Kennedy, W.G., Trafton, J.G. (2007). Long-term symbolic learning. *Cognitive Systems Research, 8,* 237-247. (Available from http://www.mllab.com)

Long-Term Symbolic Learning

William G. Kennedy and J. Gregory Trafton

Naval Research Laboratory

Abstract

What are the characteristics of long-term learning? We investigated the characteristics of long-term, symbolic learning using the Soar and ACT-R cognitive architectures running cognitive models of two simple tasks. Long sequences of problems were run collecting data to answer fundamental questions about long-term, symbolic learning. We examined whether symbolic learning continues indefinitely, how the learned knowledge is used, and whether performance degrades over the long term. We report that in both systems, symbolic learning eventually stopped, learned knowledge was used differently in different stages but the resulting production knowledge was used uniformly, and, finally, both Soar and ACT-R do eventually suffer from degraded performance with long-term continuous learning. We also discuss ACT-R implementation and theoretic causes of ACT-R's performance problems and settings that appear to avoid the performance problems in ACT-R.

Introduction

Learning has been widely studied by researchers in the fields of psychology, education, cognitive science, and Artificial Intelligence (AI). The nature and representation of learning, including forms of learning, speed, memory capacity, etc., have been investigated, and theories proposed and discussed. Most theories of learning assume that healthy people learn continuously, learn throughout their lives, and seem to have an infinite capacity for knowledge. On the other hand, relatively few computational systems have performed cognitively plausible long-term learning. In this paper we explore long-term learning using computational cognitive architectures.

Several of our terms need clarification. By learning, we mean more than simply acquiring knowledge. Learning, sometimes called skill acquisition, involves both the acquisition of knowledge and its use to improve performance. Learning is commonly defined as the improvement in performance with experience and involves both a symbolic and subsymbolic component (Anderson, 2000). We focused our work on symbolic learning, i.e., the learning of discrete pieces of knowledge. Limiting ourselves to symbolic learning also avoids issues associated with the physiology of perception and psychomotor response aspects of cognition. We emphasize the long-term aspect of learning to get past the short-term transients of perception and short-term memory. We also mean long-term in a systems sense, i.e., long enough to reach steady-state behavior. Long-term learning in humans has been studied over periods of decades and performance improvements, specifically in response time, were formalized as the "power law of learning" (Anderson, 2000; Newell & Rosenbloom, 1981). In our study of long-term, symbolic learning, we used two of the most widely used computational cognitive architectures, Soar and ACT-R.

Soar is a symbolic learning system with early success modeling the observed (long-term) power law of learning (Rosenbloom & Newell, 1986). In 1990, Newell proposed Soar as a unified theory of cognition (Newell, 1990) based on single mechanisms for the different components of cognition. Soar uses one form of short-term memory, called working memory, and a single learning mechanism, "chunking", in which the solution to a subproblem is formulated as a production. The single form of long-term memory is the retention of all productions. Soar is a symbolic AI system and does not attempt to model human learning below the symbolic level. Consistent with the learning theory that Soar implements, productions are retained forever and working memory is transient.

The ACT family of theories has a long history of integrating, matching, and explaining psychological data. The current version (November 2006), ACT-R 6, derives important constraints from asking what cognitive processes are adaptive given the statistical structure of the environment (Anderson, 1990). As a formal theory, ACT-R has been broadly tested. Within ACT-R, all declarative and procedural knowledge is retained. The ACT-R system models cognition at both the symbolic and subsymbolic levels. In ACT-R's model of human memory, activation levels are calculated which determine the system's ability to recall the knowledge and the associated latency. Both declarative and procedural memory have calculated activations associated with each piece of knowledge. (Activation for productions is called production utility.) Knowledge

with activations below a user-specified threshold is not retrieved. However, all declarative knowledge and productions are maintained and subsequent experience can raise their activation. Several model parameters control the details of the activation calculations.

Research Questions

We focus on three fundamental questions associated with the nature of long-term learning as modeled by Soar and ACT-R: (1) how long symbolic learning occurs, (2) how the learned knowledge is used or transformed, and (3) whether there are performance problems associated with long-term learning.

Does symbolic learning go on forever?

We first examined a fundamental assumption associated with long-term learning: that it continues indefinitely. When a symbolic system is learning in a domain, it seems inevitable that it would eventually reach one of two possibilities: first, it could learn all the knowledge available about the domain or, second, it could attempt to learn more knowledge than it can retain in its finite memory and fail. In either case, symbolic learning would eventually stop. Newell (1990) wrote that learning, specifically the acquisition of symbolic knowledge, occurs at an "approximately" constant rate independent of the scale of the task. Continuous learning is a tenet of his unified theory of cognition and of Soar, which implemented his theory and successfully modeled the power law of learning (Newell & Rosenbloom, 1981). It is unclear whether this broad claim was intended to be applied to finite domains because he did not add any qualification even though he was well aware that very few chunks were necessary to solve all problems in the three-block Blocks-World domain.

The theory behind the ACT-R system does acknowledge "great reductions in cognitive involvement" based on obeying the power law (Anderson, 2000) as the system gains experience with a domain. ACT-R theory explicitly acknowledges that speed up in response time could continue until the learner reaches the limitations of the embodied physical system. That suggests that the symbolic portion of learning, i.e., not including the psychomotor phase of learning, would eventually end. This analysis suggests that Soar theory predicts that symbolic learning continues indefinitely and ACT-R's theory predicts that symbolic learning eventually stops within a specific domain. Note that these different points of view concern the size of the problem domain; most theories today acknowledge continual symbolic learning in very large domains.

How is learned knowledge used?

The second question concerns how learned knowledge is used or transformed. Theorists have suggested that learning includes the retention of some facts based on a potential use rather than a demonstrated use (Anderson, 2000). But, how and when is it used?

Over the long term, the use of knowledge has been proposed to change in human learning. The learned knowledge starts as declarative knowledge used in problem solving and eventually becomes retrieved solutions without problem solving (Logan, 1988). Three stages have been proposed (Anderson, 1982; Fitts, 1964). The first stage is

the cognitive stage. The knowledge learned is primarily declarative and must be interpreted through problem solving to improve performance. General problem solving techniques are employed using the knowledge available and creating new declarative knowledge. The second stage is the associative stage. Here there is a mix of declarative and task-specific procedural knowledge and the problem solving is transitioning from general methods to methods specific to the problem domain. By the third stage, called the autonomous stage, the knowledge used is all procedural, is compiled from the declarative knowledge, is fast, and is error-free. In this last stage, there is no problem solving necessary because responses are directly recalled. The continued performance improvements in the third stage are based on psychomotor speedup up toward physical limitations (Anderson et al., 2004).

Previous experiments with Soar demonstrated that in some problem domains, even in the long term, Soar only used about half of the learned productions (Kennedy, 2003). Further, although the number of productions used on each problem was approximately constant, more recently learned productions were used more frequently but a smaller portion came from the whole range of the learning process (Kennedy & De Jong, 2003). The results reported here will highlight similarities and differences between ACT-R and Soar in how both systems perform long-term learning.

Are there performance problems in long-term learning?

The performance associated with learning is normally a combination of improvements in effectiveness, i.e., fewer steps employed to accomplish a task, and improvements in efficiency, i.e., using less resources, being quicker, cheaper. Tambe, Newell, and Rosenbloom (1990) defined the former as the cognitive effect and the latter as the computational effect. The performance problem we are concerned with is the latter, how performance, in terms of the processing time to successfully perform the task, changes with learning over long series of problems. Few AI researchers have run symbolic learning systems on long series of problems that emulate long-term learning (through c.f Lebiere, 1999). TacAir-Soar with thousands of productions has been run for hours but did not employ learning (Jones, Laird, & Nielsen, 1994). The default setting in Soar is to have learning turned off. Other more traditional AI learning systems typically have not been run long enough to achieve steady-state behavior probably because the intent was to demonstrate the effects of new learning techniques which are most evident in short runs. When AI systems have been run over the long term, performance problems have occurred in as little as 9, 20, 25, 52, or 100 problems (Bostrom, 1992; Iba, 1989; Leake, Kinley, & Wilson, 1995; Markovitch, 1988; Minton, 1988, 1990; Mooney, 1989; Tambe, Newell, & Rosenbloom, 1990). One reason for the performance problems was the cost of testing the applicability of knowledge to the current task (Minton, 1988, 1990): Problem-solving time was found to grow with the number of productions in the system. Further research suggested the problem was universal to symbolic AI systems (Holder, 1990).

In contrast to these findings, Markovitch and Scott (Markovitch, 1988) reported that their symbolic learner's performance actually improved with forgetting. After their system had learned productions (macros) based on 5,000 training problems and had established a level of performance in terms of a minimum number of nodes searched, as its learned productions were randomly and incrementally removed, its performance

actually improved. Performance peaked when approximately 90 percent of the learned productions had been removed. This work suggested that a system of keeping some productions while removing others might be a way to deal with the utility problem. For computational cognitive architectures, this is problematic because most cognitive systems do not explicitly forget learned knowledge.

At about the same time Soar was proposed as a unified theory of cognition, it was confirmed to suffer performance degradation with continued learning. It was found that some chunks were very expensive in processing time primarily due to an exponential slowdown in the process of matching variables in a rule to the current state (Tambe, Newell, & Rosenbloom, 1990). Restricting the expressiveness of chunks was found to reduce performance costs per chunk but could, in the worst case, require exponentially more chunks to represent the same knowledge (Tambe, Newell, & Rosenbloom, 1990). Further research improved the matching algorithm's performance significantly (Doorenbos, 1995), but did not eliminate the performance problem with continued learning. One component of Newell's cognitive theory implemented in Soar is that productions are the only form of long-term memory and all productions are kept indefinitely. In looking for a basis for relaxing the long-term memory retention premise, analysis of the time between uses of productions was done. This analysis revealed a characteristic transition curve separating the frequently used productions from the rarely used productions. Setting a threshold for removal of productions based on the duration since last use resulted in statistically better performance (Kennedy & De Jong, 2003).

The last issue we will investigate in long-term learning is whether and how ACT-R suffers from the performance problems that plague Soar and AI's symbolic learning systems.

Answers to all three of these questions will contribute to both the fields of AI and Cognitive Science. For Cognitive Science, results should support the development of a theory of long-term learning that may involve forgetting as well as activation. For AI, results should be directly applicable to developing learning systems that are expected to operate autonomously for long periods of time.

Method

To address these research questions, we conducted experiments using Soar and ACT-R. Both systems are available from their user groups ("ACT-R Research Group"; , "The Soar Group"). The classic Blocks-World domain (Winston, 1984) was used because it was simple to implement, has a large search space, and can be scaled easily by increasing the number of blocks. For both systems, runs were made long enough to observe steady-state performance.

Systems

The version of Soar used was 8.6.1 for Windows, run within the Soar Java Debugger (in text view). The ACT-R system, version 6 [r145] was run on several desktop PCs and Macintosh computers. Version 6 implements the latest theory on the compilation of sequentially firing productions into new productions. For both systems, common or default parameters were used as listed in Table 1. To run long series of problems in Soar, command files were generated and run in batch mode. For ACT-R, a

Lisp problem generator was run calling ACT-R to run each problem in the series. Traces of the systems' behavior were analyzed off-line.

Table 1. Systems and Parameter Settings

System	Non-default settings used
Soar 8.6.1	Learning on
ACT-R Version 6 [r145]	Subsymbolic calculations enabled, :esc t Latency factor, :lf = 0.4 Retrieval threshold, :rt = -1 Production learning, :pl=t Enable production learning, :epl=t

For Soar, the only non-default parameter adjusted was to turn learning on. For ACT-R, a small number of its parameters were set at non-default values. Subsymbolic calculations were enabled so that the system used its quantitative theories of declarative knowledge activations and retrieval. The latency factor and the retrieval threshold of ACT-R were adjusted to keep the memory of the problem available as needed during problem solving. The non-default production learning settings allowed production learning and the use of the quantitative ACT-R theory associated with learning and using productions.

Task Description

The general task was to rearrange a fixed set of three named blocks on a table from one configuration to another. As an example, one problem is to move the blocks of one configuration to be identical to the goal configuration, as shown in Figure 1. Moves consist of selecting a block and moving it to the table or on top of another block. The criteria to move a block is that the subject block be clear, i.e., not having another block on top of it, and the destination also be clear. The table is always a legal destination. There are 30 problems in this domain. The Blocks-World domain was set up so that one learned production can solve a problem, which would not be true for more complex problems or problem domains. That simplicity allows us to isolate long-term learning characteristics for both declarative and procedural knowledge. The expectation is that if we find characteristics within one-step problems, those same characteristics would manifest themselves in more complex models.

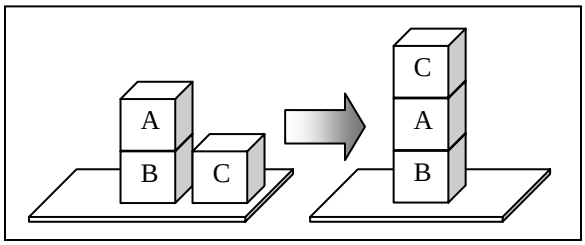

Figure 1: A Blocks World Problem.

Cognitive Models in Soar and ACT-R

Cognitive models of this task were implemented in Soar and ACT-R. Initially, both systems blindly select legal moves with no prior knowledge. There is no planning involved because that was to be learned. Both systems chose from available legal moves based on their knowledge.

When either system achieved the goal configuration, it learned the effective move. In Soar, the solution to the subproblem of choosing the best move for the problem was decided by the one-step look-ahead evaluation and resulted in a "chunk", the new production. In ACT-R, when no solution was already known, moves were tried at random and only the move that solved the problem was saved as a "chunk" in declarative memory. (Note the different uses of the term "chunk" for Soar and ACT-R.) When a solution was retrieved, a new production compiling that information was generated. After several recreations of the same production, in accordance with the ACT-R theory, the production's utility would be increased enough to compete successfully with the solution retrieval production and fire (Anderson et al., 2004; Taatgen & Lee, 2003).

For Soar, three models of Blocks World were provided with the Soar software (The Soar Group, 2005). The look-ahead model was used because it was the only one that learns. The model representation specifies the current and goal states based on what each block is on top of, either the table or another block. Soar sequentially selects a move based on a one-step look-ahead evaluation and applies moves of a block to a new location until the current configuration matched the goal. When Soar does not have the knowledge to immediately select a move, it establishes a subproblem to decide the move to make. The solution to a subproblem is saved as a learned production, a Soar "chunk", which eliminates the need to repeat the solving of the subproblem. Soar's chunks are not written for specific blocks but are generalized to descriptions of any blocks meeting specific on-top and clear conditions.

A similar model was developed for ACT-R following the approach common in the ACT-R community (e.g., Fleetwood & Byrne, 2006; Gunzelmann, 2006; Salvucci, 2006; Taatgen, 2005). The ACT-R model uses its vision module to read in a problem's initial and goal configurations in terms of what the blocks are on. It then attempts to recall a previous move from the current configuration to the goal. If it finds such a move, that move is executed. If it does not, a legal move is created based on any block that can be moved and any possible destination. After making the move, the resulting configuration is evaluated as to whether it achieved the goal. The move achieving the goal is saved as a solution to the problem of achieving that goal from the previous configuration. The model randomly tries moves and recognizes, and then saves, those that achieve the goal. (We did not use the new more general production formulation, called "dynamic pattern matching". We have made our models is available at the ACT-R website ("ACT-R Research Group").)

Both systems begin by conducting general problem solving, noting actions that achieve the goal of solving a problem, and saving that knowledge for future use. Both immediately save the new knowledge as a production. Soar makes that production immediately available for use and ACT-R requires the production to be generated several times to raise its utility enough to be used (Taatgen & Lee, 2003).

Results

For each question, experimental results, analysis, and observations are discussed.

Does symbolic learning go on forever?

When we ran both Soar and ACT-R, on three-block Blocks-World problems, symbolic learning ended as shown by the learning curves in Figures 2 and 3. The learning curves shown are plots of the cumulative number of new productions in the system against the sequential problem number. Plots of the average of five runs for both systems are shown.

Both plots have the same general shape, but their units are very different. Figure 2 shows Soar's learning over only 50 problems while Figure 3 shows ACT-R's learning over 2,500 problems. The scale is very different because Soar learns its last production on problem 15 and ACT-R's last production was learned on problem 1,751. Because Soar learns very general productions, it does not need to see every possible problem in the domain before learning stops. Our ACT-R model must see every problem to generate a production to solve it. The number of productions learned is also very different. All of the Soar runs learned exactly eight productions, all general. ACT-R eventually learned 34-37 productions, one for each problem plus a few due to model coding. Examples of rules learned are presented in Figures 4 and 5.

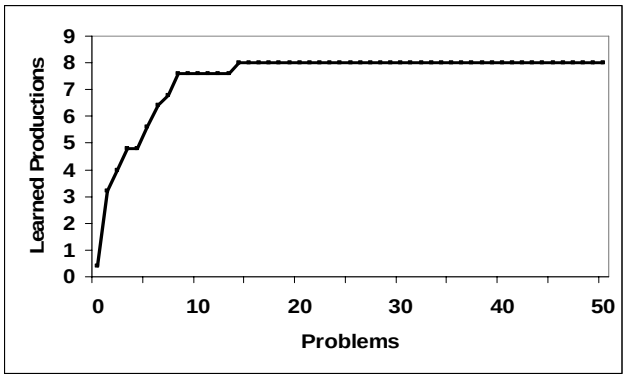

Figure 2: Productions Learned in Soar

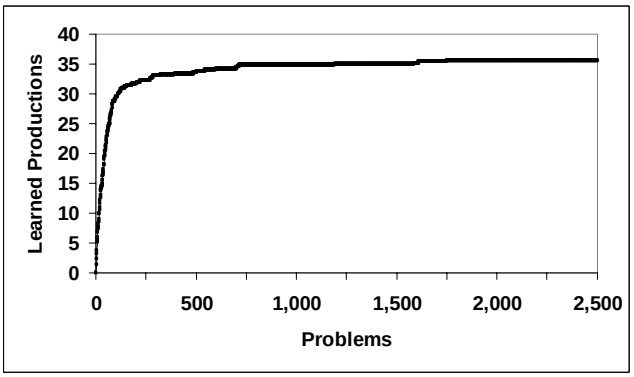

Figure 3: Productions Learned in ACT-R

Soar chunk:

```
sp {chunk-8*d5*tie*2
   :chunk
   (state <s1>     ^name            blocks-world
                   ^problem-space   <p1>
                   ^desired         <d1>
                   ^operator        <o1> +
                   ^ontop           <o2>
                   ^ontop           <o3>
                   ^ontop           <o4>)
   (<p1>           ^name            move-blocks)
   (<o1>           ^destination     <d2>
                   ^moving-block    <m1>)
   (<o2>           ^top-block       <d2>
                   ^bottom-block    <b1>)
   (<o3>           ^top-block       { <t1> <> <d2> }
                   ^bottom-block    <b1> )
   (<o4>           ^top-block       <m1>
                   ^bottom-block    <t1>)
   (<d1>           ^ontop           <o5>
                   ^ontop           { <o6> <> <o5> }
                   ^ontop           { <o7> <> <o5> <> <o6> })
   (<o5>           ^top-block       <m1>
                   ^bottom-block    <d2>)
   (<o6>           ^top-block       <d2>
                   ^bottom-block    <b1>)
   (<o7>           ^top-block       <t1>
                   ^bottom-block    <b1>)
   -->
   (<s1>           ^operator        <o1>  >  )
}
```

Translation:
A production resolving a tie between acceptable next steps is:
IF the desired arrangement is some block “ml” on top of some block “d2” with different blocks “t1” & “d2” on “b1” (i.e., the table)
AND the current state has the block called “ml” on top of block “t1” with blocks “d2” and “t1” on the table
AND a potential next action is to move block “ml” to on top of block “d2”,
THEN that action of moving “ml” to “d2” is better than any other action
(note: “ml”, “t1”, and “b1” are variable names for specific blocks).

Figure 4: A Production Learned by Soar

```
(P PRODUCTION88
  "ATTEMPT-PREVIOUS-SOLUTION-FROM-PARTIAL &
ATTEMPT-PREVIOUS-SOLUTION-SUCCESSFUL - SOLUTION14-0"
  =GOAL>
     ISA          PROBLEM
     A-ON         TABLE
     B-ON         TABLE
     C-ON         BLOCKB
     GA-ON        TABLE
     GB-ON        TABLE
     GC-ON        TABLE
     STATE        PARTIAL-SUCCESS
 ==>
  =GOAL>
     STATE        TRYSOLUTION
  +GOAL>
     ISA          PROBLEM
     A-FREE       YES
     A-ON         TABLE
     B-FREE       NO
     B-ON         TABLE
     C-FREE       YES
     C-ON         BLOCKB
     GA-ON        TABLE
     GB-ON        TABLE
     GC-ON        TABLE
     MOVER        BLOCKC
     SOLUTION     YES
     STATE        MOVE-READY
     TOLOC        TABLE
)
```

Translation:

A production that combines the recall of a previous solution and a production that acts the recall of the declarative information, specifically solution 14, is:

IF the block C is on the block B
AND the blocks A and B are on the table
AND the goal is to have all three of these blocks on the table,
THEN the move to make is block C to the table and set other resulting attributes concerning which blocks are available to be moved (“free”).

Figure 5: A Production Learned by ACT-R

Although symbolic learning in both Soar and ACT-R stopped, there are differences in the system’s behavior. Soar learned productions that are generalized while ACT-R’s productions are specific to individual block names. Therefore, Soar needs far fewer productions to cover the same domain knowledge. In addition, the ACT-R model includes its vision module which introduces noise resulting in additional production learning which is not material to the model. As a result, the point at which learning ends varies between the systems, but the fact that both stop learning is significant. This result is contrary to Alan Newell’s unified theory of cognition (1990) that predicts a constant rate of learning without any conditions, i.e., it expects that learning continues forever.

We do not make any claims with respect to Soar's learning theory applied to infinite or near infinite domains. The ACT-R theory includes both symbolic and subsymbolic components of learning and these results do not directly contradict that theory.

How is learned knowledge used?

There were two findings concerning the use of learned knowledge. The first concerned the overall frequency of use of learned productions and the second dealt with the transformations of knowledge during the stages of learning.

The rational analysis component of the ACT-R theory (Anderson, 1990) addresses the effect of the environment on the operation of rational systems, human or machine. The theory is that the order and distribution of the problems in the domain (environment) affects the operation of the learning system. The use of the learned productions was expected to be uniformly distributed due to the fact that problems were presented randomly.

For Soar, the statistics of production use confirm uniform use. Over five runs of 250 problems each, Soar learned eight new productions but used only four on each run. Those four productions were each used 25 percent of the time (standard deviation was less than 0.05 percent). In ACT-R, over five runs of 2,500 problems, the ACT-R model learned an average of 36 productions but used 26 in four runs and 27 in the other. Over the 2,500 problems, each production was each used 3.85 percent of the time (standard deviation was less than 1.25 percent). (Perfectly uniform use would have been 1/26.2 = 3.81 percent of the time.) So, although Soar used only half of its learned productions, the production use in both Soar and ACT-R was uniformly distributed. Uniformly distributed production use is consistent with the uniformly distributed random ordering of problems presented to the two systems supporting the theory that the environment affects system operation.

The other pattern of use of productions observed in ACT-R concerned the three stages of learning (Anderson, 1982; Fitts, 1964). Figure 4 shows, per problem, the number of moves created during problem solving, retrievals of previously successful moves, and firing of learned move productions per 100 problems. The number of productions fired per 100 problems became approximately 100, or one production per problem in the third stage, as expected.

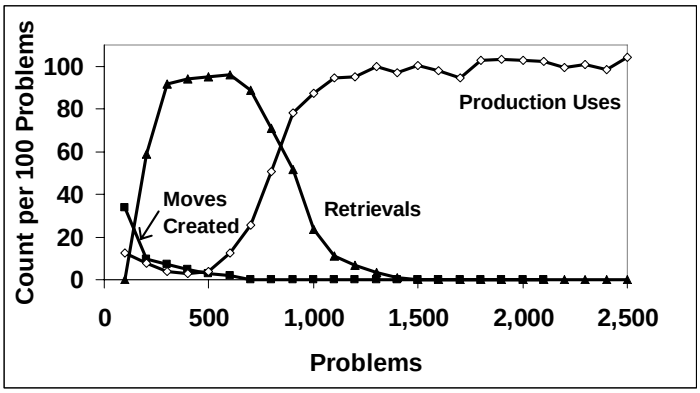

Figure 4. ACT-R Stages of learning

Figure 4 clearly shows three stages of learning. When ACT-R begins, it uses its declarative knowledge of the problem domain and general problems-solving productions to create legal moves. There was no knowledge in the model guiding the generation of moves other than their legality. After some problem solving experience, later problems are solved by retrieving previously generated moves. This is the transition of the second stage of learning. ACT-R compiles the knowledge contained in a retrieval into a new production that contains the information from memory. With continued presentation of problems, ACT-R begins to directly use the new productions it generated. These productions implement the appropriate moves directly without retrieval. Requiring repeated development of a production before allowing its use is how ACT-R controls the development of productions per its theory (Anderson & Lebiere, 1998). Soar does not delay a production's use and therefore does not display these phases of learning.

These runs demonstrated that both systems transitioned from problem solving to using learned knowledge. Soar did so immediately and ACT-R took longer. Both systems reused learned knowledge: Soar used half its learned productions and ACT-R used approximately three-fourths. Consistent with the random order of problems in the domain, both Soar and ACT-R used their productions uniformly. Finally, the ACT-R system demonstrated the three phases of learning.

Are there performance problems in long-term learning?

As discussed earlier, Soar has been shown to exhibit performance problems when run under conditions of long-term learning (Tambe, Newell, & Rosenbloom, 1990). To test whether ACT-R suffered from degraded performance with continued learning, timing data was collected during long runs of the system on Blocks World problems. We hoped to find that a cognitive modeling system would not demonstrate the performance problems found in more traditional symbolic learning AI systems.

Multiple runs were made on personal computers running the Windows operating system with the standalone version of ACT-R version 6 [r145] (Allegro Common Lisp, version 6.2) and on Apple Macintosh G4 and G5 machines running Mac OS X and Macintosh Common Lisp (MCL) version 5. No changes were made to the Lisp garbage collection parameters. Processing time was available from the Lisp programming language (implementation specific) and from the ACT-R system. Both times measurements were associated with solving a problem in ACT-R. The first was the human's response time calculated by ACT-R as a function of the Blocks-World model and ACT-R's sub-symbolic modeling. Figure 5 shows the average ACT-R run times averaged over 10 problems and five runs.

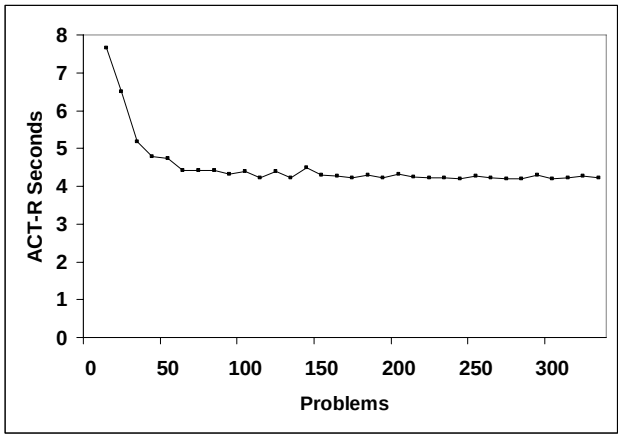

Figure 5. ACT-R Model Times for Blocks World Problems

Figure 5 shows a downward trend in the average time to solve individual Blocks Worlds problems per ACT-R's model of human cognition. This is the expected output of the ACT-R model corresponding to the observed power law of learning. With parameter adjustments for different models of different tasks, ACT-R's output has been shown to correspond to the improved performance with learning that has been seen in many human subject studies (Altmann & Trafton, 2002; Anderson, Bothell, Lebiere, & Matessa, 1998; Taatgen, 2002).

To successfully model long-term learning, ACT-R needs to run for long periods. However, in all of the long-term runs of the Blocks World problems, it eventually failed (froze). Failures ranged from as early as 340 problems on a small Apple Macintosh G4 (desktop) with 384MB of memory to over 32,000 problems on a Windows PC with 1GB of memory. Note that ACT-R was able to adequately run on the older hardware; we used it in order to see performance issues in shorter time scales than a current top of the line computer. Figures 6 shows the Lisp processing time on the Apple Macintosh G4 that ACT-R used in producing the above results up to its point of failure. The system continually required additional memory until it ran out of physical memory and failed. The causes of ACT-R's performance problems have been investigated and are discussed in the next section.

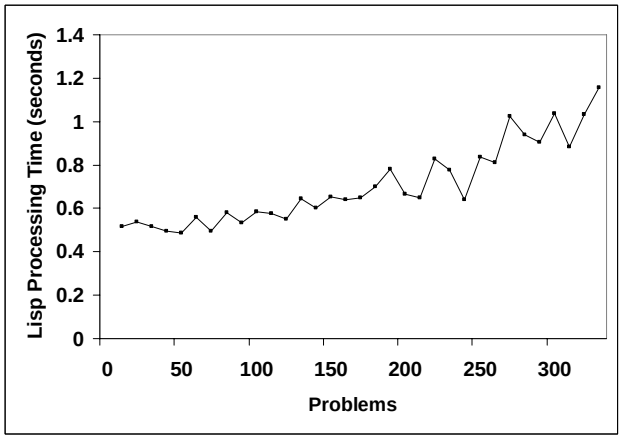

Figure 6. ACT-R Run Times for Blocks-World Problems

ACT-R and the Performance Problems

There were two major aspects to the performance problems in ACT-R. The first is the continual need for additional memory which eventually lead to requests for more memory than was available resulting in a fatal error. The second aspect of the performance problem was the increasing processing time with continued learning that caused a slowing down of the system which could lead to the system running in slower than real time.

To investigate ACT-R's performance problem, we used a simpler domain than the original Blocks-World domain. We found we could demonstrate the performance problems using the simple task of counting objects with ACT-R running on an older hardware setup (the same small Apple Macintosh G4 (desktop) with 384MB of memory that was used in earlier experiments). We used the counting model provided with the ACT-R tutorial ("ACT-R Research Group") expanded to count from one to ten. We then ran this model of counting to ten in blocks of a hundred repetitions. The code and model are available from the ACT-R website ("ACT-R Research Group"). The counting model focused primarily on declarative learning; no procedural learning occurred.

We were able to replicate the prior performance issues on the simpler task of counting, suggesting that the performance problems were not due to the Blocks-World model. Because the counting model is quite simple, we were able to systematically explore possible causes and solutions to the performance problems. Possible sources of the performance problems range from the system's hardware and operating system software environment, to ACT-R's implementation in Lisp, to the ACT-R theory itself. We will address each of these in turn.

Hardware and Operating System Software

Failure of ACT-R on long series of problems was found to be independent of the hardware and operating system: we used both Macintoshes and Windows PCs and in all cases, the system locked up, i.e., failed. The failures were also independent of the Lisp environment: we used Allegro and Macintosh Common Lisps, and ACT-R in both standalone and interpreted versions. Therefore, we ruled out the hardware and system software as the cause of the performance problems.

ACT-R's Implementation

We then turned to the ACT-R implementation as a possible cause. Previous long runs of ACT-R (Lebiere, 1999) used a different version of ACT-R and apparently did not have performance problems, possibly because is was using an optimized version of the activation formula which will be discussed below. The version we used, Version 6, was not optimized for performance but for stability and modifiability (Bothell, 2006). Therefore, it was possible that some knowledge representations or processes may have been implemented such that a memory leak occurred. Indeed, such a problem was found. To be consistent with the ACT-R theory, the current implementation creates copies of chunks of declarative knowledge for different uses. A copy is made for each retrieval from declarative memory and each change of a buffer, even if it is a duplicate of an existing chunk. Each temporary copy has an internal name. Although duplicate chunks are later recognized by the ACT-R code and not added to declarative memory, their

temporary names are retained within the ACT-R system and accumulate indefinitely. The author of the ACT-R code, Daniel Bothell, graciously made minor revisions to the ACT-R code (changes r292, r294-r297) to provide a function to clean up temporary names. These revisions allowed ACT-R with default settings to run an order of magnitude longer than previous runs (over 5,000 blocks). Figure 7 shows a series of runs of the tutorial's counting model with and without this function. (The runs labeled Full Activation Formula and Efficient Activation Formula employed this function and are discussed below.) These changes resolved the apparent memory leak caused by the retention of internal Lisp names. Note that these results used ACT-R with default settings (see Table 2). Performance problems still existed with non-default settings. Other performance issues, perhaps as a result of the ACT-R theory, are discussed next.

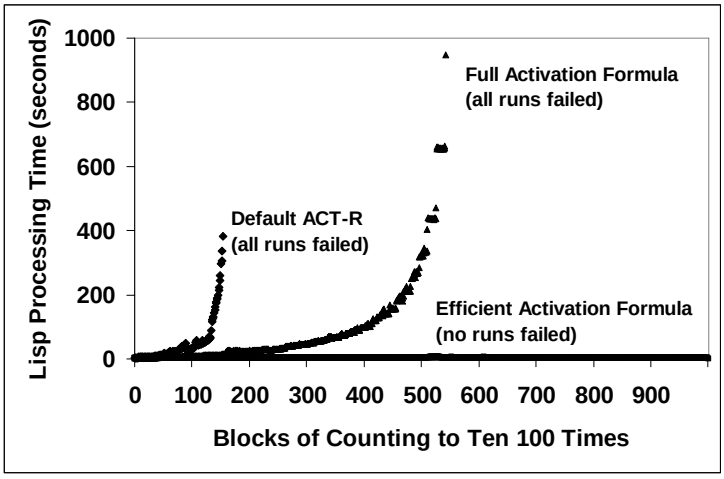

Figure 7. Comparison Long-Term Counting Runs with Different Parameters

Table 2. Settings for Long-Term Counting Runs

Parameter	Default ACT-R (Ver. 6 [r297])	Full Activation Formula (Ver. 6 [r297])	Efficient Activation Formula (Ver. 6 [r297])
Latency factor, :lf	0.04	0.04	0.04
Base Level Learning, :bll	0.5	0.5	0.5
Optimized Learning, :ol	optimized	full formula	1 (one term)

ACT-R Theory

There are parts of the ACT-R theory that require the retention of historical information indefinitely and, therefore, cause increasing memory requirements and computational burden over the long term. The ACT-R theory's representation of declarative memory calculates an activation level for every chunk of declarative memory. This activation is used to determine whether a chunk is successfully recalled and the latency of the recall process. The activation calculation includes two components, a base level of activation and an associative contribution. In accordance with the ACT-R theory, the base-level activation is a function of the frequency and recency of the uses of the subject chunk. The base-level learning formula, equation 4.1 (Anderson & Lebiere, 1998) is:

$$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right) + \beta \qquad \text{Eq. 1}$$

where the base-level activation for a chunk, B_i, is the log of the sum over all n previous uses of the chunk, of the time lag since each use, t, raised to power of a negative parameter, d, plus a constant. The formula's parameter d is set via the base-level learning ACT-R parameter. The full implementation of Equation 1 requires keeping the history of the each use of all chunks throughout the model's run. This creates a monotonically increasing demand for additional memory and causes an increase in computational resources to be used, eventually leading to a memory-related failure.

The computational burden of the full base-level learning formula has been known within the ACT-R community for years. To address the burden, ACT-R has a parameter, :ol "optimized learning". The default value for this parameter directs the use an optimized learning formulation (Anderson & Lebiere, 1998), which is:

$$B_i \approx \ln\left(\frac{n}{(1-d)} t_n^{-d}\right) \qquad \text{Eq. 2}$$

where all the terms are the same as in Equation 1: the base-level activation for a chunk, B_i, is approximately the log of the n previous uses of the chunk divided by 1 minus a parameter, d, times the time lag since the first use, t, raised to power of the negative parameter, d. This formulation does not require the history of the chunk's uses, only the number of uses, n, and assumes a uniform distribution of uses since creation of the chunk. However, that formula is considered less cognitively plausible based on the assumed uniform distribution of previous uses and its inconsistency with observations.

A more computationally efficient approximation was recently developed (Petrov, 2006). The new, improved approximation is:

$$B_i \approx \ln\left(\sum_{j=1}^{k} t_j^{-d} + \frac{(n-k)}{(1-d)} * \frac{(t_n^{1-d} - t_k^{1-d})}{(t_n - t_k)}\right) \qquad \text{Eq. 3}$$

where all the terms are the same as in Equations 1 and 2 with the additional parameter k. This formulation of base-level activation is a combination of the full and optimized formulations, Equations 1 and 2. It uses a parameter k for the number of the most recent chunk uses to be calculated as in the full base level learning, Equation 1, and then the optimized learning approximation for the rest of the terms, Equation 2. Note that this formulation is more cognitively plausible than Equation 2, but probably less plausible than Equation 1. It is an open question how much history is needed to maintain high cognitive fidelity and plausibility. This formulation is already included in ACT-R.

The implementation of Equation 3 has the user specify a finite number of previous uses which then results in finite memory requirements. Specifying a finite number of uses ends the unbounded growth in the demand for more memory. Figure 7 shows comparable long-term runs using the different equations for base level learning. Using the naming function and using Equation 3 resolved the performance problems for the counting model. We then applied these changes to the original Blocks-World domain to test the effects there.

With the implementation of the naming function and the use of Equation 3, ACT-R successfully ran 1,000 Blocks-World problems, long past the original failure point of 340 problems. This performance demonstrated that the original failure of ACT-R with long-term learning was due to these problems.

On the procedural side of memory, ACT-R theory has a similar formula for the utility of a production and supports compilation of productions into a new productions. We hypothesize that a similar memory demand and computational slowdown result from the code's implementation of the theory for procedural. However, the effects are not as great and have not been noted in the long-term learning experiments we have conducted. Therefore, this constitutes future work. Table 3 summarizes the results of our exploration of the performance problems in ACT-R.

Table 3. ACT-R and the Performance Problems

	Performance Problem	Cause	Approach	Evidence
Hardware and System Software	System locks up	Not hardware or software	--	Problem occurs with different systems, and Lisp versions
ACT-R's Implementation	Memory leak	Retention of temporary chunk names	New function to clear up names	Long runs with and without new function (Figure 7)
ACT-R Theory	Declarative memory overflow and processing slowdown	Base level learning equation requires the retention of all previous uses of each chunk	"Optimized learning" approximation or simplified base-level activation formula	Long runs with and without simplified base-level activation formulation (Figure 7)
	Procedural memory overflow and processing slowdown	Retention of production use history (hypothesized)	Future research	Future research

Conclusions and Discussion

We conclude that Soar and ACT-R have similar long-term learning characteristics. We explored three questions: does learning continue forever, how is learned knowledge used, and whether ACT-R suffers from performance problems like Soar and more traditional symbolic AI systems do.

For the first question, we found that in finite domains, symbolic learning on long-series of problems eventually stops in both Soar and ACT-R which suggests that learning stops in finite domains in humans as well. This result is contrary to Alan Newell's unified theory of cognition (1990) that predicts a constant rate of learning without any conditions, i.e., it expects that learning continues forever. Also, this finding should not be over-generalized to extremely large or infinite domains in which learning could continue for an extremely long, if not infinite, period of time.

On the use of learned productions, there are several differences between Soar and ACT-R. Soar learns from one learning opportunity and makes the new procedural knowledge available immediately. In contrast, ACT-R requires many learning opportunities before a new production is usable. Soar also creates procedural knowledge that is more general and therefore more powerful than the productions that ACT-R creates. There are advantages and disadvantages to both approaches for AI systems and Cognitive Science purposes.

Soar's learning mechanism has enabled Soar modelers to build some very powerful AI systems. However, one of the criticisms of Soar's learning mechanism is that it is too powerful (Anderson & Lebiere, 1998), specifically, Soar creates too many

chunks. Soar's powerful learning mechanisms do not hurt the AI engineering endeavors, but they can make matching human-level performance data more difficult. ACT-R's learning mechanisms are not as computationally powerful as Soar's, but ACT-R learns at both symbolic and sub-symbolic levels as humans do. Consistent with ACT-R's high-level goal of matching human cognitive behavior, it is also able to demonstrate different phases of learning.

Finally, we found that ACT-R, like Soar, suffers from performance problems with long-term learning. We found that the performance problems were caused by both an ACT-R implementation detail and ACT-R theoretical issues. A coding change resolved the performance problems caused by the implementation detail and the updated version of ACT-R with default parameters can perform long-term symbolic learning.

Two important and related theoretical issues remain, however. Both deal with the consequences of the long-tem retention of learned knowledge. First, the current ACT-R theory for the sub-symbolic activation of declarative memory requires the retention of all previous uses of each item in declarative memory. Any computational theory that must maintain a history of its operation without bounds will, eventually, run out of storage. Therefore, the activation theory and its mathematical formulation need revision to address long-term learning, not just as a computational efficiency issue (Petrov, 2006), but as a cognitively plausibility issue. A similar memory overflow issue applies to procedural learning and needs to be addressed as well. (Note that in January, 2007, a new utility equation for ACT-R will be implemented that does not require the retention of all previous uses (Anderson & Fu, 2006; Bothell, 2006).

The second theoretical issue associated with long-term, symbolic learning is more general. Any theory of learning that continually adds new symbols to memory must, eventually, run out of storage. A finite, symbolic memory capacity theoretically applies to both human and machine learning systems. In computer systems, a limited memory capacity is familiar, though no overall capacity limitations in humans have been shown.

This capacity limit applies to all symbolic learning, i.e., both declarative and procedural. The work reported here has focused on declarative memory as the source of the observed degraded performance within ACT-R. However, procedural memory is expected to have the same performance problems and theoretical issues.

This work also has implications for AI systems. Symbolic AI learning systems need to address these capacity issues to avoid system failures. Possible approaches are to limit learning based on memory capacity, incorporate forgetting of low value items to make room for continued learning, or incorporate automatic representational changes based on memory capacity.

On the cognitive science side, in spite of these theoretical issues, humans do not appear to have long-term performance problems nor appear to have reached their memory capacity. Like AI systems research, cognitive science needs to address these theoretic capacity issues. Future research may develop a cognitively plausible justification for removal of some knowledge, for example, symbols that have not been used for a very long time (e.g., they have very low activation levels) may be removed permanently.

Alternatively, it may be possible to change the representation so that those symbols require less memory.

Long-term learning is at the core of theories of cognition, either natural or artificial. Research aimed at advancing our understanding of cognition is greatly facilitated by the existence and availability of computational cognitive modeling systems implementing theories of cognition. Implemented theories of cognition provide a foundation for building and testing computational cognitive models. Our work is an example of the use of computational cognitive models built on the implementations of different theories of cognition to advance the science toward a unified theory of cognition.

Acknowledgments

The authors are deeply indebted to Daniel Bothell for clarifying ACT-R's design and implementation philosophy and for making modifications to ACT-R to address the memory leak associated with chunk names supporting this research. This work and paper benefited from discussions with the cognitive modeling group at the Naval Research Laboratory which includes: Len Breslow, Sara Kriz, Mike Brudzinski, and Malcolm McCurry. We would also like to thank anonymous peer reviewers who ensured our clarity and precision.

This work was performed while the first author held a National Research Council Research Associateship Award at the Naval Research Laboratory. This work was partially supported by the Office of Naval Reactors under job order numbers 55-8551-06, 55-9019-06, and 55-9017-06. The views and conclusions contained in this document should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U. S. Navy.

References

ACT-R Research Group. *ACT-R* Retrieved October 13, 2006, from http://act-r.psy.cmu.edu/

Altmann, E. M., & Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive Science, 26*, 39-83.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89*(4), 369-406.

Anderson, J. R. (1990). *The Adaptive Character of Thought*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (2000). *Learning and Memory: an Integrated Approach* (2nd ed.). Hoboken, NJ: John Wiley & Sons.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglas, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of mind. *Psychological Review, 111*(4), 1036-1060.

Anderson, J. R., Bothell, D., Lebiere, C., & Matessa, M. (1998). An integrated theory of list memory. *Journal of Memory and Language, 38*, 341-380.

Anderson, J. R., & Fu, W.-T. (2006). From recurrent choice to skill learning: A reinforcement-learning model. *Journal of Experimental Psychology: General, 135*(2), 184-206.

Anderson, J. R., & Lebiere, C. (1998). *The Atomic Components of Thought*. Mahwah, NJ: Erlbaum.

Bostrom, H. (1992). *Eliminating Redundancy in Explanation-Based Learning.* Paper presented at the Ninth International Workshop on Machine Learning, San Mateo.

Bothell, D. (2006). personal communication.

Doorenbos, R. B. (1995). *Production Matching for Large Learning Systems.* Carnegie-Mellon University, Pittsburgh.

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), *Categories of human learning*. New York: Academic Press.

Fleetwood, M. D., & Byrne, M. D. (2006). Modeling the visual search of displays: A revised ACT-R/PM model of icon search based on eye tracking data. *Human Computer Interaction, 21*.

Gunzelmann, G. (2006). *Understanding Similarities in the Performance on Different Orientation Tasks: Strategy Adaption.* Paper presented at the Seventh International Conference on Cognitive Modeling, Trieste, IT.

Holder, L. B. (1990). *The General Utility Problem in Machine Learning.* Paper presented at the Seventh International Conference on Machine Learning, Austin, TX.

Iba, G. A. (1989). A Heuristic Approach to the Discovery of Macro Operators. *Machine Learning, 3,* 285-317.

Jones, R., Laird, J., & Nielsen, P. (1994). *Coordinated Behavior of Computer Generated Forces in TacAir-Soar.* Paper presented at the Fourth Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL.

Kennedy, W. G. (2003). *Long-Term Learning in Soar and its Application to the Utility Problem.* George Mason University, Fairfax.

Kennedy, W. G., & De Jong, K. A. (2003). *Characteristics of Long-term Learning in Soar and its Application to the Utility Problem.* Paper presented at the Twentieth International Conference on Machine Learning (ICML-2003), Washington, DC.

Leake, D. B., Kinley, A., & Wilson, D. (1995). *Learning to Improve Case Adaptation by Introspective Reasoning and CBR.* Paper presented at the First International Conference on Case-Based Reasoning, Sesimbra, Portugal.

Lebiere, C. (1999). The dynamics of cognition: An ACT-R model of cognitive arithmetic. *Kognitionswissenschaft, 8*(1), 5-19.

Logan, G. G. (1988). Toward an Instance Theory of Automatization. *Psychological Review, 95*(4), 492-527.

Markovitch, S. a. S., P.D. (1988). *The role of forgetting in learning.* Paper presented at the Fifth International Conference on Machine Learning, Ann Arbor, MI.

Minton, S. (1988). *Quantitative Results Concerning the Utility of Explanation-Based Learning.* Paper presented at the Seventh National Conference on Artificial Intelligence, St. Paul, MN.

Minton, S. (1990). Quantitative Results Concerning the Utility of Explanation-Based Learning. *Artificial Intelligence, 42,* 363-391.

Mooney, R. (1989). *The Effect of Rule Use on the Utility of Explanation-Based Learning.* Paper presented at the Eleventh International Joint Conference on Artificaila Intelligence, Los Altos.

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

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of Skill Acquisition and the Law of Practice. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Erlbaum.

Petrov, A. A. (2006). *Computationally Efficient Approximation of the Base-Level Learning Equation in ACT-R.* Paper presented at the Seventh International Conference on Cognitive Modeling, Trieste, IT.

Rosenbloom, P. S., & Newell, A. (1986). The Chunking of Goal Hierarchies, A Generalized Model of Practice. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine Learning* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Salvucci, D. D. (2006). Modeling driver behavior in a cognitive architecture. *Human Factors, 48*, 362-380.

The Soar Group. *Soar* Retrieved July 7, 2005, from http://sitemaker.umich.edu/soar/home

Taatgen, N. A. (2002). A model of individual differences in skill acquisition in the Kanfer-Ackerman Air Traffic Control Task. *Cognitive Systems Research, 3*(1), 103-112.

Taatgen, N. A. (2005). Modeling parallelization and flexibility improvements in skill acquisition: from dual tasks to complex dynamic skills. *Cognitive Science, 29*(3), 421-455.

Taatgen, N. A., & Lee, F. J. (2003). Production Compilation: A simple mechanism to model complex skill acquisition. *Human Factors, 45*(1), 61-76.

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The Problem of Expensive Chunks and its Solution by Restricting Expressiveness. *Machine Learning, 5*, 299-348.

Winston, P. H. (1984). *Artificial Intelligence* (2nd ed.). Reading, MA: Addison-Wesley Publishing Company.