General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

NAS 12 - 2144

R-520-NASA
September 1970

COMPUTER SYSTEMS ANALYSIS METHODOLOGY: STUDIES IN MEASURING, EVALUATING, AND SIMULATING COMPUTER SYSTEMS

B. W. Boehm

A Report prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Rand
SANTA MONICA, CA. 90406

08

NAS 12 - 2144

R-520-NASA
September 1970

COMPUTER SYSTEMS ANALYSIS METHODOLOGY: STUDIES IN MEASURING, EVALUATING, AND SIMULATING COMPUTER SYSTEMS

B. W. Boehm

A Report prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Rand
SANTA MONICA, CA. 90406

PREFACE

This Report summarizes the results of a group of computer system analysis and simulation studies performed under a research contract with the National Aeronautics and Space Administration (NASA). It is basically the text of a briefing presented at four NASA centers (Goddard, Langley, Marshall, and MSC-Houston) in April 1970. Detailed results of the studies are reported in the following publications [1-4]:

Nielsen, N. R., *ECSS: Extendable Computer System Simulator*, The RAND Corporation, RM-6132-NASA, January 1970.

Bell, T. E., *Computer System Measurement and Analysis*, The RAND Corporation, R-584-NASA/PR (in process).

Watson, R. A., *Measurement and Analysis of Computer System Performance: Applications of Accounting Data*, The RAND Corporation, R-573-NASA/PR (in process).

Seven, M. J., B. W. Boehm, and R. A. Watson, *Problem-Solving with an Interactive Computer: A Study of User Behavior*, The RAND Corporation, R-513-NASA (in process).

This Report concentrates on the key results of individual studies and their applicability to the design, development, and management of complex computer systems. It should be of interest to planners, managers, and analysts of both complex computer systems and the mission-oriented systems in which they are embedded.

SUMMARY

This Report summarizes the results of a group of computer systems analysis and simulation studies performed under a NASA research contract. It represents the text of a briefing presented at four NASA centers (Goddard, Langley, Marshall, and MSC-Houston) in April 1970. The discussion, which concentrates on the key results of individual studies and their applicability to the design, development, and management of complex computer systems, should be of interest to planners, managers, and analysts of both complex computer systems and mission-oriented systems.

The studies were coordinated into three major efforts:

1) The development of a set of design principles for languages to model and simulate computer systems by attempting to design and implement a prototype language called ECSS (Extendable Computer System Simulator).
2) The evaluation and extension of the state-of-the-art of using software and hardware devices to measure and analyze the performance of complex computer systems.
3) The provision of greater perspectives on the relationship between computer performance and man-computer performance by performing and analyzing controlled experiments in man-computer problem-solving.

ECSS

The Extendable Computer System Simulator (ECSS) is a prototype language, designed and implemented to investigate ways of making the simulation of complex computer systems a less formidable task. A general-purpose language usually requires much effort to represent common computing processes.

On the other hand, existing computer system simulators are over-specialized and, in many applications, difficult to modify. Our overall approach is to provide a convenient and natural means of describing computer system characteristics and computing activities while maintaining the flexibility and power of a general-purpose simulation language. This report illustrates the strength of this approach by presenting a small model. Future plans include various improvements and additions to the language, and the development of larger and more sophisticated models for it.

Computer System Measurement and Evaluation

We briefly discussed Rand research in the application of hardware and software measurement tools to the evaluation of computer systems. The phenomenon that measurements indicate a strong instability in gross measures of multiprogrammed system performance with respect to load characteristics, disk data set allocation, and scheduling algorithms is discussed and exemplified.

Man-Computer Problemsolving Experiment

The observation and measurement of man-computer performance characteristics have received little attention compared with the great effort dedicated to the study of various computer hardware and software systems. In attempting to contribute to the currently scant store of quantitative information on man-computer problemsolving processes, and to evaluate available experimental techniques in the area, we designed and implemented an exploratory experiment in man-computer problemsolving. Twenty subjects performed a planning task, using the JOSS interactive computer system as a decision aid, to test the effects of forced temporal lockout intervals on performance. The experimental findings indicated that some of the lockout effects interacted with

the subject's level of experience. In general, however, the group of subjects having a five-minute lockout period after each trial not only achieved better solutions to the problem, but also used far less computer and personal time in doing so.

CONTENTS

FIGURES

I. INTRODUCTION

Rand's research into computer systems analysis methodology aims to provide better techniques for the design, evaluation, and analysis of computer systems. This Report describes three coordinated efforts toward achieving this goal:

1) The development of a set of design principles for languages to model and simulate computer systems by attempting to design and implement a prototype language called ECSS (Extendable Computer System Simulator).
2) The evaluation and extension of the state-of-the-art of using software and hardware devices to measure and analyze the performance of complex computer systems.
3) The provision of greater perspectives on the relationship between computer performance and man-computer performance by performing and analyzing controlled experiments in man-computer problem-solving.

Figure 1 and Table 1 indicate some of the reasons for concentration on techniques for the analysis of computer systems. Figure 1 relates the calendar time required to complete a spaceborne-software project to the size of the resulting computer program and to the degree of the project's original definition [5,6]. Figure 1 depicts that programming proceeds quite rapidly once a project is well-specified. However, the earlier "analysis" phases--e.g., determining hardware-software tradeoffs and integrating information system design with mission design--constitute a major contribution to the "technology gap," the time-lag between the availability of computer hardware

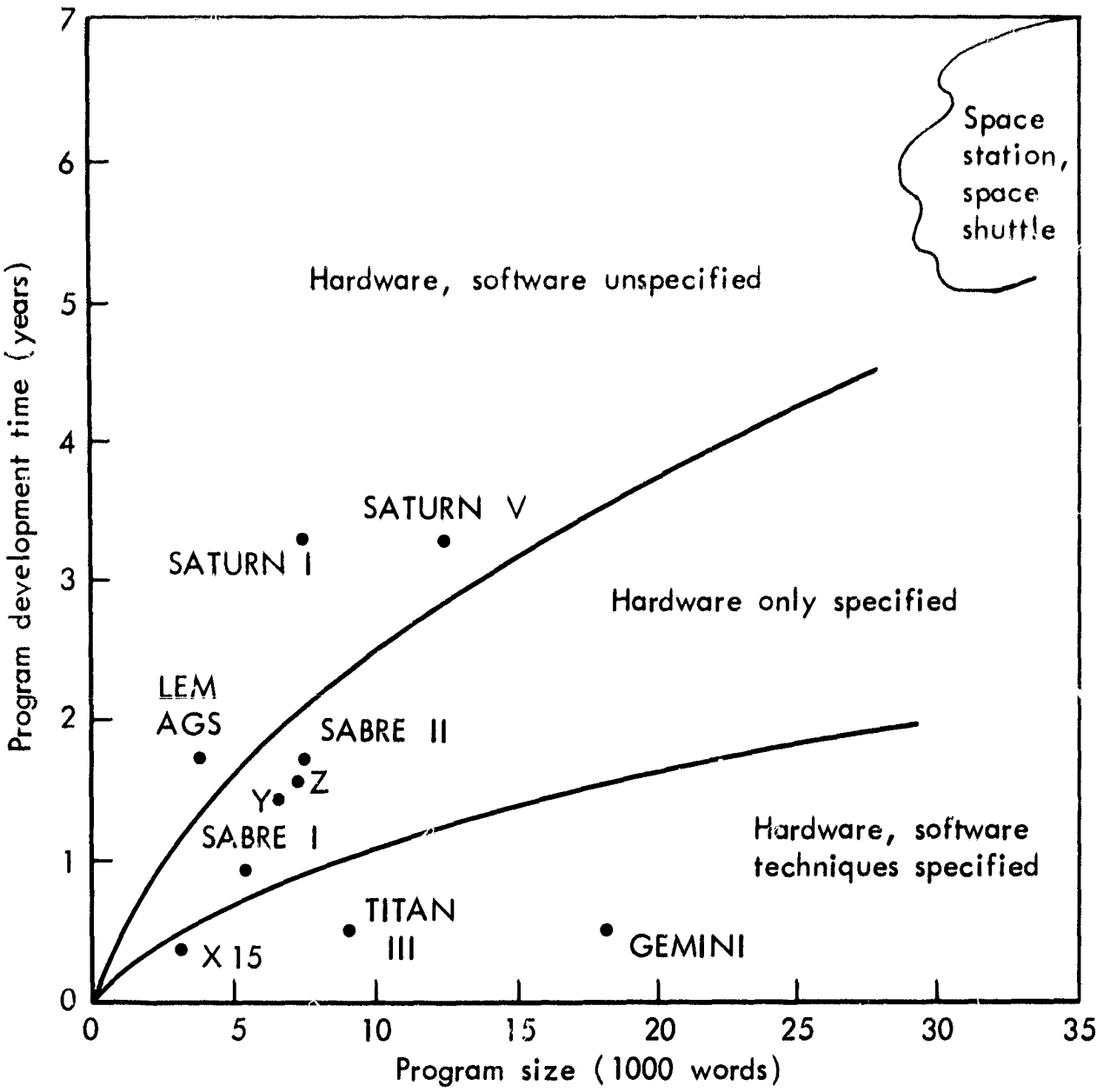


Fig. 1--Program Development Time versus Size and Degree of System Definition

and its integration into an operational mission. An attempt to place the NASA Space Shuttle and Space Station missions into the context of Fig. 1 indicates that this time-lag problem will become even more serious for these missions unless techniques to reduce it become available.

Table 1

COMPUTER PROGRAM DEVELOPMENT BREAKDOWN

Software Project	Analysis and Design	Coding and Auditing	Checkout and Test
Spaceborne: Hardware and Software Unspecified	62%	14%	24%
Spaceborne: Hardware only Specified	33	26	41
Spaceborne: Hardware, Software Techniques Specified	36	17	47
SDC General Rule of Thumb Estimation, Spaceborne	40	20	40
SAGE (Semi-Automatic Ground Environment)	39	14	47
NTDS (Naval Tactical Data System)	30	20	50
SETE (Secretariat for Electronic Test Equipment, NYU)	35	17	48
SDC General Rule of Thumb Estimation, Ground (Real-Time)	40	15	45

Table 1, derived from the SDC study of Ref. 6 and an earlier Rand study [7], confirms the indications in Fig. 1 that the major proportion of effort in an initially undefined

software project goes into analysis, and that, in any case, relatively little effort is devoted to programming individual routines. Thus, research efforts directed toward improving the "analysis" phase will have a more significant impact on the overall software development process than will research directed toward improving the "programming" phase, relatively over-emphasized at present. (A separate Rand research effort on software testing and certification techniques is also currently underway, under Air Force sponsorship.)

II. ECSS: EXTENDABLE COMPUTER SYSTEM SIMULATOR

SIMULATION OF COMPUTER SYSTEMS: CURRENT STATE

In designing and implementing simulation models of general-purpose, time-sharing systems in FORTRAN and SIMULA [8], time-shared graphics systems in GPSS [9], and computer-communications systems in FORTRAN [10], project members found many drawbacks in using existing programming languages: tedious, repetitive patterns of program statements for standard conversion and updating jobs, a lack of specialized terms to deal with common computing processes and desired outputs, and (particularly in GPSS) overspecializations that must be "programmed around" to achieve the desired model. Also, in reviewing [1] existing special purpose packages for simulating such computer systems as SCERT [11], S3 [12], and CSS [13], we found for many applications that these packages were not only overspecialized, but also extremely difficult to "program around" because this generally required a good deal of assembly-language programming and knowledge of internal table structures.

THE ECSS APPROACH

We felt that these difficulties were not necessarily inherent in languages that simulate computer systems. The design (primarily by Nielsen [1]) and preliminary implementation (primarily by D. W. Kosy of Rand) of ECSS attempts to test this hypothesis. ECSS, detailed in Ref. 1, provides a set of fairly "natural" terms and phrases for modeling computer systems, and a set of service routines that automatically handles much of the rotework involved in such models. Most important, it allows the modeler to extend ECSS, via special

definition statements or supplementary SIMSCRIPT II statements,† whenever he has difficulty describing his system with the standard ECSS facilities.

An Example

For example, suppose that one wishes to model the transmission of a message, 58 bytes long, sent with a priority of 3 over a message path between a processor and a terminal.

Modeled in SIMSCRIPT II:

```
LET COMPLETION = 58* TRANSMISSION.RATE(TERMINAL.A)
CALL TIME.OF.FREE.PATH (PROCESSOR, MESSAGE.PATH (*),3)
    YIELDING EXTRA.WAIT

IF CURRENT. PRIORITY < 3
    LET EXTRA.WAIT = 0
    REGARDLESS

SCHEDULE A MAKE.TRANSMISSION AT EXTRA.WAIT + TIME.V

SCHEDULE AN END.TRANSMISSION AT
    EXTRA.WAIT + COMPLETION + TIME.V
```

Modeled in ECSS:

```
SEND MESSAGE OF LENGTH 58 FROM PROCESSOR TO
    TERMINAL.A VIA MESSAGE.PATH WITH PRIORITY 3
```

ECSS IMPLEMENTATION

Figure 2 illustrates the relationship between ECSS and SIMSCRIPT II. Such ECSS statements as the one above, describing the computer system, its job load, and any specially

---

†SIMSCRIPT II [14], currently available and supported on IBM 360 computers, is a language with both general-purpose and simulation-oriented capabilities.

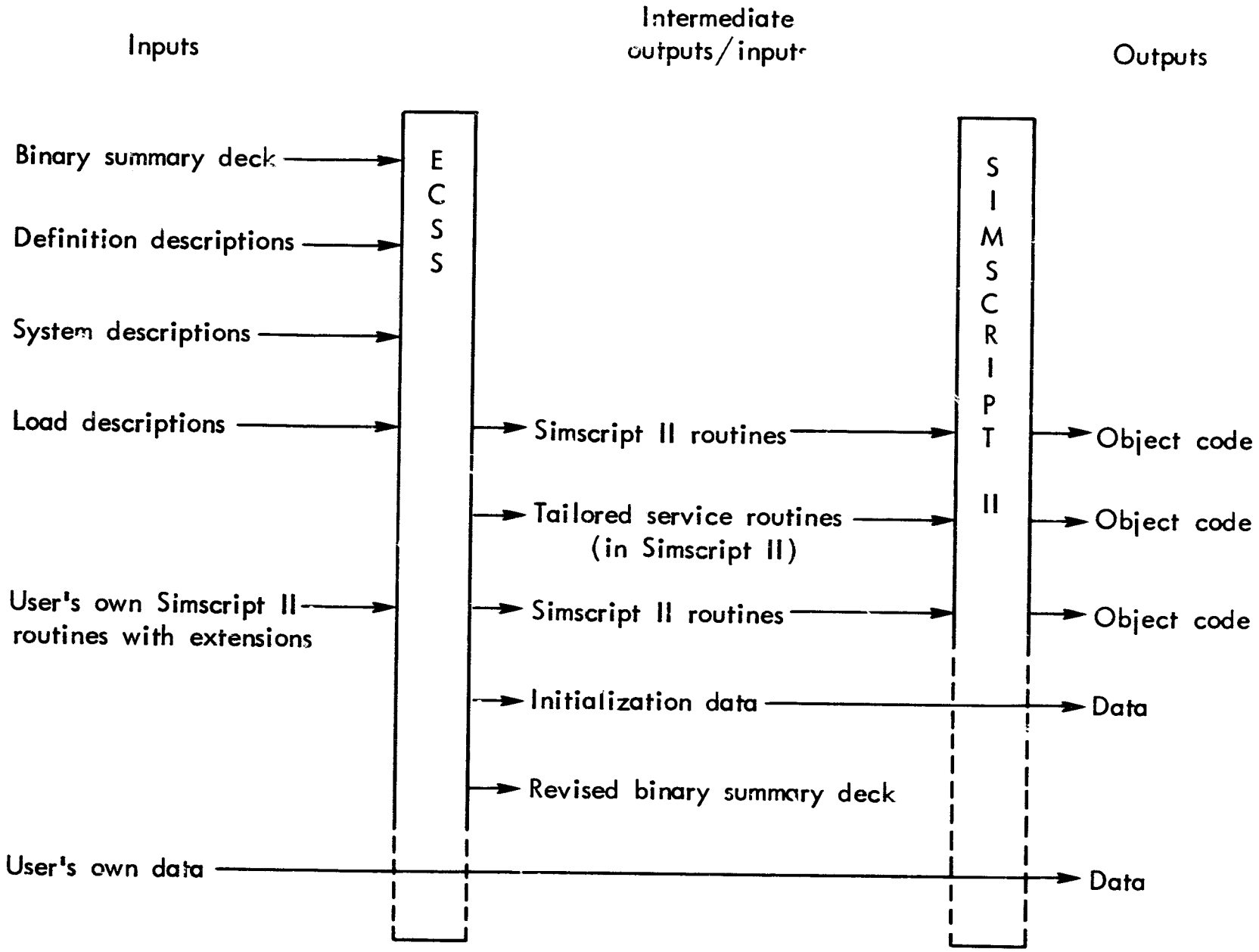


Fig. 2--Schematic of ECSS Translation

defined terms, are processed by a translation program into SIMSCRIPT statements, and augmented by both the necessary members of a set of SIMSCRIPT-coded service routines in the ECSS library and any data required to initiate the simulation. These SIMSCRIPT statements, and any supplied directly by the modeler, are then compiled into object code and executed. In order to allow small changes to ECSS models without having to recompile the entire model, a binary summary deck is also provided.

Example: A Simple Spaceborne Multiprocessor

The following simple model of a spaceborne multiprocessor represents a more complete example of an ECSS model. As seen in Fig. 3, three processing units are available to execute both periodic guidance and control (every 50 msec) and bio-monitoring programs (every 100 msec), and to process requests for data processing and display from any of five data interpretation terminals available to space-based personnel. These requests can result in demands for more memory space or processors than are available. A simulation can indicate the resulting waiting times and queue lengths, and give insights into system configuration improvements to better serve a given class of job loads or request patterns. Figures 4, 5, and 6 show the resulting ECSS code, with some annotations for clarity.

ECSS: CURRENT STATUS AND FUTURE PLANS

Currently, the essential parts of the ECSS translator are implemented and mostly debugged. Most of the associated service routines have been incorporated. We are testing the translator on a set of simple models, such as the spaceborne multiprocessor, and also validating results of a series of increasingly complex models of our IBM 360/65 system (including some extensions of OS/360 to handle time-sharing)

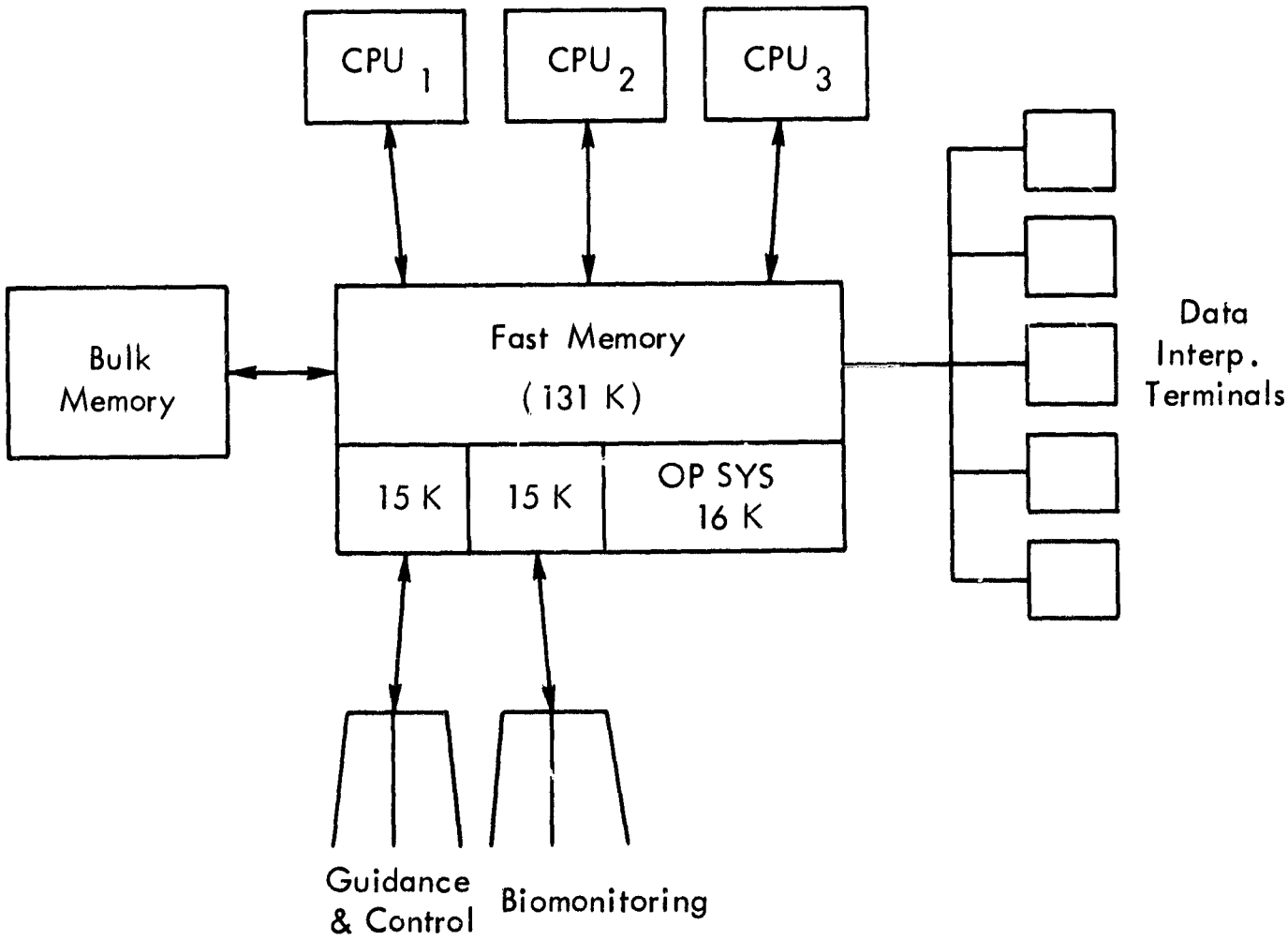


Fig. 3--Simple Spaceborne Multiprocessor

```
DEFINITION DESCRIPTION
   DEFINE UNITS  KWORDS = SPACE.UNIT  (represents
      1000 words of memory)
   DEFINE UNITS   WORDS = TRANSMISSION.UNITS,
      MIX.A.INSTRUCTIONS, MIX.B.INSTRUCTIONS  (ECSS now
      "knows" what these mean)
END


SYSTEM DESCRIPTION
   SPECIFY 3 PROCESSORS, EACH EXECUTES 500000
      MIX.A.INSTRUCTIONS PER SECOND
      EXECUTES 400000 MIX.B.INSTRUCTIONS PER SECOND
      ABSORBS 1 MILLISECOND PER MESSAGE
      CONNECTS TO FAST.MEMORY, TERMINALS
      IS INTERRUPTABLE WITH THE OPERATION CONTINUING
      HAS CAPACITY OF 1 EXECUTION USER
   SPECIFY 1 FAST.MEMORY, HAS CAPACITY OF 85 KWORDS
      CONNECTS TO BULK.MEMORY,PROCESSORS
   SPECIFY 1 BULK.MEMORY, HAS CAPACITY OF 100000 KWORDS
      CONNECTS TO FAST.MEMORY
      TRANSMITS 1000000 WORDS PER SECOND
   SPECIFY 5 TERMINALS, EACH TRANSMITS 1000 WORDS PER SECOND
      CONNECTS TO PROCESSORS
      HAS CAPACITY OF 1 EXECUTION USER
   PATH REQUEST.PATH IS TERMINALS, PROCESSORS
   PATH ANSWER.PATH IS PROCESSORS,TERMINALS
   PATH DATA.PATH IS BULK.MEMORY,FAST.MEMORY
END
```

Fig. 4--Spaceborne Multiprocessor System Description

```
(E)   LOAD DESCRIPTION          ((E): ECSS statement; (S): Simscript
                                 statement; note that they can be
                                 mixed)
(E)   JOB GUIDANCE.AND.CONTROL
(S)       IF TIME.V > 10000.0, STOP     (terminates simulation)
(S)           OTHERWISE
(E)       EXECUTE 10000 MIX.B.INSTRUCTIONS WITH PRIORITY 3
                    (3 is highest priority; will override data
                     interpreter jobs)
(S)       IF UNIFORM.F (0.0, 100.0, 1) < 50.0    (50% chance of more
                                                   execution)
(E)         EXECUTE 5000 MIX.A.INSTRUCTIONS WITH PRIORITY 3
(S)       REGARDLESS WRITE CPU(.JOB), TIME.V USING SYSTEM.LOG
                                                   (diagnostic output)
(E)   LAST

(E)   JOB BIOMONITORING
(E)       EXECUTE 8000 MIX.B.INSTRUCTIONS WITH PRIORITY 3
(S)           IF UNIFORM.F (0.0, 100.0, 1) < 70.
(E)         EXECUTE 3000 MIX.A.INSTRUCTIONS WITH PRIORITY 3
(S)       REGARDLESS WRITE CPU(.JOB), TIME.V USING SYSTEM.LOG
(E)   LAST

(E)   JOB SET.UP (TERMINAL)
(E)       EXECUTE 20000 MIX.A.INSTRUCTIONS WITH PRIORITY 1
(E)       SEND MESSAGE OF LENGTH 300 FROM CPU(.JOB) TO TERMINAL
              VIA ANSWER.PATH WAITING HERE FOR COMPLETION
                    (CPU can't proceed until transmission completed)
(E)   LAST
```

Fig. 5--Spaceborne Multiprocessor Load Description: I

```
(E)   JOB DATA.INTERPRETER   ((E): ECSS Statement; (S) Simscript)
(E)       GET 10 KWORDS FROM FAST.MEMORY
(E)       START JOB SET.UP (CPU(.JOB)) ON PROCESSORS  (log-on
                                                       processing)
(S)           LET ITERATIONS = UNIFORM.F(1.0, 7.0, 1)
                               (random number of data interpreta-
                                tion passes)
(S)   FOR I=1 TO ITERATIONS DO
(E)       WAIT FOR EXPONENTIAL.F(3.0, 1) SECONDS    (think time)
(E)       GET 20 KWORDS FROM FAST.MEMORY
(S)           LET Q = UNIFORM.F (5000.0, 20000.0, 1)
                      (size of this batch of data)
(E)       SEND MESSAGE OF LENGTH Q VIA DATA PATH  (don't start analysis
          WAITING HERE FOR COMPLETION             until all data is in memory)
(E)       START JOB DATA.ANALYSIS GIVEN TERM(.JOB),Q ON PROCESSORS
(E)       FREE 20 KWORDS FROM FAST.MEMORY      (now available for
                                                other jobs)
(S)   LOOP                    (continue for specified number of passes)
(E)       FREE 10 KWORDS FROM FAST.MEMORY      (sign off)
(E)   LAST

(E)   JOB DATA.ANALYSIS (TERMINAL,Q)
(E)       EXECUTE .01*Q**2 + 4.0*Q + 50000 MIX.B.INSTRUCTIONS
              WITH PRIORITY 1          (correlation of Q data values)
(E)       SEND MESSAGE OF LENGTH 500 FROM CPU(.JOB) TO TERMINAL
              VIA ANSWER.PATH          (display results)
(E)   LAST

(E)   INITIALLY START GUIDANCE.AND.CONTROL ON PROCESSORS AT
          0.0 SECONDS AND EVERY 50.0 MILLISECONDS AFTER
          ARRIVAL
(E)   INITIALLY START BIOMONITORING ON PROCESSORS AT 0.0
          SECONDS AND EVERY 100.0 MILLISECONDS AFTER
          ARRIVAL
(E)   INITIALLY START DATA.INTERPRETER ON EACH TERMINALS AT
          .40 SECONDS AND EVERY EXPONENTIAL.F(2.0,1)
          MINUTES AFTER COMPLETION
(E)   END
```

Fig. 6--Spaceborne Multiprocessor Load Description: II

by collecting measurements on the 360 for comparison with the simulation.

In the future, we plan to improve and complete the existing translator and service routines, extend and refine the capabilities of ECSS in modeling software control systems and automatically collecting performance statistics, improve its efficiency where possible, and use it to construct and validate more extensive models of current and future computer systems outside Rand.

III. COMPUTER SYSTEM MEASUREMENT AND EVALUATION

BACKGROUND

Language development provides only one part of the capability required for even a simulation-based analysis of a computer system. Techniques must be developed for accurately characterizing both input parameters to the simulation and appropriate models of the system and its workload. This will avoid the risk of a "garbage in, garbage out" exercise. Furthermore, the simulation must be carefully validated against actual performance data, and appropriate performance criteria must be developed so that different simulation runs may be usefully compared. These considerations have guided our companion area studies of computer system measurement and evaluation, and man-computer performance analysis.

Our studies in measurement and evaluation of computer systems have included not only such aspects directly related to computer system simulation as validation of the simulation and workload characterization, but also more extensive investigations into the relative advantages and disadvantages of hardware and software measurement tools for executing various classes of performance evaluation. These latter investigations are directly applicable to problems of computer system procurement (particularly in the use of benchmark jobs), and to performance improvement or "tuning" of existing computer systems.

CRITICAL AREAS

Detailed results of these investigations are found in the studies by Bell [2] and Watson [3]. Frequently, the measurements indicate a strong instability in gross measures of multiprogrammed system performance (central processing unit (CPU) utilization, throughput, etc.) with respect to

changes in load characteristics, disk data set allocation, and scheduling algorithms.† Small changes in load characteristics, etc., can easily produce large changes in multiprogrammed system performance. This phenomenon has the following significant operational implications:

1) Significant improvements in CPU utilization or throughput (usually at least 30 percent; sometimes over 300 percent) can be realized from investments in "tuning" multiprogrammed computer systems.
2) Computer systems selected and procured because of their performance on a series of "benchmark" jobs can lead to disastrous mismatches if great care is not taken to assure that the benchmarks are fully representative. It is relatively easy for a vendor to tune his system to look exceptionally good on a small number of benchmark jobs.
3) As workload characteristics change with time, the maintenance of a well-tuned computer requires a continuous rather than a one-shot effort.

The complexity of multiprogrammed computer systems also creates a situation in which a set of measurement trends may result from any of several dominant causes, requiring considerable detailed examination of interactions before the key contributing factors are isolated for subsequent decisions and actions. Using the simplest explanation as a basis for decisions can lead to highly dysfunctional results.

An Example

One of our studies provides a good example of this phenomenon. Figure 7 summarizes the performance of our

---

†A multiprogrammed computer system is one in which several independent programs simultaneously reside in the main memory and compete for the computer's various resources (CPU, input-output channels, disk arms, etc.) during their execution.

IBM 360/65 computer system over a period of 11 months, measured in terms of the percentage of CPU cycles productively utilized. (The remainder of the cycles were not used because the current programs in the computer were waiting for the completion of some input-output operation.) Figure 7 shows that measures for performance improvement--adding four IBM 2311 disk drives on a second channel, replacing the 2311s by a high-capacity 2314 drive, and adding 50-percent more core memory--strongly correlate with actual performance increases. (The curves represent an "eyeball" fit to the data points.) However, more detailed analysis of the evolution of performance component elements indicates that the later increase actually correlates with a decrease in the average number of jobs residing in the increased core storage (Fig. 8), and is primarily due to an otherwise undetected increase in average CPU usage by individual user jobs (Fig. 9). Similarly, detailed analysis indicates that the earlier increase in performance is due as much to decreases in input-output characteristics of the workload (Fig. 10) as to configuration changes. Thus again, the most direct explanation of computer system measurement data is often not the appropriate one.

MEASUREMENT AND EVALUATION: FUTURE PLANS

We will continue our evaluations of the utility of such software monitoring packages as IBM's Advanced Multiprogramming Analysis Procedure (AMAP) and System Measurement Facilities (SMF), and the packages developed by Stanford Linear Accelerator Center (SLAC) and Boole & Babbage, by continuing to use them in practical investigations of core fragmentation, channel and disk-arm contention, CPU slowdowns due to bulk core access, etc. These investigations have led to significant insights and improvements in the performance of Rand Computational Center computing systems over the past year.

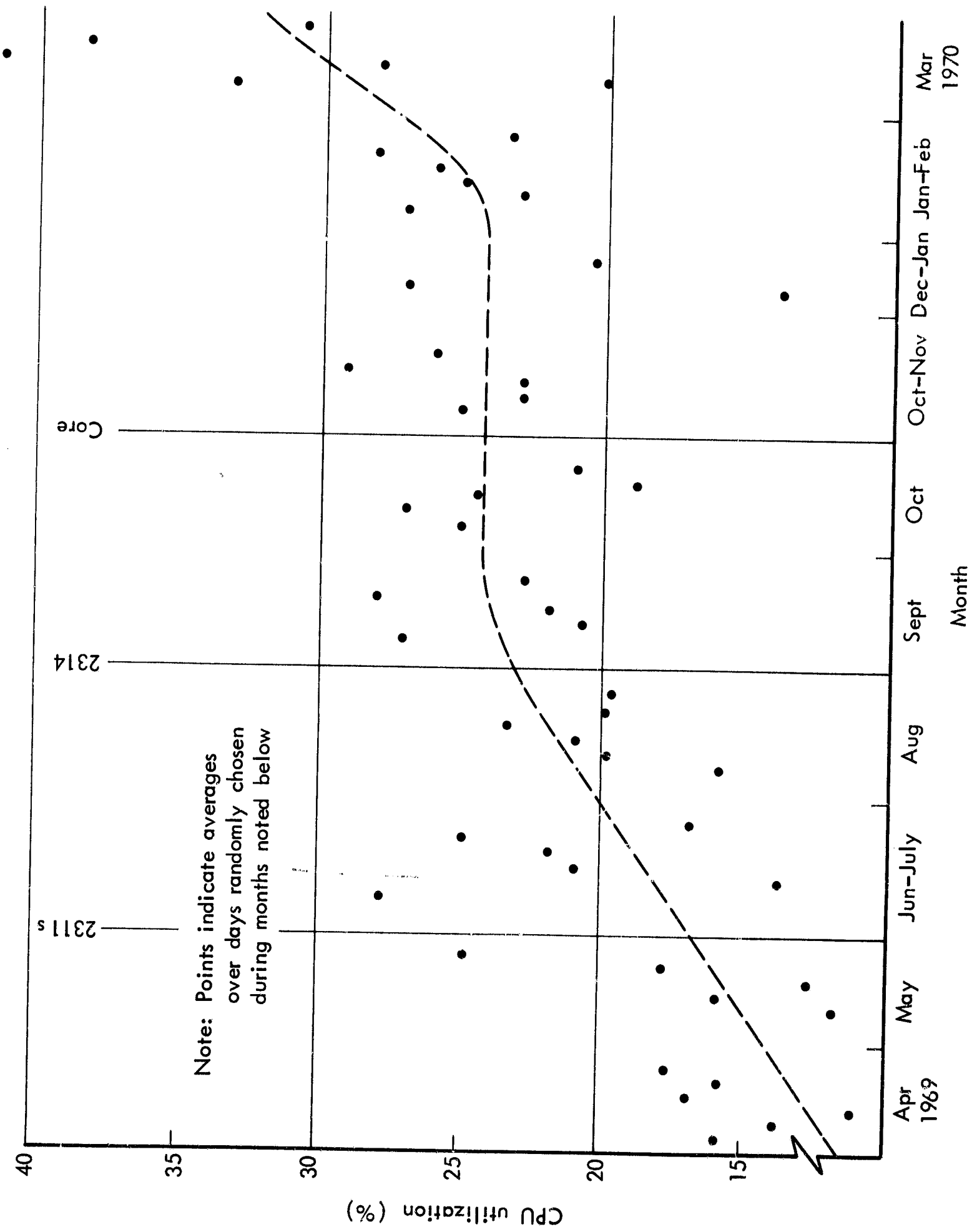


Fig. 7--CPU Utilization versus Time (IBM 360/65; OS/MVT)

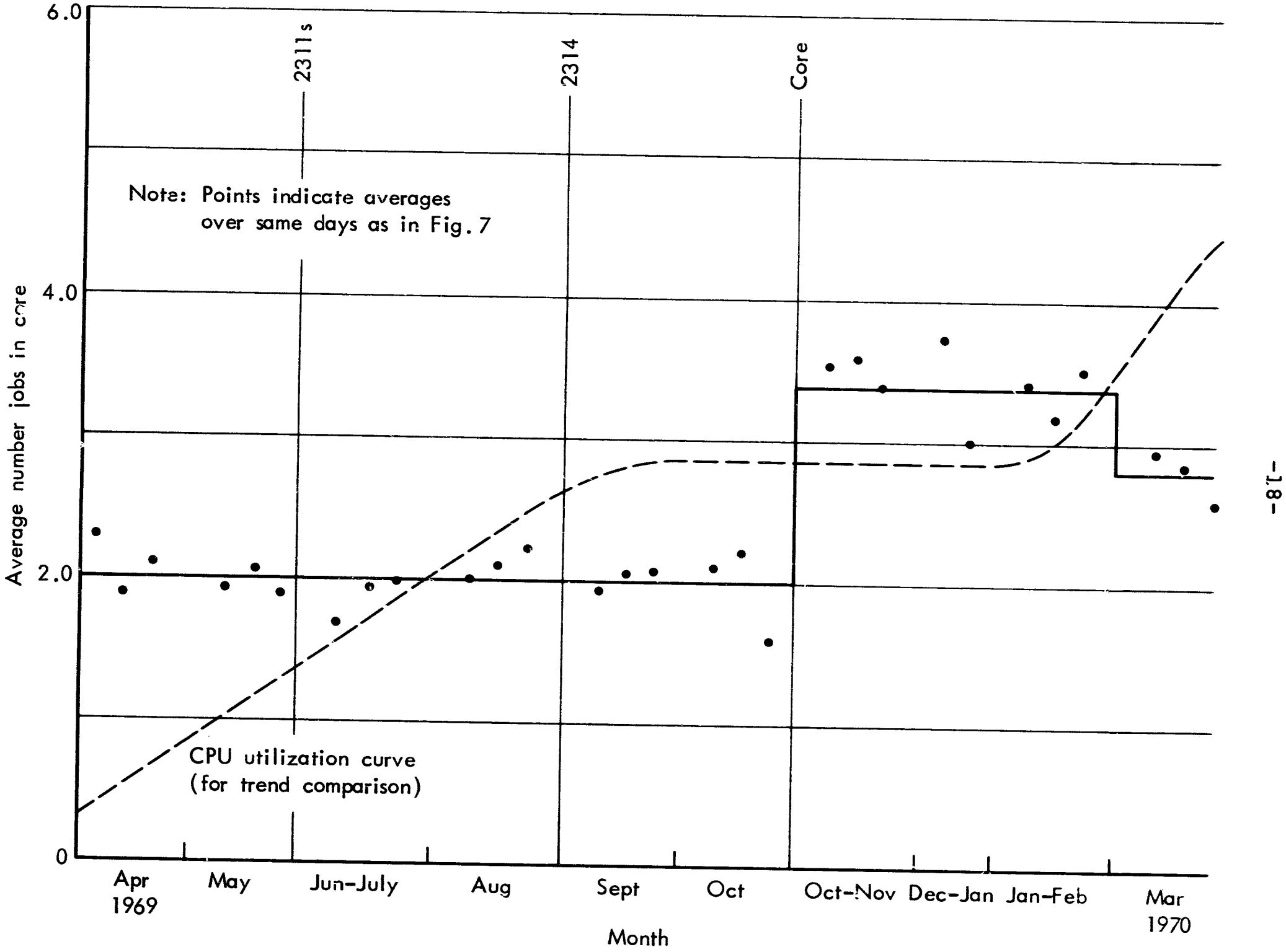


Fig. 8--Average Jobs in Core versus Time

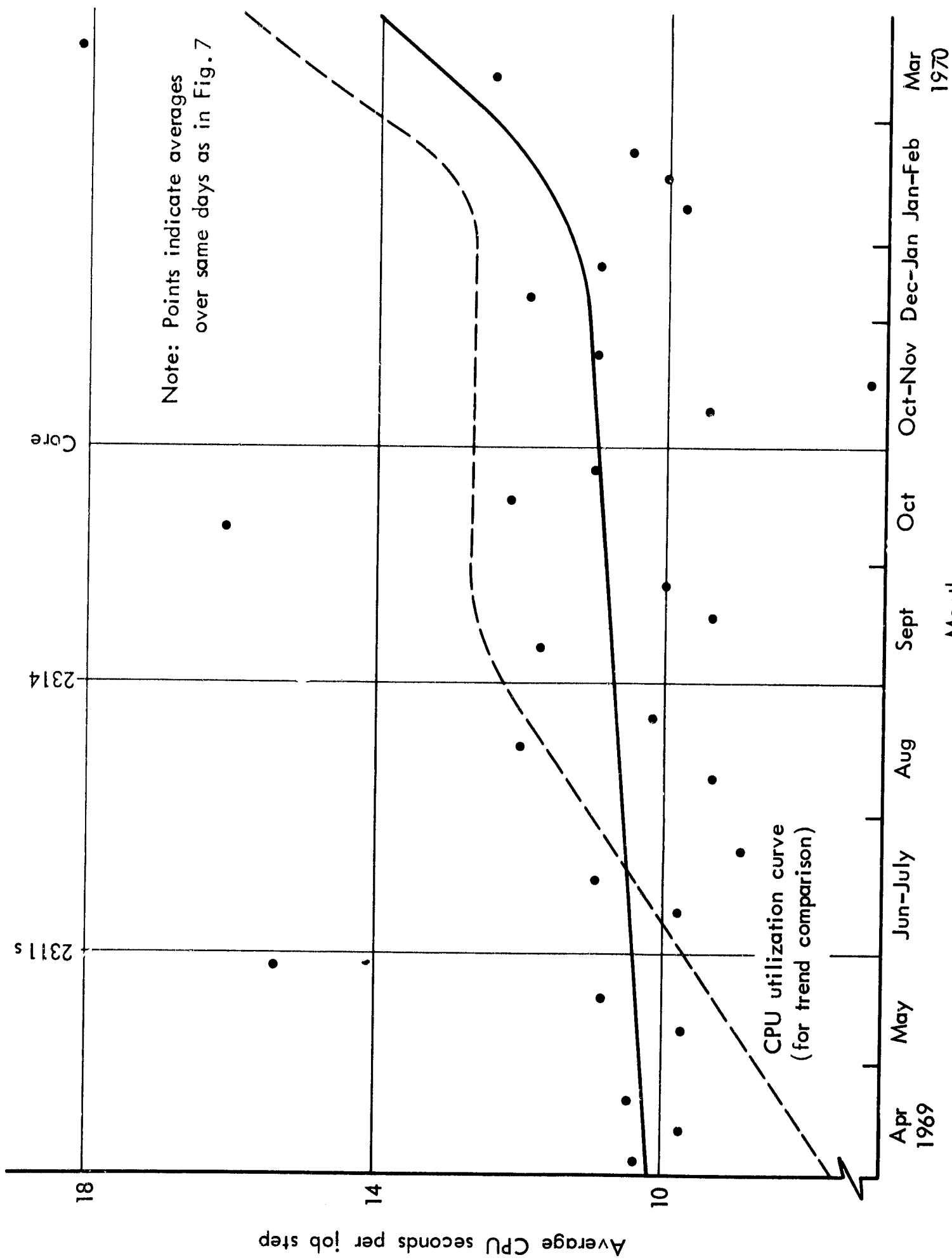


Fig. 9--CPU Load per Job Step versus Time

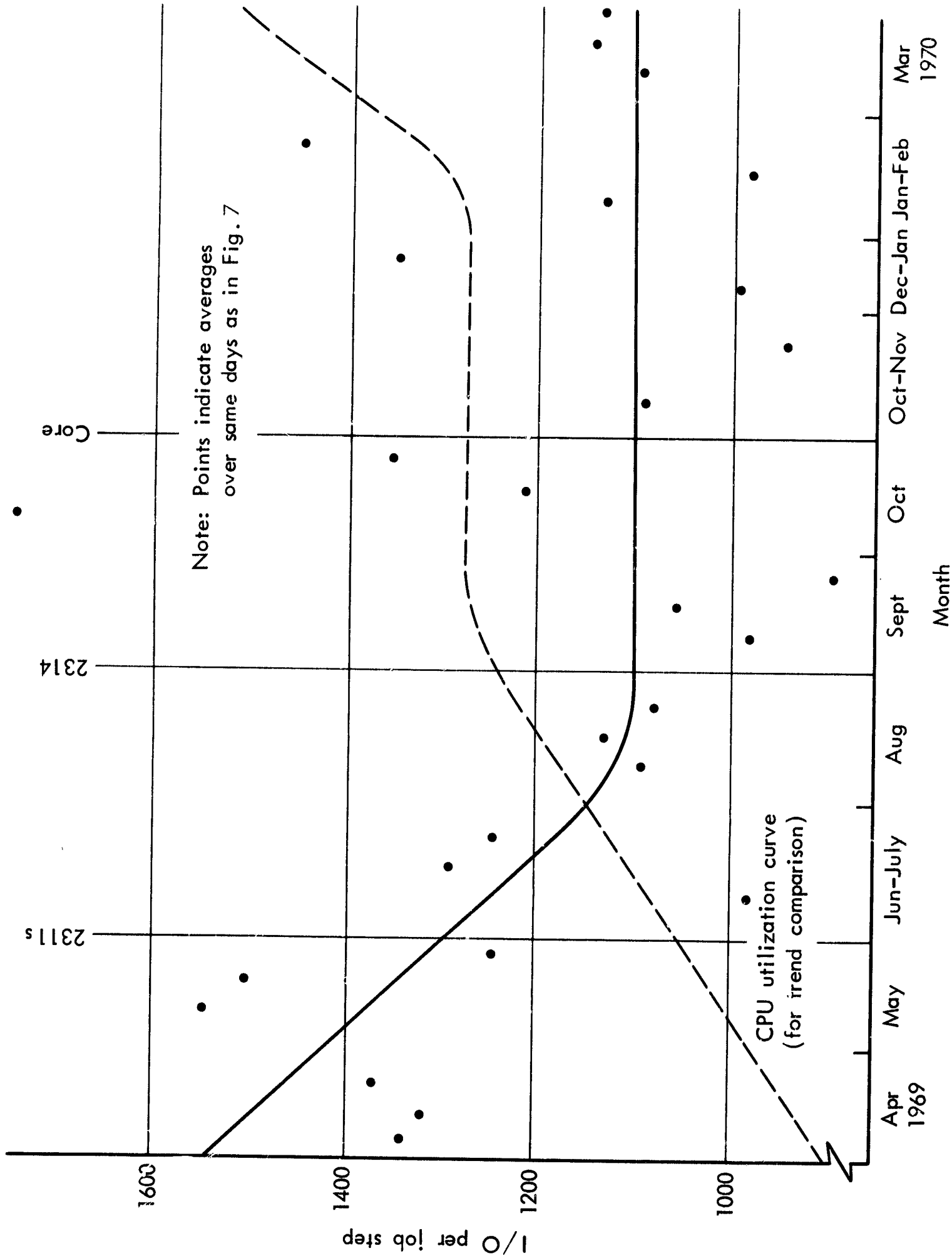


Fig. 10--Input/Output per Job Step versus Time

We are currently planning to experiment with a minicomputer hardware monitor to measure aspects of the performance of Rand's time-shared Video Graphics System. This will allow us to capture and analyze data on detailed interactions on a level impossible to attain with software monitoring, and extremely difficult with conventional hardware monitors. It will also allow us to compare the measurements with results of previous and upcoming simulations of the system.

Furthermore, we will continue and extend our work in determining the inherent variability (the practical "noise level") in computer systems, in refining methods for characterizing benchmark jobs in terms of their effects on system performance, and in developing indicators of computer system performance from low-cost data, primarily existing accounting data. We also plan to broaden the interface between our measurement and evaluation studies, the simulation studies, and the man-machine studies, particularly in validation and load characterization measurements for simulations and in deeper investigations of user responses to changing computer performance characteristics.

IV. MAN-COMPUTER PROBLEMSOLVING EXPERIMENTS

THE NEED FOR QUANTITATIVE MAN-COMPUTER DATA

One danger inherent in computer system measurement and tuning is an ever-present temptation to consider computer system performance as an end in itself, rather than as a means to better serve people. Such "performance improvement" methods as universal use of one language, large blocking of data input and output, and intricately designed code and procedures can increase machine productivity. However, it costs users an abnormally high effort to achieve any results. On the other hand, text editors, extended debugging aids, and conversational programming systems tend to reduce user-time investments at the expense of machine efficiency.

In general, then, there is a tradeoff between machine efficiency and user time invested. Philosophical arguments will yield to factual analysis of this tradeoff only when the effects on both the humans and machines can be quantitatively measured and related to overall goals. In an attempt to contribute to the currently scant store of quantitative information on man-computer problemsolving processes,[†] and to evaluate available experimental techniques in the area, we designed and implemented an exploratory controlled experiment in man-computer problemsolving.[‡]

CHOICE OF EXPERIMENT

We structured this experiment to test Gold's hypothesis that restricting one's access to the computer for a period of time *after* the presentation of current results ("lockout"

[†]Sackman [15] has provided an excellent review of results to date.

[‡]The Appendix describes some of the rationale leading to the structure of this experiment, beginning with an attempt to define a reasonably measurable and human-oriented computer system performance criterion called the "Productive Thought Ratio."

period), might improve performance by inducing the user to concentrate more on problemsolving strategy than on tactics [16]. Figure 11 shows that the lockout requires the user to spend a certain amount of what is generally called "think time."

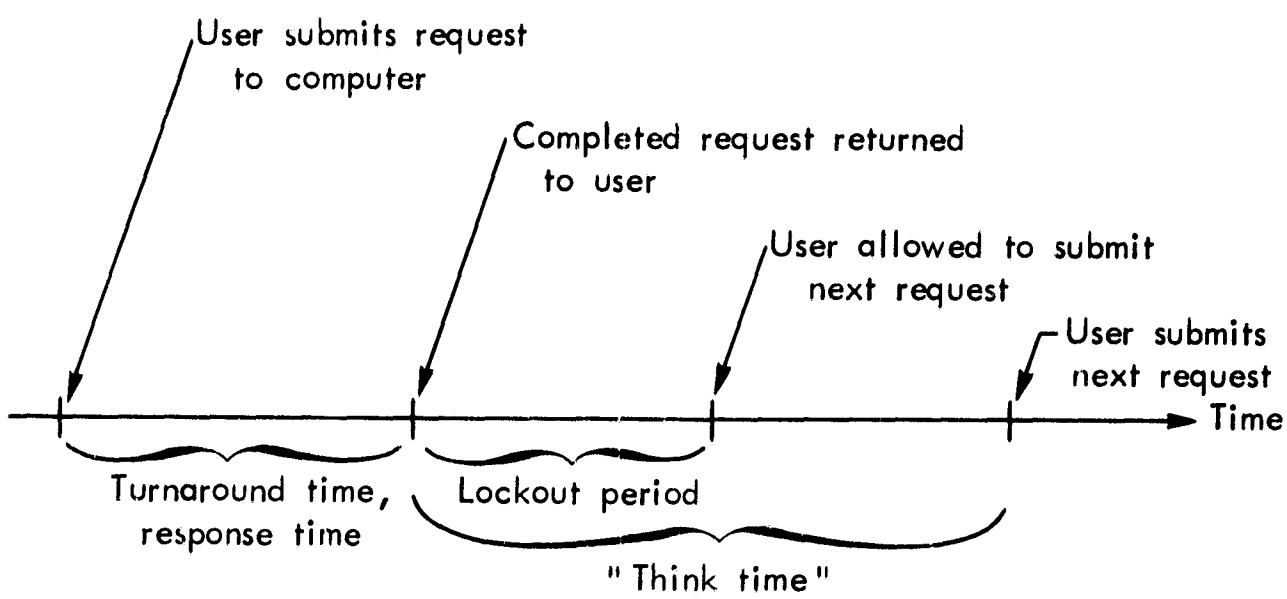


Fig. 11--Sequence of Events for Submitting a Trial Solution

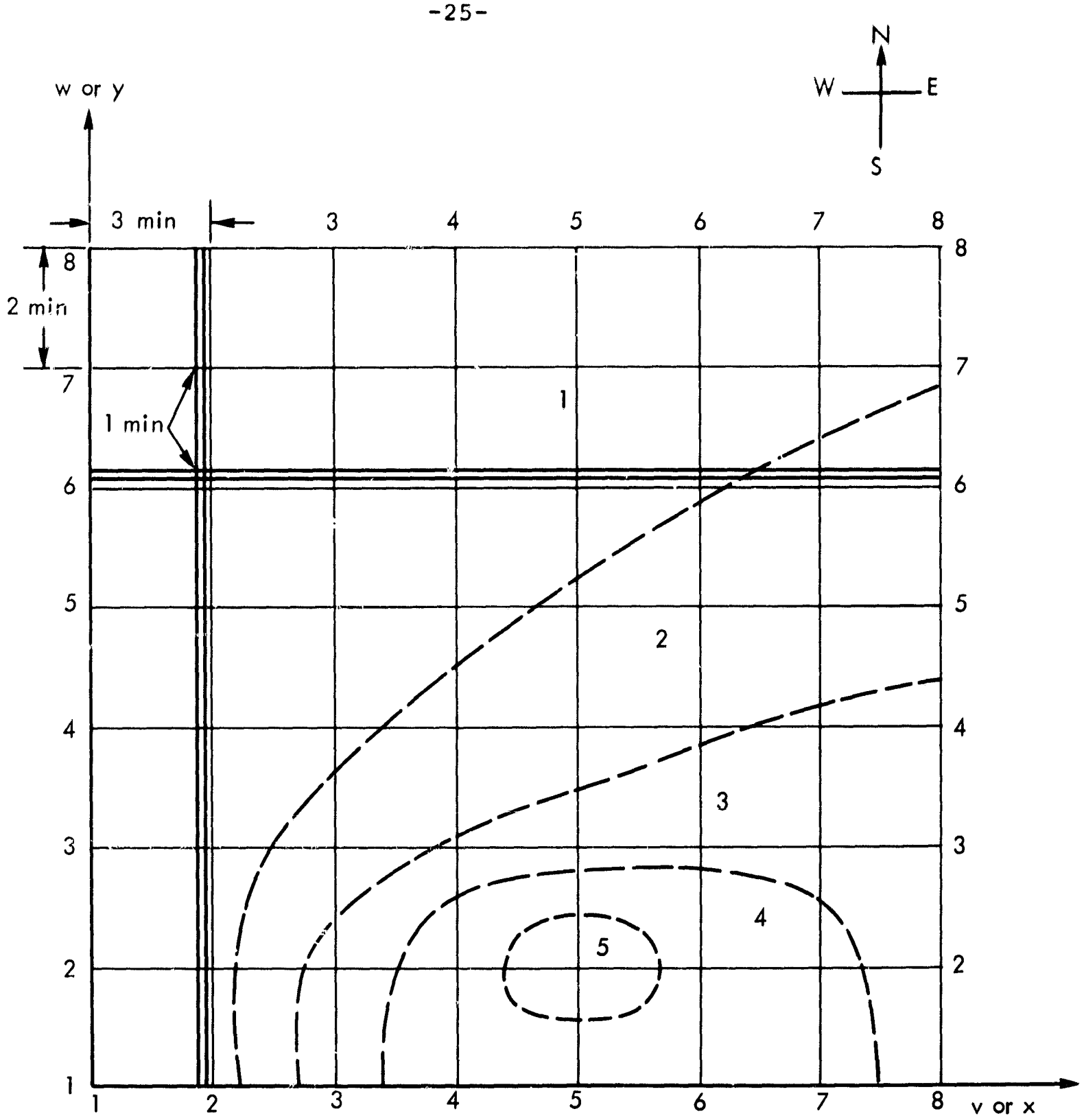
The general problemsolving situation required the subject to solve a geographical area servicing problem with the aid of JOSS, Rand's interactive computer system [17]. Subjects were allowed two hours to solve the problem, but the problem was open-ended to the extent that a range of solutions existed. A protocol of each subject's performance was generated from automatic recordings within the JOSS system, written records kept by an observer, and audio tape recordings of the subject's vocalizations. The resulting data were analyzed using analysis of variance and regression techniques.

THE TEST PROBLEM

Each subject was given a map showing a grid of surface streets, two freeways, and contour lines that indicated the frequencies of emergencies per day per intersection throughout the area (Fig. 12). Transit times between intersections were defined as two minutes on North-South surface streets, three minutes on East-West surface streets, and one minute on freeways. A time penalty of one minute was assessed for entering or leaving the freeway at any intersection.

The subject's task was to specify three surface intersections at which to locate three emergency hospitals, and to specify a set of decision rules regarding when and when not to use the freeways. His goal was to minimize the average response time per emergency for the entire area, taking into account the different accident densities. His solution was subject to the constraint that the maximum one-way response time to any given location be no more than 12 minutes. It was made clear that the number of ambulances was unlimited; scheduling and ambulance turn-around time were not factors.

The JOSS system was pre-programmed to provide the subject, on demand, with an evaluation of the effectiveness of his location and decision-rule inputs, and with certain other feedback relating to the problem. Hospital locations were specified in X,Y coordinates shown on the map. Variables used in the decision rule were specified so that the subject could refer to specific hospitals (i=1, 2 or 3), hospital locations (x,y), or emergency locations (v,w) in terms understood by the special program. As a result of an evaluation computation ("Do part 1."), the program provided 1) the average response time per emergency, and 2) the maximum response time to any emergency. If requested, the program also provided various types of information matrices:



Transit times between intersections
- 1 min on freeway
- 2 min on north-south surface streets
- 3 min on east-west surface street

Time to both enter (1 min) and leave (1 min) freeway system from surface streets: 2 min

Fig. 12--Problem Map, On-line Experiment

1) A matrix showing minimum response time to each intersection from any of the three hospitals ("Do part 210.");
2) Three individual matrices showing response times to each intersection from each of the three hospitals ("Do part 220.");
3) An individual matrix showing response times from each hospital specified ("Do part 221 for i= __."). The use of the special program was illustrated with a reproduction of an actual JOSS record of three "trials" (Fig. 13).

TREATMENT GROUPS

The primary experimental treatment was provided by programming the JOSS system to lock the subject out of the system for a specified length of time after each trial, i.e., after a current set of results had been presented to him. Lockout conditions were different for each of five groups of subjects, and included both fixed and variable intervals.

The subjects, primarily graduate students at Rand for the summer, were divided into the following five groups:

0 -- No lockout; free access to console;
5 -- Five-minute lockout period;
8 -- Eight-minute lockout period;
V -- Variable lockout period (5-min mean);
c -- Choice; subjects were instructed to "lock themselves out" as much as possible, but otherwise had free access to console.

On the basis of a questionnaire, subjects were ranked from 1 to 20 with respect to computing and operations research experience. The experimental groups were balanced in regard to experience.

Example: Using the JOSS program

X(1) 3 Y(1) 3 X(2) 6 Y(2) 3 X(3) 5 Y(3) 6	Hospital locations	You supply these
100.1 Set z 1 if v 2 or w 6. — a decision rule 200 Do part 210. — a print option Do part 1.		
Average 5.40496 Maximum 16 16 7 10 7 4 7 10 13 14 6 8 5 2 5 8 11 6 5 4 3 2 3 4 5 10 6 4 5 2 4 7 10 8 6 2 5 4 2 5 8 6 5 0 3 3 0 3 6 8 6 2 5 5 2 5 8 10 7 4 7 7 4 7 10	Shortest response times to intersections	Program types out these
Y(3) 7 200 Do part 221 for i 3. Do part 1.		You supply revisions
Average 5.44628 Maximum 14 14 9 8 5 2 5 8 11 12 8 6 3 0 3 6 9 8 7 6 5 4 5 6 7 16 8 10 7 4 7 10 13 18 9 12 9 6 9 12 15 20 10 14 11 8 11 14 17 22 11 16 13 10 13 16 19 24 12 18 15 12 15 18 21	Response times from hospital 3	Program responds
Y(3) 8 100.2 Set z 1 if w 5. 200 Do part 210. Do part 1.		You supply revisions
Average 5.89256 Maximum 14 12 10 6 3 0 3 6 9 14 9 8 5 2 5 8 11 9 8 8 7 6 7 8 9 10 7 10 9 8 9 10 11 8 6 2 5 5 2 5 8 6 5 0 3 3 0 3 6 8 6 2 5 5 2 5 8 10 7 4 7 7 4 7 10	Shortest response times	Program responds

Fig. 13— Sample JOSS printout

BASIC RESULTS

The primary measure of a subject's performance was the minimum average emergency response time he could achieve during the two-hour period. For analysis purposes, this performance was transformed into a percentage of optimal performance. Figure 14 shows the resulting performance of each individual, organized with respect to experimental group and presented with group averages. In this case, the group with a moderate lockout period (5 min) performed better than both the group with free access (0 min) and the group with a relatively severe lockout period (8 min). The variable lockout group performed almost as well as Group 5, and the "choice" group almost as well as Group 0. (However, Group C achieved this performance with half as many computer trials as Group 0).

The numbers next to each data point in Fig. 14 indicate the subject's experience ranking, "1" being highest. It is evident that performance strongly correlates with experience. This comparison is highlighted in Fig. 15, which plots each subject's performance rank versus his experience rank. The associated symbol identifies the subject's group. Most subjects fall quite close to the equivalence line bisecting the figure. However, the less experienced members of Groups 5 and V generally performed better than their experience might predict. The more experienced members of Group 8 and the less experienced members of Groups 0 and C generally performed worse than their experience might predict. Analysis of variance calculations indicates that lockout is significant at the 0.025 level, experience significant at the 0.005 level, and the interaction between lockout and experience significant at the 0.10 level.

Over 40 other performance measures were collected and analyzed along with considerable anecdotal data of interest,

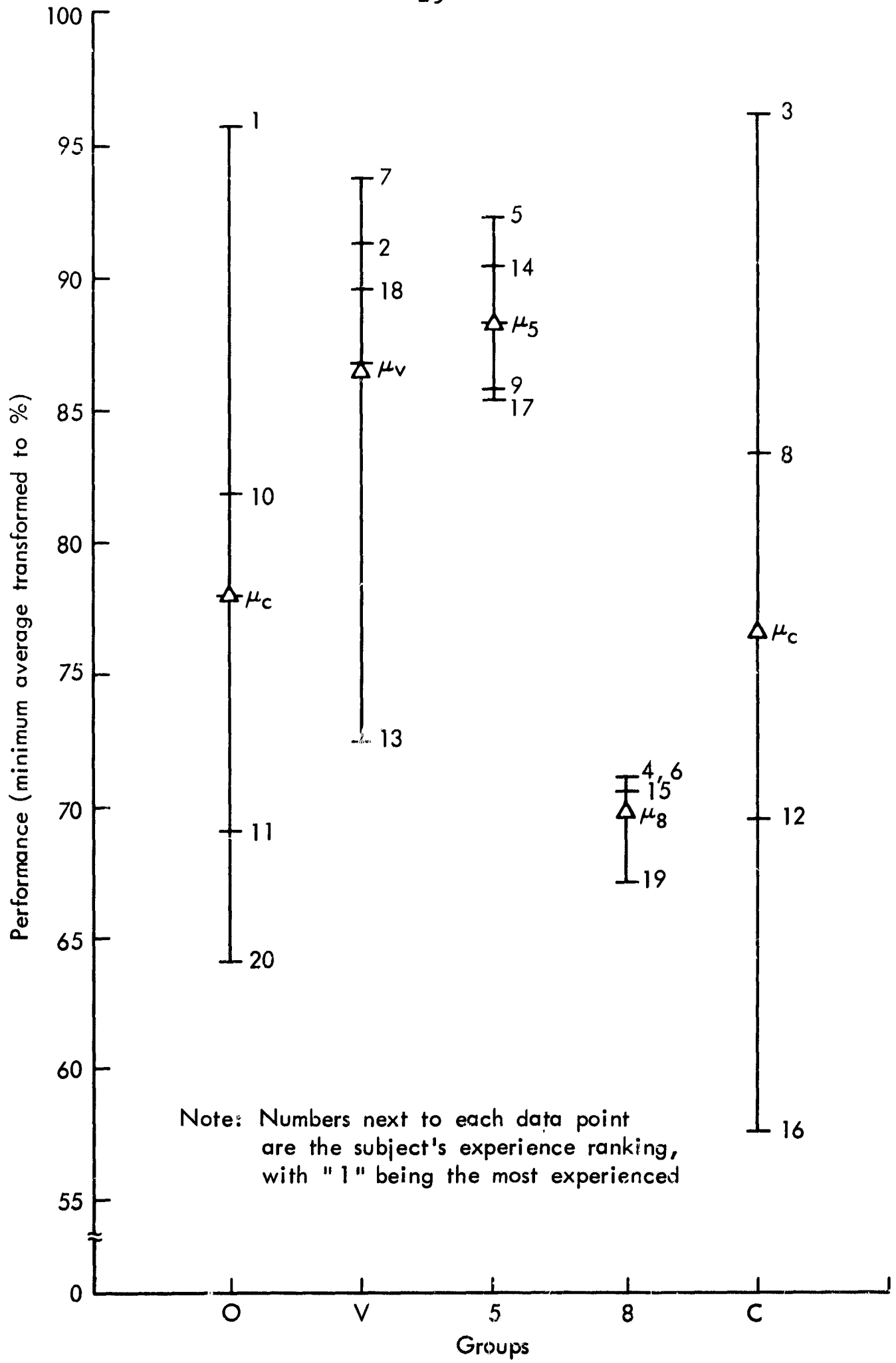


Fig. 14--Quality-of-solution Scores Attained by the Subjects

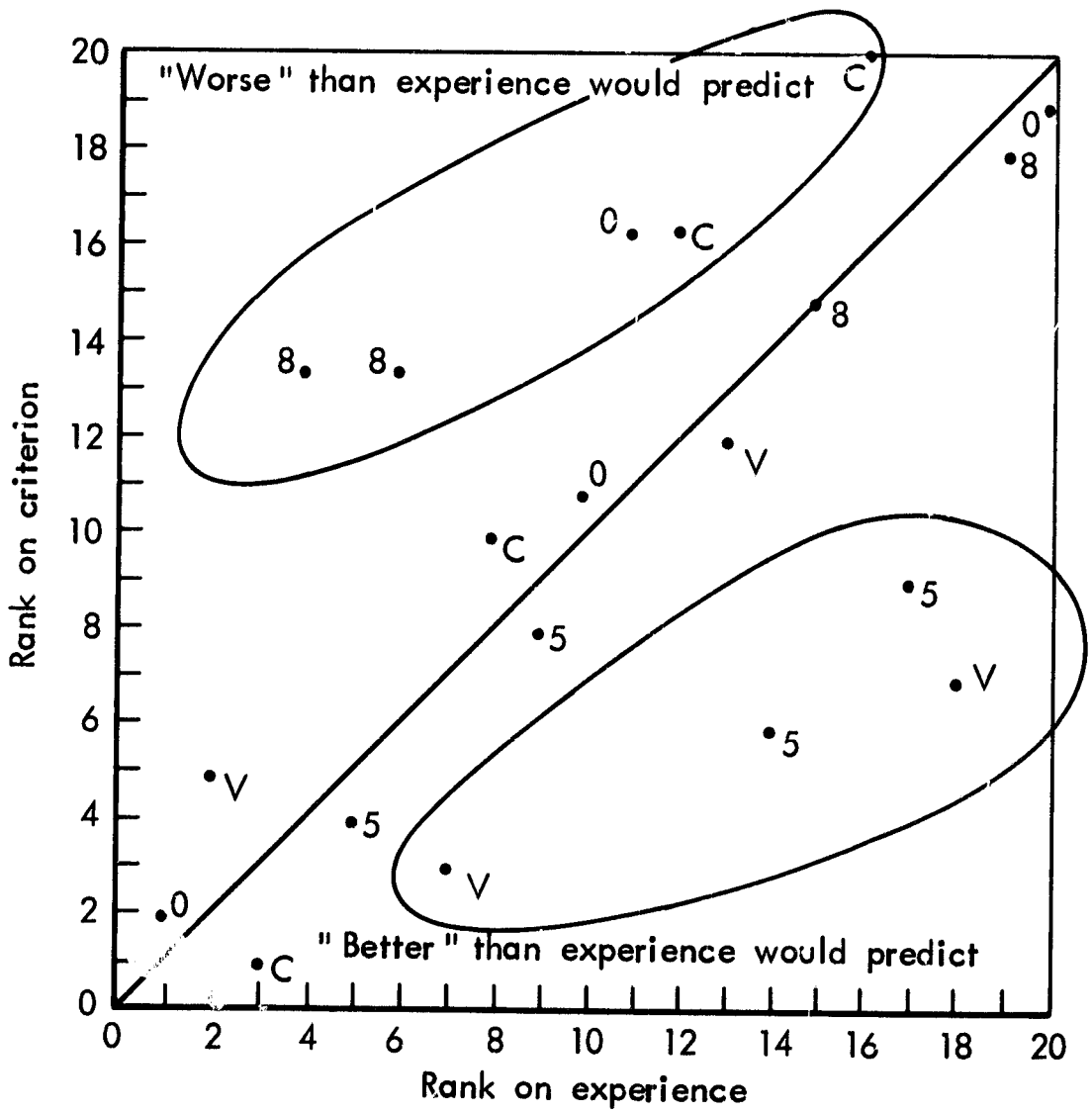


Fig. 15--Rank on Experience Compared with Rank on Criterion Measure

detailed in Ref. 4. Significant further results demonstrate that:

1) The subjects with free access (Group 0) average twice as much computer usage as groups with restricted access.
2) Group 0 subjects show no relative economies of their own time in attaining their high performance levels; however, Group 5 subjects do.
3) In general, subjects express dissatisfaction with restricted access, even in the groups with high performance.

TENTATIVE CONCLUSIONS

Perhaps the most impressive aspect of the experiment was the subjects' tremendous variability in problemsolving approaches. It is difficult to imagine anyone ever formulating a single model of man-computer problemsolving that would fit even our small group of subjects, which included some whose performances were so irregular that they had to be dropped from the analysis. For example, one subject promptly began by dumping our JOSS control program; after two hours, his only result was a set of undebugged modifications of this program. Another preferred to work almost completely by hand, saying "he didn't trust computers," and never achieved a feasible solution.

However, with respect to the large majority of problemsolvers who achieved feasible solutions in the experiment, the results of this small exploratory study raise some interesting questions regarding popular beliefs about man-machine problemsolving. Our evidence suggests that, at least in this experimental context, users tend to become dissatisfied if mild restraint is placed on their free interaction with the computer. They also tend to problemsolve more effectively, using less computer time and less of their

own time in the process. Such shibboleths as "faster is better" and "more computer time means less human time" may at times serve the computer salesman more than the consumer. The results also cast doubt on the validity of user acceptance as a general index of system effectiveness. The user may want what inconveniences him least in the short run, or he may want what he has been led to believe he should want, but the general efficacy of such desires cannot be taken for granted.

Definitive answers to questions relating to the nature of relevant parameters of problemsolving systems are of more than academic value. For example, under some circumstances, organizations under pressure to expand their hardware inventory to meet increased demand might find it far more productive to keep the system they have and introduce some form of constraint (e.g., an accounting system) that will encourage more judicious and creative use of the existing computational capabilities. However, without more information and better understanding, it would be a mistake to conclude that either approach is the "right" one. The only general conclusions that can be reached on the basis of the present work are that the relationships involved in man-machine problemsolving are neither obvious nor simple, and that there is reason to believe that further investigation could have practical significance.

MAN-COMPUTER EXPERIMENTS: FUTURE PLANS

We are currently testing the same problem on another group of subjects, under lockout conditions 0 and 5, to determine whether the initial results are confirmed by a larger sample. Also, because earlier subjects indicated in their debriefings that a graphic display capability could have helped them, and since Rand has an interactive Conversational Programming System (CPS) working on both

typewriter and graphic consoles, we are considering a modification of the current experiment to test the relative efficacy of typewriter and graphic terminals in this context.

However, before plunging into another experiment, we feel it important to devote more thought to two fundamental questions:

1) The classification of problem characteristics and problemsolving activities, at least in the neighborhood of our current study.
2) Determination of better measures of human problem-solving experience, attitudes, and capability.

Without solid foundations in these areas, future studies will progress no further in operational utility than the one reported here: provocative, useful as a cautionary indicator, but hardly a predictor for any operational situation.

V. CONCLUDING REMARKS

This Report is the product of a relatively modest effort (two to three man-years), yet it includes several results of far-reaching significance to the practical design, development, and management of computer systems, particularly during the "analysis" phase. When this is coupled with the data in Sec. I--showing that for each dollar spent for program coding on large or real-time projects, two to four dollars are spent on analysis--it indicates that a reconsideration of national computer research priorities may be warranted. Further gains in computer systems analysis techniques with strong practical significance may be as easily accessible as the results presented here.

APPENDIX

TOWARD A PERFORMANCE CRITERION FOR MAN-COMPUTER SYSTEMS: THE PRODUCTIVE THOUGHT RATIO

The following approach guided our research efforts in the analysis of man-computer systems:

1) Formulate a performance criterion for man-computer systems that appears reasonably discriminating and measurable.
2) Investigate the implications of using this criterion operationally.
3) Identify the resulting key problems and experiment for insight into them.

FORMULATE A PERFORMANCE CRITERION

Current performance criteria for such computer systems as throughput, component utilization efficiency, and turn-around or response time, tend to concentrate on the servicing of individual computer run requests rather than on the project advancement for which a given run is being made. Computer systems optimized with respect to the above criteria tend to emphasize machine efficiency at the expense of such amenities as ease of learning, programming, debugging, or modifying programs, which tend to increase human efficiency.

Suppose, however, that one could characterize the computer support of various types of projects (e.g., an engineering research and development project) as time series of individual computer run requests (Fig. 16), and that one could separate the time spent on the project into three activities, essentially mutually exclusive:

T_1 : Time spent thinking about the project;

T_2 : Time spent thinking about the programs supporting the project;

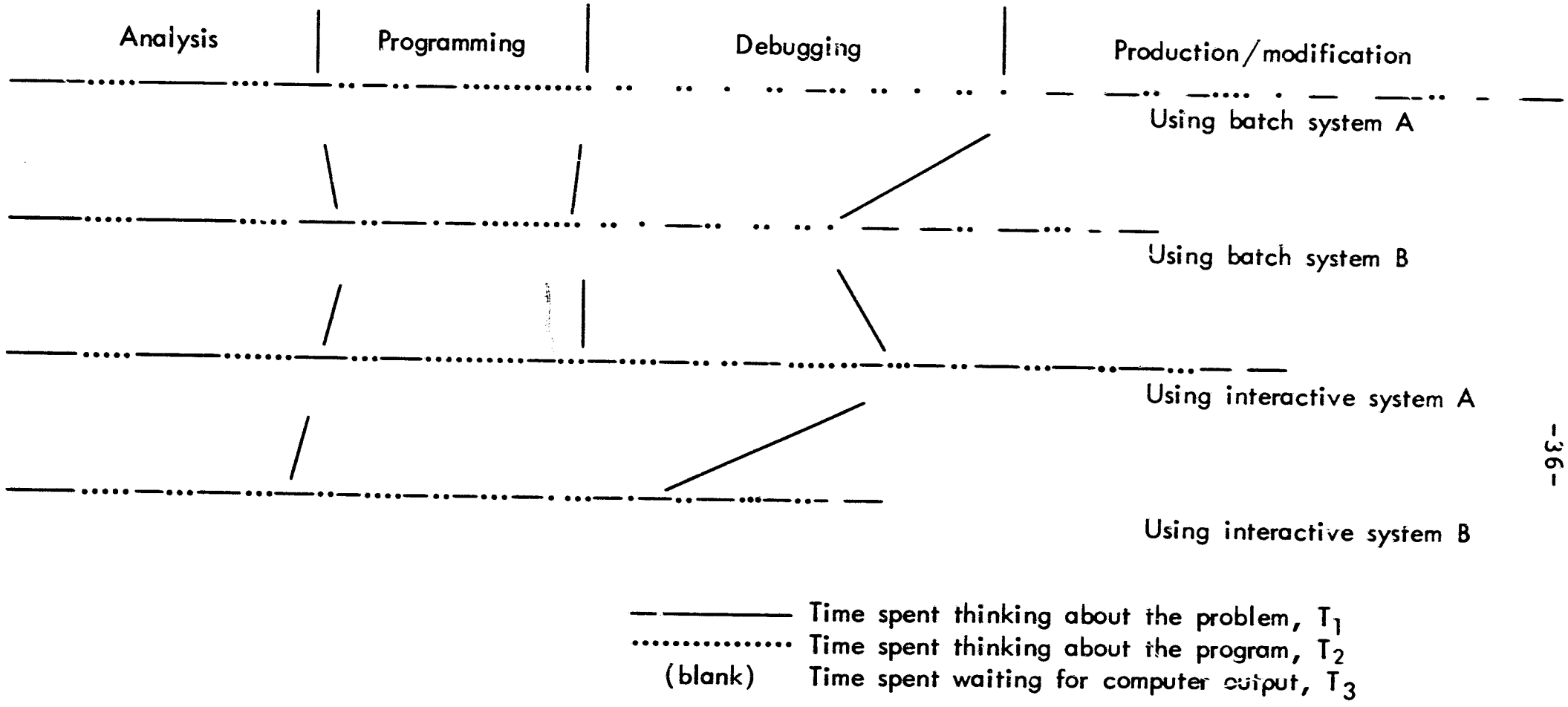


Fig. 16--Time Series Characterization of a Computer-supported Project

T_3 : Time spent waiting for the computer to respond.†

Then consider the following performance criterion, the *productive thought ratio* (P.T.R.):

$$\text{P.T.R.} = \frac{T_1}{T_1 + T_2 + T_3} .$$

A computing system that maximizes the P.T.R. (over some mix of projects) will not only try to increase machine efficiency (by decreasing T_3), but also human efficiency (by decreasing T_2 through reducing the time spent learning languages, programming, debugging, and modifying programs).‡ Also, with the time series characterization of a project, the P.T.R. is a reasonably *measurable* quantity; it requires the currently available machine measurements plus an approximate breakdown of how people use their time.

INVESTIGATE OPERATIONAL IMPLICATIONS; IDENTIFY KEY PROBLEMS

Suppose the manager of a computer system received the following statement:

> "This month our P.T.R. was 0.37. Last month it was 0.29."

What would this statement tell him?

---

†Such a characterization is appropriate for a computer service facility and the maintenance aspects of a real-time control system. It is less appropriate for the operating aspects of a real-time control system, which are better judged directly with respect to the objectives of the system.

‡The P.T.R. is intended to function best in evaluating changes relative to an existing operation. Its current form does not exclude extreme cases that produce unwarrantedly good results. For example, not using a computer yields $T_2 = T_3 = 0$ and P.T.R. = 1, an "optimal" solution. These could be fixed by adding more terms, but this would obscure the subsequent discussion of more fundamental difficulties.

1) *Nothing, unless he was sure the variation was not attributable to changes in workload.* To eliminate this difficulty, some means must be found to normalize the P.T.R. with respect to workload. One possibility would be to measure it only with respect to standard project types (large event simulations, small scientific investigations, multi-tape data analyses, etc.), under the assumption that the projects within each type are relatively homogeneous. At present, no solid data are available to test this assumption, which suggests one potential research area: the collection of detailed case histories of several projects within one of the above standard project types, and their characterization and comparison in terms of such schemes as Fig. 16.

2) *Very little, without some correlation between thinking time and insight.* This correlation can vary markedly for different systems, particularly in such areas as computer graphics.† However, some properly instrumented experiments in man-computer problemsolving could shed some light on the question. This became one of our experimental design considerations.

3) *Nothing, if based on bad measurements.* How capable and how motivated are people to separate their time accurately into categories T_1, T_2, and T_3? Our experimental observations and debriefing forms were structured to pick up such information. As one example of the results, we found that some of the subjects' estimates of time spent waiting for the computer to respond (T_3) were underestimated or overestimated by a factor of at least two, though on the average the agreement with observations was fairly close.

4) *Very little, if a significant number of users are productively using their computer wait time (T_3) on one*

---

†In practice, of course, measures of effectiveness must also be balanced with measures of cost.

project to advance another project (time-sharing themselves). This avenue leads to a host of fundamental questions involving human thinking and problemsolving processes, generally couched in such elusive terms as "concentration," "subconscious," "motivation," etc. At this early stage, the most definitive statements possible are:

- The P.T.R. is not a sufficiently delicate metric to illuminate this phenomenon, and at best its use must be restricted to "dedicated" activities.
- To make any headway with the fundamental questions above, one needs a larger problemsolving data base. Our increased appreciation of this need strengthened the case for performing experiments to gather more data.
- Such phenomena as "lockout" cannot be neatly fitted into the categories T_1, T_2, and T_3; our considerations of the possible effects of lockout led to the major hypotheses to be tested via controlled experiment.

5) *Probably about as much as can any other general man-computer performance criterion at this time.* Most of these enumerated P.T.R. difficulties would arise with any alternative criteria that attempt to assess the computer's contribution to human performance over a wide spectrum of activities. As the spectrum of activities is narrowed, ways can be found around some of the difficulties (e.g., workload variation), but others will remain quite thorny (e.g., accuracy of measurements and value of results).† However, the potential payoffs of even partial insights in this area are sufficient to warrant an increased level of activity in gathering and analyzing man-computer performance data.

---

†Some care must also be taken to avoid criteria that overemphasize the machine-like aspects of human performance (e.g., number of designs tried in a day).

REFERENCES

1. Nielsen, N. R., *ECSS: Extendable Computer System Simulator*, The RAND Corporation, RM-6132-PR, January 1970.

2. Bell, T. E., *Computer System Measurement and Analysis*, The RAND Corporation, R-584-NASA/PR (in process).

3. Watson, R. A., *Measurement and Analysis of Computer System Performance: Applications of Accounting Data*, The RAND Corporation, R-573-NASA/PR (in process).

4. Seven, M. J., B. W. Boehm, and R. A. Watson, *Problem-Solving with an Interactive Computer: A Study of User Behavior*, The RAND Corporation, R-513-NASA (in process).

5. Boehm, B. W., *Some Information Processing Implications of Air Force Space Missions, 1970-1980*, The RAND Corporation, RM-6213-PR, January 1970.

6. Manus, S. D., "SDC Recommendations for Spaceborne Software Management," *Proceedings, First Spaceborne Software Workshop*, 1966, pp. 345-360.

7. Haverty, J. P., *The Role of Programming Languages in Command and Control: An Interim Report*, The RAND Corporation, RM-3292-PR, September 1962.

8. Nielsen, N. R., "Computer Simulation of Computer System Performance," *Proceedings, 1967 ACM National Conference*, pp. 581-590.

9. Bell, T. E., *Video Graphics: A Description for Modeling*, The RAND Corporation, R-519-PR (in process).

10. Boehm, B. W., and R. L. Mobley, *A Computer Simulation of Adaptive Routing Techniques for Distributed Communications Systems*, The RAND Corporation, RM-4782-PR, February 1966.

11. *SCERT: Systems and Computers Evaluation and Review Technique*, Comress, Inc., Washington, D.C., 1967.

12. *S3: System and Software Simulator*, Defense Document Clearinghouse, AD 679269, September 1967.

13. *Computer System Simulator/360*, IBM Document Y20-0130-1, 1967.

14. Kiviat, P. J., R. E. Villanueva, and H. M. Markowitz, *The SIMSCRIPT II Programming Language*, Prentice-Hall, Englewood Cliffs, New Jersey, 1969. (Also, The RAND Corporation, R-460-PR, October 1968.)

15. Sackman, H., *Experimental Investigation of User Performance in Time-Shared Computing Systems: Retrospect, Prospect, and the Public Interest*, System Development Corporation, SP-2846, 1967.

16. Gold, M. M., *Methodology for Evaluating Time-Shared Computer Usage*, Ph.D. dissertation, Massachusetts Institute of Technology, 1967.

17. Baker, C. L., *JOSS: Introduction to a Helpful Assistant*, The RAND Corporation, RM-5058-PR, August 1966.