Modeling Single-Task Performance Improvement Using EPIC-Soar

Ronald S. Chong (rchong@umich.edu)
Artificial Intelligence Laboratory
1101 Beal Avenue
University of Michigan
Ann Arbor, MI 48109-2110

Abstract

We present a *task-independent* learning procedure which produces gradual performance improvement through experience on a perceptual-motor task. It utilizes a data structure that represents a strategy for performing the task. This learning procedure is a direct result of the constraints that occur when realistic perceptual and motor mechanisms (EPIC) are mated to a learning cognitive modeling architecture (Soar).

Introduction

What is learned when a subject's performance on a perceptual-motor task improves with experience? This is an open research question, but the improvements may be attributed to cognitive and motor learning.

In the cognitive system, there are at least three possible source of task performance improvement: 1) the proceduralization of the declarative description of the task (instructions) through *chunking* or *compilation*; 2) the chunking or compiling of any problem solving necessary to perform the task; 3) the development of task strategies and priorities.

In this work, we suggest a fourth possible source of learning that may take place in the cognitive system. We will present a *task-independent* learning procedure for modeling performance improvements for perceptual-motor tasks. The inputs to the learning procedure are: a) a representation of a task strategy; and b) a task performance model that is equivalent to what would be the product of the first three learning procedures mentioned above. We apply the learning procedure to show *additional* task performance improvement.

This work is based on a hybrid architecture called EPIC-Soar (Chong, 1995; Chong & Laird, 1997). A brief discussion of EPIC, Soar, and EPIC-Soar will be presented first, followed by a description of the learning procedure and its application to a two-choice reaction-time task.

EPIC

EPIC (Executive Process-Interactive Control) (Meyer & Kieras, 1997a, 1997b) is an architecture whose primary goal is to account for detailed human dual-task performance. It greatly extends the work begun with the Model Human Processor, MHP (Card, Moran, & Newell, 1983).

Like MHP, EPIC consists of a collection of processors and memories. However, two key points distinguish EPIC from its predecessor. First, the EPIC perceptual (*visual*, *auditory* and *tactile*) and motor (*ocular*, *vocal*, and *manual*) processors are much more elaborate, each representing a synthesis of the most recent empirical evidence. Second, EPIC is a computational system that can be programmed and executed.

Since EPIC was designed to study performance, its cognitive processor does not have the ability to learn.

Soar

Soar is a general architecture for building intelligent systems and for modeling human behavior (Rosenbloom, Laird, & Newell, 1993; Newell, 1990). Soar has been used to model human capabilities such as learning, problem solving, planning, search, natural language and HCI tasks.

Soar is a goal-oriented architecture. When a situation arises where progress ceases on a goal, Soar generates a subgoal. In the subgoal, Soar searches for knowledge to apply to allow progress to resume in the original goal. Soar incorporates a single learning mechanism called *chunking* which compiles the search in a subgoal into productions in the supergoal so that subgoaling is avoided the next time Soar is in the same or similar situation. In combination with various problem solving methods, chunking has been found to be sufficient for a wide variety of learning (Lewis, et al., 1990; Miller and Laird, 1996). However, Soar does not have embedded in it a theory for the details of perception and motor processes.

EPIC-Soar

The motivation to integrate EPIC and Soar can be considered a response to (Newell, 1990):

> *"...one thing wrong with much theorizing about cognition is that it does not pay much attention to perception on the one side or motor behavior on the other....The result is that the theory gives up the constraint on...cognition that these systems could provide. The loss is serious—it assures that theories will never..be able to tell the full story about any particular behavior."*

At present there are but a few architectures that combine perception, cognition, and action. EPIC, as we have seen, takes up where the Model Human Processor left off and represents a significant advancement of its original ideas.

ACT-R/PM (Byrne & Anderson, 1997) is a recent endeavor to add a Vision Manager and a Motor Manager to ACT-R (Anderson, 1993).[1]

Addressing this issue from a different angle, (Gordon, et al., 1997) highlight the paucity of tools to support the development of computational cognitive models of users engaged in interactive tasks. One of the many requirements suggested

[1]The Motor Manager of ACT-R/PM is based on the motor processor of EPIC.

of such tools is the support for simulated perceptual and motor action mechanisms to allow cognitive models to interact with external task simulations. Through a number of case studies, they demonstrate that user interface development environments (such as Tcl/Tk, Garnet, and Amulet) can be employed to this end. Although their perceptual and motor mechanisms are not as psychologically rigorous as those of EPIC, this is nevertheless a step in the right direction.

Finally, EPIC-Soar (Chong, 1995; Chong & Laird, 1997) is an integration of the perceptual and motor processors of EPIC with Soar. This merger is an attempt to get both the detailed predictions and explanations provided by the perceptual and motor processors of EPIC (an ability Soar does not possess) and the problem solving, planning, and learning capabilities of Soar (an ability EPIC does not possess).

An Observation From Expert Performance

Previously, we (Chong & Laird, 1997) identified and analyzed several types of executive process knowledge needed to transition from novice to expert performance on a dual-task scenario modeled in EPIC and EPIC-Soar. Of the four knowledge types identified, two have emerged as relevant.

The first type of knowledge causes *anticipatory motor programming*, i.e., it *prepares* the motor system for an upcoming command. When a command is sent to a motor processor in EPIC, the command passes through two phases—preparation, then execution—before the motor processor is free to execute another command. These phases take time to complete. If the preparation phase of a soon-to-be-sent command can be completed ahead of time, then the preparation phase (and the time required to prepare) can be avoided when the command is finally sent. This can result in a significant reduction in the time to execute the command.

The second type of knowledge causes consecutive commands for the *same* modality to be *pipelined*. The motor processors in EPIC can execute only one command at a time, but pipelining is a way to make the motor processor work on two consecutive commands at the same time. Because the motor processor has two phases, it is possible to send a command x that first gets prepared and then executed. While x is in the execution phase, another command y can be issued since the preparation phase is free. The motor processor, while continuing to execute x, will prepare the movement features for command y. When the execution of x completes, the features of y are immediately handed over to the execution phase for execution. Because the features for y were prepared and waiting, the time to execute y is significantly reduced.

The key observation of these two knowledge types is that they produce expert performance simply because they allow whole commands (or parts of command processing) to be moved, or *promoted*, to a *chronologically earlier event*. Anticipatory motor programming essentially moves the preparation phase to an earlier time. Pipelining moves the whole motor command to an earlier time such that the command will be issued as soon as the execution of a previous command (that uses the same processor) has begun. This observation of motor promotions is the inspiration for the learning procedure presented here.

Before describing the learning procedure, we will first present the representation of the initial knowledge used by the learning procedure.

Initial Knowledge

In addressing the question "what is learned when a subject's performance on a perceptual-motor task improves?", we asked a second, more fundamental, question: "what initial knowledge does a subject possess *after* receiving task instructions but *before* beginning to perform a task?" We hypothesize that this initial knowledge plays a significant role in the learning process.

Example Task: A Two-Choice Reaction Time Task

Consider this contrived example task. The instructions are:

When a trial starts, you are to look at the fixation point. A stimulus will appear in the target circle (which is displaced 16 degrees vertically below the fixation point). When the stimulus appears, look at the stimulus. If it is a left arrow, respond by pressing the key sequence L0, L1, L2, L0 *with the index, middle, ring, and index fingers of the left hand; if it is a right arrow, respond by pressing the key sequence* R0, R1, R2, R0 *with the index, middle, ring, and index fingers of the right hand. Look back to the fixation point to begin a new trial.*

Chronological Task Strategy Data Structure

It is impossible to know exactly what is in a subject's head after reading task instructions,[2] but we will assume that the subject understands the task instructions and is adequately motivated.

In developing this theory of learning, we needed to hypothesize as to the representation of the task that is available following language processing. Previous work has suggested that it is difficult to distinguish exactly which aspects of a task description are declarative data structures in Soar's working memory, and which are procedural associative structures (production rules) in long-term memory (Young & Lewis, 1997).

For this initial work, we have made the simplifying assumption that the complete structure of the task is available in declarative data structures in working memory. Our theory of acquisition is not tied to this specific representational form, but to its content, although by necessity, the algorithms have been developed for the specific representation we present.

We have invented a data structure called a *chronological task strategy*. This structure is intended to represent the initial knowledge that subjects have about the chronological ordering of perceptual events, and the motor commands that are required (per the task instructions) at the occurrence of each perceptual event. This structure and a task performance model (whose behavior is congruent with the strategy structure) are hand-coded and provided to the system.

[2]Previous Soar research (Huffman, 1994; Lewis, Newell, & Polk, 1989) has explored the acquisition of procedures from natural language instructions. Language comprehension issues however are outside the scope of this work.

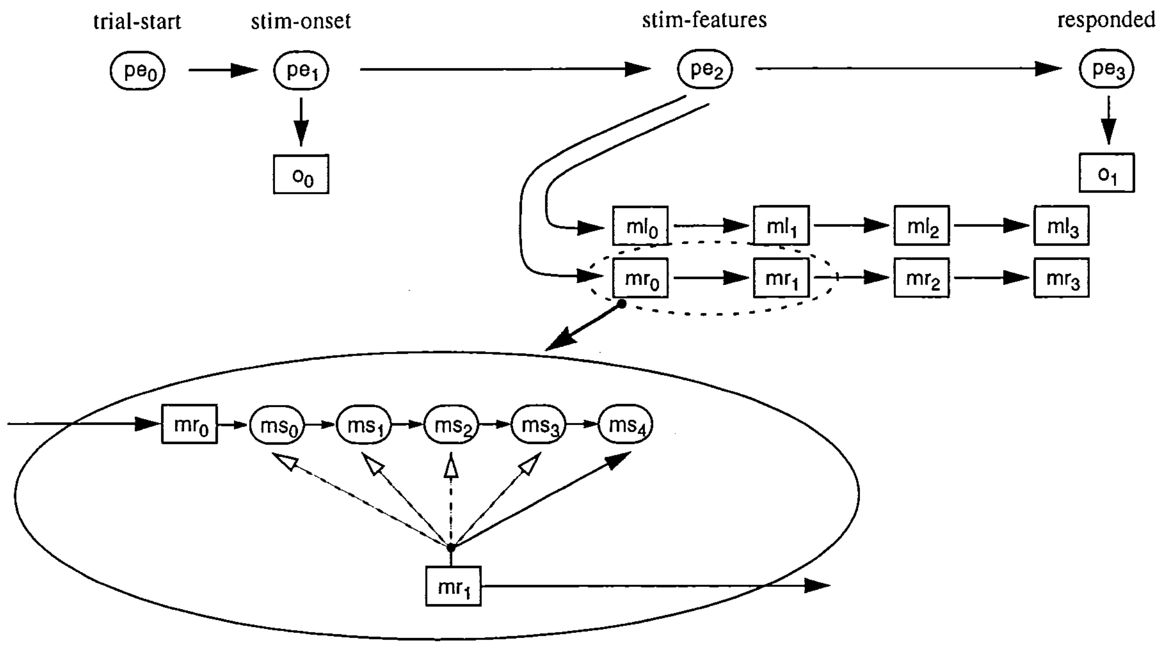

Figure 1: The data structure for the two-choice reaction-time task. The *pe-nodes* pe_0-pe_3 represent the knowledge of the chronological ordering of events in a trial. The *mc-nodes* are the command to be performed; o_0 = LOOK-AT-STIM; ml_0-ml_3 = the keypress command chain a left arrow; mr_0-mr_3 = the keypress command chain for a right arrow; o_1 = LOOK-AT-FIXATION-POINT. The inset depicts the actual link between nodes in a command chain.

The data structure itself is a sparse tree consisting of two types of nodes: event nodes and motor command nodes. Figure 1 shows a possible[3] chronological task strategy structure for the two-choice reaction-time task.

The oval nodes are perceptual event nodes or *pe-nodes*. The nodes in boxes are motor command nodes or *mc-nodes* which contain the motor commands to be executed. The nodes can be linked together to form command chains when a sequence of actions *for the same modality* are required. This structure not only represents the chronological ordering of perceptual events and the motor commands as stated, implied, or inferred from the task instructions, but it also captures the *preconditions* of motor commands; before command ml_0 can be generated, stim-features must have occurred; and command ml_1 can be generated only after command ml_0 has been generated.

There are two additional features of this structure that are present solely because we are building our model in a cognitive system that is constrained by both perception and action. The first is the stim-features perceptual event. This event is necessary because of the constraint of perception in EPIC. In the human visual system, when a stimulus appears, the features that define the object are not instantly available because they take time to propagate from the sensory system, through perception, and finally into working memory. EPIC likewise models this delay, hence the *pe-node* representing the delayed arrival of the stimulus features.

The second feature concerns the links between *mc-nodes* in the motor chain. Recall that when a command is sent to a motor processor in EPIC, the command passes through two phases—preparation, then execution—before the motor processor is free to execute another command. These phases take time to complete, but EPIC sends motor processor status messages that report on the state of each phase (whether each phase is busy or free). Additionally, EPIC may provide proprioceptive feedback on the state of the *effector* performing the command. This is true for manual (hand) commands. When a manual command is issued, the *tactile* sensory processor reports on the state of completion of the action, e.g., for a keypress command, the tactile processor reports when the key is touched, when the downstroke is completed, and when the upstroke is completed.

These messages can collectively be used to define a *precondition space* for *mc-nodes* located at position two or deeper in a command chain. This space allows a model to produce a potentially wide[4] range of performance styles,

[3]There can be no single correct structure to represent a task since a) it is possible to have many different strategies that all accomplish the same task; b) task strategies may change as task conditions change; c) biases in interpreting the task instructions can result in different strategies.

[4]Manual commands have five possible preconditions; ms_0, ms_1, ms_2, ms_3, and ms_4. Ocular commands have only two possible preconditions—ms_0 and ms_1—since there is no proprioceptive feedback for ocular commands,.

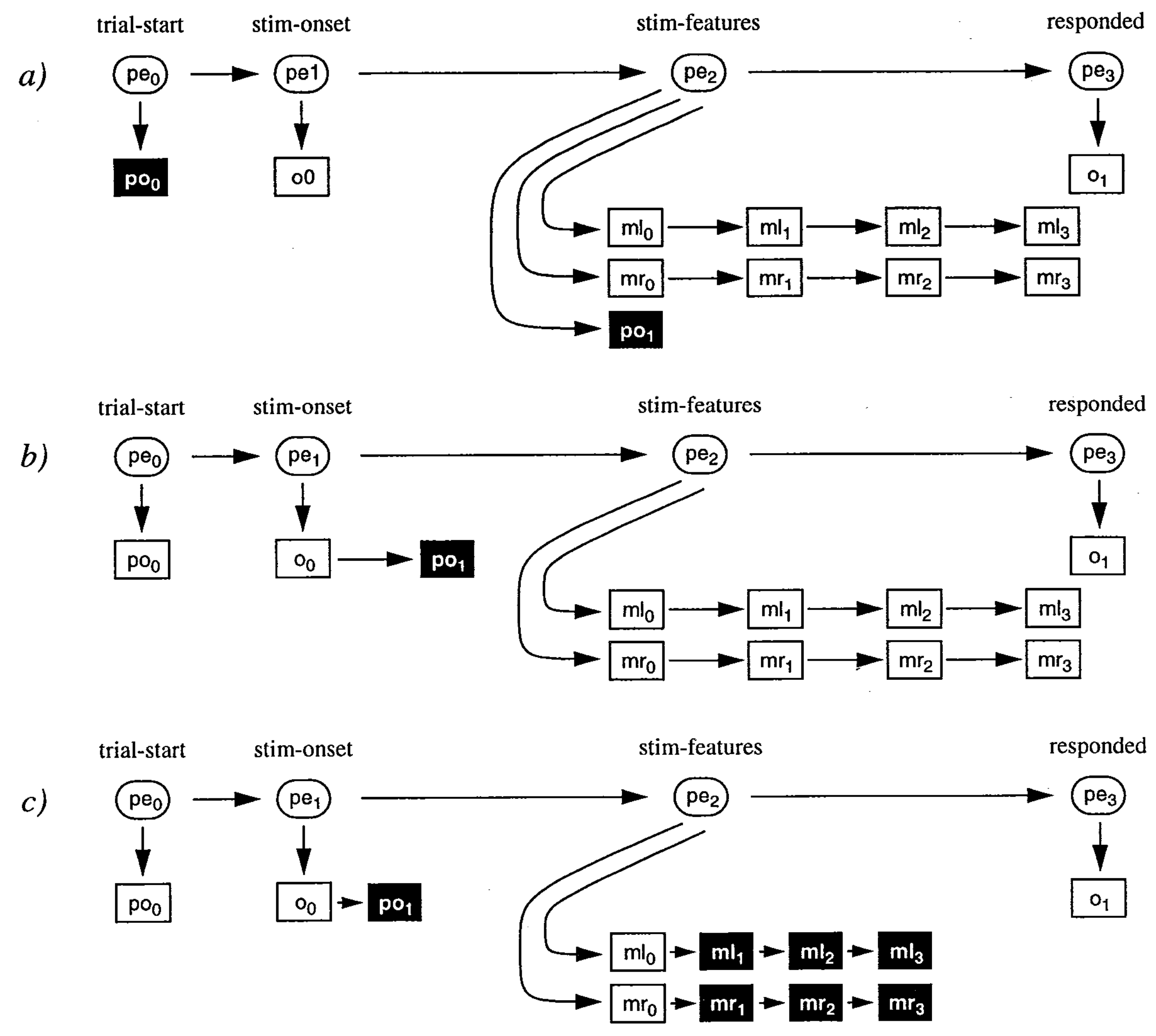

Figure 2: The progression of promotions on the two-choice reaction time task: a) *prepare-promotions* creates po_0 = PREPARE-LOOK-AT-STIM on pe_0 and po_1 = PREPARE-LOOK-AT-FIXATION-POINT on pe_2; b) a *pe-promotion* moves po_1 from pe_2 to pe_1; c) *ms-promotions*—depicted by shorter arrows between motor chain nodes—apply to the motor chains on pe_1 and pe_2.

from cautious to aggressive, or in more relevant terms, from *novice* to *expert*.

This precondition space is represented by the event (oval) nodes shown in the inset of Figure 1. They are motor processor status event nodes, or *ms-nodes*. These nodes exist only between *mc-nodes* in command chains. They are sequentially used as preconditions to the *mc-node* they precede and, like *pe-nodes*, are chronologically ordered (ms_0 first; ms_4 last).

Since our example task uses keypress responses, the *ms-nodes* are: ms_0 = processor free; ms_1 = execution free; ms_2 = key touched; ms_3 = key depressed; ms_4 = key released. In the figure, ms_4 is used as the precondition to mr_1; a cautious performance style.

The Promotion-Learning Procedure

The observation that chronological promotions can produce performance improvement has inspired a promotion-based learning procedure. This procedure performs three styles of promotions: *prepare-promotions*, *pe-promotions*, and *ms-promotions*.[5]

The chronological task strategy structure is essential to the procedure because it keeps track of the chronology of events and also identifies which promotion styles can be performed.

To use this learning procedure, the user must provide a chronological task strategy structure and an initial perfor-

[5]The *prepare-promotion* style creates anticipatory motor programming rules while the *pe-promotion* and *ms-promotion* styles combine to produce pipelining rules.

mance model which performs the task as it is represented in the structure. The model is executed. When a motor command is generated, task-independent *promotion suggestion rules* may fire to suggest that a promotion be performed for the just-generated command.

The suggested promotion style is invoked and a promoted command rule is learned. This new rule produces the same motor command (except in the case of prepare-promotions), but it is preconditioned on a chronologically earlier event. The strategy structure is updated to reflect the promotion (which also causes rules to be learned). The new command rule is immediately available for use in task performance.

The learning procedure runs concurrently with task performance, so performance is not at all hindered by the application of the procedure.[6]

We now briefly describe the three promotion styles. There are essential guidelines for the application of each of the promotion styles; these can be found in the Appendix.

Prepare-Promotion

When a command that is preconditioned on perceptual event pe_t is executed, a prepare-promotion suggestion is generated if the guidelines are satisfied. The prepare-promotion style applies and a new *prepare* rule is learned. This rule is preconditioned on *pe-node* pe_{t-1} and produces a PREPARE <action> command where <action> is the command produced by the command on *pe-node* pe_t. The strategy structure is modified to reflect this promotion.

PE-Promotion

The final promotion style is used to promote motor commands to chronologically earlier perceptual events, hence this style is called *pe-promotion*.

When a *pe-promotable* motor command (see Appendix) that is preconditioned on *pe-node* pe_t is executed, a pe-promotion suggestion is generated if the guidelines are satisfied. The pe-promotion style applies resulting in a promoted command rule which produces the same command but is now preconditioned on *pe-node* pe_{t-1}. The strategy structure is modified to reflect this promotion.

MS-Promotion

Motor status promotions, or *ms-promotions*, use the motor status nodes to allow motor commands in a command chain to be preconditioned on chronologically earlier motor processor status events.

When a motor command that is preconditioned on *ms-node* ms_t is executed, an ms-promotion suggestion is generated if the guidelines are satisfied. The ms-promotion style applies resulting in a promoted command rule which produces the same command but is now preconditioned on *ms-node* ms_{t-1}. The strategy structure is modified to reflect this promotion.

[6]We concede that this behavior is not cognitively plausible; the procedure's learning rate is independent of the workload imposed by the task—typically, a high-workload task produces slower learning rates than low-workload task. We have temporarily set aside this consideration to test and develop the procedure's core ideas.

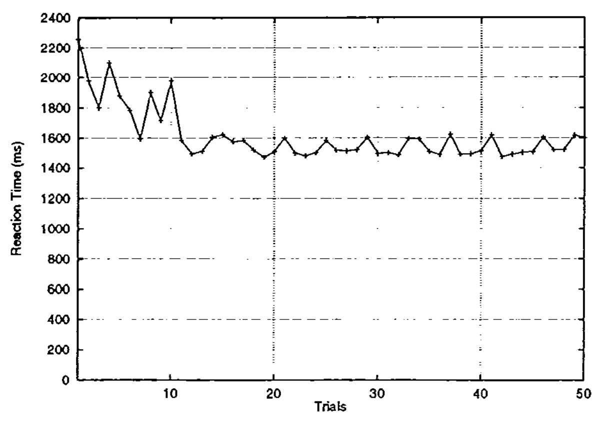

Figure 3: Raw RT data for a single run of 50 trials. Promotion learning was exhausted after 10 trials with the final RT being around 1500 +/- 50ms. This variability is small; +/-1 cognitive cycle.

Referring to the inset in Figure 1, the first time that the command represented by mr_1 is sent, an ms-promotion suggestion will be generated. The ms-promotion procedure will create a new command rule which is preconditioned on ms_3 (key depressed) instead of ms_4 (key released). The strategy structure will also be modified so that mr_1 is linked to ms_3.

When this newly learned rule fires at a later time, mr_1 will be ms-promoted to ms_2 (key touched). This continues with every application of the promoted commands until mr_1 is preconditioned by ms_0, at which point ms-promotions for mr_1 have been exhausted.

Promotion-Learning Applied To The Example Task

For our example task as shown in Figure 1, the initial performance model and the strategy structure were defined to initially give very novice-like performance; i.e. all chained *mc-nodes* and rules were configured to be preconditioned by ms_4. We ran the model for fifty trials with the learning procedure. Figure 2 shows the sequence of effects of promotion on the structure of the example task.

Prepare-promotions were first applied to both of the ocular commands. This resulted in the addition of two new *mc-nodes* po_0 and po_1. At this point, the system will prepare the motor system for the o_0 and o_1 commands on events pe_0 and pe_2, respectively.

One pe-promotion was then performed to promote the prepare *mc-node* po_1 from pe_2 to pe_1. The preparation of o_1 will now occur earlier at pe_1.

Finally, numerous ms-promotions were performed to gradually transition from cautious to aggressive behavior in the ocular and manual motor chains. This is depicted by the shortening of the arrows between the *mc-nodes*. When ms-promotions have been exhausted, all motor chain commands are preconditioned on ms_0.

Figure 3 shows the learning curve. There is a large learning effect; a difference of almost 700ms. On the first trial, the predicted reaction time is around 2250ms. After ten trials, it has leveled out to between 1500 and 1600ms.

By the end of the 50-trial run, Soar has built 75 chunks, of which thirty are motor command chunks. However, at the end of learning (after trial 10), only eight (8) of those motor chunks are applicable: one for po_0, one for po_1, and one for each keypress at position two or greater in the motor chains. The 22 other motor chunks are not applicable because when a motor command is promoted, the rule that initially produced the command is disabled by the new earlier-firing rule. Only the newest command rules are applicable.

The remaining 45 chunks are strategy modification chunks which were created as the structure was modified during learning. They capture the evolution of the strategy data structure.

We could now re-run the system, giving it the initial strategy structure as before along with these 75 new chunks. The structure modification chunks would first all fire, reproducing the order in which the strategy structure evolved during learning. After these firings, the structure would correspond to the behavior implicit in the eight applicable motor command chunks. As the task ran, these eight applicable motor command chunks would fire appropriately, and the task performance would be at the same level as shown in Figure 2 for trials >10.

The preceding discussion and Figure 2 have no doubt given the impression that promotion suggestions/styles are generated and/or applied in a predetermined order. This is not the case however. Figure 2 was made just to show an example of each promotion style. Promotion styles are actually executed serially in a "first-come first-served" fashion.

Discussion

The primary goal of this paper was to present a *task-independent* learning procedure which produces gradual performance improvement through experience on a perceptual-motor task. It utilizes a data structure that represents a strategy for performing the task. This learning procedure is a direct result of the constraints that occur when realistic perceptual and motor mechanisms (EPIC) are mated to a learning cognitive modeling architecture (Soar).

The inspiration for this learning procedure must be credited to the integration of realistic perceptual and motor processor (EPIC) with the cognitive architecture (Soar). The constraints provided by EPIC's motor processor clearly defined the *precondition space* over which a learning procedure could explore. This is just one of the many benefits of building computational models in systems that incorporate perception, cognition, and action.

Another possible benefit is that the set of promotions styles along with the *precondition space* suggests a possible sources of individual differences between subject's task performance. While this learning procedure relentlessly applied promotions styles at every possible opportunity, it is conceivable that subjects may not do the same. With the addition or removal of knowledge, it may be possible for this system to aid in explaining some individual differences.

In the Introduction, we mentioned three cognition-based sources of performance improvements. This work appears to support our suggestion that the promotion learning procedure may be a fourth source of performance improvement. By using a performance model that was the equivalent of what would be generated by the three sources, we applied the learning procedure and realized a large amount of *additional* performance improvement.

Future Work

The section that is conspicuous in its absence is one that compares human learning and performance results with that predicted by our model. We are presently searching for suitable visual-manual tasks that have existing data. However, this will permit us to validate only the final performance of the system since, typically, practice effects are specifically trained out prior to data collection. In the interim however, there is some evidence of the procedure's validity from its application in other work being done on dual-task performance improvement (Chong, 1998).

Presently, the learning procedure runs concurrently with task performance. There is no cost on task performance from the learning procedure, and no cost to the learning procedure due to task performance. As noted earlier, this is not cognitively plausible, and is now the focus of ongoing research.

Also, the strategy data structure is a declarative data structure stored and maintained in Soar's working memory. We expect to look at a more mixed declarative/procedural representation of this structure and the impact that has on the learning.

References

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

Baxter, G. D., Ritter, F. E., Jones, G., & Young, R. M. (1997). Exploiting User Interface Management Systems in Cognitive Modelling. Submitted report.

Byrne, M. D. & Anderson, J. R. (1997). Enhancing ACT-R's Perceptual-Motor Abilities. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum.

Chong, R. S. (1995). Perception, Cognition, and Action: One Small Step Towards Unification. Unpublished report.

Chong, R. S. (1998). Modeling Dual-Task Performance Improvement Using EPIC-Soar. Submitted report.

Chong, R. S. & Laird, J. E. (1997). Identifying Dual-Task Executive Process Knowledge Using EPIC-Soar. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum.

Kieras, D. E. (1994). An Introduction to the EPIC architecture for computational models of human performance, In *Proceedings of the Fourteenth Soar Workshop.*

Huffman, S. B. (1994). Instructable Autonomous Agents. Ph.D. thesis, The University of Michigan, Ann Arbor.

Lewis, R. L., Huffman, S. B., John, B. E., Laird, J. E, Lehman, J. F., Newell, A., Rosenbloom, P. S., Simon, T. & Tessler, S. G. (1990). Soar as a unified theory of cognition: Spring 1990. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum.

Lewis, R. L., Newell, A., & Polk, T. A. (1989). Toward a Soar theory of taking instructions for immediate reasoning tasks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

Meyer, D. E. & Kieras, D. E. (1997a). EPIC: A computational theory of executive cognitive processes and multiple-task performance: Part 1. *Psychological Review*, 104, 3-65.

Meyer, D. E. & Kieras, D. E. (1997b). EPIC: A computational theory of executive cognitive processes and multiple-task performance: Part 2. *Psychological Review*.

Miller, C. S. & Laird, J. E.(1996). Accounting for Graded Performance within a Discrete Search Framework. *Cognitive Science*, 20(4), 499-537.

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

Rosenbloom, P. S., Laird, J. E., & Newell, A. (1993). *The Soar Papers*. Cambridge, MA: The MIT Press.

Young, R. M. & Lewis, R. L. (1998). The Soar Cognitive Architecture and Human Working Memory. In Miyake, A. and Shah, P. (Eds), *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*. New York: Cambridge University Press.

Appendix

Here are the guidelines for the application of the three promotion styles.

Prepare-Promotion Guidelines

There are five guidelines for creating prepares; four will be presented here and the fifth will appear later.

First, prepare-promotion suggestions apply only to the first command in a motor chain; i.e. the command closest to the perceptual event node. Commands that appear later in the chain are prepared-for because it is faster to just send the command rather than to first sending a PREPARE <action> command, then immediately sending the actual command.

Second, prepare suggestions apply only to commands that are *preparable*. Commands are preparable only if it is known exactly what command is to be produced at a later time.

For example, the PUNCH motor commands in our two-choice reaction time task (Figure 2) are not preparable since the identity of the stimulus is needed before it can be known which motor command to prepare.

Another example of this would appear in modeling a continuous tracking task. Since the cursor is continuously moving, it is not possible to prepare a joystick move ahead of time since you need to know the current location of the cursor before the PLY command can be sent if you want accurate tracking performance.

The preparability of a command is indicated by a value of t for the ^preparable field in the motor command node in the data structure.

The third guideline, which is somewhat reasonable, is that a prepare cannot be made for an existing prepare.

The fourth guideline states that when the prepare command is moved to the previous perceptual event node, it must be added to the tail of the motor chain (if one exists) of the same modality type (ocular or manual).

PE-Promotion Guidelines

First, only *pe-promotable* commands can be pe-promoted. A command is pe-promotable only if it is not inextricably bound to the perceptual event to which it was initially attached. (The pe-promotability of a command is indicated by a value of t for the ^pe-promotable field of the motor command node in the data structure.)

For example, neither o_0 or o_1 in Figure 1 are pe-promotable because the task instructions specifically state that the responses should following the appearance of the stimulus, and it is implied that looking back at the stimulus should occur after the response is complete. However, the *prepares* nodes created by the prepare-promotion style are pe-promotable since they are not part of the task instructions. Note that o_1 could be made pe-promotable if the strategy structure represented an *interpretation* of the task instructions that did not require that it be sent only after a response was made.

Second, as with prepare-promotions, only the first command in a motor chain can be promoted. Of course, when the first command is promoted, the command that was second is now first and will be eligible for pe-promotion if it is pe-promotable.

The third guideline is similar to the fourth guideline for prepare-promotions. When a command is moved to the previous perceptual event node, the command must be added to the tail of the motor chain (if one exists) of the same modality type (ocular or manual). This guideline is demonstrated in Figure 2b. The first promotion is a prepare-promotion. Since there is no ocular motor command on the trial-start or stimulus-features events, po_0 and po_1 become the first/only node in those ocular motor chains. However, with the pe-promotion of po_1, there is an existing ocular motor chain (the single command, o_0) therefore, po_1 must be chained after o_0.

Prepare-Promotion Guidelines, continued

Having introduced the pe-promotions style we can now present the fifth and final prepare-promotion guideline promised earlier: prepare-promotions cannot be applied to commands that are pe-promotable.

The reasoning is as follows. Say *x* is a command that is pe-promotable. The *best* improvement possible for *x* comes by moving the whole command ahead by pe-promotion. pe-promotions of *x* will continue until a) all earlier perceptual event nodes are exhausted (i.e., *x* is hanging off trial-start), in which case a *prepare* could not possibly be made since there is no earlier *pe-node*; or b) *x* has been pe-promoted to the tail of a motor chain, and in this case, the first guideline for prepare-promotions prohibits creating prepares for all but the first command in a motor chain.

MS-Promotion Guidelines

There are presently no guidelines for restricting the application of ms-promotions.