective knowledge by learning rules that add features if the operator's preconditions were overly specific, or it removes features if the operator's preconditions were overly general. Once it corrects its reflective knowledge, it then uses it to replan, which indirectly leads to the correction of the original error in the agent's deliberate knowledge. If the inductive guess was wrong, the error may recur, as will the recovery process until the knowledge is correct.

6. Instruction to inform recovery

IMPROV alone is a weak learner, relying on only empirical accumulation of examples to correct its errors. To strengthen its learning, we have augmented IMPROV by allowing an instructor to interrupt the recovery process at any time and provide instructions that guide the recovery.

Figure 4 lists how instructions can apply to each phase of recovery. In particular, instructions can be used to warn the agent that an error is about to happen, to help the agent find the correct way to achieve its current goal or to help identify the reasons for the error.

6.1. Error detection and identifying the cause

IMPROV's instructor can indicate that an error is about to happen by interrupting deliberate processing with the command "Stop!". Our agent assumes that this command means that

the currently selected operator will lead to an execution error, instead of leading to the current goal. In our example, if the agent chooses the wrong (green) button, the instructor may say "Stop", realizing an error is about to occur as the agent moves it's hand over the wrong button. IMPROV records the operator push-button(green), along with the state when the operator was chosen, as a negative training instance for the inductive module and starts a search for the correct way to turn on the light. Thus, this instruction not only helps the agent detect the error, it also determines which operator had incorrect knowledge.

6.2. Finding a plan that reaches the goal

Once an error has been detected, IMPROV searches for a correct plan to achieve the current goal. At any time during this search, the instructor may provide guidance in the form of one or more steps in that plan. In our example, the instructor can interrupt the search by saying:

"Push the red button."

This leads the agent to prefer plans that include pushing the red button. As the agent believes pushing any button turns on the light, it believes this plan will succeed and immediately executes it. After it succeeds, the agent's knowledge is corrected just as if it had discovered the plan through search instead of instruction. In the event that the instructions are incorrect, IMPROV will try the suggested path, see that it fails and continue searching for a correct plan.

6.3. Correcting the error

Once a correct plan has been found (either through search or with the aid of instruction), IMPROV needs to identify the cause of the error. IMPROV must determine the key difference(s) between the plan that succeeded and the plan that failed, which in general is a difficult credit assignment problem. In our example, the green button is to the right of the red button. Without help from the instructor, IMPROV must guess whether the reason push-button(green) failed was because the button was on the right, or because it was green. If it makes the wrong induction, the agent may fail later. However, the instructor can interrupt the learning process and guide the choice of features for the induction. In our example, if the instructor says:

"Think about color."

this leads IMPROV to focus on the colors of the buttons (rather than their positions) and ensures that the correct induction is made.

7. Discussion

The thrust of this paper has been to explore adaptation in multi-level agents. Our current work has made some important inroads by demonstrating the feasibility of using analytic and empirical processing at the reflective and deliberate level to improve the agents abilities

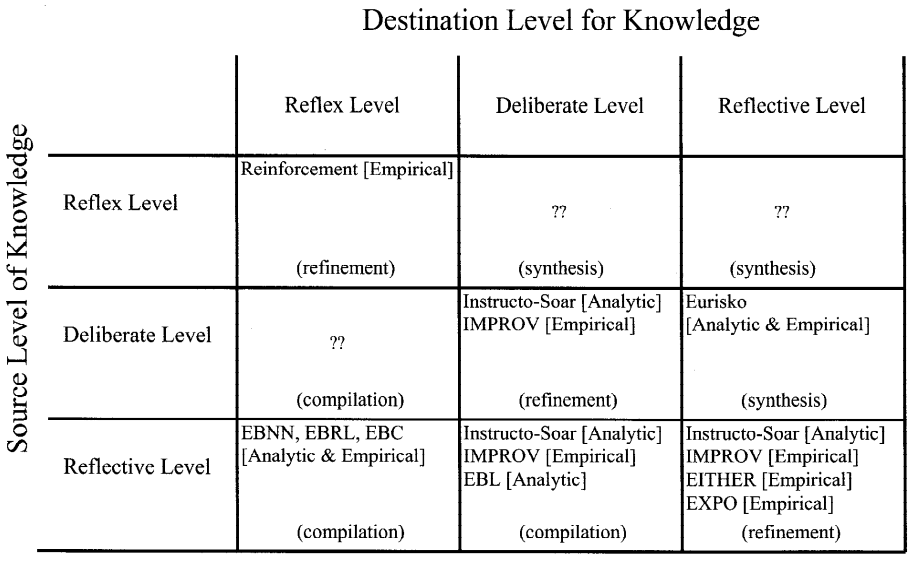


Figure 5. Matrix of Learning between Levels

at those levels. The agents learn new things about their environments that would not be possible in a single level agent because they use their reflective domain knowledge in addition to their direct experience with the environment. However, there is still more to be done before we have agents that can adapt at all levels from all available sources of knowledge.

Figure 5 is a matrix that attempts to classify current progress in learning across levels using both analytic and empirical techniques. Each box contains representative systems, with question marks (??) in boxes where there are no systems known to us that perform learning between these levels. Each row contains the problems of transferring knowledge from one specific level to another level. Implicit in these rows is that the knowledge in that level is combined with knowledge from the external environment, thus learning from the reflex level does not just use the agent's current reflex knowledge at that level, but uses that knowledge together with its experiences in its environment to improve that level. The columns are the destination of the knowledge — where the learning happens.

Along the diagonal, there is learning within a level, which we label as refinement. This region is well populated because learning often improves performance at the level where the knowledge originates (although that need not be the case). In the lower left, there are compilation approaches that move knowledge from the reflective level to the deliberate or reflex level, or knowledge from the deliberate level to the reflex level. The sparest area is the upper right, where synthesis is required to move knowledge up from reflex to deliberate or reflective, or move deliberate knowledge up to reflective. This type of learning requires detecting new patterns of behavior that is generated from the existing knowledge that can

then be used more broadly — in one sense, capturing the emerging patterns of behavior in a more deliberate or reflective representation. The one standout is Eurisko (Lenat, 1983).

Many systems appear in multiple columns because learning at one level immediately transfers to the higher levels. That is true in both IMPROV and Instructo-Soar where improvement in reflective planning knowledge transfers to improvement in the deliberate execution knowledge. One weakness in this figure is that it does not identify specific learning problems that cross all levels. For example, a particularly challenging area is the learning of new representations for tasks. This is a problem at all levels, and a solution at one level may immediately transfer to the others.

There are still some missing entries, and even when there is an entry, there are undoubtedly many research issues. Furthermore, additional research is needed on how to create agents that integrate methods from the complete matrix. However, overall the figure is more full than empty, and suggests that we are well on our way to building adaptive, multi-level agents.

Notes

1. We currently assume that a successful plan exists and therefore the agent does not have to determine when to abandon an impossible task.
2. This explanation is limited by the agent's representation language. To determine the real cause, that the red button is electrically connected to the light, the agent would need a deeper domain theory than we provided in this example.

References

DeJong, G. A case study of explanation-based control. In *Proceedings of the Twelfth International Workshop on Machine Learning*, pages 167–175, Los Altos, CA, 1995. Morgan Kaufmann Publishers, Inc.

Dietterich, T.G. and N. S. Flann. Explanation based learning and reinforcement learning: A unified view. In *Proceedings of the Twelfth International Workshop on Machine Learning*, pages 176–184, Los Altos, CA, 1995. Morgan Kaufmann Publishers, Inc.

Doorenbos, Robert B. Matching 100,000 learned rules. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 290–296. AAAI Press, 1993.

Forgy, C.L. Rete: A fast algorithm for the many pattern / many object pattern match problem. *Artificial Intelligence*, 19(1):17–38, 1982.

Gil, Yolanda. A domain-independent framework for effective experimentation in planning. In *Proceedings of the International Machine Learning Workshop*, pages 13–17, 1991.

Huffman, S.B. and J. E. Laird. Flexibly instructable agents. *Journal of Artificial Intelligence Research*, 3:271–324, 1995.

Laird, J.E. Recovery from incorrect knowledge in Soar. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 618–623, August 1988.

Laird, J.E., A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

Laird, J.E. and P. S. Rosenbloom. Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1022–1029, July 1990.

Lehman, J.F., J. E. Laird, and P. S. Rosenbloom. A gentle introduction to Soar, an architecture for human cognition. In *Invitation to Cognitive Science, Volume 4*. MIT Press, 1996.

Lehman, J.F., A. Newell, T. Polk, and R. Lewis. The role of language in cognition: A computational inquiry. In *Conceptions of the Human Mind*. Lawrence Erlbaum Associates, Inc, 1993.

Lenat, D.B. EURISKO: A program that learns new heuristics and domain concepts. The nature of heuristics III: Program design and results. *Artificial Intelligence*, 20:61–98, 1983.

Miller, C.M. *A model of concept acquisition in the context of a unified theory of cognition*. PhD thesis, The University of Michigan, Deptment of Electrical Engineering and Computer Science, 1993.

Miller, C.S. and J. E. Laird. Accounting for graded performance within a discrete search framework. *Cognitive Science*, 20:499–537, 1996.

Mitchell, T.M., R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1, 1986.

Mitchell, T.M. and S. B. Thrun. Explanation based learning: A comparison of symbolic and neural network approaches. In *Proceedings of the Tenth International Workshop on Machine Learning*, pages 197–204, Los Altos, CA, 1993. Morgan Kaufmann Publishers, Inc.

Newell, A. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

Ourston, D. and R. J. Mooney. Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the National Conference on Artificial Intelligence*, pages 815–820. AAAI/MIT Press, 1990.

Pearson, D.J. *Learning Procedural Knowledge in Complex Environments*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, 1996.

Pearson, D.J., S. B. Huffman, M. B. Willis, J. E. Laird, and R. M. Jones. A symbolic solution to intelligent real-time control. *Robotics and Autonomous Systems*, 11:279–291, 1993.

Pearson, D.J. and J. E. Laird. Toward incremental knowledge correction for agents in complex environments. In S. Muggleton, D. Michie, and K. Furukawa, editors, *Machine Intelligence*, volume 15. Oxford University Press, 1996.

Rosenbloom, P.S., J. E. Laird, and A. Newell. *The Soar Papers: Research on Integrated Intelligence*. MIT Press, 1993.

Rosenbloom, P.S., J. E. Laird, A. Newell, and R. McCarl. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47:75–111, 1991.

Tambe, M., W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, P. S. Rosenbloom, and K. Schwamb. Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1):15–39, 1995.

Winograd, T. *Understanding Natural Language*. Academic Press, New York, 1972.