Learning Hierarchical Skills from Problem Solutions Using Means-Ends Analysis

Nan Li (nan.li.3@asu.edu)
David J. Stracuzzi (david.stracuzzi@asu.edu)
Pat Langley (langley@asu.edu)
School of Computing and Informatics, Arizona State University, Tempe, AZ 85281 USA
Negin Nejati (negin@stanford.edu)
Computational Learning Laboratory, CSLI, Stanford University, Stanford, CA 94305 USA

Abstract

Humans acquire skills in different ways, one of which involves learning from worked-out solutions to problems. In this paper, we present an extension to the ICARUS cognitive architecture that lets it acquire complex hierarchical skills in this manner. Unlike previous work on this topic, our approach relies on an existing architectural mechanism, means-ends analysis, to explain steps in the problem solution and to learn new structures. We illustrate this method in the domains of multi-column subtraction and football, after which we discuss related work and consider directions for future research in this area.

Introduction

Research on cognitive architectures (Newell, 1990) attempts to explain the entire range of human cognition. In previous papers, we have described ICARUS (Langley & Choi, 2006), an architecture that, in addition to other capabilities, acquires hierarchical skills during problem solving. However, as Ohlsson (2008) has noted, humans learn skills from many different sources of input. Thus, an important research goal involves extending ICARUS to support the full range of human skill acquisition. In this paper, we report progress on modeling learning from worked-out problem solutions, which often arise in educational settings.

Our approach builds on previous work, LIGHT, a system that constructs hierarchical skills from expert solutions to problems developed by Nejati, Langley, and Konik (2006). Although LIGHT utilized ICARUS knowledge structures as input and output, it operated as a separate module that was not part of the unified architecture. At the same time, the system's approach to explaining problem solutions bore a close resemblance to ICARUS' existing mechanism for means-ends problem solving. In response, we have adapted the latter mechanism to support explanation of, and learning from, worked-out solutions to acquire complex cognitive skills, extending ICARUS' coverage of human cognition.

In the sections that follow, we briefly review the ICARUS architecture, including its assumptions about representation, performance, and learning, along with Nejati et al.'s approach to learning from solution traces. After this, we describe our adaptation of the framework's means-ends mechanism to explain and learn from such traces, followed by an example in the domain of multi-column subtraction. We then report experiments that demonstrate the generality of our approach. In closing, we discuss related research and propose avenues for additional work on this topic.

The ICARUS Architecture

In previous work, Langley and Choi (2006) have presented ICARUS, a cognitive architecture that shares many features with other frameworks like Soar (Laird, Rosenbloom, & Newell, 1986) and ACT-R (Anderson, 1993), including a distinction between short-term and long-term memories, reliance on a recognize-act cycle, and a mixture of goal-driven with data-driven behavior. ICARUS also has distinctive features, such as separate memories for concepts and skills, indexing skills by the goals they achieve, and an architectural commitment to hierarchical structures. Before describing our approach to learning from problem solutions, we should review the framework's basic assumptions.

Beliefs, Concepts, and Inference

Most cognitive architectures operate in discrete cycles that produce mental or physical action. However, before an agent takes action, it must first understand its situation. ICARUS accomplishes this by matching conceptual structures in long-term memory against dynamic percepts and beliefs that it updates on each cycle. This process begins when descriptions of the environment are deposited into a *perceptual buffer*. The architecture complements this with a *belief memory* that encodes higher-level inferences about the environment, typically about relations among entities.

ICARUS beliefs are instances of generalized concepts stated in a long-term, hierarchical *conceptual memory*. Table 1 shows some concepts for multi-column subtraction. Primitive concepts match directly against the perceptual buffer, whereas nonprimitive concepts match against instances of lower-level concepts. Each nonprimitive concept specifies subconcepts that must be present for it to match. For example, the *all-processed* concept in the table refers to *processed* and *rightmost-column*. An inference mechanism updates belief memory at the beginning of each cycle by matching the generalized concept definitions with percepts and existing beliefs in a bottom-up manner, and stops when it infers all beliefs deductively implied by the concepts and percepts.

Goals, Skills, and Execution

After inferring a set of beliefs about its environment, ICARUS uses its available skills to take action there. The system stores these structures in a *skill memory*, which also has a hierarchical organization. Skill clauses are indexed by the concepts

Table 1: Sample concepts from multi-column subtraction.

```
; PROCESSED describes the situation in which the column
; contains an answer.
((processed ?col)
  :percepts   ((column ?col below ?below))
  :tests      ((not (equal ?below nil))))

; ALL-PROCESSED describes the situation in which all the
; columns right of ?left have been processed, including ?left.
((all-processed ?col)
  :percepts   ((column ?col))
  :relations  ((processed ?col)
               (rightmost-column ?col)))

((all-processed ?left)
  :percepts   ((column ?left) (column ?right))
  :relations  ((processed ?left)  (left-of ?left ?right)
               (all-processed ?right)))
```

Table 2: Sample skills from multi-column subtraction.

```
; Achieve PROCESSED by writing down the difference in
; cases where borrowing is not needed.
((processed ?col)
  :percepts   ((column ?col top ?top bottom ?bottom))
  :start      ((top-greater ?col))
  :actions    ((*find-diff ?col ?top ?bottom)))

; Achieve PROCESSED by achieving TOP-GREATER
; followed by PROCESSED when borrowing is needed.
((processed ?col)
  :percepts   ((column ?col) (column ?left))
  :start      ((left-of ?left ?col))
  :subgoals   ((top-greater ?col) (processed ?col)))

; Achieve ALL-PROCESSED by processing any column to
; the right before processing the current column.
((all-processed ?col)
  :percepts   ((column ?col))
  :start      ((rightmost-column ?col))
  :subgoals   ((processed ?col)))

((all-processed ?left)
  :percepts   ((column ?left) (column ?right))
  :start      ((left-of ?left ?right))
  :subgoals   ((all-processed ?right) (processed ?left)))
```

they aim to achieve. Table 2 shows some sample skill clauses for multi-column subtraction. The body of a primitive clause indicates actions that the agent can directly execute in the world, as in the first example. In contrast, the body of a non-primitive skill clause specifies subgoals the agent should pursue to achieve the goal in the head, as in the second example.

On each cycle, ICARUS retrieves skill clauses that could achieve its goal, and attempts to find an applicable path downward through the skill hierarchy. Upon reaching a primitive skill, the architecture executes its actions in the environment, possibly changing its percepts and beliefs on the next cycle.

Problem Solving and Skill Learning

When its long-term memory contains relevant knowledge, ICARUS retrieves and executes skills in an effort to achieve its goals. However, in some cases the system encounters an impasse (VanLehn, 1990) in which it cannot find appropriate skills. When this occurs, it calls on a problem-solving module that carries out means-ends analysis. This mechanism reasons backwards from known skills and concepts in an attempt to construct a novel solution.

The problem solver prefers skills that, if applied, would achieve the current goal but that are not yet applicable. In this case, ICARUS creates a subgoal based on the skill's instantiated start condition and attempts to satisfy it. If it cannot find such a skill, the problem solver examines the concept definition for the current goal, selects one of its unsatisfied subconcepts, and makes this the active subgoal. Whenever ICARUS finds an applicable path that could achieve the current subgoal, it executes that skill path in the environment. Upon achieving a subgoal, the system either shifts to another unsatisfied subgoal or marks the parent goal as achieved, continuing this process until the top-level goal is satisfied.

Although ICARUS' problem solver lets it overcome impasses and achieve goals for which it has no stored skills, the process can require considerable search and backtracking. For this reason, the architecture also includes a learning mechanism that caches the results of successful problem solving in skill memory. Whenever means-ends analysis achieves a goal or subgoal G, ICARUS creates a new skill in which the head is a generalized version of G. If system achieved G by chaining off an existing primitive skill S, the new skill's subgoals are S's start condition plus S's head in the order they were achieved. If system achieved G by chaining off an existing non-primitive skill S, the new skill's subgoals are S's start condition plus S's subgoals in the order they were achieved. In both cases, the new skill's start condition is the same as that for the first subgoal. If ICARUS achieved G through chaining on concept C, the new skill's subgoals are the subconcepts of C that were initially unsatisfied, again in the order they were achieved. In this case, the start condition is the conjunction of C's subconcepts that were true initially.

In subsequent runs, ICARUS will apply the new skills whenever they are relevant to its goals and their start conditions match the current state. As a result, the agent can achieve its goals through reactive skill execution without calling the problem solver. Langley and Choi (2006) have reported encouraging results with this learning mechanism in a driving environment, the Blocks World, and Freecell solitaire.

Review of the LIGHT System

Despite ICARUS' accomplishments, it provides an incomplete account of human cognition in that it acquires skills only from its own attempts at problem solving. As Ohlsson (2008) has argued, people learn from a variety of sources, including worked-out problem solutions. Recently, Nejati et al. (2006) report one approach to acquiring knowledge in this manner.

Their LIGHT system accepts as input a goal, a sequence of skill instances and the associated state sequence. Given this information as input, the learner parses the solution by reasoning backward from the final state, at each step explaining how the action achieves the goal or one of its subgoals

by chaining over skills or concept definitions. LIGHT starts by examining the final skill instance S in the trace and, if S's effects include the goal, creates a subgoal for the earlier steps based on S's preconditions. If no skill instance in the trace could have achieved the goal, the learner decomposes it into subgoals using its conceptual definition. LIGHT then looks back through the solution trace to determine the order in which it achieved each subgoal, explaining recursively how the observed actions achieved each one. The process terminates when it links the achievement of each goal to the trace.

Using this hierarchical explanation structure, LIGHT uses the ICARUS learning methods to create new skills for each explained goal and subgoal. If a given explanation step involved chaining off a skill, then the system acquires the same structure that ICARUS would learn in this situation. If an explanation step involved chaining off a concept definition, then LIGHT constructs the same skill that the architecture would acquire under those conditions. Nejati et al. demonstrated their approach on two domains from the AI planning literature, Blocks World and Depots, showing their system acquires effective skills that solve most problems in each domain, and also captures recursive structures that generalize to situations beyond those the system has encountered.

Although LIGHT implements a novel and interesting method for acquiring hierarchical skills from solution traces, it operates as a standalone process. The system runs outside ICARUS' basic cognitive cycle and connects to it only by using the same knowledge structures and the same skill-caching mechanism. Thus, it does not directly aid our goal of accounting for learning from problem solutions within a unified theory of the human cognitive architecture.

Extending ICARUS to Learn from Solutions

Nevertheless, Nejati et al.'s system introduced some promising ideas that deserve further attention. Because LIGHT's approach to explanation bears a close relation to means-ends analysis, we decided to adapt ICARUS' problem-solving module to support a similar ability to learn from solution traces.

We had two aims in mind when pursuing this work. First, we wanted to account for people's ability to learn from worked-out solutions within a unified cognitive architecture, drawing on existing ICARUS mechanisms where possible. Second, we wanted an approach that could take advantage of previously learned skills to aid later learning, which ICARUS supports but which LIGHT did not. Both characteristics add to the psychological plausibility of our model.

A typical run begins with the architecture passively observing a state sequence that a tutor presents.[1] As the state descriptions appear in its perceptual buffer, ICARUS infers beliefs that describe each state in more abstract terms. For subtraction, these include relations between columns and numbers that are relevant to the domain skills. One difference from previous versions of ICARUS is that the inference process tracks augments beliefs with time stamps that note when they first became true and when they ceased to hold. This provides a simple episodic memory, which we introduced into ICARUS for another project, but which also proved useful for the explanation process.

Once the system has observed the problem solution and stored it in the episodic belief memory, the tutor provides it with the problem's goal and an indication that it should learn from the trace. This leads ICARUS to invoke its means-ends analysis module, which we have extended to operate slightly differently when a solution trace is available. As usual, the module begins by chaining backwards off the top-level goal, selecting a skill from primitive or non-primitive skills that would produce the goal as its effect and selecting a conceptual clause otherwise. In the first case, it creates a subgoal to achieve the skill's instantiated start condition; in the second, it creates one subgoal for each unsatisfied subconcept. The process bottoms out whenever it finds a skill that explains how a skill executed in one state in the solution leads to achieving a goal or subgoal in a later state, with this activity continuing until it accounts for the entire sequence of solution steps.

This explanatory mode of means-ends analysis differs from the traditional problem-solving mode in two key ways. One is that the system chains off a goal that is already satisfied in an effort to explain how it occurred, rather than trying to alter the environment to achieve it. However, each step in the process requires one cognitive cycle, so that explanation is deeply integrated into the architecture in the same manner as the problem solver. The other difference is that chaining is constrained by the contents of the episodic belief memory. This limits the search that arises during means-ends analysis greatly, although it may not eliminate it entirely, in which case the system backtracks and considers another path.

In addition, ICARUS prefers to explain the observed state sequence using its skills rather conceptual knowledge, and it prefers skill instances that reach farther back in the sequence. The second bias encourages the system to reuse learned, higher-level skills when they are available. In the extreme case, when the means-ends module can explain the entire solution with a single high-level skill, then no learning is necessary. However, typically the system uses a mixture of primitive skills, concept definitions, and possibly learned skills to produce an explanation.

As in means-ends problem solving, ICARUS interleaves the explanation process with skill learning. Whenever the system accounts for how an observed subsequence of states produces a goal or subgoal, it creates a new skill for that goal. The architecture uses the same learning mechanism as for standard means-ends analysis that we described earlier. After storing a new skill in memory, ICARUS returns to its efforts to account for other parts of the solution trace, continuing until it has explained, and acquired skills for, the entire sequence. The new skills become available to solve new problems that the system encounters or, if presented with additional worked-out solutions, to explain and learn from them.

[1] We wanted to remove LIGHT's assumption that traces include skill instances, since humans observe only a sequence of states.

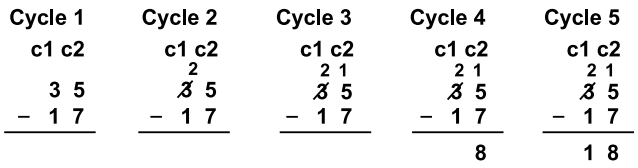

Figure 1: Solution trace for a subtraction problem.

An Example from Multi-Column Subtraction

In order to clarify the mechanism's operation further, we provide an example from multi-column subtraction, a domain that has been well studied by cognitive scientists with educational interests. Figure 1 shows a sequence of states for the problem $35 - 17$, which we provide to ICARUS along with a set of concepts that describe various situations in this domain, including the clauses for *processed* and *all-processed* in Table 1, as well as ones like *left-of*, *rightmost-column*, and *top-positive*. We also provide the system with a set of primitive skills that describe basic subtraction actions, including finding differences, adding ten to the top of a column, and subtracting one from the top number. As Figure 2 depicts, the problem's top-level goal is *(all-processed c1)*, which means that the agent should process column *c1* and all columns right of column *c1*. Since *c1* is the leftmost column, achieving *(all-processed c1)* equates to solving the entire problem.

Given the solution to this task, ICARUS first steps through the states, inferring higher-level beliefs that hold in each case. Next, the system attempts to explain the solution trace using means-ends analysis. Since it has no skill that would achieve the *all-processed* goal, it resorts to conceptual knowledge. Belief memory indicates that *(all-processed c1)* is supported by instantiated subconcepts *(left-of c1 c2)*, *(all-processed c2)*, and *(processed c1)*. The first held in the initial state, and the other two became true on cycles 4 and 5, respectively. As a result, the problem solver separates the solution trace into subtraces. Trace T_{14} from cycle 1 to cycle 4, is associated with the subgoal *(all-processed c2)*, while T_{55}, which involves only cycle 5, is associated with *(processed c1)*.

The problem solver then attempts to explain recursively how each subtrace achieves its associated subgoals. For *(all-processed c2)*, belief memory indicates that, since column *c2* is the rightmost column, *(all-processed c2)* was linked to accomplishing *(processed c2)*. Since the system has a primitive skill (see Table 2) for achieving *processed*, it uses this skill to explain T_{14}. ICARUS notes that the start condition of the skill instance *(top-greater c2)* did not become true until cycle 3, so it decomposes T_{14} into two subtraces, T_{13} and T_{44}, where T_{13} is associated with *(top-greater c2)* and T_{44} with *(processed c2)*. Among these subgoals, only *(top-greater c2)* cannot be achieved with available skills; this leads to further chaining, but we will not present the details here for simplicity's sake.

The explanation process does not proceed monolithically. Whenever the problem solver explains how a solution trace or subtrace achieves a goal or subgoal, it triggers ICARUS' learning mechanism to produce a new skill that encodes this understanding. In our example, the first subgoal explained is *(processed c2)*, using subtrace T_{14}. Since this involves a skill instance with the instantiated start condition *(top-greater c2)*, the system constructs a skill with generalized versions of *(top-greater c2)* and *(processed c2)* as its two subgoals, as Table 2 indicates. The learned skill's start condition is a generalized version of the start condition for the skill that achieved *(top-greater c2)*, that is, *(left-of ?left ?col)*.

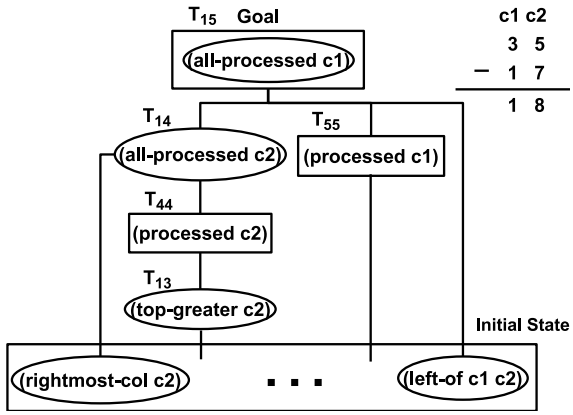

Figure 2: A multi-column subtraction explanation trace. Concept instances are shown as circles, and skill instances are shown as rectangles.

Having successfully explained *(processed c2)* with T_{14}, ICARUS acquires another skill for *(all-processed c2)* with this as the only subgoal and with *(rightmost-column c2)* as the start condition – again, with variables substituted for arguments – as Table 2 shows. Having explained *(all-processed c2)* with subtrace T_{14} and *(processed c1)* with subrace T_{55}, ICARUS constructs a skill for the top-level goal *(all-processed c1)*. In this case, the skill includes two subgoals based on *(all-processed c2)* and *(processed c1)*, along with a start condition based on *(left-of c1 c2)*. Having acquired skills for each goal in the explanation tree, learning halts at this point.

Generality of the Approach

To determine the generality of our architectural extensions, we carried out experiments with a number of problems from two domains. The first study involved multi-column subtraction. We provided ICARUS with 16 concepts and six primitive skills, then presented it with four solved subtraction problems of increasing complexity. These involved, respectively, no borrowing $(45 - 32)$, basic borrowing $(35 - 17)$, borrowing from zero $(805 - 237)$, and borrowing across zeroes $(2005 - 237)$. ICARUS learned seven new skills from the first three tasks, in each case building on ones it had acquired earlier. The final exercise produced no learning, since the system solved it using previously acquired recursive skills.

This result encouraged us to examine more closely the learned skills' ability to generalize. When presented with each of the six subtraction tasks in Table 3, ICARUS acquired, in each case, a set of skills that transferred in expected ways to the other five problems. For instance, the architecture's exe-

Table 3: Multi-column subtraction problems used for training and testing the new learning mechanism.

45 – 32 = 13	40 – 17 = 23
35 – 17 = 18	805 – 237 = 568
45 – 17 = 28	2005 – 237 = 1768

cution module could use recursive skills learned from $45 - 32$ to solve other nonborrowing tasks, even if they involved fewer or more columns, but not to solve borrowing problems. After learning on a task like $45 - 17$, it could solve problems that involved a mixture of basic borrowing and nonborrowing columns (i.e., the first four tasks in Table 3).

Because we also desired to show that our approach is not limited to controlled educational settings, we carried out a second study on learning from traces of observed football plays. Here we provided ICARUS with segmented descriptions of video footage for three separate plays that we specified with a set of 58 concepts. Hess and Fern (2007) report the methods used to transform the pixel-based videos into sequences of percepts that characterize objects in terms of attribute-value pairs. After learning skills from each video, ICARUS attempted to execute the same plays in football simulator. We evaluated the quality of the learned skills qualitatively by comparing plots of the player trajectories generated from the video to plots generated in the simulator.

During these runs, ICARUS acquired a total of 11 skills, including seven from the first play, one from the second play, and three from the third. The architecture acquired these skills cumulatively, with skills learned from later plays building upon those learned earlier. Most lower-level skills were acquired from the first trace, so that the system learned only higher-level behaviors like complex receiver patterns from later ones. A qualitative comparison of player trajectories revealed that ICARUS' execution of all three plays correspond to idealized versions of plays in the video, which makes sense because both ICARUS and the simulators are less hampered by momentum than humans on the field. Taken together, these results suggest that our approach to learning from observed behavior can acquire complex skills in both academic domains like arithmetic and physical ones like sports.

Discussion

There has been considerable research on learning within cognitive architectures. The best-known work involves Soar (Laird et al., 1986), which acquires knowledge that constrains problem-space search through a chunking mechanism, and ACT-R (Anderson, 1993), which creates new production rules through a compilation process that gradually transforms declarative representations into procedural ones (Taatgen & Lee, 2003). The learning mechanisms in ICARUS, Soar, and ACT-R are all analytic but differ in their details, although ICARUS' method is similar to Soar's in that it reduces search and akin to ACT-R's in that it transforms knowledge from one form to another. However, it has closer ties to PRODIGY, which used an analytical technique to acquire control rules for means-ends problem solving (Minton et al., 1989).

Although research on cognitive architectures has not focused on the acquisition of skills from worked-out solutions, there have been some efforts along these lines. Van Lent and Laird (2001) describe analogous work on learning Soar operators, but their approach relied on annotated traces that our method does not require. Work by Neves (1978) and Matsuda et al. (2008) on learning production rules for algebra also took advantage of solution traces, but neither acquired complex hierarchical procedures.

Other research in AI has examined learning complex procedures from problem solutions. Segre (1987), Mooney (1990) and VanLehn and Jones (1993) report analytical approaches to this task, but none of their systems acquired hierarchical or recursive structures. More recent work by Hogg, Muñoz-Avila, and Kuter (2008) acquires hierarchical skills from solution traces, but requires knowledge about high-level tasks that our approach does not assume. Reddy and Tadepalli's (1997) X-LEARN comes closest to our own in terms of inputs and outputs, but it used a nonincremental method to learn conditions on its hierarchical skills.

Another closely related system, VanLehn's (1990) Sierra, also models the impasse-driven acquisition of hierarchical procedures for multi-column subtraction from sample solutions. However, his work focused on explaining the origin of bugs, which we have not attempted. Also, Sierra examined similarities among a number of problem solutions during learning, whereas ICARUS acquires multiple skills from individual problem solutions in an incremental manner.

Although our results to date provide a promising account of skill learning within a theory of cognitive architecture, there remain many avenues for additional research. We should evaluate the extended ICARUS' abilities in subtraction and football more extensively, as well as demonstrate the ability to learn from solution traces on other tasks of educational interest, such as physics problem solving.

In addition, we should make our framework more consistent with results on human skill acquisition. In particular, our studies of multi-column subtraction revealed that ICARUS learns more rapidly than people, in that it masters all seven skills from a single problem $2005 - 237$. A human student typically requires a variety of simple training problems before he moves on to ones that involve complex combinations. The system can acquire skills in a more gradual, cumulative fashion, but this is not necessary. The most promising response would limit the architecture's episodic memory to retain only the most recent N beliefs. The revised version would still be able to learn from complex solutions, but it would only acquire simpler skills that occur low in the explanation tree before it forgot steps higher in the tree. Once ICARUS had mastered these skills, it could benefit from more complex problems, since it would not need to explain the lower levels and could focus its efforts on higher levels of the solution trace.

Our longer-term agenda includes extending ICARUS on two additional fronts. The first involves even stronger unification between its explanation mechanism and other capabilities. An improved system would resort to regular means-ends problem solving when presented solutions with missing steps, using search to fill in the gaps. We should also make explanation process as interruptable as problem solving, which the current ICARUS will suspend if a higher-priority goal becomes unsatisfied unexpectedly. The second involves augmenting the architecture to learn from the additional sources of information that Ohlsson (2008) discusses, such as external feedback and violated constraints. As in the current work, we should model these abilities with as few changes to ICARUS as possible, since our goal is a unified theory of cognition.

Concluding Remarks

Learning from demonstrated solutions is one of the main ways that humans acquire skills, making it a crucial ability for any broad theory of cognition to explain. In this paper, we reported extensions to an existing cognitive architecture, ICARUS, that provided it with the ability to learn in this manner. The most important aspect of our account is that it utilizes the framework's mechanism for means-ends analysis to explain the observed solution trace. This extension required almost no changes to the architecture and used the same machinery for creating subgoals, marking them as satisfied, and learning new skills as the standard means-ends process.

We demonstrated the extended ICARUS' behavior on multi-column subtraction, showing that it can learn complex hierarchical and recursive skills in this domain that generalize correctly to new problems. Although the system learns subtraction procedures more rapidly than humans, it provides a promising initial account of learning from problem solutions that is embedded within a unified theory of the human cognitive architecture. We hope to take a similar approach when we provide the framework with additional learning abilities.

Acknowledgements

This material is based in part on research sponsored by DARPA under agreement FA8750-05-2-0283. The views contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, expressed on implied, of DARPA or the U. S. Government.

References

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Hess, R., & Fern, A. (2007). Improved video registration using non-distinctive local image features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recogntion.* Minneapolis, MN: IEEE Press.

Hogg, C., Muñoz-Avila, H., & Kuter, U. (2008). HTN-Maker: Learning HTNs with minimal additional knowledge engineering required. In *Proceedings of the Twenty-Third Conference on Artificial Intelligence* (pp. 950–956). Chicago: AAAI Press.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence.* Boston: AAAI Press.

Matsuda, N., Cohen, W. W., Sewall, J., Lacerda, G., & Koedinger, K. R. (2008). Why tutored problem solving may be better than example study: Theoretical implications from a simulated-student study. In *Proceedings of the Nineth International Conference on Intelligent Tutoring Systems* (pp. 111–121). Berlin: Springer-Verlag.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., & Etzioni, O. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*, 63–118.

Mooney, R. J. (1990). *A general explanation-based learning mechanism and its application to narrative understanding*. San Mateo, CA: Morgan Kaufmann.

Nejati, N., Langley, P., & Konik, T. (2006). Learning hierarchical task networks by observation. In *Proceedings of the Twenty-Third International Conference on Machine Learning.* Pittsburgh, PA: ACM Press.

Neves, D. M. (1978). A computer program that learns algebraic procedures by examining examples and working problems in a textbook. In *Proceedings of the Second Biennial Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 191–195). Toronto, Canada.

Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA, USA: Harvard University Press.

Ohlsson, S. (2008). Computational models of skill acquisition. In R. Sun (Ed.), *The cambridge handbook of computational psychology*. New York: Cambridge University Press.

Reddy, C., & Tadepalli, P. (1997). Learning goal-decomposition rules using exercises. In *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 278–286). San Francisco: Morgan Kaufmann.

Segre, A. (1987). A learning apprentice system for mechanical assembly. In *Proceedings of the Third IEEE Conference on AI for Applications* (pp. 112-117).

Taatgen, N. A., & Lee, F. J. (2003). Production compilation: A simple mechanism to model complex skill acquisition. *Human Factors, 45*, 61–76.

Van Lent, M., & Laird, J. E. (2001). Learning procedural knowledge by observation. In *Proceedings of the First International Conference on Knowledge Capture* (pp. 179–186). Victoria, BC: ACM Press.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions*. Cambridge, MA, USA: MIT Press.

Vanlehn, K., & Jones, R. M. (1993). Learning by explaining examples to oneself: A computational model. In S. Chipman & A. L. Meyrowitz (Eds.), *Cognitive models of complex learning* (pp. 25–82). Kluwer Academic Publishers.