THE EFFICIENCY OF CERTAIN PRODUCTION SYSTEM IMPLEMENTATIONS

CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA

SEPTEMBER 1976

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR - TR - 77 - 0011	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE EFFICIENCY OF CERTAIN PRODUCTION SYSTEM IMPLEMENTATIONS		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. McDermott, A. Newell & J. Moore		8. CONTRACT OR GRANT NUMBER(s) F44620-73-C-0074
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Dept. Pittsburgh, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101D AO 2466
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington, VA 22209		12. REPORT DATE September 1976 13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Office of S ientific Research (NM) Bolling AFB, DC 20332		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distributinn unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

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

Abstract. In this paper, methods for increasing the efficiency of the class of control structures called production systems are discussed. After briefly describing three algorithms for processing production systems, we attempt to isolate the characteristics which most affect their cost. We then present three production systems to display these dependencies.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

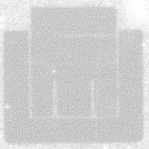
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

THE EFFICIENCY OF CERTAIN PRODUCTION SYSTEM IMPLEMENTATIONS

J. McDermott, A. Newell and J. Moore [1]

September, 1976

DEPARTMENT
of
COMPUTER SCIENCE



Carnegie-Mellon University

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is approved for public release IAW AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

THE EFFICIENCY OF CERTAIN PRODUCTION SYSTEM IMPLEMENTATIONS

J. McDermott, A. Newell and J. Moore [1]
September, 1976

Abstract. In this paper, methods for increasing the efficiency of the class of control structures called production systems are discussed. After briefly describing three algorithms for processing production systems, we attempt to isolate the characteristics which most affect their cost. We then present three production systems to display these dependencies.

This work was supported in part by the Defense Advanced Research Projects Agency (F44620-73-C-0074) and is monitored by the Air Force Office of Scientific Research.

1 J. Moore is now at Information Sciences Institute, University of Southern California.

I. INTRODUCTION

There has recently been considerable investigation in Artificial Intelligence of a class of control structures called production systems. Production systems provide one alternative for how to organize an intelligent system [Davis and King, 1976; Feigenbaum, Buchanan and Lederberg, 1971; Newell, 1973; Newell and Simon, 1972]. This paper is concerned with the efficiency of implementation of production systems. If implemented straightforwardly according to their definition (as is done in most initial and demonstration versions), they are quite expensive, especially for large systems. Thus, alternative implementations must be developed. In this paper we explore several schemes and give some experimental results.

A production system consists of a set of productions and a working memory. Working memory, WM, is a collection of memory elements. A production, P, is a conditional statement composed of zero or more condition elements and zero or more action elements. Whenever each condition element, C, in a production is supported by a memory element, M, in working memory, the production is satisfied and may be fired. When a production is fired, each action element, A, in the production is executed, and this ordinarily causes working memory to be modified in some way; for example a new element might be added to working memory or an element in working memory might be deleted. The basic unit of behavior is the recognition-act cycle. The cycle has two parts: (1) discovering the conflict set, the subset of satisfied productions, and selecting one of the productions in the conflict set as the production to be fired; and then (2) firing the production selected, ie, executing the action elements of the production. Processing continues, cycle follows cycle, until either no production is satisfied or an action element explicitly stops the processing.

As an example, consider the following production system whose production memory contains three productions and whose working memory is a list of fixed length containing six elements:

P1: (4 --> (SAY GO) STOP)
P2: (<DIGIT> --> (SAY <DIGIT>) (<DIGIT> + 1))
P3: (READY --> 1)

WM: (READY JUNK1 JUNK2 JUNK3 JUNK4 JUNK5)

Assume that when there is more than one production in the conflict set, the first satisfied production in order from the top is fired. Given this assumption, at the beginning of the first cycle, P3 is discovered to be satisfied; it is fired and this results in the number 1 being added to working memory.

WM: (1 READY JUNK1 JUNK2 JUNK3 JUNK4)

At the beginning of the second, third, and fourth cycles, both P2 and P3 are found to be satisfied; P2 is selected to be fired; the first time it is fired, it outputs 1 and then 2 is added to working memory; the second time, 2 is output and 3 is added to working memory; the fourth time, 3 is output and 4 is added to working memory.

WM: (2 1 READY JUNK1 JUNK2 JUNK3)
WM: (3 2 1 READY JUNK1 JUNK2)
WM: (4 3 2 1 READY JUNK1)

Finally, at the beginning of the fifth cycle, all three productions are found to be satisfied; P1 is selected; when it is fired, it outputs GO and then stops the processing.

A production system forms a complete computational system. The properties that make it of interest are its homogeneous control structure, the modularity of its encoding, the almost complete procedural representation and its embodiment of a recognition-act philosophy to determine what to do next. We are interested here only in implementation schemes on standard uniprocess computing systems.

The cost of a production system is simply the per cycle cost of finding the conflict set and executing the action elements in the production selected to be fired. If one uses the most obvious method of determining which productions are members of the conflict set -- testing the condition elements of each production against the elements in working memory at the beginning of each cycle -- then the time it takes to discover the conflict set can be much greater than the time taken by the second stage of the cycle. Indeed, if a production system has either a large number of productions or a large number of memory elements, then the cost of the first stage can be prohibitive.

In this paper we will present some results of an attempt to reduce the cost of discovering the conflict set. Throughout, we will use a particular production system architecture, PSG [Newell, 1973; Newell and McDermott, 1975], but the considerations will be applicable generally. In section II we describe how production systems processed by PSG discover the conflict set and the heuristics we have used to make this processing more efficient. In section III we present formulas for the costs of these various schemes. In section IV we compare experimentally the time requirements of PSG when the heuristics are used with the time requirements of PSG alone for three different production systems.

II. IMPLEMENTATION ALGORITHMS

A production system processed by PSG has a list of elements (M) as its working memory and contains a list of productions whose condition elements (C) are either named symbols or unnamed lists of named symbols. To determine if a production is satisfied, the following three criteria are used: (1) every C must match a distinct M; (2) a C and an M match only if they have the same name, or the C is a matching variable, or they match as expressions; (3) a C and an M match as expressions only if the subelements of the C and the subelements of the M are in the same order, the first subelement of the C matches the first subelement of the M, and each subsequent subelement of the C matches a subelement of the M. Thus, consider the following working memory containing three elements

WM: ((E (F) G) D ((I H)))

and the six productions

P4: ((E (F) X) ((I X)) --> ...)
P5: (((F) G) --> ...)
P6: (D (E G) --> ...)
P7: (((I H G)) --> ...)
P8: (((I)) --> ...)
P9: (E --> ...)

If X is a variable, P6 and P8 would both be satisfied, while the other four productions would not.

PSG determines which productions are members of the conflict set in the following way: At the beginning of each cycle, the first condition element in the first production is matched against each memory element until a match is found. Then the second condition element is matched against each memory element. This continues until an attempted match fails or until all of the condition elements have been found to be supported, in which case the production is inserted into the conflict set. This procedure is repeated for each of the productions. We can schematize this as follows:

```
for every P
   SUCCESS ← true
   while SUCCESS
      for every C
         FOUND ← false
         while not FOUND
            for every M
              if match(C, M)
                then
                   FOUND ← true
                   if final C then add P to CONFLICT SET
                else if final M then SUCCESS ← false
```

PSG provides two alternative methods for resolving conflict in the case of more than one production being satisfied. If production order is used, productions are tested in order until the first satisfied one is found; then that production is fired. Thus for production order, no explicit conflict set is formed. If memory order is used, all productions are tested; then that production in the conflict set supported by the memory element most recently added to working memory is fired.

The basic procedure that PSG uses to determine which productions are in the conflict set does not avail itself of any of the knowledge that could be obtained before execution by determining which productions contain identical (or similar) condition elements. Nor does it make use of the information actually computed during previous cycles. In other words, knowledge of which productions are satisfied, which condition elements are supported, and what memory elements are in working memory is lost at the end of each cycle and thus must be continuously rediscovered. The absence of

this information leads, of course, to much redundancy in processing. One obvious way to eliminate this redundancy is to build a machine to augment PSG that can store and update information that can be helpful in determining whether productions are satisfied. A wide variety of such machines are possible; they differ from one another in the information they store and in the ways that they make use of it. Basically, however, all of the knowledge which is relevant comes from one of three sources. These knowledge sources (KSs) can be characterized as follows:

1. The Condition-membership KS provides knowledge about the occurrence of condition elements in productions.
2. The Memory-support KS provides knowledge about the memory elements that support condition elements.
3. The Condition-relationship KS provides knowledge about the relationship among condition elements within the conditions of each production.

We give the name filter to any machine which uses such knowledge sources to reduce the number of productions tested by a production system architecture. As shown in Figure 1, a filter admits to further testing that subset of productions not known to be unsatisfied on the basis of whatever information is available in the KSs which it uses. The set of productions passed through the filter to full-fledged evaluation is called the P+ set. To be useful in a filter, a KS must contain enough information so that a significant number of productions can be excluded from consideration. The filter must also be able to store this information in such a way that it can be accessed cheaply and (when necessary) updated, so that the cost of using the KS is significantly less than the cost of processing the excluded productions.

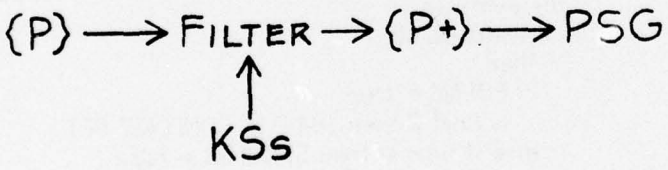


FIGURE 1. PSG AUGMENTED BY A FILTER

Since a filter bears no responsibility for assuring that a production is satisfied, it depends on the basic process (here represented by the PSG algorithm) to check the final candidates in the P+ set. Clearly, an alternative to using a few inexpensive tests to eliminate large numbers of productions from consideration is to embed the sources of knowledge directly into an algorithm that makes the full selection of the production to be executed. An effort by Forgy [1976] takes exactly this course, based on ideas very similar to the ones explored here.

Knowledge of what conditions occur in the productions permits rejection of productions whose conditions elements are not supported by elements in working memory. In other words, filters based on the Condition-membership KS can pass productions all of whose conditions elements appear to be supported by memory elements. This does not guarantee that such productions will be satisfied: the condition

elements may not match the memory elements in detail (if the filter is using only partial knowledge); or all conditions might match individually, but not when considered together (eg, due to instantiation of variables). Given, for example,

P10: (D E --> ...)
P11: (E F --> ...)
P12: (F D --> ...)
P13: (D D --> ...)

WM: (D F)

and given a KS containing the information that P10, P12, and P13 contain D, P10 and P11 contain E, and P11 and P12 contain F, a filter could match the condition elements D, E, and F against the elements in working memory and then generate P+: {P12 P13}. Only three condition elements would be matched against the elements in working memory by PSG, rather than the eight (actually seven) which would have to be matched if the information in the Condition-membership KS were not available.

The Memory-support KS contains the knowledge of which condition elements are supported by a memory element at the beginning of a cycle. This knowledge in the Memory-support KS complements that in the Condition-membership KS. That is, if a filter makes use of both of these KSs, then (continuing with the above example), since the Memory-support KS would contain the information that the condition elements D and F are supported, no matching would be necessary at the beginning of the cycle to generate P+: {P12 P13}. Of course the knowledge in the Memory-support KS must be updated at the end of each cycle. This updating involves matching the elements being added to and deleted from working memory with each of the condition elements, and the cost of doing this is primarily the cost of doing the matching. To determine whether the updating cost would be more or less than the cost of matching each condition element with the elements in working memory at the beginning of each cycle, one must know the ratio of the number of memory elements added or deleted per cycle to the number of elements in working memory. If the ratio is 1 to 1, then the two costs are approximately the same.

The third KS, the Condition-relationship KS, is probably of value only if the other two KSs are also being used. Like the Memory-support KS, it contains knowledge which must be updated during each cycle; this knowledge can be used to determine whether a production, all of whose condition elements are supported when considered in isolation, is actually satisfied. Consider, for example,

P13: (D D --> ...)
P14: (X (E X) --> ...)

WM: (D (E F))

where X is a variable which will match any memory element. The knowledge stored in both the Condition-membership and Memory-support KSs would not indicate that either

of these productions are unsatisfied, though both are. The Condition-relationship KS would contain the information that P13 can be satisfied only if D occurs twice in working memory and that P14 can be satisfied only if the variable X can be bound to the same symbol each time it is instantiated.

If one builds a filter that makes full use of all three KSs, then this filter can generate conflict sets (ie, P+ sets that contain no unsatisfied productions). Such a filter, of course, eliminates the cost of testing (essentially retesting) each production in the P+ set to determine whether it is satisfied. However, the cost of updating the Condition-relationship KS can be considerable; whether or not such a filter can be cost effective is largely dependent on the production system architecture which the filter is augmenting. For an architecture like PSG, it is doubtful that the extra power of such a filter would be worth the additional cost. PSG does not look for more than one instantiation of the condition side of each production. Since the probability is ordinarily high that an instantiation will be found for each production in the P+ sets generated by a filter using the Condition-membership KS (or both the Condition-membership and Memory-support KSs), the high cost of insuring that there is such an instantiation is likely to be a wasted expense. Consequently, we have not built a filter for PSG which uses the Condition-relationship KS. Other architectures, however, particularly those which look for all instantiations of the condition side of each production, are more likely to benefit from being augmented by a filter which generates conflict sets [see Forgy, 1976].

Filters need not use full knowledge, eg, of whether a condition element can match a memory element. Good filter design will trade off the use of more precise knowledge to obtain simpler computation. For example, consider

P15: ((D) (E F) --> ...)
P16: ((D X) (I G) --> ...)
P17: ((D X Y) (I Y) --> ...)

WM: ((D E F) (I E F G))

where X and Y are variables. A filter using the Condition-membership KS, as described above, would have to match each of the six condition elements with the elements in working memory to generate P+: {P16 P17}. If, on the other hand, the KS contained no explicit references to specific elements, but instead contained only information about classes of elements -- where the classes might be condition elements whose first symbol is D, condition elements whose first symbol is E, and so on -- then only three matches would be needed to generate the same P+ set, and each of the matches would involve comparing only a single pair of symbols. Such a heuristic does have its failings. Consider

P18: ((D E) --> ...)
P19: ((D F) --> ...)
P20: ((D G) --> ...)

WM: ((D E))

Using the heuristic, P+: {P18 P19 P20} would be generated, whereas not using the heuristic would yield P+: {P18}. Nevertheless, the idea of exploring class representations for the KSs is attractive. Both of the filters which we have implemented are efforts in this direction.

Productions may also contain negative condition elements; in this case, a production is satisfied only if the condition element is not supported by any element in working memory. In PSG, a condition element is negated by placing the symbol ABS after it. For example, consider

P21: ((F X) (E X) ABS --> ...)
P22: ((D X) (E X) ABS --> ...)

WM: ((F H) (D D) (E H))

where X is a variable. P21 is not satisfied (X will be bound to H and the element (E H) is not absent from working memory); but P22 is satisfied (X will be bound to D and the element (E D) is absent from working memory). The ABS construct causes a problem for any filter which does not make use of the Condition-relationship KS. If, for instance, we made no special provision for ABS condition elements and constructed a filter which made use of either of the Condition-membership KSs representations described above, then since the condition element (E X) considered in isolation is supported, the filter would incorrectly generate an empty P+ set. One solution to this problem, and the one we have adopted here, is to design filters that ignore ABS condition elements. No information at all about the second condition elements in P21 and P22 would be stored in the KS. Thus from the point of view of the filter, the two productions above would each contain only a single condition element.

A Condition-membership Filter

One of the two filters we have constructed makes use of the Condition-membership KS exclusively; we have (somewhat unimaginatively) given it the name CmF. The KS is a discrimination net which indicates which productions contain similar condition elements. Here similar means having the same first named symbol or primary feature (PF). Each node in the network is a single primary feature and thus represents all condition elements which have that feature as their first named symbol. The full CmF has nodes corresponding to each feature in the production system being processed. A partial CmF has nodes corresponding to only some of the features.

At the beginning of a run (a sequence of recognition-act cycles), a list of pairs is generated; each pair contains a primary feature and the set of productions which do not contain a condition element with that feature. This list is ordered by increasing size of the sets of productions. Then the primary feature with the smallest set of associated productions (ie, the feature excluding the largest number of productions) is selected as the top node of the discrimination net. This node has two successors. Its no successor is the feature on the list which excludes the second largest number of productions; the no successor of this node is the feature on the list which excludes the third largest number of productions, and so on. The yes successor of the top node

(and of each of the other nodes in the tree) is found by reordering the list on the basis of how many additional productions are excluded by each remaining feature; the feature which excludes the largest number of additional productions is the yes successor; its no successor is the feature on the reordered list which excludes the second largest number of productions, and so on, as above. Nodes continue to be generated until the list of features is empty or until the net is of a stipulated size. [2] Given the productions

P23: ((D E) ((G)) --> . . .)
P24: ((D (F)) (D F) --> . . .)
P25: (H --> . . .)

the discrimination net in Figure 2a would be produced. (Note that the arcs that extend to the left are all no arcs and those that extend to the right are yes arcs.)

At the beginning of each cycle, a set containing the primary feature of each memory element in working memory is generated. Then the discrimination net is traversed by testing whether the current node is a member of this feature set. If it is not, then only the productions associated with the terminal node descendants of that node can possibly be satisfied, so the yes-successor is tested. If, on the other hand, the current node is a member of the feature set, no information has been gained; to discover which productions (if any) can be excluded, the no-successor is tested. When a terminal node is encountered, the list of possibly satisfied productions associated with that node becomes the P+ set. Schematically we have:

```
for every M add PF of M to WM FEATURE SET
for I ← 1 thru N where N is depth of NET
   if NODE N in WM FEATURE SET
     then descend to the NO-SUCCESSOR
     else descend to the YES-SUCCESSOR
P+ SET ← productions associated with terminal node
```

A Condition-membership Memory-support Filter

We have implemented three versions of a second filter which makes use of both the Condition-membership and the Memory-support KSs. The three versions differ from one another only in the set of features they use to represent condition elements. Since they use both the Condition-membership and the Memory-support KSs, we call them CmMsF1, CmMsF2, and CmMsF3.

CmMsF1, like CmF, uses only the primary feature to represent a condition element. At the beginning of a run a set of associations are set up, as illustrated in Figure 2b. Each primary feature is associated with the list of those condition elements which it represents; then each condition element is associated with the list of productions containing it. In addition, the number of condition elements contained in

2 The reordering procedure is useful only for partial nets; it guarantees that the net produced is the most effective net of its size.

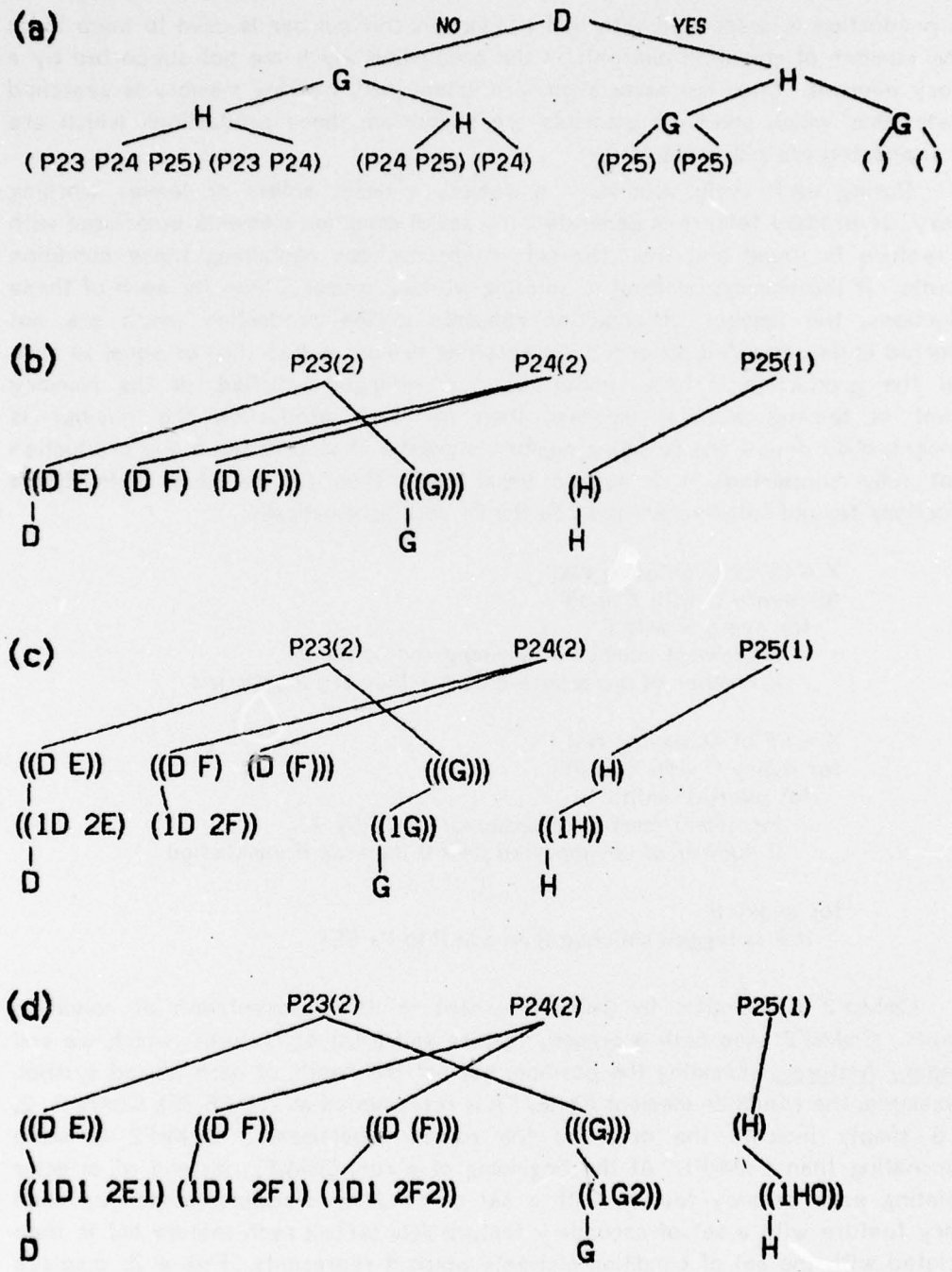


FIGURE 2. REPRESENTATION OF P23-P25 FOR (a) CmF, (b) CmMsF1, (c) CmMsF2, AND (d) CmMsF3

each production is associated with that production; this number is used to keep track of the number of condition elements in the production which are not supported by a memory element. Once the associations are established, working memory is searched to determine which condition elements are supported; those productions which are fully supported are put in the P+ set.

During each cycle, whenever a memory element enters or leaves working memory, its primary feature is generated; the set of condition elements associated with this feature is found and then the sets of productions containing these condition elements. If the memory element is entering working memory, then for each of these productions, the number of condition elements in the production which are not supported is decremented by one; if the resulting number is less than or equal to zero (ie, if the production is fully supported), it is retagged satisfied. If the memory element is leaving working memory, then for each production the number is incremented by one; if the resulting number is greater than zero (ie, if the production is not fully supported), it is tagged unsatisfied. Then before each cycle, those productions tagged satisfied are put into the P+ set. Schematically,

```
X ← PF of M entering WM
for every C with X as PF
   for every P with C
      decrement number of unsupported Cs by 1
      if number of unsupported Cs < 1 then tag P satisfied

X ← PF of M leaving WM
for every C with X as PF
   for every P with C
      increment number of unsupported Cs by 1
      if number of unsupported Cs > 0 then tag P unsatisfied

for every P
   if P is tagged satisfied then add P to P+ SET
```

CmMsF2 is identical to CmMsF1, except in its representation of condition elements. CmMsF2 uses both a primary feature and a list of features, which we call secondary features, indicating the position, but not the depth, of each named symbol. For example, the condition element (D ((E) F)) is represented as (1D 2E 3F), where 1, 2, and 3 simply indicate the order of the named subelements. CmMsF2 is more discriminating than CmMsF1. At the beginning of a run, CmMsF2, instead of directly associating each primary feature with a set of condition elements, associates each primary feature with a set of secondary feature lists (SFLs); each feature list is then associated with the set of condition elements which it represents. Figure 2c displays these indirect associations. During each cycle, after the primary feature of the memory element entering or leaving working memory has been generated, the list of secondary features of the memory element is generated; the secondary feature lists associated with the primary feature are then matched with the list of secondary features of the memory element. A condition element feature list matches a memory

element feature list if all of its elements are contained in the memory element feature list and if they are in the same order. A feature which represents a variable is never bound and thus can always match any element. The productions containing the condition elements represented by the secondary feature lists which matched successfully are updated as in CmMsF1.

```
X ← PF of M entering WM
Y ← SFL of M entering WM
for every SFL associated with X
  if match(SFL, Y) then add associated C to CSET
for every C in CSET
   for every P with C
      decrement number of unsupported Cs by 1
      if number of unsupported Cs < 1 then tag P satisfied

X ← PF of M leaving WM
Y ← SFL of M leaving WM
for every SFL associated with X
  if match(SFL, Y) then add associated C to CSET
for every C in CSET
   for every P with C
      increment number of unsupported Cs by 1
      if number of unsupported Cs > 0 then tag P unsatisfied

for every P
   if P is tagged satisfied then add P to P+ SET
```

CmMsF3 uses both a primary feature and a list of secondary features indicating the depth as well as the position of each named symbol to represent condition elements. For example the condition element (D ((E) F)) is represented as (1D1 2E3 3F2), where the numbers following the named subelements indicate the depth of that subelement. While CmMsF3 is more discriminating than CmMsF2, it differs from CmMsF2 only in the procedure it uses to match the secondary feature lists with the list of secondary features of the memory element entering or leaving working memory. Figure 2d illustrates the associations used by CmMsF3.

III. DERIVATION OF COST FORMULAS

In this section we give formulas that predict the cost of PSG and PSG augmented by the various filters. The formulas arise from an analysis of the structure of the algorithm (as embedded in the actual code) coupled with various simplifying statistical assumptions about the nature of the task environment. We are primarily interested in assessing the major factors affecting the cost of a cycle, T.CYCLE. This could be accomplished by using the formulas to argue for the <u>form</u> of the relationship and then verifying this with experimental data, determining the constants so as to

produce a good fit to the data. Alternatively, the formulas can be made sufficiently precise to produce absolute predictions of the time taken under various experimental conditions. We have chosen this later course, since it is an interesting side question how good such cost formulas can be.

It should be evident from our description of PSG's basic algorithm that two important factors determining the cost of PSG when it is not augmented by a filter are the size of production memory and the size of working memory. This shows up in the simplest approximation to the actual per cycle cost of PSG, which is

(1) T.CYCLE = CONDITION COST + ACTION COST
= (P * M * T.MCH) + ((A/P) * T.ACT)

P is the number of productions in production memory
M is the number of elements in working memory
T.MCH is the time to match a condition element with a memory element
A is the number of action elements
A/P is the average number of actions per production
T.ACT is the time to execute one action element.

This formula actually yields a value which is close to a lower bound for the per cycle cost of a basic implementation for a production system with memory order for conflict resolution. If the number of condition elements in each production is close to 1, or if the number of first condition elements which are supported by elements in working memory is close to 0, then this formula is a fair approximation.

For production systems whose conflict resolution strategy is production order, a better approximation is

(2) T.CYCLE = (((P + 1)/(CS + 1)) * M * T.MCH) + ((A/P) * T.ACT)

CS is the average number of productions in the conflict set

This takes into account that the system quits as soon as it finds the first acceptable production, so that it searches only (P+1)/(CS+1) productions instead of P.

These two formulas make clear that the basic algorithm is linear jointly in P and M, with either conflict resolution regime. The cost addition of the action side will be negligible for large systems (either large P or M), as long as the number of actions per production remains modest (which is true of all systems envisioned to date, since the action sequence is required to be unconditional, hence is basically limited). Though most architectures opt for small or modest working memory (M from 5 to 30), production systems to be effective must have a large P (thousands or even much more). Though production order (formula (2)) effectively divides P by (CS + 1), the size of the conflict set is not under deliberate control and it is unclear whether a production system with very large CS would be viable, since it implies substantial lack of directionality.

When a filter is added to PSG, the per cycle cost can be approximated as

(3) T.CYCLE = T.FILTER + (P' * M * T.MCH) + ((A/P) * T.ACT)

P' is the number of productions in the P+ set which are tested
For production order, P' = (P+ + 1)/(CS + 1)
For memory order, P' = P+

The main effect of the filter according to formula (3) is to cut down the effective number of productions, leaving everything else unchanged, and adding a cost due to the filter which is unknown until specific designs are considered.

Formulas (1)-(3) have some serious weaknesses. They do not take into account the number of supported condition elements which are likely to be tested. Formula (3) is particularly weak since it does not take into account the possibility that, besides decreasing the number of productions, the filter may also decrease the number of memory elements with which a condition element is matched. A more adequate formula, then, is the following:

(4) T.CYCLE=
T.FILTER +
(MCH+P- * T.MCH+) + (MCH-P- * T.MCH-) +
(MCH+P+ * T.MCH+) + (MCH-P+ * T.MCH-) +
(A/P) * T.ACT

The second and third lines of formula (4) separate the cost of processing productions that are not satisfied from the cost of processing satisfied productions; within each line, the cost of matching condition elements with their supporting memory elements is separated from the cost of attempting to match condition elements with memory elements which do not support them. Thus:

MCH+P-	The number of successful matches on Ps that fail
MCH-P-	The number of unsuccessful matches on Ps that fail
MCH+P+	The number of successful matches on Ps that succeed
MCH-P+	The number of unsuccessful matches on Ps that succeed

The adequacy of (4) depends on the closeness with which MCH+P-, MCH-P-, MCH+P+, and MCH-P+ can be approximated in terms of basic characteristics of production systems. These values can be estimated using the following information:

P	The number of productions in production memory
CS	The average number of productions in the conflict set
M	The average number of elements in working memory
C	The number of condition elements
CD	The number of distinct, non ABS condition elements
C.ABS	The number of ABS condition elements
PF	The number of distinct primary features
SFL	The number of distinct feature lists
F	The number of features (named symbols)
FV	The number of features which are free variables
LPF	The average depth of the primary feature
NTC	The number of NTCs
A	The number of action elements (excluding NTCs)

All but one of these quantities are static; that is, they can be determined without running the production system. The other, CS, though it can be computed exactly only at the end of a run, can be estimated with some accuracy by the designer of the production system since the capabilities of a production system are to a considerable extent a function of the size of its conflict set. NTC is one of the commands which can occur in an action element. Its function is to find an element in working memory which matches its argument; thus we include the cost of NTC as part of the condition cost.

The estimated-cost formula for PSG using this information is given below. The formula reflects accurately the structure of the algorithm down to the level of the matching routines. Thus, each time constant corresponds to a particular isolated subpiece of code. Each routine was timed independently by running loops involving only that piece of code. All times are in milliseconds (ms). [3]

(5) T.PSG =

```
    0 +
    P'- * (1/(CD - 1)) * T.MCH+ +
    P'- * (((1/(CD - 1)) * ((M - 1)/2)) + M) * T.MCH- +
    P'+ * (((C - C.ABS) + NTC)/P) * T.MCH+ +
    P'+ * (((((C - C.ABS) + NTC)/P) * ((M - 1)/2)) + ((C.ABS/P) * M)) * T.MCH- +
    (A/P) * T.ACT
```

P'- is the number of unsatisfied productions tested:
For production order, P'- = (P - CS)/(CS + 1).
For memory order, P'- = P - CS.
T.MCH+ is the time it takes to match a supported C with its supporting M:
T.MCH+ = (LPF * 28) + (((F - FV)/C) * 6) + ((FV/C) * 18) + 15.
T.MCH- is the time it takes to attempt to match a C with an M which does not support it:
T1.MCH- = (LPF * 28) + 15.
P'+ is the number of satisfied productions tested:
For production order, P'+ = 1.
For memory order, P'+ = CS.
T.ACT is the time it takes to execute one action element:
T.ACT = 72.

In the second line of (5), MCH+P- is approximated by taking the product of the number of unsatisfied productions tested and the number of supported condition elements in each production before the first unsupported one is found. In the third line, MCH-P- is approximated by taking the sum of two products: the first gives the number of unsuccessful matches which occurred in conjunction with the condition elements which were found to be supported; the second product is simply the number of unsatisfied productions tested multiplied by the number of elements in working memory. In the fourth line, MCH+P+ is approximated by multiplying the number of satisfied productions tested by the number of non-ABS condition elements (including NTCs) per production.

---

3 The version of PSG which we used is coded in the list processing, implementation language L*(H) and was run on a PDP-10, model KA. The system was running in a basically interpretive mode and no attempt was made to optimize the code locally.

In the fifth line, MCH-P+ is approximated by taking this same product and multiplying it by the number of elements in working memory with which a match was attempted before a successful match was found; then the number of ABS condition elements per production times the number of memory elements in working memory is added to that product. In the sixth line, the action cost is estimated by multiplying the average number of action elements in each production by the cost of executing an action element.

The estimated-cost formula for PSG.CmF is

(6) T.PSG.CmF =
M * (T.SET + (PF * T.NET)) +
P'- * (1/((P+ * (C/P)) - 1)) * T.MCH+ +
P'- * (((1/((P+ * (C/P)) - 1)) * ((M - 1)/2)) + M) * T.MCH- +
P'+ * (((C - C.ABS) + NTC)/P) * T.MCH+ +
P'+ * (((((C - C.ABS) + NTC)/P) * ((M - 1)/2)) + ((C.ABS/P) * M)) * T.MCH- +
(A/P) * T.ACT

T.SET is the time it takes to generate the PF of one M:
T.SET = 10.
T.NET is the time it takes to test whether the PF of a C is the same as the PF of one M:
T.NET = 2.
P'- is the number of unsatisfied productions tested:
For production order, P'- = (P+ - CS)/(CS + 1).
For memory order, P'- = P+ - CS.
P+ is the number of productions admitted to further testing:
P+ = CS * (((C - C.ABS)/PF) ↑ (P/(C - C.ABS))).
P'+ is the number of satisfied productions tested:
For production order, P'+ = 1.
For memory order, P'+ = CS.

In the first line of (6), T.FILTER is approximated by multiplying the time it takes to generate the PF of one M plus the time it takes to search the net given one M by the number of elements in working memory. The second and third lines of this formula are the same as the second and third lines of formula (5) except P+ is substituted for P in P'- and the probability that a C (in a production in the P+ set) matches an M is 1/(P+ * (C/P)) instead of 1/CD. P+ is approximated by multiplying CS by the average number of condition elements that are the same with respect to their primary feature raised to the inverse of the average number of condition elements in each P. The remaining lines of (6) are identical to the fourth, fifth, and sixth lines of (5).

PSG.CmF is useful as a partial filter only when some subset of features, PRTF, essentially partitions the production system into a number of sets equal to the number of features in that subset. When this situation obtains, the cost formula is the same as (6), except that P+ is (1/PRTF) * P and the filter cost is M * (T.SET + (PRTF * T.NET)).

The estimated-cost formula for PSG.CmMsF is

(7) T.PSG.CmMsF =
((A/P) * (C/F') * LPF * (T.PFT + (K * T.FLT)) +
P'- * (1/((P+ * (C/P)) - 1)) * T.MCH+ +
P'- * (((C - C.ABS)/C) + ((C.ABS/C) * (M/2))) * T.MCH- +
1 * (((C -C.ABS) + NTC)/P) * T.MCH+ +
1 * ((C.ABS/P) * M) * T.MCH- +
(A/P) * T.ACT

F' is the number of distinct primary features or distinct feature lists:
For PSG.CmMsF1, F' = PF.
For PSG.CmMsF2 and PSG.CmMsF3, F' = SFL.
K is the factor by which T.FLT is multiplied depending on which version of CmMsF is being used
For PSG.CmMsF1, K = 0.
For PSG.CmMsF2, K = 1.
For PSG.CmMsF3, K = 2.
T.PFT is the time it takes PSG.CmMsF1 to keep track of which Cs have a primary feature in working memory:
T.PFT = 30.
T.FLT is the time it takes PSG.CmMsF2 to keep track of which Cs have a feature list in working memory:
T.FLT = 78.
P'- is the number of unsatisfied productions tested:
P'- = (P+ - CS)/(CS + 1)
P+ is the number of productions admitted to further testing:
P+ = CS * (((C - C.ABS)/F') ↑ (P/(C - C.ABS))).
T.MCH- is the time it takes to attempt to match a C with an M which does not support it:
T.MCH- = (LPF * 28) + 21.

In the first line of formula (7), T.FILTER is approximated by multiplying the number of modifications to working memory by the number of condition elements associated with each distinct primary feature or distinct feature list and that by the average depth of the primary feature and then multiplying this product by the time it takes to keep track of which condition elements are at least partially supported by which memory elements. The second and third lines of this formula are the same as the second and third lines of (6), except: P'- is always (P+ - CS)/(CS + 1); each C is matched with approximately one M rather than with half of the elements of working memory for supported Cs and all of the elements of working memory for unsupported Cs; and the time it takes to attempt to match a C with a non-supporting M is slightly greater since no C is matched with an M which does not have at least the same primary feature. The fourth and fifth lines of the formula are the same as the fourth and fifth lines of (6), except: P'+ is always 1; and each non ABS condition element is matched with just one memory element. The sixth lines of the two formulas are identical.

If the formulas (5)-(7) are accurate, then we can make the following general statements about the efficiency of production system architectures: (1) An architecture which does not make use of a filter has a cost which is roughly linear in the product of P and M. Each unsatisfied production tested involves at least M matches. An additional

N * (M/2) matches will be needed for each unsatisfied production whose first N condition elements are supported by memory elements; however, for large P and small M, this number will be insignificant. The number of matches needed to test a satisfied production is C/P * ((M - 1)/2 + 1); this number likewise will be insignificant for large P unless M (or CS) is correspondingly large. (2) A production system architecture augmented by a filter which makes use of a Condition-membership KS has a cost which is roughly linear in the product of P+ and M. The number of matches needed to test an unsatisfied production will ordinarily be higher for a system augmented by a filter since presumably the productions in the P+ set contain a larger number of supported condition elements than do the productions not in the P+ set. The number of matches needed to test a satisfied production is the same as for an architecture not augmented by a filter. Because production systems typically have many productions, but only a small working memory, the Condition-membership KS can be viewed as the most crucial of the knowledge sources since it eliminates the cost dependency on P. (3) An architecture augmented by a filter which makes use of both the Memory-support KS and the Condition-membership KS has a cost which is linear only in P+. Since each non-negated condition element is associated with those memory elements that appear to support it, often only one match will be involved in testing an unsatisfied production, and only rarely will more than C/P matches be necessary. The number of matches needed to test a satisfied production is just C/P. By providing information which makes the searching of working memory unnecessary, the Memory-support KS eliminates the cost dependency on M.

IV. EXPERIMENTAL RESULTS

This section is divided into three parts. Each part considers one of the three production systems we used for gathering cost data, and includes a brief description of the production system involved, a table displaying the predicted and measured costs for the runs, with and without filtering, and a discussion of the discrepancies between the predicted and measured costs.

PSCC1: Testing Variations in Processing

The first set of cost data we collected was for the purpose of testing the reliability of our estimated-cost formulas. We wanted to determine whether the discrepancies between the measured and predicted costs of a production system would vary if the number of productions and the size of working memory were held constant, but the stimuli presented to the production system were varied. We reasoned that if the discrepancies between measured and predicted costs did not remain relatively constant, then the cost of a production system would be too highly task dependent for the formulas to be considered meaningful. The production system we selected for this test, PSCC1, was designed to simulate the performance of human subjects on a task described by Chase and Clark [1972]. They used this task to study the mental processes people use in deciding whether of not a sentence is an accurate description of a picture.

The task itself involves presenting a subject with (1) one of of the eight sentences which can be formed by using star or plus as subject, is or isn't as verb, above or below as preposition, and then plus or star (whichever was not used as subject) as the object of the preposition, and (2) a picture of a * and a +, one above the other. The subject then indicates whether the sentence is true of the picture. For instance, star is above plus and */+ is presented; and the subject says yes.

We presented three of the 16 possible sentence-picture combinations to PSCC1. For each set of inputs, runs on PSG, PSG.CmMsF1, PSG.CmMsF2, and PSG.CmMsF3 were made. PSG.CmF was not used. Given the number of productions in PSCC1, space limitations made the use of the full CmF impossible, and the diversity of features in PSCC1 was such that no partial CmF was more than minimally discriminatory. PSCC1 has the following parameters:

	Run 1	Run 2	Run 3
P	40	40	40
CS	3.09	2.93	2.91
M	16	16	16
C	65	65	65
CD	47	47	47
C.ABS	6	6	6
PF	25	25	25
F	105	105	105
FV	37	37	37
LPF	.77	.77	.77
NTC	20	20	20
A	81	81	81

All the static parameters are identical; only CS varies slightly for each run. For CmMsF2's representation of condition elements, SFL = 43, for CmMsF3's representation, SFL = 45. PSCC1 uses production order for conflict resolution; thus, only one (the first) satisfied production is found, not the entire conflict set.

Although there are, as Table 1 shows, significant discrepancies between the predicted and measured costs for this task, the discrepancies remain constant across all of the runs and are due to some of the distribution assumptions which we made, not to an incorrect characterization of the interdependencies among the production system parameters.

The measured costs for each of the PSSC1 runs on PSG are greater than our predicted costs by about a factor of 1.5. This discrepancy can be accounted for almost entirely by our inability to estimate accurately the number of productions which need to be considered before the first satisfied production is found. The formula which we use to estimate this number, (P - CS)/(CS + 1), presupposes that the satisfied productions are evenly distributed among the productions in production memory and thus that roughly 1/CS productions will be tried in each cycle. In PSCC1, however, one production, the thirty-fourth, is fired far more frequently than any of the others, and consequently, on the average, 19 productions are tried before the first satisfied one is found. Since production systems could be written containing a production functionally equivalent to the thirty-fourth, but either much closer to the front or to the end of production memory, an accurate prediction of the number of

	Stimulus Set 1		Stimulus Set 2		Stimulus Set 3	
	predicted	measured	predicted	measured	predicted	measured
PSG						
Filter cost	0	0	0	0	0	0
Condition cost	5988	9423	6230	9328	6261	9389
P.considered	10.02	19.18	10.43	18.47	10.48	18.47
C.considered	10.84	21.52	11.26	20.81	11.31	20.82
MCH+P-	.20	3.18	.21	3.16	.21	3.11
MCH-P-	145.83	285.93	152.45	275.95	153.27	277.07
MCH+P+	1.48	1.30	1.48	1.33	1.48	1.33
MCH-P+	13.46	5.23	13.46	4.95	13.46	4.98
Action cost	145	170	145	183	145	179
Total cost	6134	9593	6376	9512	6406	9569
PSG.CmMsF1						
Filter cost	121	139	121	160	121	151
Condition cost	252	948	252	1093	252	1118
P+	5.53	7.80	5.24	7.81	5.21	7.76
P.considered	1.60	2.82	1.59	2.84	1.59	2.80
C.considered	2.30	4.73	2.29	4.74	2.29	4.69
MCH+P-	.07	2.75	.08	2.72	.08	2.64
MCH-P-	.98	8.23	.97	9.07	.97	9.47
MCH+P+	1.48	1.30	1.48	1.33	1.48	1.33
MCH-P+	2.40	3.09	2.40	2.74	2.40	2.71
Action cost	145	173	145	198	145	199
Total cost	520	1262	519	1452	519	1469
PSG.CmMsF2						
Filter cost	254	296	254	307	254	338
Condition cost	221	928	221	930	221	967
P+	3.83	5.78	3.63	5.60	3.61	5.68
P.considered	1.18	2.52	1.18	2.51	1.18	2.49
C.considered	1.84	4.39	1.84	4.37	1.84	4.33
MCH+P-	.03	2.78	.04	2.67	.04	2.60
MCH-P-	.30	7.66	.29	8.23	.29	8.67
MCH+P+	1.48	1.30	1.48	1.33	1.48	1.33
MCH-P+	2.40	3.07	2.40	2.74	2.40	2.71
Action cost	145	186	145	194	145	197
Total cost	621	1411	621	1432	621	1502
PSG.CmMsF3						
Filter cost	418	463	418	491	418	498
Condition cost	219	929	219	922	219	952
P+	3.71	5.52	3.52	5.42	3.50	5.42
P.considered	1.15	2.43	1.15	2.42	1.15	2.40
C.considered	1.81	4.30	1.81	4.28	1.81	4.24
MCH+P-	.03	2.78	.03	2.67	.03	2.60
MCH-P-	.25	7.57	.25	8.14	.25	8.58
MCH+P+	1.48	1.30	1.48	1.33	1.48	1.33
MCH-P+	2.40	3.07	2.40	2.74	2.40	2.71
Action cost	145	192	145	197	145	197
Total cost	783	1585	783	1610	783	1648

Table 1. Per cycle times (in milliseconds) and counts for PSCC1

productions which will be tried in each cycle is impossible to obtain without a fairly clear idea of the intended role of each production. If it had been known that 19 productions would be considered in each cycle, then our formula would have yielded a far more accurate estimate of the cost. In the first run, for example, the predicted number of condition elements considered would have been 20.23, MCH-P- would have been 292.25, and the condition cost would have been 10540 ms.

The other significant discrepancies in the PSG runs are in MCH+P-, MCH-P+, and T.MCH-. Again, using the first run as an example, if P.considered had been accurately estimated, MCH+P- would have been .38. Of the six productions containing ABS condition elements, three are designed in such a way that the ABS element (which is the second element in all three productions) is the principal discriminator; that is, the first condition element in each of these three productions matched a memory element .64 times per cycle. If this had been taken into account as well, then MCH+P- would have been 2.30. MCH-P+ is inaccurately predicted because our formula presupposes that, on the average, each condition element will match the middle element in working memory, which in the case of PSSC1 is the seventh or eighth. Had we assumed instead that the fourth or fifth memory element would be the supporting element, then MCH-P+ would have been approximately 5. The actual cost of matching a condition element in PSSC1 with a memory element which does not support it is approximately 32 ms; our estimate was 36.6 ms. To predict this cost more accurately, one would have to collect relatively complete information about the make-up of each condition element.

The discrepancies between the predicted and measured costs in PSG.CmMsF1, PSG.CmMsF2, and PSG.CmMsF3 are primarily due to our inability to estimate accurately the size of the P+ set. Our estimate is low for all three of the runs with each of the three filters because our PSG.CmMsF formula is built on the assumption that ABS condition elements can be ignored when trying to determine the probability that any given production will be satisfied. This assumption is, of course, simplistic, since as we saw in the discussion of the PSG discrepancies above, there is a production writing style in which an ABS condition element is the principal discriminator in a production. In the case of the first PSG.CmMsF1 run, for example, an average of three productions containing an ABS condition element are in the P+ set in each cycle. That is, 39% of the productions which contain an ABS condition element are, on the basis of the information in the KSs, possibly satisfied. However, only 8% of the productions not containing an ABS condition element are in the P+ set. If our formula were modified so that it could take this additional information into account, the estimated size of the P+ set would be 7.1. The only other significant discrepancies between the predicted and measured costs in the PSG.CmMsF runs are with MCH+P- and MCH-P-. Here, as well, the difference arises because of the ABS condition elements. One production (as it happens, the first production in production memory) causes most of the trouble. This production has two condition elements: In the case of the first PSG.CmMsF1 run, the first of these is supported by a memory element 87% of the time; the second condition element is an ABS condition element. The production itself is satisfied only 5% of the time. Thus, .82 times per cycle, two condition elements in an unsatisfied production are found to be supported, and so taking just this one production into account, MCH+P-

would be 1.64. This same production accounts to a considerable extent for the MCH-P- discrepancy. If our formula had assumed that half of the elements in working memory would be examined before each of the two supporting memory elements were found, the predicted value of MCH-P- would have been 12.3.

PS20Q: Testing Variations in the Number of Productions

Our purpose in gathering the second set of cost data was to determine the relationship between the number of productions in production memory and the cost of processing a production system, both with and without a filter. In order to measure the effect of varying production memory size, we used a production system containing productions almost all of which are of the same form. The production system, PS20Q, plays the game of twenty questions. Except for three control productions, each production in the system is a member of a three production set which encodes information about a category. One production in the set associates a category with a sub-category. Another associates the category with another category within the same super-category. The third provides a mechanism by which the production system can acquire knowledge of additional sub-categories. For example, the following information about the category citrus fruit might be represented: (1) A lime is a citrus fruit. (2) A banana is another kind of fruit. (3) If there is another citrus fruit besides the lime, then it should be associated with the lime. Because information about each category is encoded in a homogeneous fashion, the values of the production system parameters vary directly with the number of productions.

The first run of PS20Q was with 50 productions; before each of the two subsequent runs, an additional 50 productions were added to production memory. In all three cases, the production system (unbeknownst to itself) was looking for the same answer. For each set of productions, runs on PSG, PSG.CmF, PSG.CmMsF1, and PSG.CmMsF2 were made. PSG.CmMsF3 was not used since PSG.CmMsF2 was fully discriminating; that is, the P+ set was the conflict set. The PSG.CmF runs were with a partial filter; only four features were used. PS20Q has the following parameters:

	Run 1	Run 2	Run 3
P	50	100	150
CS	1.13	1.13	1.13
M	7	7	7
C	160	322	485
CD	45	76	113
C.ABS	0	0	0
PF	29	48	72
SFL	44	75	112
F	197	405	620
FV	0	0	0
LPF	.28	.30	.30
NTC	57	105	157
A	413	787	1163

PS20Q uses working memory order for conflict resolution; thus in this case, PSG finds all satisfied productions before selecting one to fire.

As is evident from Table 2, the predicted costs for the PS20Q runs are much closer to the measured costs than was the case in the previous task. This is due

	50 Productions predicted	50 Productions measured	100 Productions predicted	100 Productions measured	150 Productions predicted	150 Productions measured
PSG						
Filter cost	0	0	0	0	0	0
Condition cost	8407	9422	16816	18880	25009	27269
P.considered	50.00	50.00	100.00	100.00	150.00	150.00
C.considered	53.60	60.96	103.83	117.13	153.85	173.17
MCH+P-	1.11	7.17	1.32	13.35	1.33	19.39
MCH-P-	345.42	393.13	696.04	799.22	1046.07	1205.17
MCH+P+	3.62	2.52	3.64	2.52	3.66	2.52
MCH-P+	10.85	1.35	10.92	1.35	10.97	1.35
Action cost	594	450	566	454	558	443
Total cost	9002	9872	17383	19334	25567	27713
PSG.CmF						
Filter cost	126	122	126	130	126	141
Condition cost	2331	2266	4428	4615	6477	6681
P+	12.50	11.39	25.00	21.65	37.50	32.13
P.considered	12.50	11.39	25.00	21.65	37.50	32.13
C.considered	15.28	22.17	27.81	38.61	40.33	55.13
MCH+P-	.29	7.00	.30	13.17	.30	19.22
MCH-P-	80.46	84.30	167.99	172.48	255.49	262.26
MCH+P+	3.62	2.52	3.64	2.52	3.66	2.52
MCH-P+	10.85	1.35	10.92	1.35	10.97	1.35
Action cost	594	422	566	448	558	448
Total cost	3051	2811	5121	5194	7162	7122
PSG.CmMsF1						
Filter cost	382	252	475	586	478	758
Condition cost	144	250	147	324	147	377
P+	1.93	1.17	2.04	1.26	2.04	1.35
P.considered	1.37	1.04	1.43	1.04	1.43	1.04
C.considered	3.65	2.52	3.72	2.52	3.74	2.52
MCH+P-	.07	.09	.08	.09	.08	.09
MCH-P-	.37	.04	.43	.04	.43	.04
MCH+P+	3.20	2.39	3.22	2.39	3.23	2.39
MCH-P+	.00	.00	.00	.00	.00	.00
Action cost	594	464	566	476	558	478
Total cost	1129	966	1188	1387	1176	1615
PSG.CmMsF2						
Filter cost	988	775	1894	1298	1887	1894
Condition cost	140	251	143	325	143	412
P+	1.69	1.13	1.78	1.13	1.78	1.13
P.considered	1.26	1.00	1.30	1.00	1.30	1.00
C.considered	3.52	2.39	3.59	2.39	3.60	2.39
MCH+P-	.06	.00	.06	.00	.06	.00
MCH-P-	.26	.00	.30	.00	.30	.00
MCH+P+	3.20	2.39	3.22	2.39	3.23	2.39
MCH-P+	.00	.00	.00	.00	.00	.00
Action cost	594	478	566	472	558	479
Total cost	1643	1504	1804	2096	1789	2786

Table 2. Per cycle times (in milliseconds) and counts for PS200

partly to the similarity in form of most of the productions and partly to the use of working memory order rather than production order as the conflict resolution strategy. Figure 3, which plots total cost as a function of the size of production memory, shows very clearly the value of the two filters.

Because PS20Q uses working memory order, there is no difficulty in predicting for PSG the exact number of productions which will be considered in each cycle. However, because of the particular character of the productions in PS20Q, there is a discrepancy between the predicted number of condition elements considered and the measured number. The character of the productions is such that production memory can be partitioned into four subsets; one of these sets contains three productions; the other three sets each contain approximately 1/3 of the remaining productions, and each production in each set has the same first condition element. Since one of these three condition elements is supported every other cycle, we could predict that C.considered would be 60.82 when production memory contained 50 productions, 119.17 when production memory contained 100 productions, and 177.53 when production memory contained 150 productions. Because of these three sets of productions with the same first condition element, there is, of course, a significant discrepancy between the predicted and measured values of MCH+P-; but if this fact were taken into account, the predicted values of MCH+P- for the three runs would be approximately 7.3, 15.6, and 24. The discrepancy in MCH+P+ exists because the three control productions which are fired about half of the time each have only two condition elements, rather than the 3 or 4 of the remaining productions.

The two remaining significant discrepancies are in MCH-P+ and MCH-P-. The MCH-P+ discrepancy in PS20Q is similar to, but more pronounced than, the MCH-P+ discrepancy in PSCC1. Here, as before, the difficulty is with our assumption that on the average, half of the elements in working memory will be considered before the supporting element is found. In the case of PS20Q, only one element is considered. The discrepancy between the predicted and measured values of MCH-P- is due to an implementation feature of PSG which was not discussed above and which is not reflected in our formulas. When the conflict resolution strategy being employed is working memory order, PSG adds a special symbol to working memory to mark the location of the memory element closest to the front of working memory that supports a condition element in a satisfied production. If the number of matches in which this symbol participates were subtracted from the measured value of MCH-P-, the resulting measured value for MCH-P- would be very close to the predicted value; in the first run, for example, the measured value would be 344.26. This adjustment would, in turn, lead to a modification in the measured condition cost of approximately 1100 ms.

The discrepancies in the three runs with the partial CmF are almost identical to the discrepancies in the PSG runs. The one discrepancy which perhaps needs some comment is the MCH-P- discrepancy. If the measured value for the first PSG.CmF run were adjusted as above, MCH-P- would be 74.04 and the measured condition cost would then be approximately 2000 ms. Given these adjustments, the predicted total cost for these three runs would increase slightly.

There are two significant discrepancies in the PSG.CmMsF1 and PSG.CmMsF2

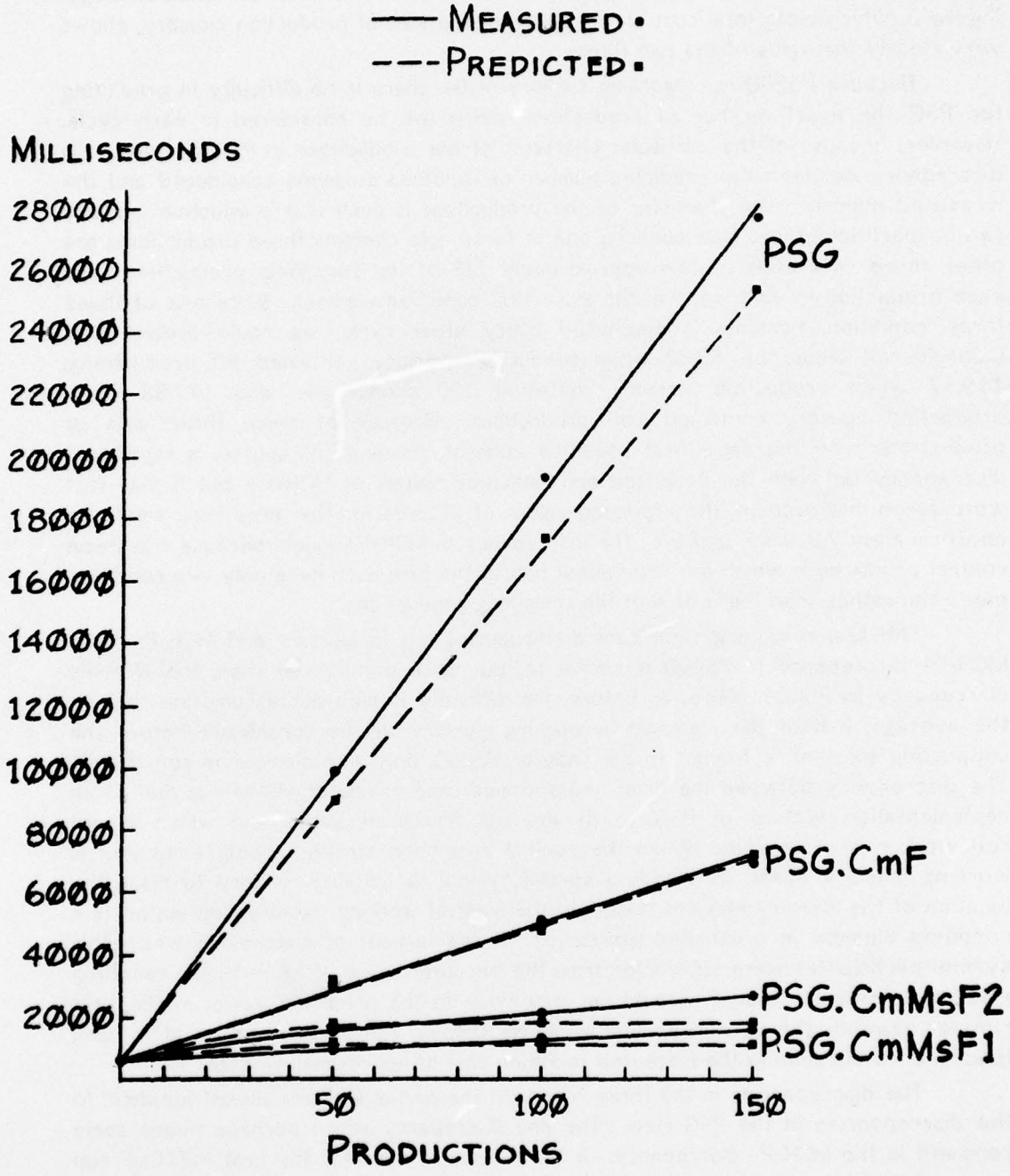


FIGURE 3. PER CYCLE COST OF PS20Q ON PSG, PSG.CmF, PSG.CmMsF1, AND PSG.CmMsF2

runs. In both sets of runs, the measured filter costs and the measured condition costs increase with the size of production memory whereas the predicted costs remain relatively constant. This would suggest that there is a dependency in both cases on the number of productions in production memory -- a dependency which our formula does not capture. In the case of the condition cost, there is such a dependency. The filters have been implemented in such a way that rather than updating the P+ set at the beginning of each cycle, the filter simply updates a flag on each production which indicates whether or not it is satisfied. PSG then generates the P+ set by checking the flag on each production. The cost of this check is approximately 1.5 ms per production. In the case of the filter cost, on the other hand, the dependency is only apparent. The actual dependency is on the number of condition elements associated with each primary feature. In PS20Q, however, the number of condition elements associated with four of the primary features is directly proportional, as we mentioned above, to the number of productions in production memory.

PSSTRN: Testing Variations in the Size of Working Memory

The purpose of the third set of cost data was to determine the relationship between the size of working memory and the cost of processing a production system, both with and without a filter. To determine the effect of working memory size, we used a production system, PSSTRN, which can do three related tasks: (1) It can do the Sternberg binary classification task [Newell, 1973]. In this task, the subject is given a small set of symbols, called the positive set, and then another symbol, called the probe; the task is to determine whether or not the probe is in the positive set. (2) It can determine whether the same probe has been given before. (3) It can look for a pattern in its yes or no answers to the Sternberg task. The effect of working memory size was determined by selecting three different-sized working memories in such a way that the same sequence of productions fired, but a larger number of condition elements (the number being proportionate to the number of memory elements added) were satisfied during each run.

For the first run of PSSTRN, four elements were in working memory, for the second, an additional three, and then three more for the third run. For each working memory, runs on PSG, PSG.CmF, PSG.CmMsF1, and PSG.CmMsF2 were made. As with PS20Q, PSG.CmMsF2 was fully discriminating and so PSG.CmMsF3 was not used. PSSTRN has the following parameters:

	Run 1	Run 2	Run 3
P	24	24	24
CS	2.40	6.50	6.70
M	4	7	10
C	50	50	50
CO	27	27	27
C.ABS	2	2	2
PF	17	17	17
SFL	26	26	26
F	85	85	85
FV	27	27	27
LPF	.79	.79	.79
NTC	28	28	28
A	54	54	54

The predicted costs for the PSG.CmF runs are for the complete 17 level (17 PF) net; however, space limitations made it impossible to use the full filter and so the runs were made with a net of ten levels instead. This partial filter was as discriminating as the full filter would have been and so except for a slight difference in filter cost, our predicted costs are the same as they would have been had we been able to use the full filter. Working memory order was used as the conflict resolution strategy.

As Table 3 shows, the discrepancies between the predicted and measured costs for PSSTRN are not completely consistent across the runs. This is primarily because there is a wide variance in the number of condition elements that each of the memory elements support or appear to support. The effect of this lack of uniformity of support can be seen in Figure 4, where total cost is plotted as a function of working memory size.

In the first PSG run the predicted costs are very close to the actual costs when one takes into account the special symbol added to working memory to effect production selection during conflict resolution. The measured value of MCH-P- would be 85.80 and the adjusted condition cost would be approximately 4500 ms. [4] In the second PSG run the adjusted value of MCH-P- would also be approximately the same as the predicted value, but in the third run the adjusted value would be significantly less than the predicted value. In all three runs the predicted value of MCH+P- is too low because our formula presupposes, as we indicated above, that no condition element is more likely to be supported than any other condition element. However, in PSSTRN, the sets of productions that do each subtask are partially distinguished from one another by control elements (condition elements whose only function is to make the order in which productions are fired easier to control). Since the control elements in this production system are always the first elements in the productions, the first element in each of the productions involved in the subtask being addressed is more likely to be supported than other condition elements. There is a discrepancy in the second and third runs which is not present in the first. In both, the predicted value of MCH+P+ is too high because the average number of condition elements in the productions which are satisfied is much less than the average number of condition elements in the entire set of productions. In the third run, but not in the other two, the predicted value of MCH-P+ is too high. It is clear that since the three memory elements added to working memory did not appreciably affect the size of the conflict set, these three memory elements are minimally involved in supporting condition elements; consequently, the supporting memory element is almost always found before these elements could be matched with a condition element.

The discrepancies in all three runs of PSSTRN on PSG.CmF are for the most part due to our inaccurate estimates of the size of the P+ sets. In both the first and the third run, our estimate is too low for reasons analogous to those given in our discussion of PSCC1. In the second run, our estimate is too high, primarily because

4 This is still too high because in PSSTRN the cost of matching a condition element with a memory element which does not support it is approximately 40 ms, rather than 36.

	4 Memory Elements		7 Memory Elements		10 Memory Elements	
	predicted	measured	predicted	measured	predicted	measured
PSG						
Filter cost	0	0	0	0	0	0
Condition cost	4130	5595	8057	8498	11254	11760
P.considered	24.00	24.00	24.00	24.00	24.00	24.00
C.considered	27.43	30.10	31.71	32.50	31.92	35.80
MCH+P-	.83	3.90	.67	4.10	.67	7.00
MCH-P-	87.65	107.40	124.52	138.90	175.99	209.50
MCH+P+	4.80	3.40	13.00	8.70	13.40	9.30
MCH-P+	8.00	7.10	42.79	41.00	65.88	48.90
Action cost	162	222	162	225	162	226
Total cost	4292	5817	8219	8723	11416	11986
PSG.CmF						
Filter cost	176	117	388	197	448	286
Condition cost	1098	1562	4582	3560	6422	5865
P+	4.03	5.60	10.92	9.60	11.26	12.10
P.considered	4.03	5.60	10.92	9.60	11.26	12.10
C.considered	6.85	8.90	18.17	15.20	18.72	19.90
MCH+P-	.22	1.10	.20	1.10	.20	3.00
MCH-P-	6.86	16.60	31.57	24.70	46.50	70.50
MCH+P+	4.80	3.40	13.00	8.70	13.40	9.30
MCH-P+	8.00	7.10	42.79	41.00	65.88	48.90
Action cost	162	220	162	220	162	218
Total cost	1436	1899	5052	3978	7024	6369
PSG.CmMsF1						
Filter cost	156	106	156	107	156	108
Condition cost	209	386	224	418	237	495
P+	4.03	3.60	10.92	7.70	11.26	11.90
P.considered	1.48	1.40	1.59	1.40	1.59	1.40
C.considered	2.63	2.30	2.70	2.30	2.70	2.30
MCH+P-	.06	.50	.03	.50	.03	.50
MCH-P-	.50	.20	.65	.20	.69	.20
MCH+P+	2.00	1.50	2.00	1.50	2.00	1.50
MCH-P+	.33	.20	.58	.50	.83	.80
Action cost	162	225	162	227	162	227
Total cost	528	718	543	753	556	831
PSG.CmMsF2						
Filter cost	369	206	369	207	369	208
Condition cost	198	356	211	417	222	494
P+	3.26	3.50	8.83	7.60	9.10	11.80
P.considered	1.25	1.40	1.31	1.40	1.31	1.40
C.considered	2.38	2.30	2.41	2.30	2.41	2.30
MCH+P-	.04	.50	.02	.50	.02	.50
MCH-P-	.26	.20	.34	.20	.36	.20
MCH+P+	2.00	1.50	2.00	1.50	2.00	1.50
MCH-P+	.33	.20	.58	.50	.83	.80
Action cost	162	226	162	228	162	228
Total cost	729	789	742	852	754	931

Table 3. Per cycle times (in milliseconds) and counts for PSSTRN

production memory contains two productions which are in the conflict set on almost every cycle, but which are similar to no other productions in production memory. The MCH+P+ and MCH-P+ discrepancies arise, of course, for the reasons given in our discussion of the runs on PSG. Except for the P+ set predictions in the second run on PSG.CmMsF1 and the third run on PSG.CmMsF2, our PSG.CmMsF formula was quite accurate. These two discrepancies, as well as the filter cost discrepancies, arise for reasons given above.

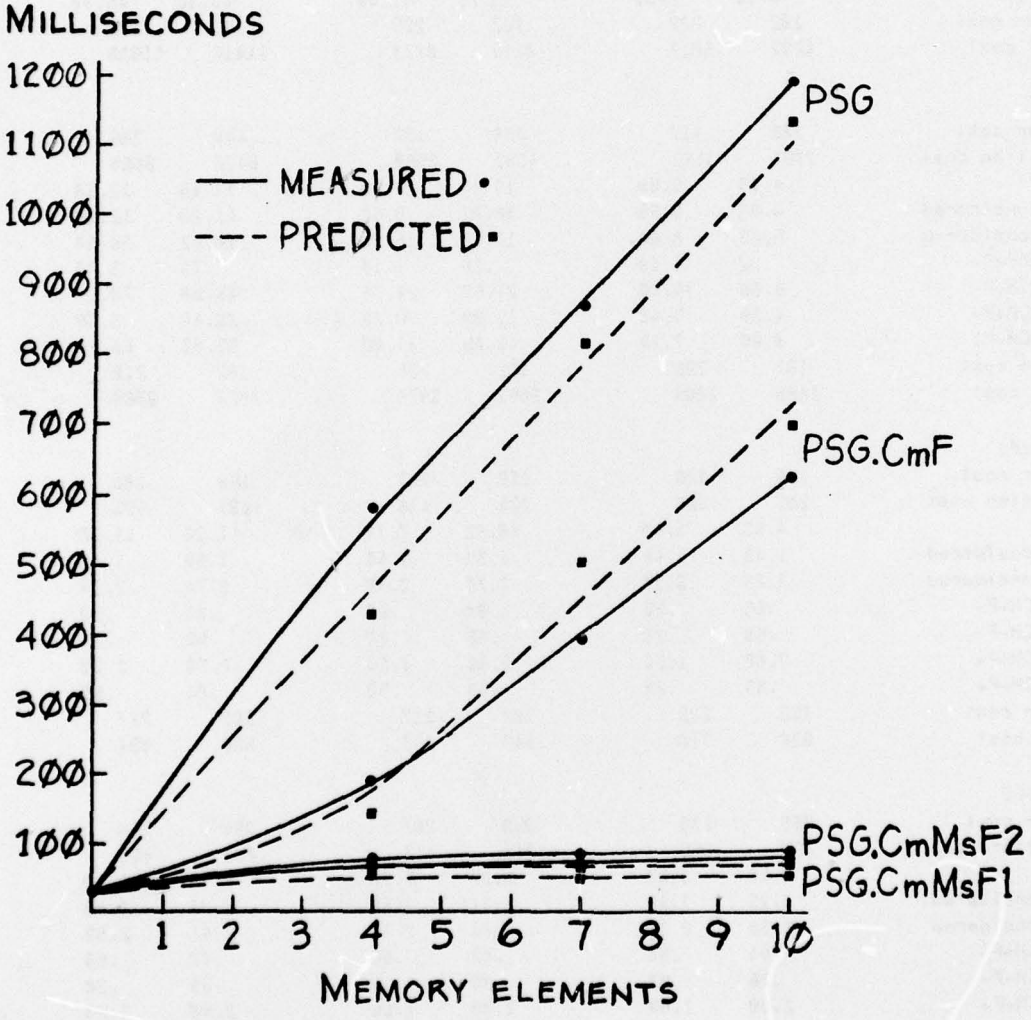


FIGURE 4. PER CYCLE COST OF PSSTRN ON PSG, PSG.CmF, PSG.CmMsF1, AND PSG.CmMsF2

Conclusions

As we stated at the beginning of section III, our purpose in generating the estimated-cost formulas was primarily to provide a means for studying the effect of a variety of production system parameters on the cost of processing a production system. Our discussion of the discrepancies between the predicted and actual costs of processing the three production systems has, we think, shown that in spite of the inadequacy of our various statistical assumptions, our cost formulas do capture the most important dependencies. Though many things are involved, the factor which stands out as the single most important determinant for a production system architecture not augmented by a filter is MCH-P-, the number of attempted matches of a condition element in an unsatisfied production with a memory element which does not support it. It is this factor that our two filters deal with most successfully.

The two graphs (Figures 3 and 4) show very clearly that the cost of running a production system on PSG is almost directly proportional to the product of the number of productions in production memory and the number of elements in working memory. The filter (CmF) which makes use of just the Condition-membership KS eliminates the dependency on the number of productions by making the number of unsatisfied productions tested dependent on CS rather than on P. The filter (CmMsF) which utilizes both the Condition-membership KS and the Memory-support KS eliminates the memory size dependency as well by eliminating the need to search working memory.

CmF and CmMsF both show that large increases in efficiency can be gained by using simple filtering mechanisms. Although a CmF, because it cannot eliminate the working memory dependency, is perhaps not of much use in practical applications, CmMsFs appear to have the necessary power: neither the cost of maintaining the KSs nor the cost of finding the satisfied productions is dependent on production memory or working memory size. Moreover, CmMsFs are highly effective even when the information in the Condition-membership and Memory-support KSs is very incomplete, as is the case with CmMsF1.

REFERENCES

Chase, W. and Clark, H. Mental operations in the comparison of sentences and pictures. In Gregg, L. (ed), **Cognition in Learning and Memory**. John Wiley and Sons, 1972, pp. 205-232.

Davis, R. and King, J. An overview of production systems. **Technical report.** Department of Computer Science, Stanford University, 1976.

Feigenbaum, E., Buchanan, B., and Lederberg, J. On generality and problem solving: a case study using the DENDRAL program. In B. Meltzer and D. Michie (eds), **Machine Intelligence 6**. Edinburgh University Press, 1971, pp. 165-190.

Forgy, C. A production system monitor for parallel computers. **Technical Report.** Department of Computer Science, Carnegie-Mellon University, 1976 (forthcoming).

Newell, A. Production systems: models of control structures. In Chase, W. (ed), **Visual Information Processing**. Academic Press, 1973, pp. 463-526.

Newell, A. and McDermott, J. PSG manual. Department of Computer Science, Carnegie-Mellon University, 1975.

Newell, A. and Simon, H. **Human Problem Solving**. Prentice-Hall, 1972.