

Interoperating Learning Mechanisms in a Cognitive Architecture

Dongkyu Choi and Stellan Ohlsson

Department of Psychology
University of Illinois at Chicago
1007 W Harrison Street (M/C 285), Chicago, IL 60607
{dongkyuc, stellan}@uic.edu

Abstract

People acquire new knowledge in various ways and this helps them to adapt to changing environment properly. In this paper, we investigate the interoperation of multiple learning mechanisms within a single system. We extend a cognitive architecture, ICARUS, to have three different modes of learning. Through experiments in a modified Blocks World and a route generation domain, we test and demonstrate the system's ability to get synergistic effects from these learning mechanisms.

Introduction

Human learning involves a variety of ways to acquire new knowledge. People learn from both positive and negative experiences. They also learn from both explicit and implicit instructions or demonstrations. This variety gives people the flexibility they need to adapt to changing environments, which has a direct impact on survival. Ohlsson (2011) argues that there are nine different modes of learning and emphasizes that the interoperation of such modes is the key to human success in this world.

However, most research in cognitive architectures dealt with each learning mode individually, focusing on the representational and the operational aspects of a learning mechanism and its benefits (e.g., Langley and Choi, 2006; Laird, Rosenbloom, and Newell, 1986). In contrast, our ongoing work (Choi and Ohlsson, 2010a; 2010b; 2011) concerns multiple learning mechanisms that interoperate within a single system. We started with a cognitive architecture, ICARUS, that has the built-in ability to learn from positive problem solving experiences. The system uses a version of means-ends problem solver to generate solution traces for its goals, from which it learns new procedural knowledge for the goals.

We then extended the architecture with two new mechanisms for learning from observations and learning from failures. The former augments ICARUS's bottom-up belief inference by providing a way to retain previous beliefs about the surroundings until they are proven to be false. This is useful when an ICARUS agent operates in a partially observable environment, since it allows retaining relevant pre-

vious beliefs even if the current sensory data does not include their perceptual supports. Hence, the system can make better-informed decisions with previous observations that are still reasonable in addition to the limited set of currently perceived information.

The second mechanism involves learning from failures (Ohlsson, 1996). It enables ICARUS to revise its procedural knowledge based on the observed or expected violations of constraints. When a constraint ICARUS cares about gets violated (or is expected to get violated at the next step), the architecture invokes the learning mechanism that adds extra preconditions to its procedure that caused (or would cause) the violation. The system distinguishes two different types of violations and revise procedures accordingly.

With the addition of these two learning modes, the extended ICARUS possesses three different modes of learning and it can support our research on the interoperation of multiple learning mechanisms. In this paper, we present some preliminary results from our efforts in this direction. We use a modified Blocks World and a route generation domain for examples. In the following sections, we first introduce the two domains briefly and review some basic assumptions of ICARUS. Then we describe the three learning mechanisms in some detail and present experimental results that show the interoperation of learning mechanisms and the synergistic effects among them. We also discuss related and future work before we conclude.

Illustrative Domains

In this work, we use two domains to test and demonstrate our system. The first one is a modified Blocks World, where the blocks have some additional features like color and size. This domain provides fully observable states and, therefore, ICARUS can perceive all the features of all the blocks that exist on table at any given time. As with the classic implementation of the domain, the ICARUS agent has available actions like grasping and ungrasping a block and moving the gripper in horizontal or vertical direction.

We also added several constraints, some of which ICARUS should consider at any given run. These include: the *color* constraint that requires stacked blocks to have the same color; the *top-block* constraint that forces certain blocks to not have anything on them; and the *tower* constraint that re-

quires the blocks above to be smaller than the blocks below in a stack. In our experiments, the number of constraints an agent should consider is a measure of the complexity of the task, along with the total number of blocks.

The second domain we use is a route generation domain. This includes fixed start and end locations with waypoints between them. Figure 1 shows an example setting in this domain. This domain features the partial observability and, at any given location, the agent can perceive only its current location and the neighboring locations. There are more than one routes available, giving the ICARUS agent multiple options to choose from. The only available action is to move from one location to a neighboring, connected location. Each location can be either dangerous or not, and we have only one constraint in this domain, which requires the agent to not be in a dangerous location.

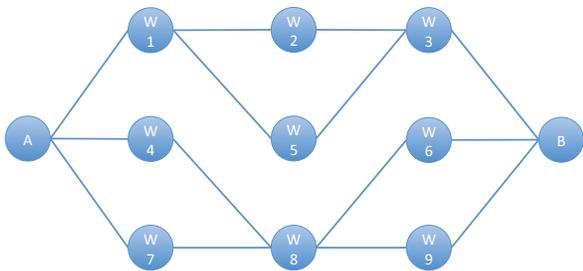


Figure 1: An example setting in the route generation domain. Locations *A* and *B* are the start and end locations, respectively, while *W1* through *W9* are waypoints.

In the route generation domain, we vary the complexity of the task by increasing the length of the shortest route between the start and end locations, which, in turn, increases the number of possible routes. The example shown in Figure 1 corresponds to the complexity level 1 we will discuss later in the experiments section. Next, we review some basic assumptions of ICARUS.

ICARUS Review

As with other cognitive architectures (Laird, Rosenbloom, and Newell, 1986; Anderson, 1993), ICARUS makes commitments to a specific way to represent knowledge, infer beliefs, perform execution and learn new knowledge. In this section, we review ICARUS’s basic assumptions before presenting the details of its learning mechanisms.

Representation and Memories

ICARUS makes distinctions in two separate dimensions. The first exists between concepts and skills. Concepts give ICARUS a language to describe its surroundings by enabling the system to infer beliefs about the current state of the world. Skills, on the other hand, are procedures that ICARUS believes to achieve certain concept instances. The second

distinction lies between long-term knowledge and short-term structures. Long-term concepts and skills are general descriptions of situations and procedures, and ICARUS instantiates them for a particular situation at hand. Instantiated concepts and skills are short-term structures, in that they are constantly created and destroyed as the situation changes. These two distinctions result in four separate memories in ICARUS.

In its long-term conceptual memory, the architecture encodes concept definitions that are similar to Horn clauses (Horn, 1951). As shown in Table 1, concepts include a head and a body that includes perceptual matching conditions or references to other concepts. The first concept with the head, (*at ?location*), matches against an object of type, *self*, and its attribute, *location*, in its *:percepts* field. The second concept, (*connected ?from ?to*), matches against an additional object of type, *location*, and tests if its *accessible* attribute is not null and the two locations, *?from* and *?to*, are different. These two concepts do not have any reference to other concepts in their definitions, so they are *primitive* concepts. On the other hand, the third concept, (*not-dead-end ?location*), matches against a *location* object and refers to two subconcepts in addition to a test. This makes the last concept a *non-primitive* one. This way, ICARUS builds a hierarchy of concepts that provides multiple levels of abstraction.

Another long-term memory stores ICARUS’s skills that resemble STRIPS operators (Fikes and Nilsson, 1971). The head of each skill is the predicate it is known to achieve, making all skills indexed by their respective goals. Each skill has a body that includes perceptual matching conditions, some preconditions, and either direct actions to the world or a subgoal decomposition. Skills with no references to subgoals are *primitive*, while the ones with subgoals are *non-primitive*. Table 2 shows some sample ICARUS skills. The first skill that achieves (*at ?location*) has two preconditions, (*at ?from*) and (*connected ?from ?location*), in its *:start* field and an action in its *:actions* field. Without any reference to subgoals, this skill is *primitive*. The second skill, however, is a *non-primitive* one that provides a subgoal decomposition to achieve (*at B*). Namely, this skill instructs ICARUS to consider two ordered subgoals, (*at W6*) and (*at B*), to achieve the eventual goal.

In addition, ICARUS has two short-term memories to store instantiated concepts and skills. While a short-term conceptual memory holds the current beliefs of the system, its short-term skill memory stores the selected skill instances indexed by their corresponding goals or subgoals. When ICARUS works on complex problems, information on goals and subgoals tends to dominate the short-term skill memory since it also serves as the goal stack for the system. Next, we explain the processes that generate contents of these short-term memories from their long-term counterparts.

Inference and Execution

ICARUS operates in cycles. On each cycle, it performs a series of processes as shown in Fig 2. The system first instan-

Table 1: Some sample ICARUS concepts for the route generation domain. Question marks denote variables.

```

((at ?location)
 :percepts ((self ?self location ?location)))

((connected ?from ?to)
 :percepts ((self ?self location ?from)
            (location ?to accessible ?access))
 :tests    ((not (equal ?from ?to))
            (not (null ?access))))

((not-dead-end ?location)
 :percepts ((location ?location))
 :relations ((connected ?location ?to1)
            (connected ?location ?to2))
 :tests    ((not (equal ?to1 ?to2))))

```

Table 2: Some sample ICARUS skills for the route generation domain. Question marks denote variables. The second skill is a part of a specific route that is learned.

```

((at ?location)
 :start    ((at ?from)
            (connected ?from ?location))
 :actions  ((*move-to ?location)))

((at B)
 :subgoals ((at W6) (at B)))

```

tiates its long-term concepts based on the data from its sensors. The bottom-up inference of concepts creates beliefs in the form of instantiated conceptual predicates. The inference process starts with the perceptual information about objects in the world. The system attempts to match its concept definitions to the perceptual information and, when there is a match, it instantiates the head of the definitions to compute its current beliefs.

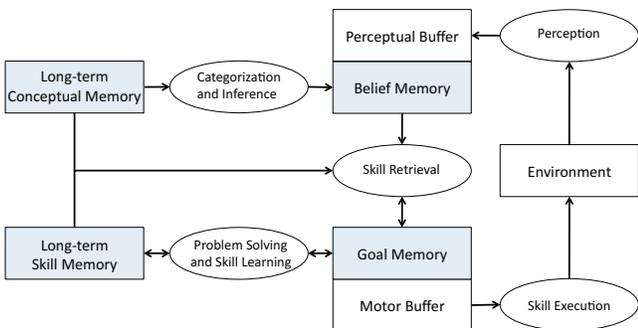


Figure 2: ICARUS’s memories and the processes that work over them.

Once the architecture computes all its beliefs, it starts the skill retrieval and execution process. ICARUS’s goals guide this process, and the system retrieves relevant long-term skills based on the current beliefs. When it finds an executable path through its skill hierarchy, from its goal at the top to actions at the bottom, ICARUS executes the actions specified at the leaf node of the path. This execution, in turn, changes the environment, and the system starts another cycle by inferring beliefs from new data received from the environment.

Learning Mechanisms

The original ICARUS includes a single learning mechanism that acquires new skills from successful problem solving traces (Langley and Choi, 2006). It uses a version of means-ends problem solver to decompose its goals into subgoals and generate a solution trace. The system then uses it to compose a new skill for each subgoal. To study the interoperation of multiple learning mechanisms, we needed at least one more learning mechanism in the architecture.

While searching for possible candidate mechanisms, we noticed that the original architecture’s problem solver implicitly assumes fully observable domains. For the means-ends problem solving to work, the system should have enough knowledge to chain backward from its goal to the current state, and this means the domain should be fully observable from the outset. To move away from this assumption, we added a new mechanism for learning from observations, which allows retaining previous beliefs as long as they do not contradict the current observations. This extension enables ICARUS to gradually accumulate its knowledge about the surroundings as it explores.

As part of our ongoing work, we added another mechanism for learning from failures using a new constraint language (Choi and Ohlsson, 2010b). Given constraints that are expressed as relevance–satisfaction condition pairs, the system revises its skills that cause violations of such constraints by adding new preconditions to them. In this section, we explain these three learning mechanisms in more detail.

Learning from Problem Solving

When ICARUS hits an impasse with no executable skills for the current goal, it invokes its means-ends problem solver to find a solution. As shown in Figure 3, the system has two options, either using a skill definition to propose an unsatisfied precondition as the next subgoal (*skill chaining*), or using a concept definition to decompose the current goal into subgoals (*concept chaining*). By default, ICARUS gives priority to the former and proceeds to the latter only when there is no skill chains available.

The architecture applies problem solving chains recursively until it finds a subgoal for which it can execute immediately. When such a subgoal is found, ICARUS proceeds with the execution to achieve it. Once the system satisfies the subgoal in the world, it learns a new skill from this experience by generalizing the situation and the procedures used.

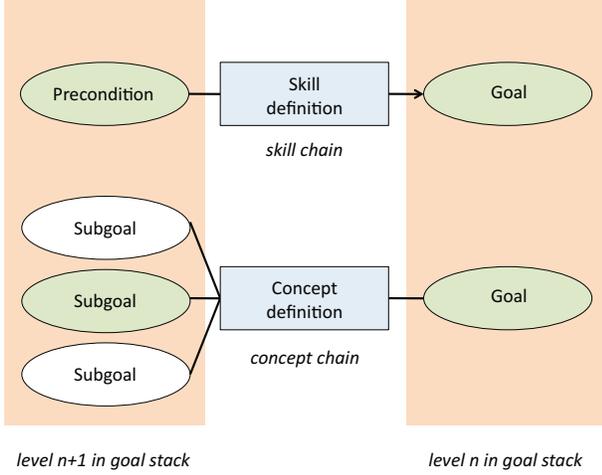


Figure 3: Two types of problem solving chains in ICARUS. For a skill chain, the system uses a skill definition to push the unsatisfied precondition as subgoal, while in a concept chain it uses a concept definition to decompose a goal into subgoals.

Learning from Observations

This learning mechanism outputs a different kind of knowledge. Instead of creating new skills, it retains certain beliefs of the ICARUS agent even after their perceptual supports disappear. The original architecture performs bottom-up inference of its beliefs from scratch on every cycle, but the extended system carries over some of the previous beliefs while it still infers new ones based on the updated perceptual information on a cycle.

This process happens in a straightforward fashion. ICARUS first performs the bottom-up inference with updates from the environment. It then compares this belief state to the previous one and finds conflicting beliefs in the previous state that get removed. The rest of the previous beliefs get added to the current belief state. What is crucial in this process is the mechanism for removing conflicting beliefs from the previous state. ICARUS uses negations in the definitions of concepts to find the beliefs that are, in some sense, opposed to new beliefs in the current state. However, this is not enough to find all conflicts, and it causes a catastrophic expansion of beliefs when it operates alone.

Therefore, the latest extension also includes a new field `:delete` in concept definitions that stores what is similar to a *delete list*. Since not all conflicting relations are explicitly expressed in the form of negated subconcepts, developers can manually add such relations in concept definitions. This is particularly useful to ensure uniqueness of some concept instances like an agent’s current location. For example, in the definition of `(at ?location)` that shows the current location of the ICARUS agent in a route generation domain (see the first concept in Table 1), we can guarantee that only one instance of this concept exists in the belief state on a given cycle by including `(at ?other)` in the delete list

for this concept. This process is shown in Figure 4.

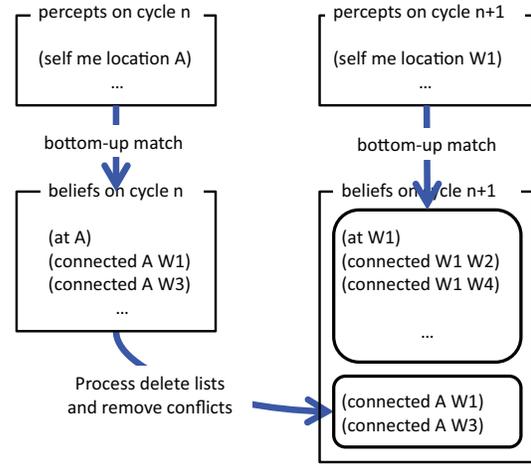


Figure 4: After inferring the current beliefs from the perceptual information on cycle $n+1$, ICARUS combines this result with previous beliefs that are both not in conflict with the current ones and not removed while processing the delete lists. In this example, only static beliefs for connectivity are maintained in the belief state for cycle $n+1$.

Learning from Constraint Violations

The extended ICARUS has the notion of constraints, expressed as pairs of relevance and satisfaction condition adopted from Ohlsson (1996). On every cycle, the system checks if the relevance conditions of each constraint is true in its belief state, and if so, it also checks the satisfaction conditions. When a constraint is violated, namely, when it is relevant but not satisfied, ICARUS invokes its constraint-based specialization mechanism to revise skills that caused the violation.

There are two different cases of violations. One is when a constraint has been irrelevant but it becomes relevant and not satisfied, and the other is when a constraint has been relevant and satisfied but it becomes unsatisfied. The system treats the two cases differently, using two distinct rules as shown in Table 3 to compute additional preconditions it adds to the corresponding skills. See Ohlsson (2011) for a more detailed description of this learning mechanism.

Table 3: Added preconditions computed differently based on the type of the constraint violation. C_r , C_s , O_a , and O_d denote relevance conditions, satisfaction conditions, add list, and delete list, respectively.

Type \ Revision	1	2
A	$\neg(C_r - O_a)$	$(C_r - O_a) \cup (C_s - O_a)$
B	$\neg C_r$	$C_r \cup \neg(C_s \cap O_d)$

Experiments

To prove the synergistic effect of having multiple learning mechanisms, we performed experiments in both the modified Blocks World and the route generation domain we have reviewed earlier. But we focus on the result from the latter in this paper, since this is where ICARUS requires one of the new learning mechanisms, namely, learning from observations due to the partial observability. In this section, we describe our experimental setup and the results from our experiments in this domain.

Experimental Setup

In our route generation domain, the agent starts at a location on one side, and it has the goal to get to a target location on the other side. Using the connectivity information between various neighboring locations, an agent should traverse from its origin to the target. Although there are multiple possible routes in the environment, some of the routes might become unavailable for travel due to various reasons such as criminal activity or damage to a bridge. When this happens, the agent can encounter situations where it is unable to use routes it has learned before, requiring it to adapt to the new situation.

The domain is modified from its original form to give partial knowledge of the environment to ICARUS agents, restricting the available connectivity information to the visible ones from the agent’s current location. We give only the basic concept and skill sets to the system at the beginning, along with a constraint. This means that the system knows how to operate in the world, but not at the level of expertise that enables it to satisfy the constraint at all times.

A typical run in this domain goes as follows. We give the system a goal to get to a target location, B , starting from the initial location, A . The two locations are connected by two alternate routes using waypoints $W1$ and $W2$, respectively. From the location A , the agent sees connections to the neighboring locations, $W1$ and $W2$. Without a complete connectivity information from the current location to the target, both execution and problem solving fails, and the system falls back to random exploration. This gets the agent to a waypoint, $W1$. From this location, the agent can see a direct connection to its target, which it takes by executing its skill for moving between neighboring locations (the first skill in Table 2).

Once the agent reaches the target, it is transported back to its origin for repeated trials. During the first trial, the system has remembered all the connectivity information it saw using declarative learning. Therefore, on the second trial, the problem solver can generate the route, $A - W1 - B$, from the beginning. It then takes the route by executing its skill twice for the two segments and achieves its goal. From this successful problem solving, the agent learns the route as specific skills like the second one in Table 2.

Before continuing subsequent trials, we designate the waypoint $W1$ as dangerous. This causes a violation of the constraint ICARUS is given, namely:

(at ?location) \rightarrow (not (dangerous ?location))

which simply says that it should not be at a location that is dangerous. During the next trial, the system attempts to use the known route, $A - W1 - B$, but it realizes that taking this route would cause a constraint violation. ICARUS learns from this expected violation by revising its skill to include an additional precondition, which ensures that the location the skill takes it to is not dangerous. Then again, there is no executable skill from A , and the system finds an alternate route $A - W2 - B$ through problem solving and learns a specific skill for this route. Starting from the next trial, the agent can simply execute its specific route skills to get to the target without any problem solving.

Evidence of Interoperation

While running the experiments, we sampled some runs under various conditions to investigate the interoperation of the three learning mechanisms in ICARUS. Our findings indicate that each learning mode takes the result of another as its input and other mechanisms use the outputs as part of their inputs. More specifically, we found the interoperational pattern shown in Figure 5.

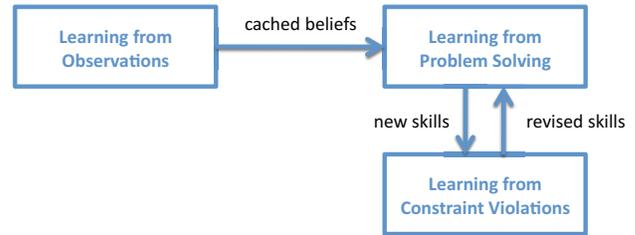


Figure 5: The interoperational pattern in the current setup of the learning mechanisms.

A typical example of this interoperation is as follows (see Figure 1). Initially, the ICARUS agent is at A and it perceives only three connections, (connected $A W1$), (connected $A W4$), and (connected $A W7$). Since these are not sufficient for the problem solver to generate a chain from the goal, (at B), the system falls back to random exploration and the agent moves to $W4$. From this location, the agent sees two more connections, (connected $W4 A$) and (connected $W4 W8$), but the five connections so far are still not enough to generate a problem solving chain. Therefore, ICARUS executes at random and this continues until cycle 14, on which the agent has wandered to $W3$ and it has 19 connection information accumulated. The problem solver is able to generate the trivial route, $W3-B$, at this point.

On the next trial, ICARUS invokes its problem solver immediately, using the previously accumulated information. It finds the route, $A-W1-W2-W3-B$, and learns this route as new skills. Later at the fifth trial, we introduce dangerous locations, $W1$ and $W2$. This renders the previously learned route obsolete, and it causes the system to revise its skill for moves to consider this constraint properly. Once again,

the system uses previously accumulated connection information to generate an alternate route. But this time, the revised skill rejects any moves into the dangerous locations, and forces ICARUS to find the new route, $A-W7-W8-W6-B$, and learns it as new skills. From the next trial, the system uses the new route and achieves its goal through straightforward execution until the environment changes yet again. Next, we present some quantitative results from our experiments in this domain.

Experimental Results

We ran similar experiments at three different levels of complexity, with 100 simulated subjects for each. There are six waypoints and four different routes between the origin and the target at the first level, while there are nine waypoints and four possible routes at the second level and 12 waypoints and eight possible routes at the third level. We performed experiments in four different conditions, in which 1) we turn on all learning modes, 2) turn off learning from constraint violations, 3) turn off learning from problem solving, or 4) turn off both of these learning modes. Since turning off learning from observations causes the system to fall back to exploration all the time, we did not include any conditions that involve turning off this learning mode.

Figure 6 summarizes main findings. The measure of computational effort is the total number of cycles per trial. The first four trials show the initial learning of a route to target. After the fourth trial, the learned route was declared out of bounds by marking some waypoints on that route to have become dangerous. The fifth trial is thus the one in which ICARUS discovers this change and faces the challenge of adapting to it. The subsequent trials trace the discovery and learning of a novel route. The figure shows four curves for each complexity level, corresponding to the four learning conditions outlined above.

The curve at the top of each graph on the later trials (marked with 'x') shows the result for the fourth condition where only learning from observations is active. After the first trial where the system can do nothing but randomly exploring, the system shows no change in effort from the second trial to the fourth. This is because it has accumulated enough connection information during the successful exploration on the first trial and invokes problem solving from the second trial. However, without learning from problem solving under this condition, ICARUS cannot take advantage of prior efforts and the performance on the second, third, and fourth trials are virtually equivalent. Once the environment is changed between the fourth and the fifth trials, the system should perform search to find a path around the dangerous waypoints, so computational effort is higher on the fifth trial than the prior ones and then it stays high because the system does not acquire new skills from its experience.

The next curve marked with triangular shapes is for the third condition, in which learning from constraint violations is active along with learning from observations. Since the system still does not learn any specific routes under this condition, there is no noticeable difference from the above condition on the first four trials. However, after the environment is changed (and therefore a constraint comes into play), the

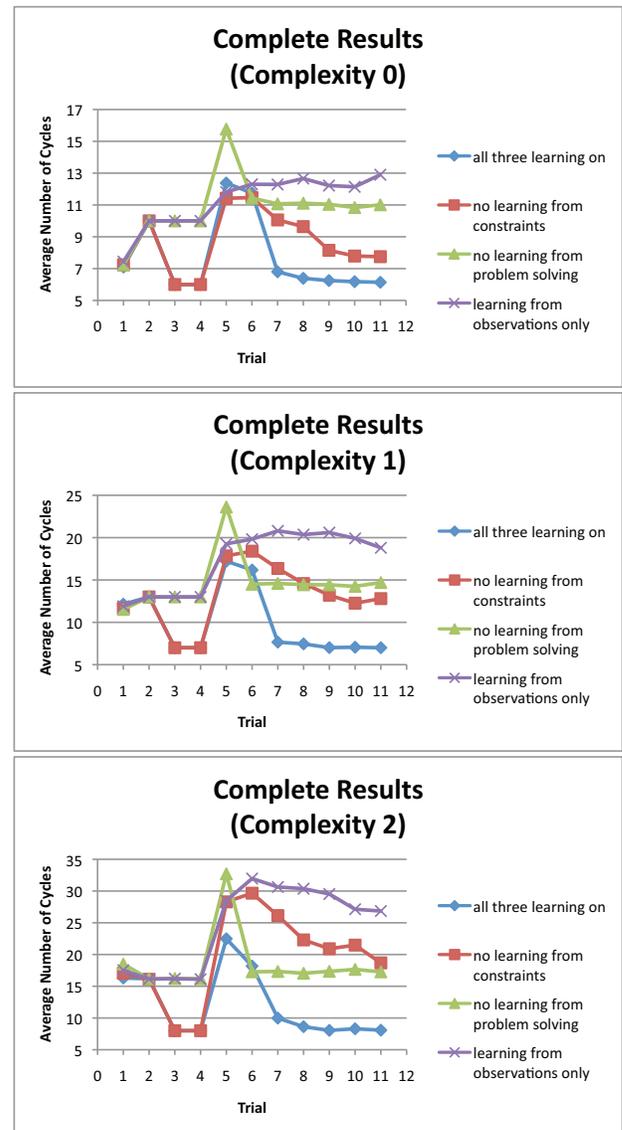


Figure 6: Number of cycles taken to reach the target location in situations with three different levels of complexity. Four conditions are shown in different shapes and colors consistently throughout the three graphs.

system revises its skill for moving to a neighboring location based on the constraint violation it detects when attempting to move to a dangerous location. After this learning, the problem solver finds a route without ever incurring any constraint violations. For this reason, the system performs noticeably better than the above condition from the sixth trial and on. But the learning does not happen for free, and the overhead causes worse performance than the last condition on the fifth trial.

In the other two conditions where learning from problem solving is active, the system rapidly learns an initial route during the first four trials. There is no measurable differ-

ence between the two conditions with respect to the system’s ability to learn an initial path, and the number of cycles required to traverse the landscape decreases significantly from the first trial to the fourth, for about 14% in the first complexity level and about 50% in the other levels. After the peaks at their fifth trials, however, the two conditions start to diverge. As complexity increases, the gap between the first and the second conditions widens. This shows the benefit of revising skills through learning from constraint violations, which reduces the system’s problem solving efforts by making it impossible to cause violations during the process. From the sixth trial, the performance gap between the two conditions decreases gradually, but the system without learning from constraint violations does not recover completely even on the last trial where it seems to have reached the steady state.

Another interesting finding in higher-complexity cases is that, on the fifth trial, the system adapts quicker to the changed environment when it runs with all three learning than when either learning from problem solving or learning from constraint violations is turned off. As shown in Figure 7, the synergy is substantial: the number of cycles is 17 compared to 19 for learning from constraint violations at complexity level 1 and the difference increases to 20 versus 26 at complexity level 2. These represents savings of 10% and 22%, respectively. When compared to learning from problem solving, the savings are even higher at 17% and 32%.

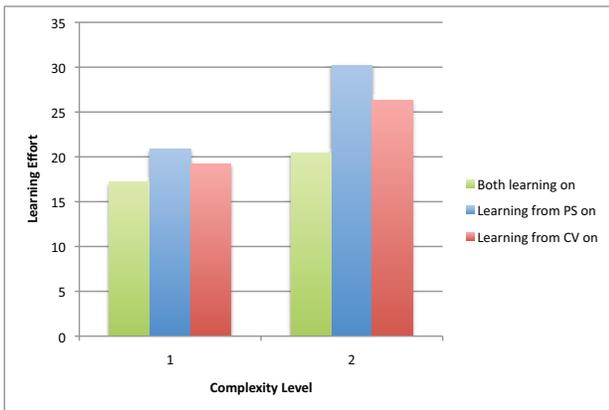


Figure 7: Learning efforts measured by the number of cycles at the fifth trial. The different between the two conditions shows the synergistic advantage of adding another learning mode as complexity increases.

In summary, both the improved performance at the final steady state and the reduced learning effort at the fifth trials suggest synergistic effects of having multiple learning mechanisms in a single system. We found that the performance of the system improves significantly as additional learning mechanisms are added. ICARUS solves problems faster in the conditions with two active learning mechanisms than in the condition with a single learning mechanism, and it performs even better with all three learning mechanisms.

Related and Future Work

This paper covers an ongoing research effort toward human-level variety of learning capabilities. In the current state, the system includes three different modes of learning, each of which has a vast amount of related work in the literature. First of all, learning from problem solving is closely related to previous research on macro-operators (Fikes, Hart, and Nilsson, 1972; Mooney, 1989; Shavlik, 1989). The ICARUS approach shares the basic principle of composing knowledge elements into larger structures. However, it supports disjunctions and recursions in the skill hierarchy, in addition to the simple fixed sequences learned in systems with macro-operators.

The mechanism for learning from constraint violations also has important similarities to previous work in explanation-based learning literature (Ellman, 1989; Wusteman, 1992). These methods assume a significant amount of domain theories presumed to be perfect. To augment this limitation, researchers worked on the similar problems of blame assignment and theory revision, although the exact formulations were different from ours. Unlike most of these work, our approach includes explicit descriptions of constraints, which the system uses to detect failures and revise existing procedural knowledge accordingly.

The interoperation of these two learning mechanisms in ICARUS has some similarity to the version space method (Mitchell, 1997; Winston, 1992), which has two learning modes for generalization from positive instances and specialization from negative instances. But the resemblance stops at the conceptual level, since ICARUS learns and revises its procedures using the two mechanisms, while the version space method is exclusively for concept learning.

In contrast to the two learning methods above, the topic of learning declarative knowledge from observations is significantly less studied. Researchers agree on the fundamental differences between declarative and procedural knowledge (Anderson, 1976), and the both types of learning are popular research topics among neuroscientists in relation to particular brain regions (e.g., Weis et al., 2004; Quintero-Gallego et al., 2006). However, research for simulating declarative learning through computational means is not common. Chi and Ohlsson (2005) classified various types of changes to declarative knowledge as learning proceeds, but the work does not attempt to model them computationally. We extended ICARUS to support declarative learning, but the research is in an early stage and requires further investigation.

Although our current work successfully shows the synergistic effects of multiple learning mechanisms in ICARUS, this research is still at an early stage. We plan to extend the architecture with yet another learning mechanism, possibly learning by analogy, to further verify our hypothesis of synergy among learning mechanisms. As this paper suggests, it is not our focus to implement a powerful single learning mechanism. Rather, we aim to build a collection of distinct learning capabilities that are written in a straightforward manner. Learning by analogy will not be an exception, and we plan to start with a simple mechanism that maps ob-

jects to similar objects or predicates to related ones. We find research on representation mapping by Könik et al. (2009) as a good inspiration in this direction.

Conclusions

The human ability to adapt to changing situations depends on a variety of learning mechanisms. Therefore, an intelligent agent cannot be limited to a single learning mode to simulate human behavior properly. We extended ICARUS to support three different modes of learning to model this behavior. Our initial results in a route generation domain show synergistic effects of having multiple learning mechanisms, especially evident at higher levels of complexity in the environment. We plan to collect more empirical evidences of the synergy and continue to explore additional types of learning capabilities in this framework.

Acknowledgments

This research was funded by Award # N0001-4-09-1025 from the Office of Naval Research (ONR) to the second author. No endorsement should be inferred.

References

- Anderson, J. R. 1976. *Language, memory, and thought*. Hillsdale, NJ: Lawrence Erlbaum.
- Anderson, J. R. 1993. *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.
- Chi, M. T. H., and Ohlsson, S. 2005. Complex declarative learning. In Holyoak, K., and Morrison, R., eds., *The Cambridge handbook of thinking and reasoning*. Cambridge, UK: Cambridge University Press. 371–399.
- Choi, D., and Ohlsson, S. 2010a. Cognitive flexibility through learning from constraint violations. In *Proceedings of the Nineteenth Annual Conference on Behavior Representation in Modeling Simulation*.
- Choi, D., and Ohlsson, S. 2010b. Learning from failures for cognitive flexibility. In *Proceedings of the Thirty-Second Annual Meeting of the Cognitive Science Society*. Portland, OR: Cognitive Science Society, Inc.
- Choi, D., and Ohlsson, S. 2011. Effects of multiple learning mechanisms in a cognitive architecture. In *Proceedings of the Thirty-Third Annual Meeting of the Cognitive Science Society*. Boston, MA: Cognitive Science Society, Inc.
- Ellman, T. 1989. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys* 21(2):163–222.
- Fikes, R., and Nilsson, N. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.
- Fikes, R.; Hart, P.; and Nilsson, N. 1972. Some new directions in robot problem solving. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 7*. Edinburgh: Edinburgh University Press. 405–430.
- Horn, A. 1951. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic* 16(1):14–21.
- König, T.; O’Rourke, P.; Shapiro, D.; Choi, D.; Nejati, N.; and Langley, P. 2009. Skill transfer through goal-driven representation mapping. *Cognitive Systems Research* 10(3):270–285.
- Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning* 1:11–46.
- Langley, P., and Choi, D. 2006. Learning recursive control programs from problem solving. *Journal of Machine Learning Research* 7:493–518.
- Mitchell, T. 1997. *Machine Learning*. New York: McGraw Hill.
- Mooney, R. J. 1989. The effect of rule use on the utility of explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 725–730. Detroit, MI: Morgan Kaufmann.
- Ohlsson, S. 1996. Learning from performance errors. *Psychological Review* 103:241–262.
- Ohlsson, S. 2011. *Deep learning: How the mind overrides experience*. New York, NY: Cambridge University Press.
- Quintero-Gallego, E. A.; Gómez, C. M.; Casares, E. V.; Márquez, J.; and Pérez-Santamaría, F. J. 2006. Declarative and procedural learning in children and adolescents with posterior fossa tumours. *Behavioral and Brain Functions* 2(9).
- Shavlik, J. W. 1989. Acquiring recursive concepts with explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 688–693. Detroit, MI: Morgan Kaufmann.
- Weis, S.; Klaver, P.; Reul, J.; Elger, C. E.; and Fernández, G. 2004. Temporal and cerebellar brain regions that support both declarative memory formation and retrieval. *Cerebral Cortex* 14(3):256–267.
- Winston, P. 1992. *Artificial Intelligence*. Reading, MA: Addison-Wesley Publishing Company.
- Wusteman, J. 1992. Explanation-based learning - a survey. *Artificial Intelligence Review* 6(3):243–262.