N89-20060

PRODUCTION SYSTEM CHUNKING IN SOAR: CASE STUDIES IN AUTOMATED LEARNING

Final Report

NASA/ASEE Summer Faculty Fellowship Program -- 1988

Johnson Space Center

Prepared By:	Robert Allen, Ph.D.
Academic Rank:	Assistant Professor
University & Department:	Univeristy of Houston Dept. of Mechanical Engineering Houston, TX 77204-4792
NASA/JSC	
Directorate:	Mission Support
Division:	Mission Planning and Analysis
Branch:	Technology Development and Applications
JSC Colleague:	Robert T. Savely
Date Submitted:	27 July 1988
Contract Number:	NGT 44-005-803

ABSTRACT

A preliminary study of SOAR, a general intelligent architecture for automated problem solving and learning, is presented. The underlying principles of universal subgoaling and chunking were applied to a simple, yet representative, problem in artificial intelligence. A number of problem space representations were examined and compared. It is concluded that learning is an inherent and beneficial aspect of problem solving. Additional studies are suggested in domains relevant to mission planning, as well as, in aspects related to SOAR itself.

available, the subgoal terminates and pops from the goal stack. The goal stack also serves as the anchor to information in working memory (WM). Each working memory element (WME) is connected to some goal in the stack and can be accessed only by specifying the connection from the goal to the WME via **augmentations**. An example of one such connection is:

(goal <g> ^state <s>)
(state <s> ^binding <b>)
(binding <b> ^cell c11 ^tile <t1>)

where **state** and **binding** are the goal augmentations that provide the connection between the goal <g> and the value of the action attribute. A **chunking mechanism** is provided to summarize the system behavior in terms of subgoals, and also enables the system to learn aspects of problem solving related to subgoals. The overall architecture is presented in Figure 1.

Some of the important characteristics of SOAR are: (1) separation between architecture level and the knowledge level; (2) problem-spaces for representing knowledge; (3) universal subgoaling to resolve impasses ; and (4) production system representation that serves as access paths to information in long term memory. In addition, there is no conflict resolution mechanism in SOAR. All matched productions are fired "in parallel" and add one type of WMEs to the working memory. The process of collecting available information is called the **elaboration phase**. The second type of WME is the **preference element** that is used by the architecture in the **decision cycle** to determine the next goal context. The decision procedure, sketched in Figure 2, controls the elaboration and decision cycles. More detailed descriptions can be found in elsewhere (1-3).

PROBLEM DOMAIN

SOAR's capabilities are illustrated below in an output trace of the problem-solving process for the "eight-puzzle" problem. The board in the eight-puzzle is represented by nine cells occupied by tiles numbered one through nine with one blank cell. The numbers on the beginning of each line are the decision cycle numbers; elaboration cycles are not shown. The **Build:P** notation signifies a

INTODUCTION

SOAR is a production system architecture for a system capable of exhibiting general intelligence. SOAR has three principal characteristics separating it from other architectures. These are: (1) SOAR can be used to solve a range of problems, from routine problems to open-ended problems; (2) SOAR applies a wide range of problem-solving methods required for these tasks; and (3) SOAR learns about aspects of the problem-solving process and is capable of reporting about its performance.

This document summarizes the work performed in implementing simple, yet representative, problems in SOAR. One purpose of this exercise was to be acquainted with the SOAR architecture and implementation. In the course of this work, some general issues were raised and found to coincide with current research topics in SOAR.

This paper is organized as follows. First, a brief description of SOAR is presented. Next, some "toy" AI problems are briefly described and their implementations in SOAR are presented. Penultimately, a comparison of some problems decompositions is examined and the affect of problem presentation on learning and performance is discussed. Finally, future studies that can be performed are recommended.

SOAR: AN OVERVIEW

SOAR is an architecture for exhibiting general intelligent behavior. SOAR has evolved from a series of production system architectures (1,2). SOAR is embedded in about 255 kilobytes of LISP code, and extends 100 kilobytes of modified OPS5.code.

In SOAR, each task is represented in **problem-spaces**. The problem solving process begins from an initial state, working through the state to subsequent states by applying operators. Stages in the problem solving process are characterized by a goal context, which consists of the current **goal**, **problem-space**, **state** and **operator**. When one stage in the problem-solving process does not have enough information, it creates a subgoal (hence the name "universal subgoaling") to collect the needed information. The new subgoal is then added to the goal stack that keeps track of the goals that were created for this problem-space. When the needed information is

Soar Architecture

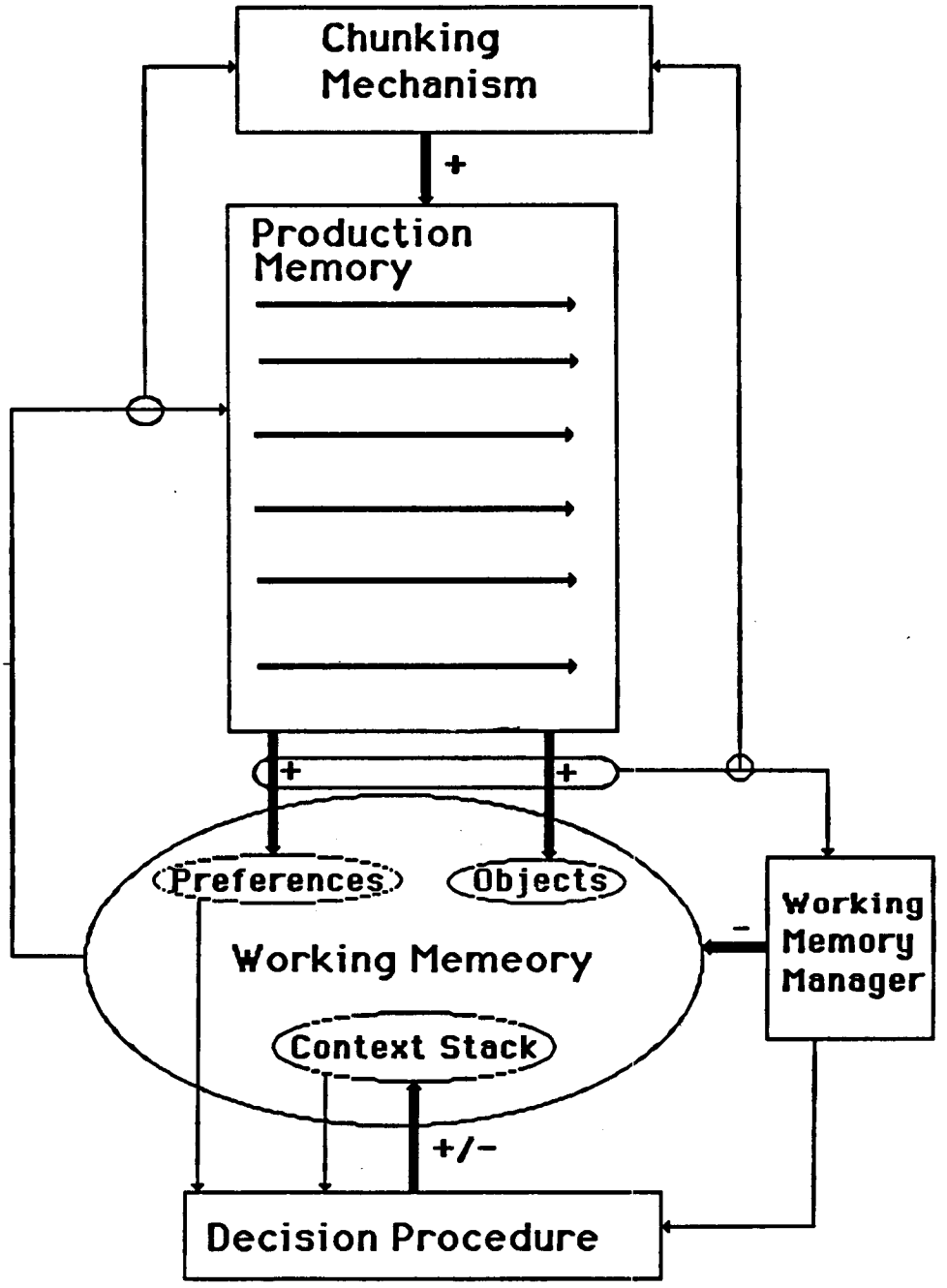

Figure 1

Decision Procedure

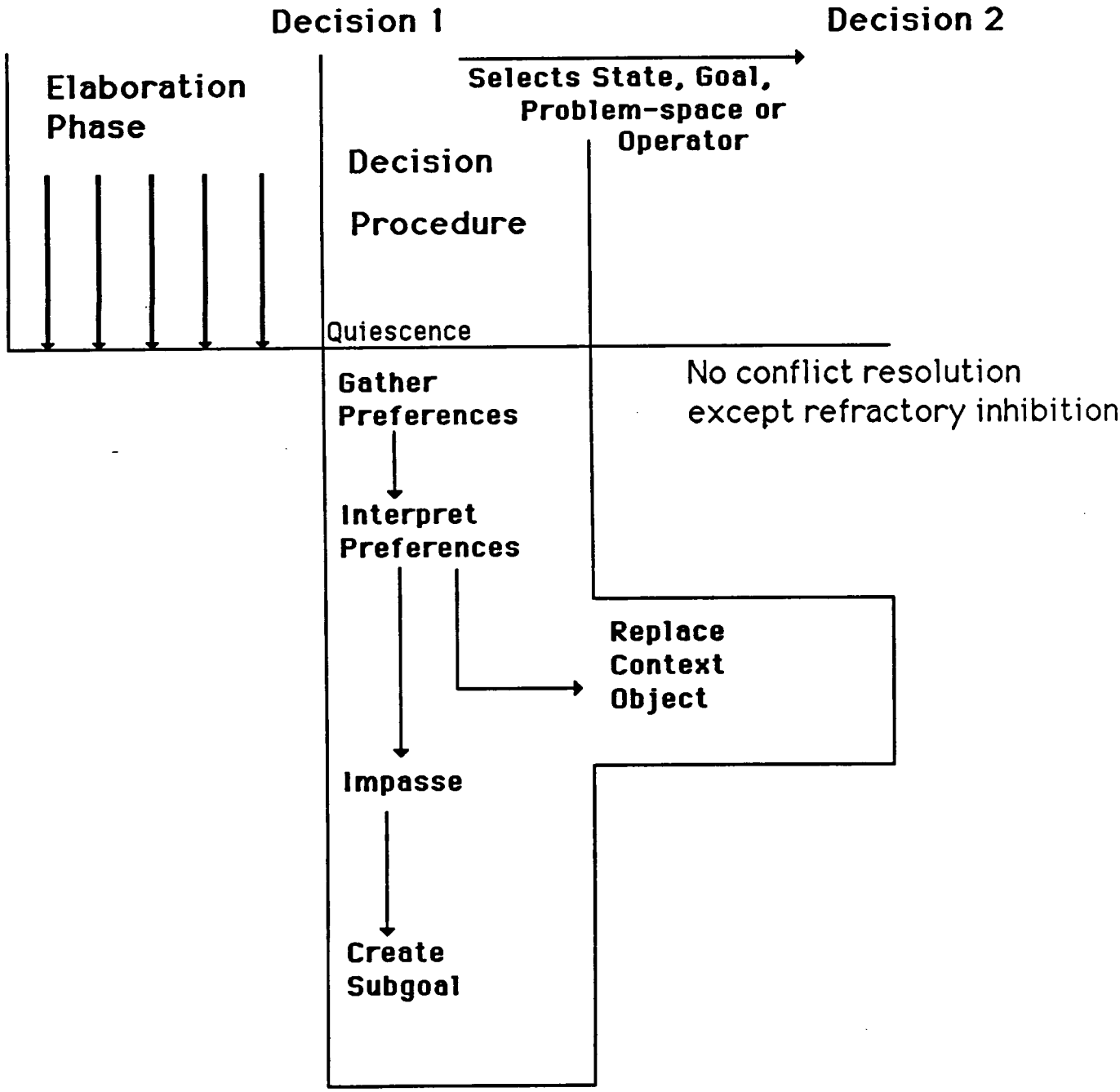

Figure 2

new chunk (rule) being created during the trace. The letters G, P, O and S represent the current goal, problem-space, operator and state, respectively.

Learn status: on all-goals print trace

```
0  G: G1
1  P: P3 EIGHT-PUZZLE
2  S: S4
3  O: O34 MOVE-TILE(C22)

C22(S4)  -->  S35
          -------------
     | 2 | 8 | 3 |
         |---|---|---|
     | 1 | 0 | 4 |
         |---|---|---|
     | 7 | 6 | 5 |
          -------------

4  S: S35
5  ==>G: G185 (UNDECIDED OPERATOR TIE)
6     P: P186 SELECTION
7     S: S42
8     O: (O47 O46 O45)
9     ==>G: G190  (EVALUATE-OBJECT (MOVE-TILE(C21)) OPERATOR
NO-CHANGE)
10       P: P3 EIGHT-PUZZLE
11       S: S35
12       O: O40 MOVE-TILE(C12)

C21(S35)  -->  S48
          -------------
     | 2 | 0 | 3 |
         |---|---|---|
     | 1 | 8 | 4 |
         |---|---|---|
     | 7 | 6 | 5 |
          -------------
```

```
C12(S35) --> S53
          -------------
     | 2 | 8 | 3 |
        |---|---|---|
     | 0 | 1 | 4 |
        |---|---|---|
     | 7 | 6 | 5 |
          -------------

13        S: S53
Duplicate chunk
Build:P200
Duplicate chunk
Build:P202
Build:P204
Build:P205
14  O: O41 MOVE-TILE(C21)

C21(S35) --> S59
          -------------
     | 2 | 0 | 3 |
        |---|---|---|
     | 1 | 8 | 4 |
        |---|---|---|
     | 7 | 6 | 5 |
          -------------

15  S: S59
16  O: O74 MOVE-TILE(C11)

C11(S59) --> S64
          -------------
     | 0 | 2 | 3 |
        |---|---|---|
     | 1 | 8 | 4 |
        |---|---|---|
     | 7 | 6 | 5 |
          -------------
```

17 S: S64
18 O: O76 MOVE-TILE(C12)

C12(S64) --> S70

```
 -------------
| 1 | 2 | 3 |
|---|---|---|
| 0 | 8 | 4 |
|---|---|---|
| 7 | 6 | 5 |
 -------------
```

19 S: S70
20 O: O80 MOVE-TILE(C22)

C22(S70) --> S77

```
 -------------
| 1 | 2 | 3 |
|---|---|---|
| 8 | 0 | 4 |
|---|---|---|
| 7 | 6 | 5 |
 -------------
```

21 S: S77
goal SOLVE-EIGHT-PUZZLE achieved

```
>(print-stats)

Soar 4.4 (external release: created April 19, 1987)
Run statistics on July  26, 1988
81 productions (1345 / 5703 nodes)
229.41667 seconds elapsed (43.133335 seconds chunking overhead)
21 decision cycles (10924.604 ms per cycle)
46 elaboration cycles (4987.319 ms per cycle)
    (2.1904762 e cycles/d cycle)
211 production firings (1087.2828 ms per firing)
    (4.5869565 productions in parallel)
717 RHS actions after initialization (319.96747 ms per action)
226 mean working memory size (339 maximum, 296 current)
2890 mean token memory size (10696 maximum, 3893 current)
22363 total number of tokens added
18470 total number of tokens removed
40833 token changes (5.6184134 ms per change)
    (56.7125 changes/action)
```

The particular implementation of this problem in SOAR requires twelve productions, which can be divided as follows: (1) four rules for setting-up the problem space; (2) one rule to create an operator instantiation; (3) two rules for applying an instantiated operator; (4) one rule for search control; (5) one rule for monitoring states; and (6) three rules for evaluating operators. While the rules described are specific to this problem, encoding other tasks typically involve a similar division of a problem and hence a similar division of rules.

LEARNING

The chunk below is an example of a rule that becomes part of production memory during the session outlined above.

```
(SP P218 ELABORATE
  (GOAL <G1> ^OPERATOR { <> UNDECIDED <O2> })
  (OPERATOR <O2>
      ^NAME EVALUATE-OBJECT ^ROLE OPERATOR
      ^SUPERPROBLEM-SPACE <P1> ^OBJECT <O1> ^SUPERSTATE <S1>
      ^DESIRED <D1> ^EVALUATION <E1>)
  (PROBLEM-SPACE <P1> ^NAME EIGHT-PUZZLE)
  (OPERATOR <O1>
      ^NAME MOVE-TILE ^BLANK-CELL <C2> ^TILE-CELL <C1>)
  (STATE <S1> ^BLANK-CELL <C2> ^BINDING <B1> { <> <B1> <B2> })
  (BINDING <B1> ^CELL <C2>)
  (BINDING <B2> ^CELL <C1> ^TILE <T2>)
  (DESIRED <D1> ^BINDING <D2>)
  (BINDING { <> <B1> <D2> } ^CELL <C2>)
  (BINDING <D2> ^TILE <T2>)
  -->
  (DISPLACED (MAKE EVALUATION <E1> NUMERIC-VALUE 1)
^(NLAM-MAKE (QUOTE (EVALUATION <E1> NUMERIC-VALUE 1)))))
```

In more transparent form the rule states:

> IF any tile is being moved into its final position
> THEN the next state will yield an evaluation of 1,
> i.e., the next state will be evaluated favorably

This particular rule is one that is created because of the current implementation; the search control rule creates a **worst** preference for those operators not placing tiles in their desired locations. Other search control productions are responsible for the creation of chunks more in line with their method of search control (4).

The search control rules are of importance in the outcome of the problem-solving process. To explore the effect of these rules on the parameters to measure performance, sample runs were performed using three different search control strategies. The results are presented in Table 1. The third column refers to a search control procedure that rejects an operator that will move back a tile into its previous location. The fourth column refers to a search control mechanism that creates a worst preference to a tile that is to be moved out of its desired location. The fifth column corresponds to no

Type of run	Data	Reject * undo	Worst * Preference	No Search
No Learning	number of productions	73	74	82
	elapsed time	177	53	268
	decision cycles	48	21	39
	elaboration cycles	84	43	73
	production firings	442	181	383
	working memory elements	276	301	339
Learning	number of productions	81	77	72
	elapsed time	112 (24)	54 (9)	281 (114)
	decision cycles	30	21	62
	elaboration cycles	55	43	94
	production firings	303	182	572
	working memory elements	323	301	453
Learned	number of productions	81	77	82
	elapsed time	51 (0)	28 (0)	48 (0)
	decision cycles	12	12	12
	elaboration cycles	29	25	29
	production firings	120	104	116
	working memory elements	295	279	291
	chunks used	6	3	7

Table 1: Eight-Puzzle Results

prespecified search control strategy. Each row contains data for each type of run. The data contain the number of productions in the system, the elapsed time to complete the task, the number of decision cycles, the number of elaboration cycles, the number of production firings and the maximum number of working memory elements. The last row, corresponding to runs that have already "learned," indicates the number of chunks that were fired during a problem solving session. The numbers in parentheses indicate the amount of time required for chunking.

As expected, chunking added productions to each system. However, a learned system did not chunk again. Also as expected, a "stronger" search control mechanism (column Four) corresponded to *a* better overall performance: fewer cycles, fewer rule firings, more efficient memory and more efficient chunking The reason for this is a *priori* control of the preferences in the "worst*preference" strategy. Applying specific preference orientations in the search control strategy appears to be an effective way to structure a problem implementation (1,4). Finally, it is noted that even without a specified search control strategy (column Five), SOAR's default strategy is sufficient to solve this problem prior to chunking.

CONCLUSIONS

A preliminary examination of SOAR has been performed and, from the above-mentioned results, the following conclusions are drawn:

- Learning is a beneficial aspect of automated problem.solving in that code can be made more efficient and previously unrecognized knowledge, in the form of chunks, can be created.

- Problem decomposition is the key to an efficient (or even successful implementation). The search control mechanism(s) used in a specific implementation strongly influences the problem-solving process and the learning process.

- Measuring the difficulty of a problem is a nontrivial task. The performance criteria measured by SOAR need to be scrutinized before a definitive measure can be ascertained.

It is clear that additional studies are needed, with more practical problems (such as those presented in (4)), to see if the SOAR architecture can be useful in NASA-related applications.

Acknowledgment

The author thanks Lui Wang, Aerospace Engineer for the Artificial Intelligence Section, for his assistance with using the microEXPLORER™ and the Lisp environment.

References

1. Laird, J.E., "SOAR User's Manual," Version 4, Xerox Palo Alto Research Center, Palo Alto, CA 1986.

2. Laird, J..E., A. Newell and P.S. Rosenbloom, "SOAR: An Architecture for General Intelligence," Artificial Intelligence, 33 (1987)1-64.

3. Laird, J., P. Rosenbloom and A.Newell, "Universal Subgoaling and Chunking, Kluwer Academic Press, 1986.

4. Reich, Y. and S.J. Fenves, "Floor-system Design in SOAR: A Case Study in Learning to Learn," Technical Report EDRC-12-26-88, Carnegie-Mellon University, Pittsburgh, PA 1988.