THE CHUNKING OF SKILL AND KNOWLEDGE

Techinical Report AIP-7

Paul S. Rosenbloom, John E. Laird and Allen Newell

Stanford University
University of Michigan
Carnegie-Mellon University

The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

4

THE CHUNKING OF SKILL AND KNOWLEDGE

Techinical Report AIP-7

Paul S. Rosenbloom, John E. Laird and Allen Newell

Stanford University
University of Michigan
Carnegie-Mellon University

29 September 1987

This research was supported by the Computer Sciences Division, Office of Naval Research and DARPA under Contract Number N00014-86-K-0678; Defense Advanced Research Projects Agency (DOD) under Contract N00039-86-C-0133 and by the Sloan Foundation. Computer facilities were partially provided by NIH grant RR-00785 to Sumex-Aim. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the U.S. Government, the Sloan Foundation, or the National Institutes of Health. Reproduction in whole or in part is permitted for purposes of the United States Government. Approved for public release; distribution unlimitied.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AIP - 7	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Carnegie-Mellon University; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Computer Sciences Division, Office of Naval Research (Code 1133)
6c. ADDRESS (City, State, and ZIP Code): Department of Psychology, Pittsburgh, Pennsylvania 15213	7b. ADDRESS (City, State, and ZIP Code): 800 N. Quincy Street, Arlington, Virginia 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: Same as Monitoring Organization; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS p400005ub201/7-4-86

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO
N/A	N/A	N/A	N/A

11. TITLE (Include Security Classification): The Chunking of Skill and Knowledge

12. PERSONAL AUTHOR(S): P.S. Rosenbloom, J.E. Laird and A. Newell

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	87 September 29	18

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): Artificial Intelligence, Machine Learning, Cognitive Architecture. (SDW)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This article describes recent work that utilizes the concept of chunking as the basis for an integrated model of the acquisition of both skill and knowledge. We look at results in the areas of practice (skill) and verbal learning (knowledge). The approach is based on viewing task performance as a problem solving process and chunking as a learning process that stores away information about the results of problem solving. In practice tasks, chunks acquired during the solution of one problem can be used during later problems to speed up the system's performance. This chunking process produces the same type of power-law practice curves that appear so ubiquitously in human practice. In verbal learning tasks, chunks acquired during training are used at test time to determine how to respond. This psychological model is a manifestation of a set of processes that provide the basis of a general architecture. Such an architecture is not only interesting in its own right, but provides support for the more narrowly based psychological phenomena. Keywords:

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Dr. Alan L. Meyrowitz	(202) 696-4302	N00014

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

To appear in the Proceed. of
the "Working Models of Human
Perception" workshop, 1987,
Einhoven, The Netherlands

The Chunking of Skill and Knowledge[1]

Paul S. Rosenbloom,[2] John E. Laird,[3] & Allen Newell[4]

August 18, 1987

Abstract

This article describes recent work that utilizes the concept of chunking as the basis for an integrated model of the acquisition of both skill and knowledge. We look at results in the areas of practice (skill) and verbal learning (knowledge). The approach is based on viewing task performance as a problem solving process and chunking as a learning process that stores away information about the results of problem solving. In practice tasks, chunks acquired during the solution of one problem can be used during later problems to speed up the system's performance. This chunking process produces the same type of power-law practice curves that appear so ubiquitously in human practice. In verbal learning tasks, chunks acquired during training are used at test time to determine how to respond. This psychological model is a manifestation of a set of processes that provide the basis of a general architecture. Such an architecture is not only interesting in its own right, but provides support for the more narrowly based psychological phenomena.

The concept of chunking has played a major theoretical role in cognitive psychology ever since Miller's classic paper (Miller, 1956). Through much of this history it has been used primarily in models of memory organization. According to this view, chunking is a process of creating symbols (chunks) which represent the combination of several other symbols. In this long and productive tradition, chunking has been used to model a wide variety of memory phenomena (Miller, 1956; DeGroot, 1965; Bower & Winzenz, 1969; Johnson, 1972; Chase & Simon, 1973; Chase & Ericsson, 1981).

In recent years, chunking has also been proposed as the basis for a model of human

---

[1]This research was sponsored by the Defense Advanced Research Projects Agency (DOD) under contract N00039-86-C-0133 and by the Sloan Foundation. Computer facilities were partially provided by NIH grant RR-00785 to Sumex-Aim. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the US Government, the Sloan Foundation, or the National Institutes of Health.

[2]Knowledge Systems Laboratory, Departments of Computer Science and Psychology, Stanford University, 701 Welch Road (Bldg. C), Palo Alto, CA 94304 (After September 1987, University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292)

[3]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109

[4]Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213

practice (Newell & Rosenbloom, 1981; Rosenbloom, 1983; Rosenbloom & Newell, 1987a). According to this view, chunking is a process that acquires new productions (chunks) by the caching of goal-based performance. This version of chunking was successfully able to model the ubiquitous power law shape of human practice curves. In building on these results, chunking has been reimplemented as a learning mechanism within an artificial intelligence problem solver called Soar (Laird, 1983; Laird, 1986; Laird, Newell, & Rosenbloom, 1987). This combination of Soar and chunking has provided a learning architecture with the flexibility and power needed to demonstrate learning behaviors that go considerably beyond simple practice effects (Steier et al, 1987). The results have been good enough for chunking to be proposed as a general learning mechanism for artificially-intelligent systems (Laird, Rosenbloom, & Newell, 1984; Laird, Rosenbloom, & Newell, 1986).

The generality of the combination of chunking and Soar as an AI learning system raises the possibility that this combination can be transferred back into psychology to provide the basis for a general model of human learning. In this article we report on a preliminary examination of this possibility that involves the application of Soar to two major domains of human learning: skill acquisition and knowledge acquisition. Within skill acquisition we return to the subdomain of practice, asking whether Soar can replicate the earlier results by producing power law practice curves. Within knowledge acquisition we look within the subdomain of verbal learning at simple recognition, recall, and cued recall tasks. In this preliminary investigation we are not yet looking for close matches to experimental data. Instead, we focus on the nature of the processing that allows these tasks to be performed by chunking.

In the next two sections we give brief descriptions of Soar and chunking. This background material is followed by sections on skill acquisition and knowledge acquisition in Soar, and then by a section containing concluding remarks.

1. Soar

Soar is based on formulating all symbolic goal-oriented processing as search in problem spaces (Newell, 1980). The problem space determines the set of states and operators that can be used during the processing to attain a goal. The states represent situations. There is an initial state, representing the initial situation, and a set of desired states that represent the goal. An operator, when applied to a state in the problem space, yields another state in the problem space. The goal is achieved when a desired state is reached as the result of a sequence of operator applications starting from the initial state.

Each goal defines a problem solving context ("context" for short). A context is a data structure in Soar's working memory – a short-term declarative memory – that contains, in addition to a goal, roles for a problem space, a state, and an operator. Problem solving for a goal is driven by the acts of selecting problem spaces, states, and operators for the appropriate roles in the context. Each deliberate act of the Soar architecture – a selection of a problem space, a state or an operator – is accomplished via a two-phase decision cycle. First, during the elaboration phase, the description of the current situation (that is, the contents of working memory) is elaborated with relevant information from Soar's production memory – a long-term procedural memory. The elaboration phase proceeds in a sequence of synchronous cycles. During each cycle of the elaboration phase, all of the productions in the production memory are matched against working memory, and then all of the resulting production instantiations are executed. The net effect of these production firings is to add information to the working memory. New objects are created, new knowledge is added about existing objects, and preferences are generated.

There is a fixed language of preferences that is used to describe the acceptability and desirability of the alternatives being considered for selection. By using different preferences, it is possible to assert that a particular problem space, state, or operator is acceptable (should be considered for selection), rejected (should not be considered for selection), better than another alternative, and so on. When the elaboration phase reaches quiescence – that is, no more productions can fire – the second phase of the decision cycle, the decision procedure, is entered. The decision procedure is a fixed body of code that interprets the preferences in working memory according to their fixed semantics. If the preferences uniquely specify an object to be selected for a role in a context, then a decision can be made, and the specified object becomes the current value of the role. The decision cycle then repeats, starting with another elaboration phase.

If, when the elaboration phase reaches quiescence, the preferences in working memory are either incomplete or inconsistent, an impasse occurs in problem solving because the system does not know how to proceed. When an impasse occurs, a subgoal with an associated problem solving context is automatically generated for the task of resolving the impasse. The impasses, and thus their subgoals, vary from problems of selection (of problem spaces, states, and operators) to problems of generation (e.g., operator application). Given a subgoal, Soar can bring its full problem solving capability and knowledge to bear on resolving the impasse that caused the subgoal. When impasses occur within impasses, then subgoals occur within subgoals, and a goal hierarchy results (which therefore defines a hierarchy of contexts). The top goal in the hierarchy is a

task goal; such as, to recognize an item. The subgoals below it are all generated as the result of impasses in problem solving. A subgoal terminates when its impasse (or some higher impasse) is resolved.

2. Chunking

The traditional view of chunks is that they are symbols representing the combination of several other symbols (Miller, 1956). Newell and Rosenbloom (1981) turned this concept into a model of practice by describing how performance could be improved by the acquisition of chunks that represent patterns of objects in the task environment. This model was instantiated in a production system architecture, first in a domain-specific form (Rosenbloom & Newell, 1987b), and then in a domain-independent form (Rosenbloom, 1983; Rosenbloom & Newell, 1987a). A modified version of the domain-independent chunking mechanism was then implemented as part of the Soar architecture (Laird, Rosenbloom, & Newell, 1986).

In its Soar implementation, chunking is a learning mechanism that acquires new productions which summarize the processing that leads to results of subgoals. The actions of the new productions are based on the results of the subgoal. The conditions are based on those aspects of the pre-goal situation that were relevant to the determination of the results. Relevance is determined by using the traces of the productions that fired during the subgoal. Starting from the production trace that generated the subgoal's result, those production traces that generated the working-memory elements in the conditions of the trace are found, and then the traces that generated their condition elements are found, and so on until elements are reached that existed prior to the subgoal. Productions that only generate preferences do not participate in this backtracing process -- preferences only affect the efficiency with which a goal is achieved, and not the correctness of the goal's results.

An example of this chunking process is shown schematically in Figure 2-1. The circled letters are objects in working memory. The two striped vertical bars mark the beginning and ending of the subgoal. The objects to the left of the first bar (A, B, C, D, E, and F) exist prior to the creation of the subgoal. The objects between the two bars (G, H, and I) are internal to the subgoal. The objects to the right of the second bar (J) are results of the subgoal. P1, P2, P3, and P4 are production traces; for example, production trace P1 records the fact that a production fired which examined objects A and B and generated object G. The highlighted production traces are those that are involved in the backtracing process.

Chunking in this figure begins by making the result object (J) the basis for the

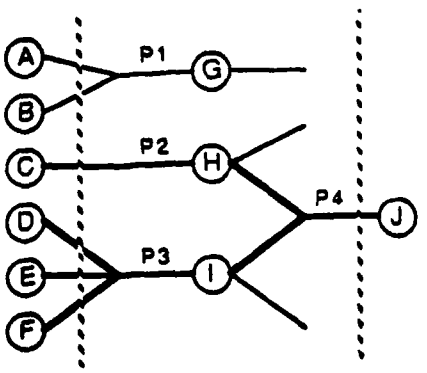

Figure 2-1: Schematic view of the chunking process in Soar.

action of the chunk. The condition-finding process then begins with object J, and determines which production trace produced it – trace P4. It then determines that the conditions of trace P4 (objects H and I) are generated by traces P2 and P3, respectively. The condition elements of traces P2 and P3 (objects C, D, E, and F) existed prior to the subgoal, so they form the basis for the conditions of the chunk. The resulting chunk is

$$C \wedge D \wedge E \wedge F \longrightarrow J. \qquad (1)$$

Once a chunk has been learned, the new production will fire during the elaboration phase in relevantly similar situations in the future, directly producing the required information. No impasse will occur, and problem solving will proceed smoothly. Chunking is thus a form of goal-based caching which avoids redundant future effort by directly producing a result that once required problem solving to determine. It bears a family resemblance to other learning mechanisms which move the system along the store-versus-compute trade-off, such as production composition (Lewis, 1978; Neves & Anderson, 1981; Anderson, 1982; Anderson, 1983), memo functions (Michie, 1968), macro-operators (Fikes, Hart, & Nilsson, 1972; Korf, 1985), and explanation-based generalization (Mitchell, Keller, & Kedar-Cabelli, 1986).

In the pre-Soar implementation of chunking, chunks were learned bottom-up in the goal hierarchy, reflecting the accepted view of human chunking. On any trial, only the terminal subgoals – the subgoals that did not themselves have subgoals – were chunked. Higher-level subgoals could be chunked only after all of their subgoals had been chunked. This bottom-up approach was critical in producing power law practice curves. However, chunking has been implemented in Soar with a settable option. Chunking can proceed for only the terminal subgoals (bottom-up chunking) or for all of the goals in the hierarchy (all-goals chunking). Given a sufficient number of trials, the results of the two approaches should be essentially indistinguishable. However, because all-goals chunking yields faster learning, most research on learning in Soar has employed

it. In this article, all results except for those specifically intended to model practice curves were generated with all-goals chunking. Practice curves were generated with bottom-up chunking.

3. Skill Acquisition

There is a ubiquitous regularity in human practice — *the power law of practice* — that states that the time to perform a task (T) decreases as a power-law function of the number of times the task has been performed (that is, the number of trials, N):

$$T = BN^{-\alpha}. \tag{2}$$

While the power law of practice was originally recognized in the domain of motor skills (Snoddy, 1926), it has recently become clear that it holds over a much wider range of human tasks — possibly extending to the full range of human performance (Newell & Rosenbloom, 1981).

The driving force behind the initial development of chunking as a learning mechanism was a desire to develop a model of practice that would produce power law practice curves. The experimental task used during these early investigations was a 1023-choice reaction-time task (Seibel, 1963). This is a perceptual-motor task in which there is a stimulus display of ten lights, arranged horizontally, and a response apparatus of ten buttons, arranged horizontally in such a way that each finger rests on one of them. The stimulus and response environments are set up so that there is a highly compatible one-one correspondence between the lights and buttons, each light directly above a button. On each trial of the experiment, some of the lights are on and some are off. The subject's task is to respond as quickly as possible by pressing the buttons corresponding to the lights that are on.

The performance algorithm used in modeling this task is based on a top-down divide-and-conquer algorithm. The top goal is to process a horizontal region of the display, containing a pattern of on and off lights. This goal is recursively decomposed into subgoals to process smaller and smaller regions until patterns are reached which can be directly converted into corresponding patterns of button presses. Chunking works in a bottom-up fashion in this goal hierarchy. Productions are first learned for the bottom-most level of subgoals. These chunks relate small patterns of lights to their corresponding patterns of button presses. In later trials, these chunks can be used in place of problem solving in the low-level subgoals, speeding up task performance. This also allows chunking to proceed up the goal hierarchy, acquiring chunks which relate increasingly larger patterns of lights to patterns of button presses.

A simple analysis of this learning process suggests exponential speed-ups with

practice. Each time a task is practiced, a new and higher level of chunks can be acquired, reducing the time to perform the task by a constant factor (the size of the chunks). However, learning slows down because the larger chunks that are learned later in practice are encountered less often – for example, in the Seibel task, a chunk that contains three lights is encountered half as often as a chunk that contains two lights – and therefore contribute less to the overall speedup than do the smaller chunks which are encountered more often. Learning (chunk acquisition) actually continues at a constant pace, but what is learned becomes progressively less helpful. The net result is a slow down of the learning process to the point where it becomes more like a power law than an exponential. Approximate mathematical analyses of this process can be found in Newell & Rosenbloom (1981) and Rosenbloom (1983).

In addition to the mathematical analyses, the pre-Soar implementation of chunking was evaluated by generating and analyzing simulated practice curves (Rosenbloom, 1983; Rosenbloom & Newell, 1987a). For the Seibel task, the simulated practice curves were shown to provide close approximations to power laws. To determine whether these results continue to hold for the implementation of chunking in Soar, a version of Seibel's task has been implemented in Soar. The divide-and-conquer algorithm is implemented by having a problem space with a single operator – process-a-region-of-lights – which can be instantiated for different regions. In the top goal, two instances of this operator are generated, one for each half of the display. When one of these operators is selected, an impasse occurs because there is no knowledge directly available in productions about how to process regions that contain more than one light. In the subgoal for this impasse, two more instances of the operator are generated, this time for the two halves of the region that the higher-level operator was processing. This recursion continues until operator instances are generated for regions that contain only a single light. Such operators can be processed directly by productions.

With practice on this task, chunks are learned which summarize the processing that goes on in the subgoals. These chunks directly implement particular instances of the process-a-region-of-lights operator, avoiding the need for problem solving in a subgoal. Figure 3-1 shows a log-log plot – power laws plot as straight lines in log-log plots – of 125 trials of the Seibel task, as performed by Soar. In this figure, the data has been aggregated at a rate of five trials per point. Simulated time is computed by counting the number of decisions required to perform the task. The best-fit power law curve for this data is:

$$T = 110N^{-0.80}. \tag{3}$$

In addition to the Seibel task, well over 20 other tasks have been implemented in

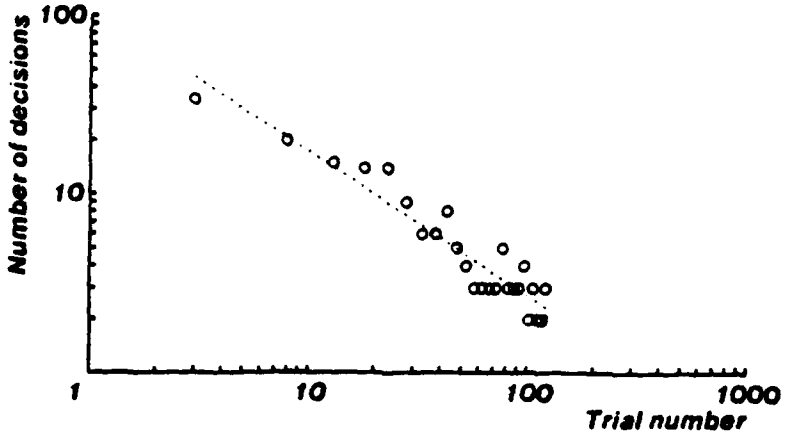

Figure 3-1: Simulated practice curve for a 1023-choice RT task.

Soar. Practice curves have not been generated for most of these tasks, but one does exist for the task of computer configuration. This is a difficult task that involves configuring a working computer by assembling a list of components that have been ordered by a customer. About 25% of the R1/XCON computer-configuration expert system (McDermott, 1982) has been reimplemented in Soar by designing and implementing a set of problem spaces in which the computer components can be assembled (Rosenbloom, Laird, McDermott, Newell, & Orciuch, 1985; van de Brug, Rosenbloom, & Newell, 1987). The final system had 9 task problem spaces with a total of 34 operators.

One interesting phenomenon that shows up in this task (as well as in others) is that chunks can transfer within a single trial (improving performance the first time a problem is solved), across trials (to later instances of the same problem), and across tasks (to other configuration problems). As long as two subproblems are relevantly similar, chunks can transfer between them. Figure 3-2 shows a thirty trial practice curve generated by Soar for this task.[5] The data presented in this figure is unaggregated (accounting for the apparent increase in variance). The thirty trial sequence consists of two complete passes through a sequence of 15 orders. The best-fit power law curve for this data is:

$$T = 111N^{-0.21}. \quad (4)$$

The chunks that Soar learns while performing these two tasks fall into two categories. The first category of chunks are for operator implementation. In the Seibel

[5]Interestingly, this curve was produced by plotting existing data generated for a different purpose. This was not a deliberate experiment to produce power law practice curves.

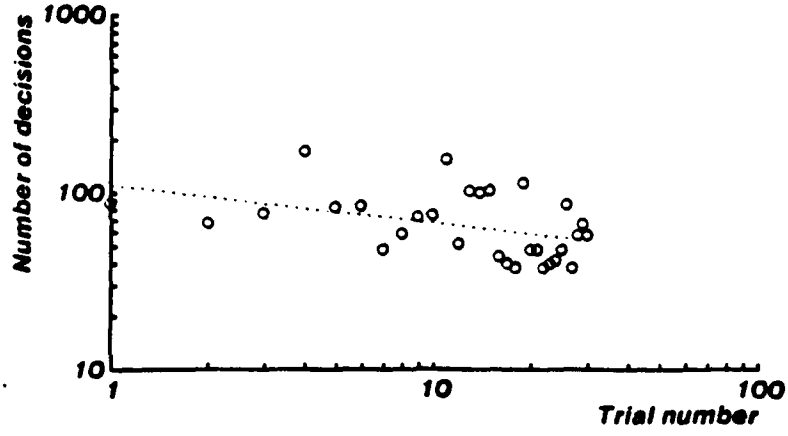

Figure 3-2: Simulated practice curve for a computer configuration task.

task, chunks are learned which directly relate patterns of lights to patterns of button presses. These chunks act as implementation productions for the process-a-region-of-lights operator. In the computer configuration domain there is a single top-level problem space for the task. All of the other task problem spaces are used as subspaces for the implementation of the operators in this space, or at lower levels to implement operators of subspaces. Implementation chunks are acquired for all of these complex operators.

The second category of chunks are for selection – usually of operators, but also potentially of states and problem spaces. When it is unclear which among several operators should be selected for a state in a problem space, an impasse and subgoal are generated. The results of problem solving in the subgoal should be preferences that lead to the selection of one of the alternatives. Chunking this type of subgoal leads to the acquisition of search control productions; that is, productions which generate preferences that can be used to control the problem space search. This category of chunks does not appear in the relatively simple implementation of the Seibel task, but does appear in the computer configuration task. In other tasks, such as Tic-Tac-Toe, for which we do not yet have practice curves, search control chunks are the dominating factor in learning.

4. Knowledge Acquisition

In the previous section it was demonstrated how the acquisition of chunks could lead to improvements in performance. Procedural chunks were learned that performed operator implementation, and control chunks were learned which helped in selection. However, chunking originated as a model of declarative memory, and only later was it converted into a model of skill learning. Given this conversion, the question naturally

arises as to whether it can, in its current form, be used to acquire declarative chunks which represent new knowledge. For example, can chunking support the types of learning that are required in simple verbal learning experiments? There have been good reasons for doubting that this form of *data chunking* is possible. Chunking, as a skill acquisition mechanism, improves performance by creating productions which cache the effects of subgoal-based problem solving. The new productions thus summarize processing that the system can already perform. They do not, at least in any obvious fashion, lead to qualitatively new behaviors or to the acquisition of new knowledge that is not already known to the problem solver. Nonetheless, as was demonstrated in Rosenbloom, Laird, & Newell (1987), chunking can be used as the basis for a data chunking capability in Soar.

Consider one of the simplest of the verbal learning tasks, a recognition task. Let's assume that the objects to be learned are strings of letters, such as "ab", and that the performance task is to recognize whether a string has ever been seen before.[6] There are two types of trials: training and performance. On each training trial the system must perceive a new string, and then store into its long-term memory a representation of the string that will enable it to perform an old-versus-new judgement on a performance trial. Thus, the first step in Soar's learning to recognize a new string is for it to use its perceptual capabilities to generate a representation of the string in its working memory.[7] At this point, the new string is available for use by the system, but it has not yet been learned – working memory is only a temporary memory which holds the current data upon which the system is working. The learning act occurs when a production is created which can, at appropriate points in the future, recognize the string. If Soar is to use its chunking mechanism to do this, it must take advantage of the fact that chunking learns from goal-based experience. The key is for it to set up the right internal tasks so that its problem solving experience in subgoals leads to the creation of chunks that represent the new knowledge.

To learn to recognize a new string, an internal task is set up in which, in a subgoal, the system first examines each of the letters out of which the string is composed, and then generates a name for the string. The name is an internally generated symbol for the new string; for example, G3297. The name becomes the result of the subgoal, and

---

[6] The system described in Rosenbloom, Laird, & Newell (1987) actually learns about hierarchically structured objects that are grounded in a set of primitive objects that represent letters. However, in this article only one-level strings of letters are discussed.

[7] Soar does not yet have an appropriate I/O interface, so in the current implementation this perceptual phase is performed by special purpose Lisp code.

thus forms the basis for the action of a chunk. The conditions of the chunk explicitly test for the contents of the string. For example, when the system learned to recognize the string "ab", the following production was acquired that augments (/) the string with its name.[8]

```
"ab" --> /G3297                                    (5)
```

On a performance trial, a string is presented and an old-versus-new judgement must be made. This judgement is made by determining whether a name for the string has been retrieved from long-term memory. There is one tricky aspect to this judgement. Suppose a name does appear for the string in working memory. Was the name retrieved from long-term memory by the execution of a recognition chunk, or was it just now invented? An "old" response should only be produced for a string if a recognition chunk has been learned for it on a previous training trial. The key to making this discrimination lies in realizing that the data generated by a production can be used in more than one way. The most straightforward way to use a recognition chunk is as a means of speeding up the generation of a name for a string — it allows a name to be generated during the elaboration phase, rather than via problem solving in a subgoal. However, the task at hand is to determine whether the string has been seen before, not to generate a name for it.

To actually use the chunk for the recognition task it needs to be treated as episodic knowledge representing the fact that the string has been perceived. This involves a small inductive leap in assuming that chunks are only created for strings that are perceived — it can be wrong if the system learns about a string that it has imagined rather than perceived. To use the recognition chunk as episodic knowledge in Soar, responses on performance trials are based on the contents of working memory after the completion of one elaboration phase. If a recognition chunk has been learned for the string, its name will be retrieved during elaboration. If, on the other hand, quiescence is reached without a name appearing, then the string is not recognized. This recognition task is described in detail in Rosenbloom, Laird, & Newell (1987).

Also described in Rosenbloom, Laird, & Newell (1987) is a simple recall task which involves the memorization of a set of strings, which are later to be generated on demand. From the point of view of the internal task, it is the dual of the recognition task. Instead of incorporating information about a new string into the conditions of a production, the information must be incorporated into the actions. As with recognition, there are training and performance trials. On each training trial the system is

[8] This is a simplified representation of the actual production.

presented with a new string, and it must learn to generate it on demand. On a performance trial, the system receives a recall request, and must respond by producing the strings that it learned to generate on the training trials.

To accomplish this, on training trials chunks need to be acquired that can generate the presented strings when the demand arises. The appropriate internal task for this problem would appear to be simply to copy a presented string in a subgoal. The chunk that is learned from this experience has actions which generate a string that is a copy of the presented string. The problem with this simple solution is that, if the copy is based on an examination of the presented string, then the conditions of the chunk will test for the existence of the presented string before generating the copy, thus allowing the string to be recalled in only those circumstances where it is already available. The solution to this problem that we have discovered is to split recall learning into separate generate and test phases. Generation is performed in a problem space that contains operators which generate (and combine) a primitive set of known letters. The result of the generation process is a new string constructed from known objects, rather than a copy of the input string. As with other generate-and-test models of recall, a recognition capability – a recognition chunk – is used as the basis for the test (see, for example, Watkins & Gardiner, 1979), but in contrast with other models, generate-and-test is being used here during training trials rather than performance trials.

The entire training trial consists of three steps: (1) learn to recognize the presented string, (2) in a subgoal, generate a new string by selecting and executing a sequence of operators that build up a string one letter at a time (without examining the presented string in the process), (3) if the generated string is recognized, return it as a result of the subgoal and learn a chunk which will generate the string. Constructing the new string from scratch ensures that the chunk will not test the presented string. However, it does introduce an additional problem of how to control the generation process so that the to-be-learned string will be generated rather than any of the other infinite possibilities. The solution to this problem is to use the presented string as search-control knowledge during the process of generating the new string. As described in Section 2, search control knowledge (productions which generate preferences) does not enter into the chunking process because it only affects the efficiency with which a problem is solved, and not its correctness. The goal test – that is, the recognition chunk – determines correctness. In consequence, the generation process can proceed efficiently, but the chunk created for it will not depend on the presented object.

The following production is a typical recall chunk. It generates a string and its name if there is not already a string with that name in working memory. The *'s denote

wildcards which can match any letter.

¬"**"/G3297 --> "ab"/G3297 (6)

On a performance trial, all of the available recall chunks execute, retrieving into working memory representations of all of the strings that the system has so far learned to recall. To satisfy the recall request, all of these retrieved objects must then be produced.

One peculiarity of this recall task is that to recall a single object (or a subset of the learned objects), all known objects must be retrieved from long-term memory into working memory. To solve this problem, and to lead up to more complex verbal learning tasks, such as paired-associate learning, a form of cued recall has been implemented. In cued recall there are a set of cues associated with each string to be learned. A string can only be retrieved from long-term memory if its cues are present in working memory. When a new string is to be learned, its features may provide cues which lead to the retrieval of old strings from long-term memory. The cues acquired for the new string are determined by finding the features which will discriminate it from these old strings.

The procedure followed during a cued recall training trial is similar to that used for the simple recall task. The principal difference is that during the generation phase, search control is used to suggest both the letters used in the presented string, and the letters used in any of the old strings that are retrieved but not rejected. An impasse will get generated whenever the generation process reaches a position in the string for which more than one letter has been suggested. To resolve this impasse, the presented string is first examined to determine which letter it contains. Then, based on this examination, the conflicting letter and string are rejected, and the letter from the presented string is selected. Below are some of the chunks that are acquired as the system learns to recall the sequence of strings "ab", "ba", "aa", "bb", "ca", "cb", "cc", "bc", and "ac" (not shown are the recognition chunks that are also acquired and the chunks which reject the incorrect letters). When more than one chunk is learned on the same trial, they are shown on the same line, separated by a semi-colon. Single quotes denote a pattern which is to match the input string, while double quotes denote a pattern that is to match a retrieved string.

¬"**"/G3297 --> "ab"/G3297 (7)

'b*' ∧ "a*" ∧ ¬"**"/G3298 --> "ba"/G3298; 'b*' ∧ "a*" --> Reject("a*") (8)

'*a' ∧ "*b" ∧ ¬"**"/G3299 --> "aa"/G3299; '*a' ∧ "*b" --> Reject("*b") (9)

'*b' ∧ "*a" ∧ ¬"**"/G3300 --> "bb"/G3300; '*b' ∧ "*a" --> Reject("*a") (10)

'c*' ∧ "a*" ∧ ¬"* "/G3301 --> "ca"/G3301; 'c*' ∧ "a*" --> Reject("a*") (11)

'c*' ∧ "b*" ∧ ¬"** /G3302 --> "cb"/G3302; 'c*' ∧ "b*" --> Reject("b*") (12)

'*c' ∧ "*a" ∧ ¬"**"/G3303 --> "cc"/G3303; '*c' ∧ "*a" --> Reject("*a") (13)

'b*' ∧ "c*" ∧ ¬"**"/G3304 --> "bc"/G3304; 'b*' ∧ "c*" --> Reject("c*") (14)

'*c' ∧ "*b" ∧ ¬"**"/G3305 --> "ac"/G3305; '*c' ∧ "*b" --> Reject("*b") (15)

On the first learning trial, the string "ab" is presented, and production 7 is learned. It looks just like the corresponding simple recall chunk, production 6, because there are no previously learned strings from which it must be discriminated. On the second learning trial, the string "ba" is presented, resulting in the retrieval of string "ab" (because production 7 retrieves "ab" for every string). The two strings are then discriminated based on their first letters, resulting in the creation of the two productions on line 8. The first production generates "ba" if: (1) the first letter of the input string is "b", (2) a string has been retrieved that has "a" as its first letter, and (3) no string named G3298 has already been retrieved into working memory. The second production rejects any string beginning with "a" if the first letter of the input string is "b". Consider what happens when the cue "b*" (or "ba", for that matter) is presented on a training trial after these two strings have been learned. First production 7 fires, retrieving "ab". Then the productions on line 8 fire in parallel, retrieving "ba" and rejecting "ab". The string "ba" is then recalled because it is the only non-rejected string that has been retrieved.

As more strings are learned, multiple cycles of retrieval and rejection often occur before a desired string is recalled. The following lines show the sequences of retrievals that occur, as a function of input cue, after all of the strings have been learned. Sets of strings are bracketed when they are retrieved in parallel.

```
Input("**") ∨ Input("a*") ∨ Input("*b") ∨ Input("ab"): "ab"
Input("b*"): "ab", "ba"
Input("*a"): "ab", "aa"
Input("c*"): "ab", "ca"
Input("ba"): "ab", {"ba" "aa"}
Input("aa"): "ab", "aa"
Input("bb"): "ab", "ba", "bb"
Input("ca"): "ab", {"aa" "ca"}
Input("cb"): "ab", "ca", "bb", "cb"
Input("cc"): "ab", {"ca" "ac"}, "cc"
Input("bc"): "ab", {"ba" "ac"}, "cc", "bc"
Input("*c") ∨ Input("ac"): "ab", "ac"
```

Another way to look at the set of productions that have been learned in this cued recall situation is as the implementation of a discrimination network, as in EPAM (Feigenbaum & Simon, 1984). In EPAM, the discrimination network was a tree in which objects were stored at the leaf nodes, and tests were stored at the internal nodes. Given a set of features, tests would be performed and branches would be followed until a leaf node was reached. In the Soar version of cued recall, every node in the network contains an object (a string), and each pair of productions performs a test and branches to a new node (rejecting the old node in the process). Discrimination stops when there

are no further branches to follow – essentially, a node acts like a leaf when it has been retrieved but not rejected. In this discrimination network, multiple tests can be pursued in parallel on one or more features, and multiple branches can be followed in parallel. However, for each string, there is only one path through the network that results in that string appearing as a leaf.

This network can potentially support at least three distinct types of behavior (only the first has so far been demonstrated). First, given a set of letter features, it can be used to retrieve the string which is the "best" match to those features. The evaluation of what match is "best" is not based on some ideal notion of a closest match; rather, it is based on the structure of the network, which has been built up in response to the need to make the necessary discriminations. For example, given "xc" as input, the network retrieves "ac" as the best match. Second, if the recognition chunk fails – because, for example, some of the input string's features are missing or modified – an old-versus-new judgement could be made by comparing the input string with the string that it causes to be retrieved from the network. Third, strings could be recalled in a free recall task by repeatedly prompting the network with potential cues.

5. Conclusions

In this article we have taken a step towards an integrated model of human learning that is based on the concept of chunking. All learning occurs via the acquisition of chunks (productions) that summarize goal-based problem solving experiences. Chunking has been shown to produce forms of both skill acquisition and knowledge acquisition.

In skill acquisition, chunks speed up task performance by directly implementing operators (procedural knowledge) and by controlling problem space search (control knowledge). Chunks can transfer within a trial, across trials, and across tasks. For both a simple perceptual-motor task and a complex cognitive task, the practice curves do appear to be power law in form. However, we have not yet performed a careful comparison between these curves and alternative functional forms. Future work along these lines should include a more careful look at practice curves from a wider variety of tasks, particularly tasks in which most of the chunks are learned for search control rather than operator implementation, and an expansion of the coverage of the model to other aspects of skill acquisition.

In knowledge acquisition, chunking can be used to support verbal learning. Procedural knowledge that is learned through experience is used to answer episodic questions about what the system has perceived. The result is successful performance in

simple recognition and recall tasks. The most complex form of verbal learning presented, cued recall, involves discrimination, generation, and test (recognition) processes. Future work along these lines should include extending the model to deal with more complex forms of verbal learning, such as paired-associate learning, and with other domains, such as semantic memory. Also required is a more detailed comparison of the model to human experimental data. Until this is done, these results must be considered tentative. As such comparisons are made, and as the scope of the model is extended, the model will undoubtedly need to be refined in a variety of ways.

In addition to the fact that the model has qualitatively reasonable properties in the domains of practice and verbal learning, it is important to note that it is embedded in a total architecture that is capable of a wide variety of cognitive activities. For instance, the chunking mechanism which does these two types of learning also does other types of learning. This larger context provides additional support for the model from outside of the experimental domains explicitly covered here.

References

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

Anderson, J. R. Knowledge compilation: The general learning mechanism. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Proceedings of the 1983 Machine Learning Workshop.* , 1983.

Bower, G. H. & Winzenz, D. Group structure, coding, and memory for digit series. *Journal of Experimental Psychology Monograph*, 1969, *80*, 1-17. (May, Pt. 2).

Chase, W. G., & Ericsson, K. A. Skilled memory. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

Chase, W. G. & Simon, H. A. Perception in chess. *Cognitive Psychology*, 1973, *4*, 55-81.

DeGroot, A. D. *Thought and Choice in Chess.* The Hague: Mouton, 1965.

Feigenbaum, E. A., & Simon, H. A. EPAM-like models of recognition and learning. *Cognitive Science*, 1984, *8*, 305-336.

Fikes, R. E., Hart, P. E., & Nilsson, N. J. Learning and executing generalized robot plans. *Artificial Intelligence*, 1972, *3*, 251-288.

Johnson, N. F. Organization and the concept of a memory code. In Melton, A. W. & Martin, E. (Eds.), *Coding Processes in Human Memory.* Washington, D.C.: Winston, 1972.

Korf, R. E. Macro-operators: A weak method for learning. *Artificial Intelligence*,

1985, *26*, 35-77.

Laird, J. E. *Universal Subgoaling.* Doctoral dissertation, Carnegie-Mellon University, 1983. (Available in Laird, J. E., Rosenbloom, P. S., & Newell, A. *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Hingham, MA: Kluwer, 1986).

Laird, J. E. *Soar User's Manual (Version 4)* (Tech. Rep. ISL-15). Xerox Palo Alto Research Center, 1986.

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

Laird, J. E., Rosenbloom, P. S., & Newell, A. Towards chunking as a general learning mechanism. In *Proceedings of AAAI-84.* Austin: , 1984.

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

Lewis, C. H. *Production system models of practice effects.* Doctoral dissertation, University of Michigan, 1978.

McDermott, J. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 1982, *19*, 39-88.

Michie, D. "Memo" functions and machine learning. *Nature*, 1968, *218*, 19-22.

Miller, G. A. The magic number seven plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 1956, *63*, 81-97.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. Explanation-based generalization: A unifying view. *Machine Learning*, 1986, *1*, 47-80.

Neves, D. M. & Anderson, J. R. Knowledge compilation: Mechanisms for the automatization of cognitive skills. In Anderson, J. R. (Ed.), *Cognitive Skills and their Acquisition.* Hillsdale, NJ: Erlbaum, 1981.

Newell, A. Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.), *Attention and Performance VIII.* Hillsdale, N.J.: Erlbaum, 1980.

Newell, A. & Rosenbloom, P. S. Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive Skills and their Acquisition.* Hillsdale, NJ: Erlbaum, 1981.

Rosenbloom, P. S. *The Chunking of Goal Hierarchies: A Model of Practice and Stimulus-Response Compatibility.* Doctoral dissertation, Carnegie-Mellon University, 1983. (Available in Laird, J. E., Rosenbloom, P. S., & Newell, A. *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*, Hingham, MA: Kluwer, 1986).

Rosenbloom, P. S., & Newell, A. An integrated computational model of stimulus-response compatibility and practice. In G. H. Bower (Ed.), *The Psychology of Learning and Motivation (Vol. 21).* Academic Press, 1987. In press.

Rosenbloom, P. S., & Newell, A. Learning by chunking: A production-system model of practice. In D. Klahr, P. Langley, R. Neches (Eds.), *Production System Models of Learning and Development.* Cambridge, MA: Bradford Books/MIT Press, 1987.

Rosenbloom, P. S., Laird, J. E., & Newell, A. Knowledge level learning in Soar. In *Proceedings of AAAI-87.* Seattle: , 1987.

Rosenbloom, P. S., Laird, J. E., McDermott, J., Newell, A. & Orciuch, E. R1-Soar: An experiment in knowledge-intensive programming in a problem-solving architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1985, *7*, 561-569.

Seibel, R. Discrimination reaction time for a 1,023 alternative task. *Journal of Experimental Psychology*, 1963, *66*, 215-226.

Snoddy, G. S. Learning and stability. *Journal of Applied Psychology*, 1926, *10*, 1-36.

Steier, D. M., Laird, J. E., Newell, A., Rosenbloom, P. S., Flynn, R., Golding, A., Polk, T. A., Shivers, O. G., Unruh, A., & Yost, G. R. Varieties of Learning in Soar: 1987. In P. Langley (Ed.), *Proceedings of the Fourth International Workshop on Machine Learning.* Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1987.

van de Brug, A., Rosenbloom, P. S., & Newell, A. Some Experiments with R1-Soar. 1987. In preparation.

Watkins, M. J., & Gardiner, J. M. An appreciation of generate-recognize theory of recall. *Journal of Verbal Learning and Verbal Behavior*, 1979, *18*, 687-704.