Towards Chunking as a General Learning Mechanism

John E. Laird, Paul S. Rosenbloom and Allen Newell
Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

ABSTRACT

Chunks have long been proposed as a basic organizational unit for human memory. More recently chunks have been used to model human learning on simple perceptual-motor skills. In this paper we describe recent progress in extending chunking to be a general learning mechanism by implementing it within a general problem solver. Using the ***Soar*** problem-solving architecture, we take significant steps toward a general problem solver that can learn about all aspects of its behavior. We demonstrate chunking in ***Soar*** on three tasks: the Eight Puzzle, Tic-Tac-Toe, and a part of the ***R1*** computer-configuration task. Not only is there improvement with practice, but chunking also produces significant transfer of learned behavior, and strategy acquisition.

1 Introduction

Chunking was first proposed as a model of human memory by Miller [8], and has since become a major component of theories of cognition. More recently it has been proposed that a theory of human learning based on chunking could model the ubiquitous power law of practice [12]. In demonstrating that a practice mechanism based on chunking is capable of speeding up task performance, it was speculated that chunking, when combined with a general problem solver, might be capable of more interesting forms of learning than just simple speed ups [14]. In this paper we describe an initial investigation into chunking as a general learning mechanism.

Our approach to developing a general learning mechanism is based on the hypothesis that all complex behavior — which includes behavior concerned with learning — occurs as search in problem spaces [11]. One image of a system meeting this requirement consists of the combination of a performance system based on search in problem spaces, and a complex, analytical, learning system also based on search in problem spaces [10]. An alternative, and the one we adopt here, is to propose that all complex behavior occurs in the problem-space-based performance system. The learning component is simply a recorder of experience. It is the experience that determines the form of what is learned.

Chunking is well suited to be such a learning mechanism because it is a recorder of goal-based experience [13, 14]. It caches the processing of a subgoal in such a way that a chunk can substitute for the normal (possibly complex) processing of the subgoal the next time the same subgoal (or a suitably similar one) is generated. It is a task-independent mechanism that can be applied to all subgoals of any task in a system. Chunks are created during performance, through experience with the goals processed. No extensive analysis is required either during or after performance.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-81-K-1539. 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 or the US Government.

The essential step in turning chunking into a general learning mechanism is to combine it with a general problem-space problem solver. One candidate is ***Soar***, a reflective problem-solving architecture that has a uniform representation and can create goals to reason about any aspect of its problem-solving behavior [5]. Implementing chunking within ***Soar*** yields four contributions towards chunking as a general learning mechanism.

1. Chunking can be applied to a general problem solver to speed up its performance.

2. Chunking can improve *all* aspects of a problem solver's behavior.

3. Significant transfer of chunked knowledge is possible via the implicit generalization of chunks.

4. Chunking can perform strategy acquisition, leading to qualitatively new behavior.

Other systems have tackled individual points, but this is the first attempt to do all of them. Other work on strategy acquisition deals with the learning of qualitatively new behavior [6, 10], but it is limited to learning only one type of knowledge. These systems end up with the *wandering bottle-neck* problem — removal of a performance bottleneck from one part of a system means that some other locale becomes the bottleneck [10]. Anderson [1] has recently proposed a scheme of knowledge compilation to be a general learning mechanism to be applied to all of cognition, although it has not yet been used on complex problem solving or reasoning tasks that require learning about all aspects of behavior.

2 Soar — A General Problem-Solving Architecture

Soar is a problem solving system that is based on formulating all activity (both problems and routine tasks) as heuristic search in problem spaces. A problem space consists of a set of *states* and a set of *operators* that transform one state into another. Starting from an initial state the problem solver applies a sequence of operators in an attempt to reach a desired state. ***Soar*** uses a production system[1] to implement elementary operators, tests for goal satisfaction and failure, and *search control* — information relevant to the selection of goals, problem spaces, states, and operators. It is possible to use a problem space that has no search control, only operators and goal recognizers. Such a space will work correctly, but will be slow because of the amount of search required.

In many cases, the directly available knowledge may be insufficient for making a search-control decision or applying an operator to a state. When this happens, a *difficulty* occurs that results in the automatic creation of a subgoal to perform the necessary function. In the subgoal, ***Soar*** treats the difficulty as just another problem to solve; it selects a problem space for the subgoal

[1]A modified versions of ***Ops5*** [3], which admits parallel execution of all satisfied productions.

in which goal attainment is interpreted as finding a state that resolves the difficulty. Thus, *Soar* generates a hierarchy of goals and problem spaces. The diversity of task domains is reflected in a diversity of problem spaces. Major tasks, such as configuring a computer will have a corresponding problem space, but so also will each of the various subtasks. In addition, problem spaces will exist in the hierarchy for performing tasks generated by problems in the system's own behavior, such as the selection of an operator to apply, the application of an operator to a state, and testing for goal attainment. With such an organization, all aspects of the system's behavior are open to problem solving when necessary. We call this property *universal subgoaling* [5].

Figure 1 shows a small example of how these subgoals are used in *Soar*. This is the subgoal/problem-space structure that gets generated while trying to take steps in a task problem space. Initially (A), the problem solver is at State1 and must select an operator. If search control is unable to uniquely determine the next operator to apply, a subgoal is created to do the selection. In that subgoal (B), a *selection* problem space is used that reasons about the selection of objects from a set. In order to break the tie between objects, the selection problem space has operators to evaluate each candidate object.

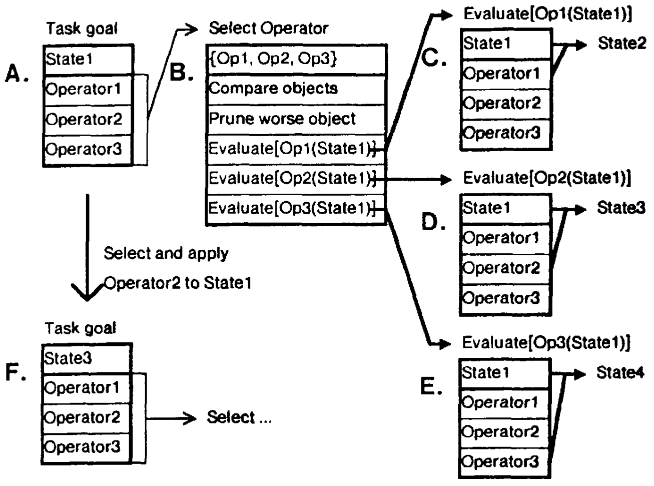

Figure 1: Eight Puzzle subgoal/problem space structure.

Evaluating an operator, such as Operator1 in the task space, is a complex problem requiring a new subgoal. In this subgoal (C), the original task problem space and state (State1) are selected. Operator1 is applied, creating a new state (State2). The evaluation for State2 is used to compare Operator1 to the other operators. When Operator1 has been evaluated, the subgoal terminates, and then the whole process is repeated for the other two operators (Operator2 and Operator3 in D and E). If, for example, Operator2 creates a state with a better evaluation than the other operators, it will be designated as better than them. The selection subgoal will terminate and the designation of Operator2 will lead to its selection in the original task goal and problem space. At this point Operator2 is reapplied to State1 and the process continues (F).

3 Chunking in Soar

Chunking was previously defined [14] as a process that acquired chunks that generate the results of a goal, given the goal and its parameters. The parameters of a goal were defined to be those aspects of the system existing prior to the goal's creation that were examined during the processing of the goal. Each chunk was represented as a set of three productions, one that encoded the parameters of a goal, one that connected this encoding in the presence of the goal to (chunked) results, and a third production that decoded the results. These chunks were learned bottom-up in the goal hierarchy; only terminal goals — goals for which there were no subgoals that had not already been chunked — were chunked. These chunks improved task performance by substituting efficient productions for complex goal processing. This mechanism was shown to work for a set of simple perceptual-motor skills based on fixed goal hierarchies [13].

At the moment, *Soar* does away with two of the features of chunking that existed for psychological modeling purposes: the three production chunks, and the the bottom-up nature of chunking. In *Soar*, single-production chunks are built for every subgoal that terminates. The power of chunking in *Soar* stems from *Soar*'s ability to automatically generate goals for problems in any aspects of its problem-solving behavior: a goal to select among alternatives leads to the creation of a production that will later control search; a goal to apply an operator to a state leads to the creation of a production that directly implements the operator; and a goal to test goal-satisfaction leads to a goal-recognition production. As search-control knowledge is added, performance improves via a reduction in the amount of search. If enough knowledge is added, there is no search; what is left is a *method* — an efficient algorithm for a task. In addition to reducing search within a single problem space, chunks can completely eliminate the search of entire subspaces whose function is to make a search-control decision, apply an operator, or recognize goal-satisfaction.

The conditions of a chunked production need to test everything that was used in creating the results of the subgoal and that existed before the subgoal was invoked. In standard problem solvers this would consist of the name of the goal and its parameters. However, in *Soar* there are no fixed goal names, nor is there a fixed set of parameters. Once a subgoal is selected, all of the information from the prior goal is still available. The problem solver makes use of the information about why the subgoal was created and any of the other information that it needs to solve the problem.

For each goal generated, the architecture maintains a *condition-list* of all data that existed before the goal was created and which was *accessed* in the goal. A datum is considered accessed if a production that matched it fires. Whenever a production is fired, all of the data it accessed that existed prior to the current goal are added to the goal's condition-list. When a goal terminates (for whatever reason), the condition-list for that goal is used to build the conditions of a chunk. Before being turned into conditions, the data is selectively variablized so that the conditions become tests for object descriptions instead of tests for the specific objects experienced. These variables are restricted so that two distinct variables can not match the same object.

The actions of the chunk should be the results of the goal. In traditional architectures, a goal produces a specific predefined type of result. However, in *Soar*, anything produced in a subgoal can potentially be of use in the parent goal. Although the potential exists for all objects to be relevant, the reality is that only a few of them will actually be useful. In figuring out the actions of the chunk, *Soar* starts with everything created in the goal, but then prunes away the information that does not relate directly to objects in any supergoal.[2] What is left is turned into production actions after being variablized in accordance with the conditions.

At first glance, chunking appears to be simply a caching mechanism with little hope of producing results that can be used on other than exact duplicates of tasks it has already attempted. However, if a given task shares subgoals with another task, a chunk learned for one task can apply to the other, yielding *across-task*

[2] Those that are pruned are also removed from memory because they are intermediate results that will never be used again.

transfer of learning. *Within-trial* transfer of learning can occur when a subgoal arises more than once during a single attempt on a task. Generality is possible because a chunk only contains conditions for the aspects that were accessed in the subgoal. This is an *implicit generalization*, by which many aspects of the context — the irrelevant ones — are automatically ignored by the chunk.

4 Demonstration

In this section we describe the results of experiments on three tasks: the Eight Puzzle, Tic-Tac-Toe, and computer configuration (a part of the ***R1*** expert-system implemented in ***Soar*** [15]). These tasks exhibit: (1) speed ups with practice; (2) within-trial transfer of learning; (3) across-task transfer of learning; (4) strategy acquisition (the learning of paths through search spaces); (5) knowledge acquisition in a knowledge-intensive system; and (6) learning of qualitatively different aspects of behavior. We conclude this section with a discussion of how chunking sometimes builds over-general productions.

4.1 Eight Puzzle

The states for the Eight Puzzle, as implemented in ***Soar***, consist of different configurations of eight numbered tiles in a three by three grid; the operators move the blank space up (U), down (D), left (L) and right (R) [5]. Search-control knowledge was built that computed an evaluation of a state based on the number of tiles that were moved in and out of the desired positions from the previous state.[3] At each state in the problem solving, an operator must be selected, but there is insufficient search-control knowledge to intelligently distinguish between the alternatives. This leads to the selection being made using the set of selection and evaluation goals described in Section 2. The first column of Figure 2 shows the behavior of ***Soar*** without chunking in the Eight Puzzle problem space. All of the nodes off the main path were expanded in evaluate-operator subgoals (nodes on the main path were expanded once in a subgoal, and once after being selected in the top goal).[4]

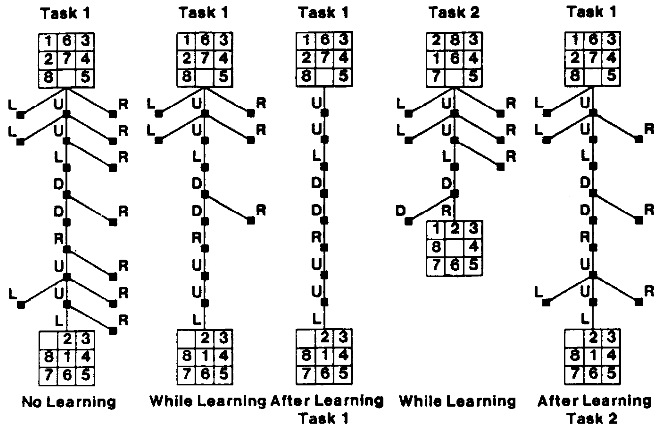

Figure 2: Within-trial and Across-task Transfer in Eight Puzzle.

[3]To avoid tight loops, search-control was also added that avoided applying the inverse of the operator that created a given state.

[4]At two points in the search the correct operator had to be selected manually because the evaluation function was insufficient to pick out the best operator. Our purpose is not to evaluate the evaluation function, but to investigate how chunking can be used in conjunction with search-control knowledge.

When ***Soar*** with chunking is applied to the task, both the selection and evaluation subgoals are chunked. During this run (second column of Figure 2), some of the newly created chunks apply to subsequent subgoals in the search. This within-trial transfer of learning speeds up performance by dramatically reducing the amount of search. The third column in the figure shows that after one run with learning, the chunked productions completely eliminate search.

To investigate across-task learning, another experiment was conducted in which ***Soar*** started with a learning trial for a different task — the initial and final states are different, and none of the intermediate states were the same (the fourth column). The first task was then attempted with the productions learned from the second task, but with chunking turned off so that there would be no additional learning (the final column). The reduced search is caused by across-task transfer of learning — some subgoals in the second trial were identical in all of the relevant ways to subgoals in the first trial. This happens because of the interaction between the problem solving only accessing information relevant to the result, and the implicit generalization of chunking only recording the information accessed.

4.2 Tic-Tac-Toe

The implementation of Tic-Tac-Toe includes only the basic problem space — the state includes the board and who is on move, the operators make a mark on the board for the appropriate player and change who is on move — and the ability to detect a win, loss or draw [5]. With just this knowledge, ***Soar*** searches depth-first through the problem space by the sequence of: (1) encountering a difficulty in selecting an operator; (2) evaluating the operators in a selection subgoal; (3) applying one of the operators in an evaluation subgoal; (4) encountering a difficulty in selecting an operator to apply to the resulting state; and (5) so on, until a terminal state is reached and evaluated.

Chunking in Tic-Tac-Toe yields two interesting results: (1) the chunks detect board symmetries, allowing a drastic reduction in search through within-trial transfer, (2) the chunks encode search-control knowledge so that the correct moves through the space are remembered. The first result is interesting because there is no knowledge in the system about the existence of symmetries, and without chunking the search bogs down terribly by re-exploring symmetric positions. The chunks make use of symmetries by ignoring orientation information that was not used during problem solving. The second point seems obvious given our presentation of chunking, however, it demonstrates the *strategy acquisition* [6, 10] abilities of chunking. Chunking acquires strategic information on the fly, using only its direct experience, and without complex post-processing of the complete solution path or knowledge learned from other trials. The quality of this path depends on the quality of the problem solving, not on the learning.

4.3 R1

Part of the ***R1*** expert system [7] was implemented in ***Soar*** to investigate whether ***Soar*** can support knowledge-intensive expert systems [15]. Figure 3 shows the subgoal structure that can be built up through universal subgoaling, including both subgoals that implement complex operators (heavy lines) and subgoals that select operators (thin lines to Selection subgoals). Each box shows the problem-space operators used in the subgoal. The actual subgoal structure extends much further wherever there is an ellipsis (...). This subgoal structure does not pre-exist in ***Soar***, but is built up as difficulties arise in selecting and applying operators.

Table 1 presents statistics from the application of ***R1-Soar*** to a small configuration task. The first three runs (Min. S-C) are with a minimal system that has only the problem spaces and goal detection defined. This base system consists of 232 productions (95 productions come with ***Soar***, 137 define ***R1-Soar***). The final three runs (Added S-C) have 10 additional search-control

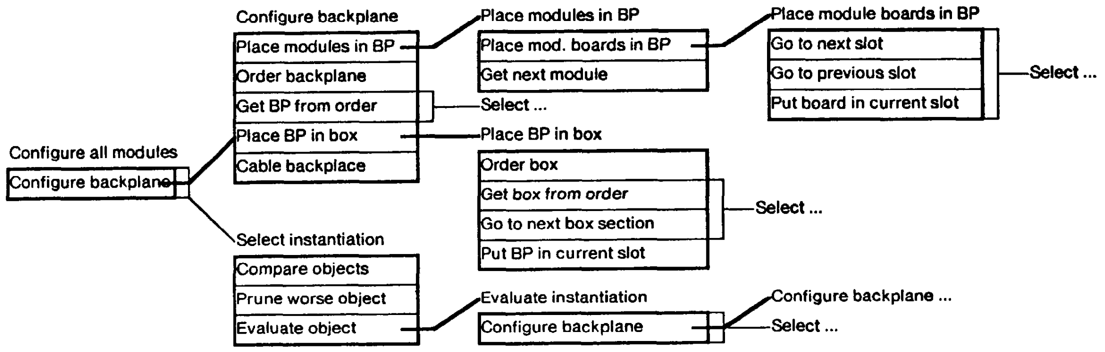

Figure 3: Subgoal Structure in ***R1-Soar***.

productions that remove much of the search. In the table, the number of search-control decisions is used as the time metric because decisions are the basic unit of problem-solving.[5]

Run Type	Initial Prod.	Final Prod.	Decisions
Min. S-C	232	232	1731
Min. S-C with chunking	232	291	485
Min. S-C after chunking	291	291	7
Added S-C	242	242	150
Added S-C with chunking	242	254	90
Added S-C after chunking	254	254	7

Table 1: Run Statistics for ***R1-Soar***.

The first run shows that with minimal search control, 1731 decisions are needed to do the task. If chunking is used, 59 productions are built during the 485 decisions it took to do this task. No prior chunking had occurred, so this shows strong within-trial transfer. After chunking, rerunning the same task takes only 7 decisions.

When ***Soar*** is run with 10 hand-crafted search-control rules, it only takes 150 decisions. This is only little more than three times faster than ***Soar*** without those rules took when chunking was used. When chunking is applied to this situation — where the additional search control already exists — it still helps by decreasing to 90 the number of decisions for the first trial. A second trial on this task once again takes only 7 decisions.

4.4 Over-generalization

The within-trial and across-task transfer in the tasks we have examined was possible because of implicit generalization. Unfortunately, implicit generalization leads to over-generalization when there is special-case knowledge that was *almost* used in solving a subgoal. In ***Soar*** this would be a production for which most but not all of the conditions were satisfied during a problem solving episode. Those conditions that were not satisfied, either tested for the absence of something that is available in the subgoal (using a negated condition) or for the presence of something missing in the subgoal (using a positive condition) . The chunk that is built for the subgoal is over-general because it does not include the *inverses* of these conditions — negated conditions for positive conditions, and positive conditions for negated conditions. During a later episode, when all of the conditions of a special-case production would be satisfied in a subgoal, the chunk learned in the first trial bypasses the subgoal. If the special-case production would lead to a different result for the goal, the chunk is over-general and produces an incorrect result.

Figure 4 contains an example of how the problem solving and chunking in ***Soar*** can lead to over-generalization. Consider the situation where O is to move in state 1. It already has the center (E), while X is on a side (B). A tie arises between all the remaining moves (A,C,D,F-I) leading to the creation of a subgoal. The Selection problem space is chosen in which each of the tieing moves are candidates to be evaluated. If position I is evaluated first, it leads to a line of play resulting in state 2, which is a win for O because of a fork. On return to the Selection problem space, move I is immediately chosen as the best move, the original tie-subgoal terminates, move I is made, and O goes on to win. When returning from the tie-subgoal, a chunk is created, with conditions sensitive to all aspects of the original state that were tested in productions that fired in the subgoals. All positions that have marks were tested (A-C, E, I) as well as those positions that had to be clear for O to have a fork (G, F). However, positions D and H were not tested. To see how this production is over-general consider state 3, where O is to move. The newly chunked production, being insensitive to the X at position D, will fire and suggest position I, which leads to a loss for O.

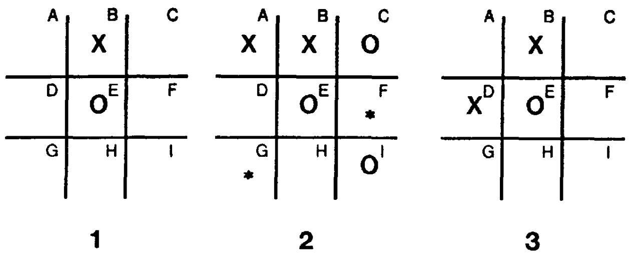

Figure 4: Over-generalization in Tic-Tac-Toe.

Over-generalization is a serious problem for ***Soar*** if we want to encode real tasks that are able to improve with experience. However, over-generalization is a problem for any learning system that works in many different environments and it leads to what is called *negative-transfer* in humans. We believe that the next step in handling over-generalization is to investigate how a problem solver can recover from over-general knowledge, and then carry out problem solving activities so that new chunks can be learned that will override the over-general chunks. This would be similar to John Anderson's work on discrimination learning using knowledge compilation [1].

[5]On a Symbolics 3600, ***Soar*** usually runs at 1 second per decision. Chunking adds an overhead of approximately 15%, mostly to compile new productions. The increased number of productions has no affect on the overall rate if the chunked productions are fully integrated into the existing production-match network.

5 Conclusion

In this paper we have taken several steps towards the establishment of chunking as a general learning mechanism. We have demonstrated that it is possible to extend chunking to complex tasks that require extensive problem solving. In experiments with the Eight Puzzle, Tic-Tac-Toe, and a part of the *R1* computer-configuration task, it was demonstrated that chunking leads to performance improvements with practice. We have also contributed to showing how chunking can be used to improve many aspects of behavior. Though this is only partial, as not all of the different types of problem solving arose in the tasks we demonstrated, we did see that chunking can be used for subgoals that involve selection of operators and application of operators. Chunking has this generality because of the ubiquity of goals in ***Soar***. Since all aspects of behavior are open to problem solving in subgoals, all aspects are open to learning. Not only is ***Soar*** able to learn about the task (chunking the main goal), it is able to learn about how to solve the task (chunking the subgoals). Because all aspects of behavior are open to problem solving, and hence to learning, ***Soar*** avoids the wandering bottle-neck problem.

In addition to leading to performance speed ups, we have shown that the implicit generalization of chunks leads to significant within-trial and across-task transfer of learning. This was demonstrated most strikingly by the ability of chunks to use symmetries in Tic-Tac-Toe positions that are not evident to the problem solving system. And finally, we have demonstrated that chunking, which on first glance is a limited caching function, is capable of strategy acquisition. It can acquire the search control required to turn search-based problem solving into an efficient method.

Though significant progress has been made, there is still a long way to go. One of the original goals of the work on chunking was to model human learning, but several of the assumptions of the original model have been abandoned on this attempt, and a better understanding is needed of just why they are necessary. We also need to understand better the characteristics of problem spaces that allow interesting forms of generalization, such as use of symmetry to take place. We have demonstrated several forms of learning, but others, such as concept formation [9], problem space creation [4], and learning by analogy [2] still need to be covered before the proposal of chunking as a general learning mechanism can be firmly established.

References

1. Anderson, J. R. Knoweldge compilation: The general learning mechanism. Proceedings of the 1983 Machine Learning Workshop, 1983.
2. Carbonell, J. G. Learning by analogy: Formulating and generalizing plans from past experience. In *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, Eds., Tioga, Palo Alto, CA, 1983.
3. Forgy, C. L. *OPS5 Manual.* Computer Science Department, Carnegie-Mellon University, 1981.
4. Hayes, J. R. and Simon, H. A. Understanding complex task instructions. In *Cognition and Instruction*, Klahr, D., Ed.,Erlbaum, Hillsdale, NJ, 1976.
5. Laird, J. E. *Universal Subgoaling.* Ph.D. Th., Computer Science Department, Carnegie-Mellon University, 1983.
6. Langley, P. Learning Effective Search Heuristics. Proceedings of IJCAI-83, IJCAI, 1983.
7. McDermott, J. "R1: A rule-based configurer of computer systems." *Artificial Intelligence 19* (1982), 39-88.
8. Miller, G. A. "The magic number seven, plus or minus two: Some limits on our capacity for processing information." *Psychological Review 63* (1956), 81-97.
9. Mitchell, T. M. *Version Spaces: An approach to concept learning.* Ph.D. Th., Stanford University, 1978.
10. Mitchell, T. M. Learning and Problem Solving. Proceedings of IJCAI-83, IJCAI, 1983.
11. Newell, A. Reasoning, problem solving and decision processes: The problem space as a fundamental category. In *Attention and Performance VIII*, R. Nickerson, Ed.,Erlbaum, Hillsdale, NJ, 1980.
12. Newell, A. and Rosenbloom, P. Mechanisms of skill acquisition and the law of practice. In *Learning and Cognition*, Anderson, J. A., Ed.,Erlbaum, Hillsdale, NJ, 1981.
13. Rosenbloom, P. S. *The Chunking of Goal Hierarchies: A Model of Practice and Stimulus-Response Compatibility.* Ph.D. Th., Carnegie-Mellon University, 1983.
14. Rosenbloom, P. S., and Newell, A. The chunking of goal hierarchies: A generalized model of practice. Proceedings of the 1983 Machine Learning Workshop, 1983.
15. Rosenbloom, P. S., Laird, J. E., McDermott, J. and Newell, A. R1-SOAR: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture. Department of Computer Science, Carnegie-Mellon University, 1984.