TM-3628/009/00

COMPUTER-ASSISTED PLANNING

Final Semiannual Technical Summary Report to the Director, Advanced Research Projects Agency, for the period 16 March 1971 to 15 September 1971

Sponsored by the Advanced Research Projects Agency
ARPA Order No. 1327

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
System Development Corporation 2500 Colorado Avenue Santa Monica, California 90406	Unclassified
	2b. GROUP

3. REPORT TITLE

COMPUTER-ASSISTED PLANNING: Final Semiannual Technical Summary Report to the Director, Advanced Research Projects Agency, for the Period 16 March 1971 to 15 September 1971

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Technical Final Semiannual 16 March 1971 to 15 September 1971

5. AUTHOR(S) (First name, middle initial, last name)

Clark Weissman (Project Director)

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
15 October 1971	88	34

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DAHC15-67-C-0149, ARPA Order No. 1327	TM-3628/009/00
b. PROJECT NO. 1D30	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) NONE
d.	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

This final report describes research and development activities carried out by System Development Corporation during the period 16 March 1971 through 15 September 1971 under contract number DAHC15-67-C-0149 with the Advanced Research Projects Agency, Information Processing Techniques Office. The report also summarizes activities for the contract year ending 15 September 1971. The purpose of the activities described in this report, which have been carried out under the title of Computer-Assisted Planning, has been explore the practical implications and potential uses of computer technology in comprehensive military planning. These activities have built on earlier work in Computer-Aided Command and extend the application to military planning problems of the ADEPT prototype military computer utility and time-sharing system. Four tasks are reported on: (1) Computation and Communication Tradeoff Studies, which is developing computer-based models for use in the analysis and acquisition of military computer networks; (2) Natural Computer Input and Output, which is directed at the research and implementation considerations associated with English-language data management, computerized speech recognition and synthesis, and hand-printed mathematical and symbolic input and output; (3) Systems Research, focusing on computer-network integration and communication techniques and on the mathematical foundations for computer-program optimization; and (4) Interactive Systems, focusing on techniques and languages for interactive man-machine problem solving and on the extension of the ADEPT time-sharing system to support increasingly sophisticated interactive terminal capabilities. Progress toward stated project goals during the reporting period is described, and the achievements of the contract year are summarized.

DD FORM 1 NOV 1473

Unclassified

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ADEPT Time-Sharing System						
ARPA Network						
Compiler Technology						
Computer-Assisted Planning						
Computer Networks						
CONVERSE System						
Distributed Data Management						
Distributed Intelligence						
English Data Management						
Graphic Input/Output						
Graph Theory						
Man-Machine Synergy						
Problem Solving						
Program Optimization						
Speech Understanding						

SERIES	BASE NO. / VOL	REISSUE
TM	3628 /009	/00

TECHNICAL MEMORANDUM

(TM Series)

The work reported herein was supported by the Advanced Research Projects Agency of the Department of Defense under contract DAHC15-67-C-0149, ARPA Order No. 1327, Program Code No. 1P10.

COMPUTER-ASSISTED PLANNING

Final Semiannual Technical Summary Report to the Director, Advanced Research Projects Agency

for the period
16 March 1971 to 15 September 1971

C. Weissman, Project Director
Tel. (213) 393-9411

15 October 1971

SYSTEM DEVELOPMENT CORPORATION
2500 COLORADO AVE.
SANTA MONICA
CALIFORNIA
90406

ABSTRACT

This final report describes research and development activities carried out by System Development Corporation during the period 16 March 1971 through 15 September 1971 under contract number DAHC15-67-C-0149 with the Advanced Research Projects Agency, Information Processing Techniques Office. The report also summarizes activities for the contract year ending 15 September 1971.

The purpose of the activities described in this report, which have been carried out under the title of Computer-Assisted Planning (CAP), has been to explore the practical implications and potential uses of computer technology in comprehensive military planning. These activities have built on earlier work in Computer-Aided Command and extend the application to military planning problems of the ADEPT prototype military computer utility and time-sharing system. Four task areas are reported on: (1) Computation and Communication Tradeoff Studies, which is developing computer-based models for use in the analysis and acquisition of military computer networks; (2) Natural Computer Input and Output, which is directed at the research and implementation considerations associated with English-language data management, computerized speech recognition and synthesis, and hand-printed mathematical and symbolic input and output; (3) Systems Research, focusing on computer-network integration and communication techniques and on the mathematical foundations for computer-program optimization; and (4) Interactive Systems, focusing on techniques and languages for interactive man-machine problem solving and on the extension of the ADEPT time-sharing system to support increasingly sophisticated interactive terminal capabilities. Progress toward stated project goals during the reporting period is described, and the achievements of the contract year are summarized.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

(Page vi Blank)

LIST OF FIGURES

LIST OF TABLES

1. SUMMARY

This final report describes the progress and results of System Development Corporation's research in Computer-Assisted Planning for the six months from 16 March 1971 through 15 September 1971. Our research in Computer-Assisted Planning (CAP) has built on earlier work both on Computer-Aided Command and on the development of the Advanced Development Prototype (ADEPT) computer utility and time-sharing system, which serves as the basic CAP support facility.

The CAP program has been directed toward the long-range goal of clarifying and systematizing the strategic planning process, with the eventual objective of constructing an experimental, prototype computer-based planning system. The program's immediate goal has been to explore the effects of improved communications on the use of computers by Department of Defense planners. "Communications," in this context, includes both communication between planners and their computer resources and the flow of communications among computer facilities, particularly those that are time-shared or networked. The program has focused on (1) developing models and procedures that will assist military planners in strategic and tactical planning and in planning the acquisition and use of computation and communications resources; (2) designing and developing prototype communications-interface systems that will enable planners to communicate with computers in ordinary English, by voice, and by means of conventional hand-drawn mathematics; (3) advancing the state of the art of time-shared and networked computer systems in order to contribute to the development and evolution of the ARPA Network, of which SDC is a part; and (4) providing the bases for enhanced interaction between planners and computer systems. These activities, which are summarized below, are the basis for the organization of the body of this report.

Computation and Communication Tradeoff Studies (CACTOS)

The CACTOS project is examining the 1975-80 Department of Defense requirements for computation and communications resources, with particular attention to the tradeoffs between concentrated and distributed computational power. Having focused during the first half of the contract year on the development of a computer-based tradeoff-analysis model and on the selection of an existing military computer network for analysis, the project has spent the past six months refining the model and conducting two experimental analyses--one of the Marine Manpower Management System centered in Kansas City, Missouri, and one of the throughput characteristics of a hypothetical network, using the CACTOS model and several alternative hardware configurations. The project staff has also conferred with military personnel responsible for data-processing planning and operations and has prepared several forecasts of technological developments during the 1975-80 time period. These activities are described in detail in a separate report (SDC document number TM-4743/012/00).

Natural Computer Input and Output

Although communication between men and computers is unlikely ever to be as "natural" as communication between men, we are making considerable progress in the design of communications interfaces that enable computers to process and interpret ordinary English sentences input to them via a tele-typewriter terminal, words and phrases spoken into a microphone, and mathematical expressions and flow diagrams written or drawn on a data tablet.

The CONVERSE project staff has now completed the implementation of Versions 1 and 2 of the CONVERSE system, which incorporate syntactic and semantic extensions to the CONVERSE intermediate language--the language processor that accepts a wide variety of English sentence types, analyzes them into their constituent parts, and passes them to the natural-language compiler, which then translates them into statements in the LISP 1.5 programming language. A major extension of the grammatical-analysis component of the system was achieved in Version 1 by the addition of a series of transformational rules that generalize syntactic analysis and simplify the production of well-formed semantic interpretations of sentences. Version 2 adds to CONVERSE the ability to process both declarative and imperative sentences (in addition to the interrogative sentences that have been the primary form of input to the system in prior years). Version 2 also incorporates features that permit a user to describe and maintain a variety of data-base contents and structures by means of ordinary English sentences. These features represent an important step toward the achievement of an on-line system with which planners can conveniently communicate with data files of substantial size and expanding content.

The Voice Input/Output project, now completing its first full year, is pursuing speech understanding by computer on the basis of the hypothesis that both a human's "inquiry state" and the words and phrases he uses can be dynamically modeled and predicted--an hypothesis that, if it proves valid and can be implemented, will greatly increase the likelihood that the computer will recognize and interpret human speech. During the past year, the project has developed and implemented the basic components of a system that will accept continuous speech. A high-quality audio-recording environment has been provided that allows precise control of speech sampling. Computer interface hardware and software have been developed, and several software modules for recording and producing acoustic waveforms have become operational. With these achievements, the project is now prepared for full-scale research and development efforts directed toward the implementation of a Vocal Data Management System and, ultimately, a vocal CONVERSE.

The Graphic Input/Output project has focused, during the year, on the refinement of existing components of an interactive data-tablet display system, on the development of a new component that recognizes and interprets hand-drawn flowcharts, and on the synthesis of these components into a system, called TAM (The Assistant Mathematician), that processes and evaluates conventional mathematical expressions. The long-range goal of our research in computational graphics is the development of a system that will replace traditional computer-programming techniques which are almost totally tied to expertise in programming languages, with a facility by means of which engineers, planners, and mathematicians (as well as computer programmers) can write programs and input them to the computer via ordinary mathematics, using the conventions that are most suitable, or even idiomatic, to the types of problems to which their programs will be addressed. Although such a system is still only a vision, our success in developing and implementing TAM gives us confidence that it is a vision capable of realization.

Systems Research

Our systems-research activity encompasses two projects: the Networks project, which is focused on integrating the ADEPT system into the ARPA Network and on making available to other members of the Network our natural input/output facilities, and the Graph-Meta project, concluding this year, which has focused on the applications of graph theory to the optimization of computer-program compilers.

During the past six months, the Networks project staff has completed Version 2 of HOSTOSS, the subsystem of ADEPT that interfaces with the ARPA Network. HOSTOSS permits programs running under ADEPT to communicate both with each other and with programs running elsewhere in the Network. Together with our TELNET program, it also permits SDC users to log in on systems running at remote Network nodes and makes ADEPT available to as many as five remote users. The Networks staff has also completed the initial phase of a study whose purpose is to define and implement techniques that will permit the sharing of data by different data-management systems within a computer network. The search for a "common" data-management language that would permit data exchange by disparate data-management systems has long been a dream of computer-science researchers, and, were such a language found, it would have a major impact on military user agencies, to whom the quick and efficient communication of computerized data is essential. Although we are yet in the early phase of this study, it does appear that the CONVERSE natural-language compiler can be adapted to serve as such a common language, and that translation modules could be implemented at the various nodes of a computer network to process requests for data stated in the CONVERSE intermediate language. With this initial investigation completed, the study will be focused during the coming year on data-sharing experiments with other members of the ARPA Network.

Several years of research into the theoretical foundations of optimizing compilers--compilers that, in addition to transforming programming-language statements into machine code, carry out operations that render the code maximally efficient--have culminated in the completion and draft publication of a textbook describing the theory and its applications, particularly to programs written in the FORTRAN IV programming language. Although other researchers have devoted themselves to the problems of program optimization, we believe that our work is the first to develop a rigorous and viable mathematical foundation for their solution. The most dramatic practical implication of this achievement is that it may now become possible for engineers, scientists, and planners who have only modest skills in computer programming to write programs whose efficiency approaches that of programs written by the most expert professional programmers.

Interactive Systems

Our work on Gaku, an on-line, interactive system designed specifically for use by planners, concluded this year with the completion of the User Adaptive Language (UAL), which is the vehicle for expressing and modifying the conceptual structure of complex planning problems. A complete description of the language and its applications has been published. In addition, during the past six months, we have published the results of an earlier extensive program of experiments (co-sponsored by ARPA and the Office of Naval Research) in human intellectual problem-solving behavior.

As our research in natural input/input and computer networking progresses, increasingly complex demands are placed on the ADEPT system. The task of responding to these demands while maintaining ADEPT as a highly reliable time-sharing facility is the responsibility of the Time-Sharing project, which this year increased the capabilities of the Programmed Multiline Controller (PMLC), a hardware/software complex, centered in a Honeywell DDP-516 computer, that serves as the interface between our interactive terminals and the ADEPT system. The PMLC is now capable of supporting terminals operating at rates of 10, 30, and 60 characters per second with a dramatic reduction in overhead and storage requirements. A capability was also added that will support five users at remote nodes of the ARPA Network, and the planning work to increase the number of interactive ADEPT users from 10 to 15 has been completed.

Finally, our work to extend the capabilities of our LISP 1.5 system, which is the primary support system for CONVERSE, concluded with the implementation of several techniques for providing additional space in the system for its users' program data.

2. COMPUTATION AND COMMUNICATION TRADEOFF STUDY (CACTOS)

The goal of the Computation and Communication Tradeoff Study is to determine specific DoD requirements for computation and communication networks on a regional, functional, and categorical basis for the 1975-80 time period. With the continuing advance of technology in computer hardware, software, and communications, it is important that an overall analysis be done of DoD requirements in these areas. Implicit in such an analysis are the tradeoffs between various characteristics of computer and communication networks, which must be examined several years prior to the procurement of equipment and facilities, and which must be related not only to cost and time, but to each other. Such a tradeoff study must be at least in part quantitative and be capable of wide usage so as to relate to specific network configurations within DoD.

To achieve this goal, the following objectives have been identified:

1. Determine amounts of data processed, stored, retrieved, and transmitted, including response time and tradeoffs between security, vulnerability, throughput, reliability, cost, and time.

2. Construct analytic models for evaluating and modifying selected networks.

3. Perform tradeoff studies based, in part, on the results of the model analysis.

4. Validate the analysis using an existing military network.

5. Describe future technology in areas relating to computers and communications.

The interrelationships of these objectives are illustrated in Figure 2-1.

2.1 PROGRESS

During the past six months, the CACTOS project has focused on two groups of experiments, in which the prototype network-analysis model developed earlier in the year was used to study (1) an existing military computer network (the Marine Manpower Management System headquartered in Kansas City, Missouri) and (2) the effects on throughput in a hypothetical network of variations in hardware configurations at its nodes. The detailed results of these experiments and a detailed description of the model as it is presently constituted are contained in separate reports listed in section 2.4; the experiments are summarized here.

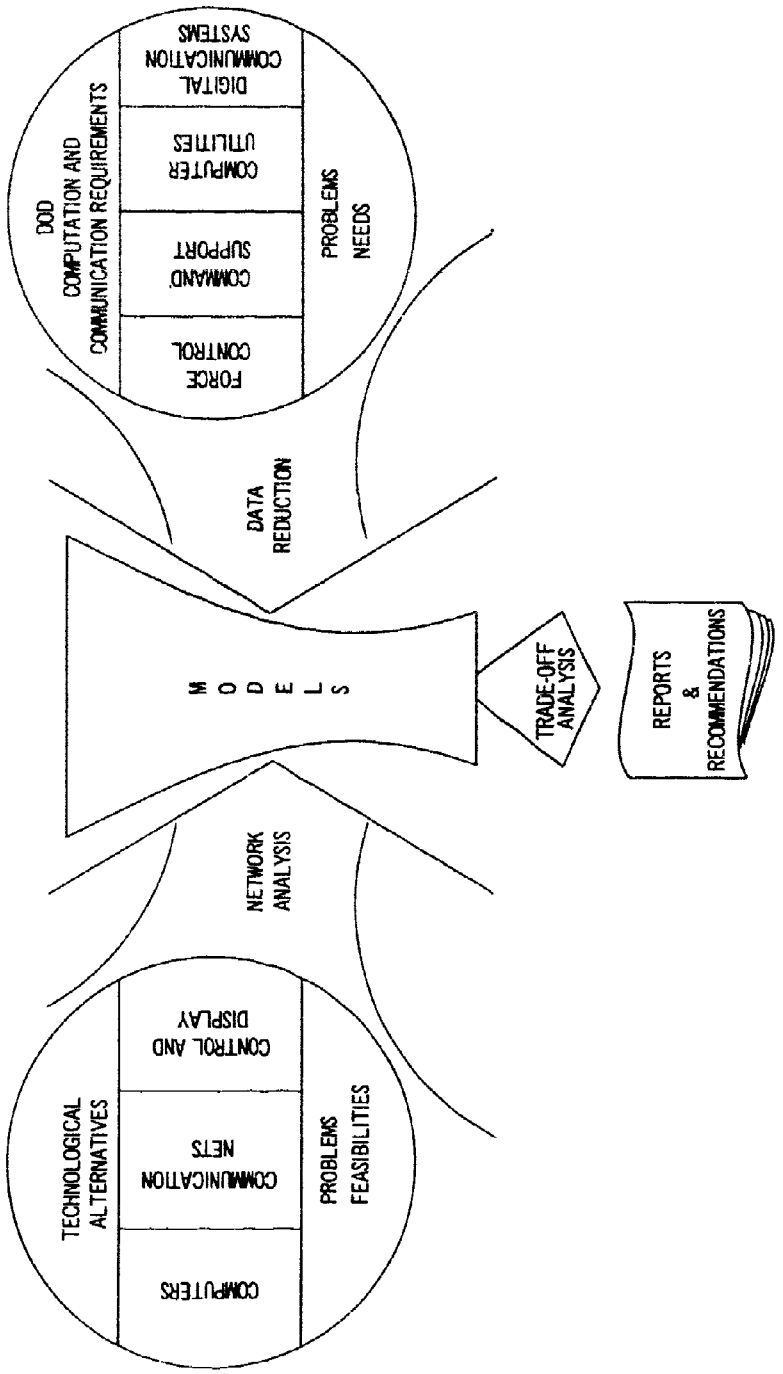


Figure 2-1. CACTUS Overview

Marine Manpower Management System (MMS) Experiments

The experiments with the Marine Manpower Management System (MMS) had a dual purpose: first, to validate the prototype network-analysis model by testing it against the operations of an existing military data network; second, to produce immediately practical results of the CACTOS activity.

The Marine Manpower Management/Joint Uniform Military Pay System (MMS/JUMPS) is centered in the Marine Corps Automated Service Center (MCASC) in Kansas City, Missouri, with satellite Data Processing Installations at Marine Corps bases in the continental United States and overseas. It is a part of AUTODIN. A major part of the study of MMS/JUMPS was to discover improved topologies and/or functional allocations of tasks and traffic to improve the timeliness of system performance. Of perhaps greatest concern is the potential 5-to-10-day response delay in processing a change throughout the system. Using the MMS/JUMPS data, the model gave results for delays within 5 percent of actual figures. At the request of Marine Corps Headquarters, the CACTOS project compared the MMS/JUMPS data with that drawn from a dedicated system outside AUTODIN. Table 2-1 presents the results of the comparison. Column 1 in the table shows the computer times taken at MCASC, and, on an average, throughout the network, to process MMS/JUMPS data in a system dedicated to Marine personnel business alone. Column 2 shows the results of loading the system up to a 90 percent utilization rate. Not much change is seen at MCASC, where the computer is already quite busy with MMS applications. However, the network average jumps beyond the MCASC response time, largely because wait time is larger for the smaller nodes.

Table 2-1. Dedicated Versus Public Operations

	Dedicated MMS Only Hrs/Min	Shared 90% Load Hrs/Min
Single Message	:11	:37
All Messages	1:10	19:25
Network Average		
Single Job	1:06	3:59
All Jobs	16:50	27:04
MCASC Average		
Single "Job"	:31	1:33
All Jobs	20:42	23:34
Total Response		
MCASC	21:52	46:29
Network	18:00	42:59

CACTOS also studied the effect of having a priority assignment for certain percentages of jobs. Experiments were run for priority processing for jobs and messages of 10 percent, 20 percent, and 30 percent of the MMS jobs. It was observed that even assuming a 20-percent penalty for processing the priority tasks, enough wait time is avoided to decrease total response time somewhat. The network average, where the MMS load is less than MCASC, benefits more from prioritizing than does MCASC, which has a substantial MMS computation load.

Experiments were also run on a distributed processing system split between MCASC and HQMC. Although there was some improvement, it was not judged significant as compared to cost.

Throughput Experiments

The other major part of the CACTOS activity during the past six months was a set of computer-throughput experiments with the CACTOS model. The purposes of these experiments were to:

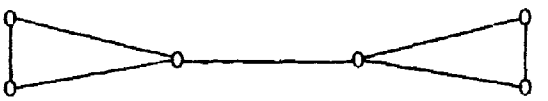
1. Evaluate the usability, credibility, and potential application value of the assumptions underlying the model, including those related to queueing.

2. Gain insight into the effect of controllable computer hardware characteristics on system performance.

3. Discover the effects of hardware throughput on job (software) characteristics.

4. Demonstrate the effectiveness of the computer-throughput model as a tool for use in developing computer selection and replacement strategies.

To concentrate on the computation characteristics of a network, communication characteristics (such as job load) were fixed. The network configuration, as illustrated below, consisted of six nodes and seven fully duplexed lines.



Twenty-five users were assumed at each node, 50-kilobit lines were used throughout, all internode distances were assumed to be 1,000 miles, the average job size was 1,000,000 modified bits, and the average message size for all jobs was 400 bits. Computer hardware, while varied from run to run, was assumed to be the same at all six nodes during any given run. Twenty-five hardware configurations were tested, each against five types of jobs, for a total of 125 runs. To simplify cost calculations and comparisons, all equipment was selected from the IBM 360 line. A variety of core, CPU, and disc matches were run. Jobs varied according to overlap charactersitics and the relative amounts of CPU and I/O required. Figures 2-2 through 2-6 show the results of the runs for job types A through E. The vertical axis represents total response time per job; the horizontal axis represents system cost. Only the varying and pertinent costs were used, which included (1) the CPU with the appropriate amount of core; (2) the disc unit; and (3) the appropriate controller, if any, for interfacing with the disc unit. Costs were computed from actual monthly lease rates, as quoted in Auerbach's Standard EDP Reports. Each point on a job's graph represents an expected response time for that job on a given system and is located according to the cost of the system.

The graphs have been found useful for at least three types of applications:

1. Developing general insights into hardware cost-performance relationships and tradeoffs.

2. System selection guidance.

3. System replacement and upgrading guidance.

The graphs exhibit a hyperbolic relationship between cost and response time. The asymptotic lines of least possible cost and best possible performance may be quickly located and understood as boundaries among the alternatives. The shell of dominant points (such as 1, 5, 2, 6, 10, 7, 11, 15, 23, 24, and 25 in Figure 2-4), which represent the alternatives with the greatest overall cost-effectiveness, are readily located, and all other choices might logically be eliminated from further consideration.

Detailed inspection reveals other interesting points. As an example, on the graph (Figure 2-5) for Job Type D, we see that the lines reflecting changes in CPU speed are nearly horizontal. This means that for this type of job and in this range of equipment, performance cannot be significantly improved simply by getting a faster CPU. The only situation in which such a change would be justified would be to get a computer for which more core or faster peripherals are available. Note that this situation does not prevail for job types that have no I/O-CPU overlap--a result very much to be expected.

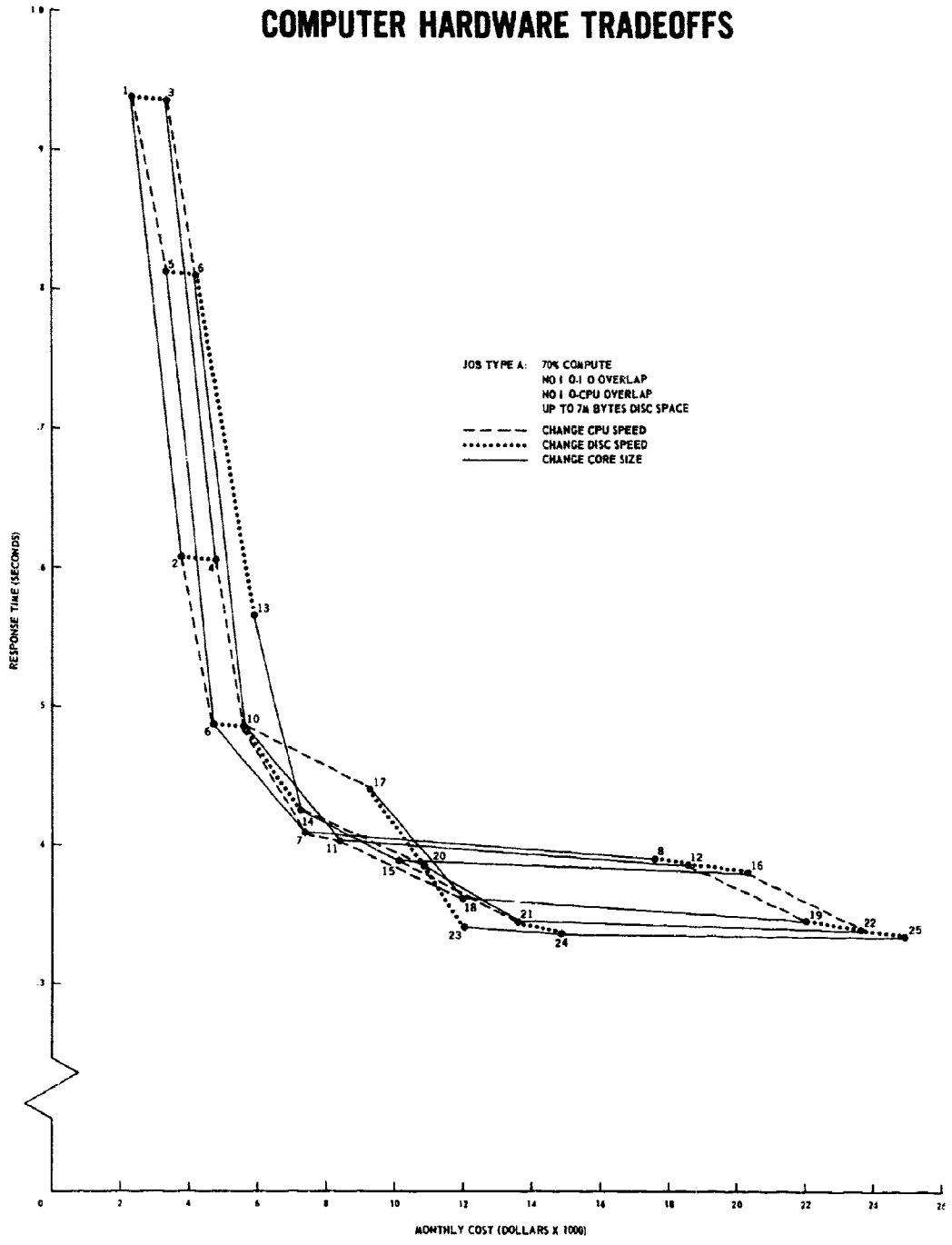


Figure 2-2. Computer Hardware Tradeoffs: Job Type A

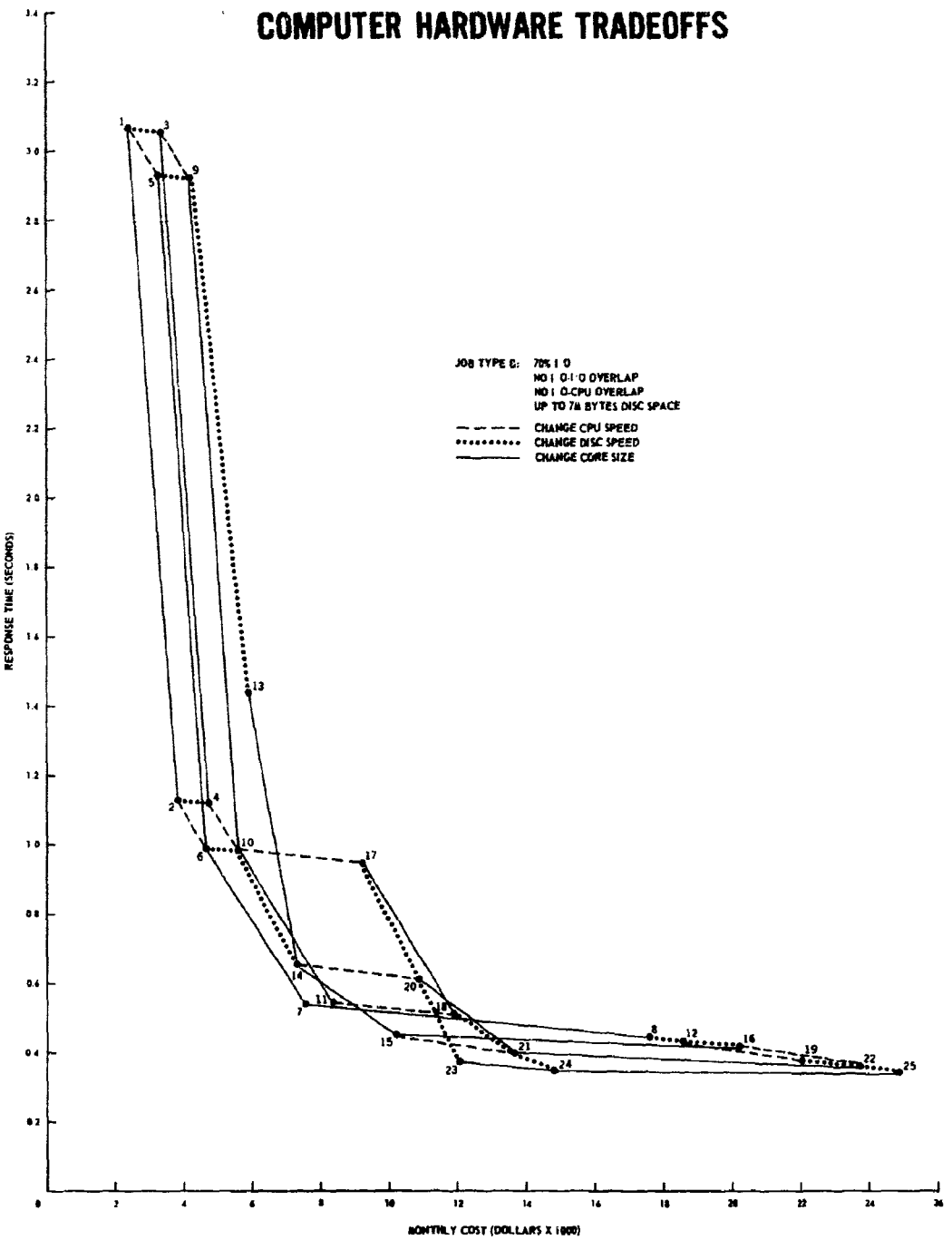


Figure 2-3. Computer Hardware Tradeoffs: Job Type B

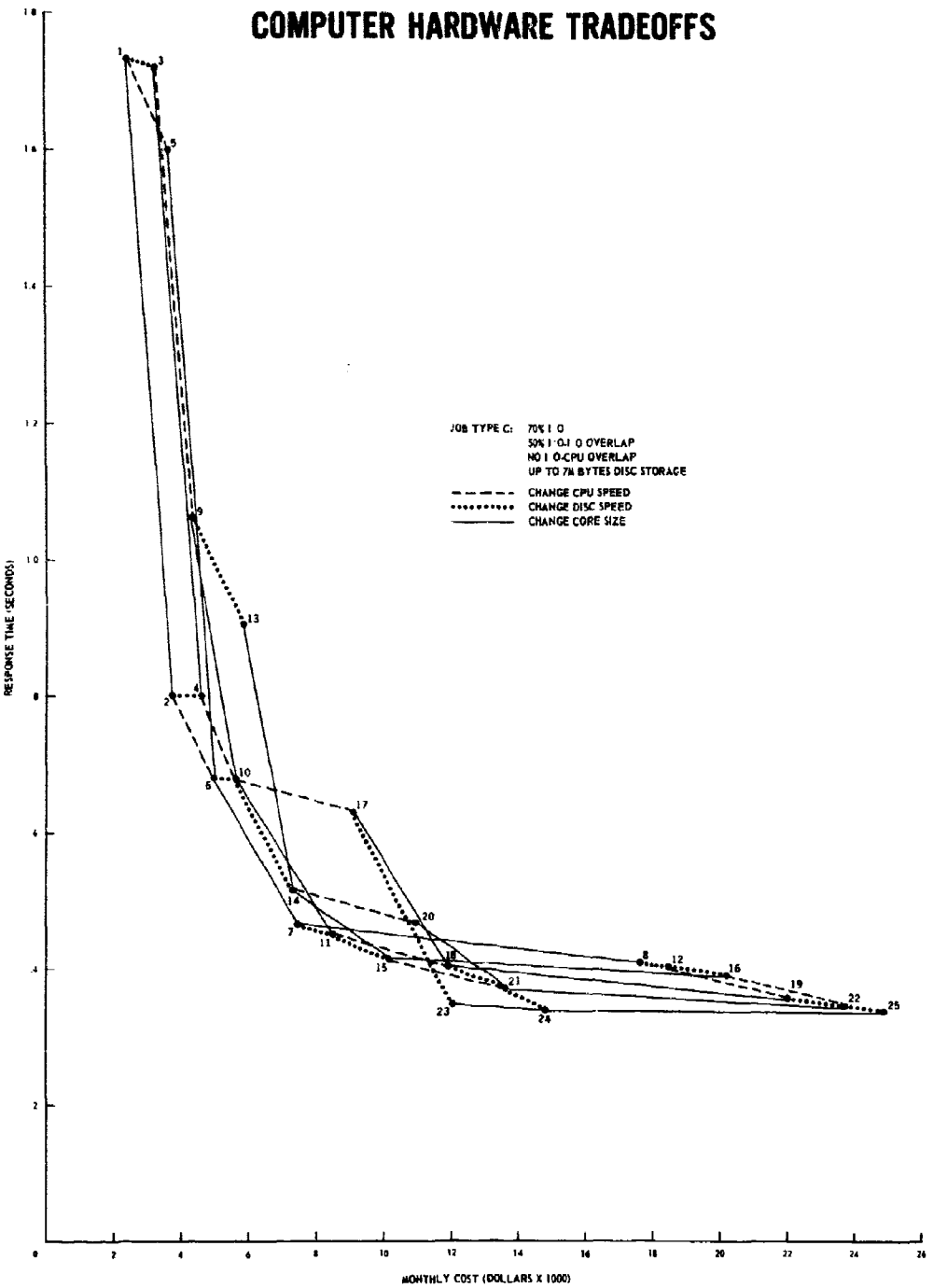


Figure 2-4. Computer Hardware Tradeoffs: Job Type C

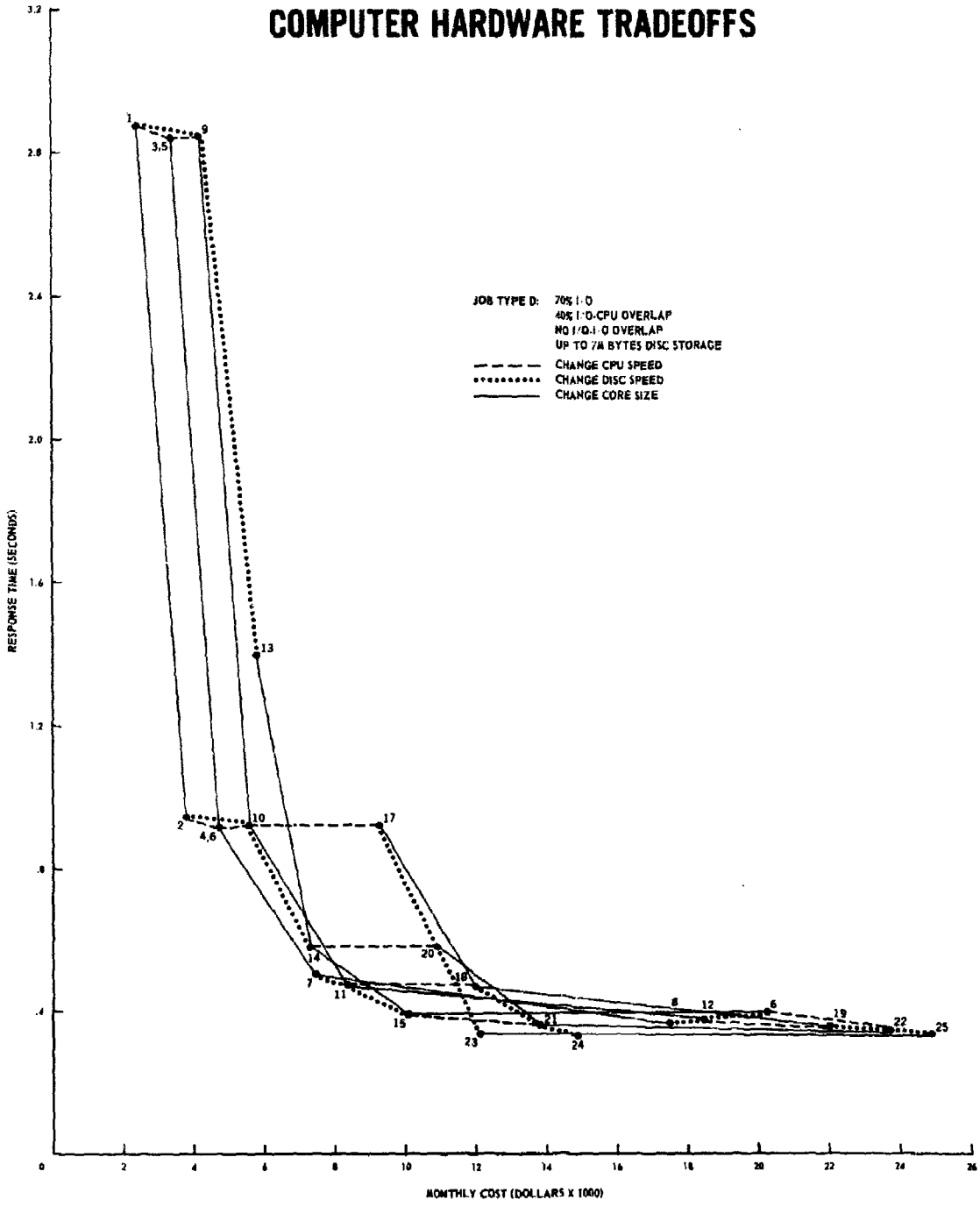


Figure 2-5. Computer Hardware Tradeoffs: Job Type D

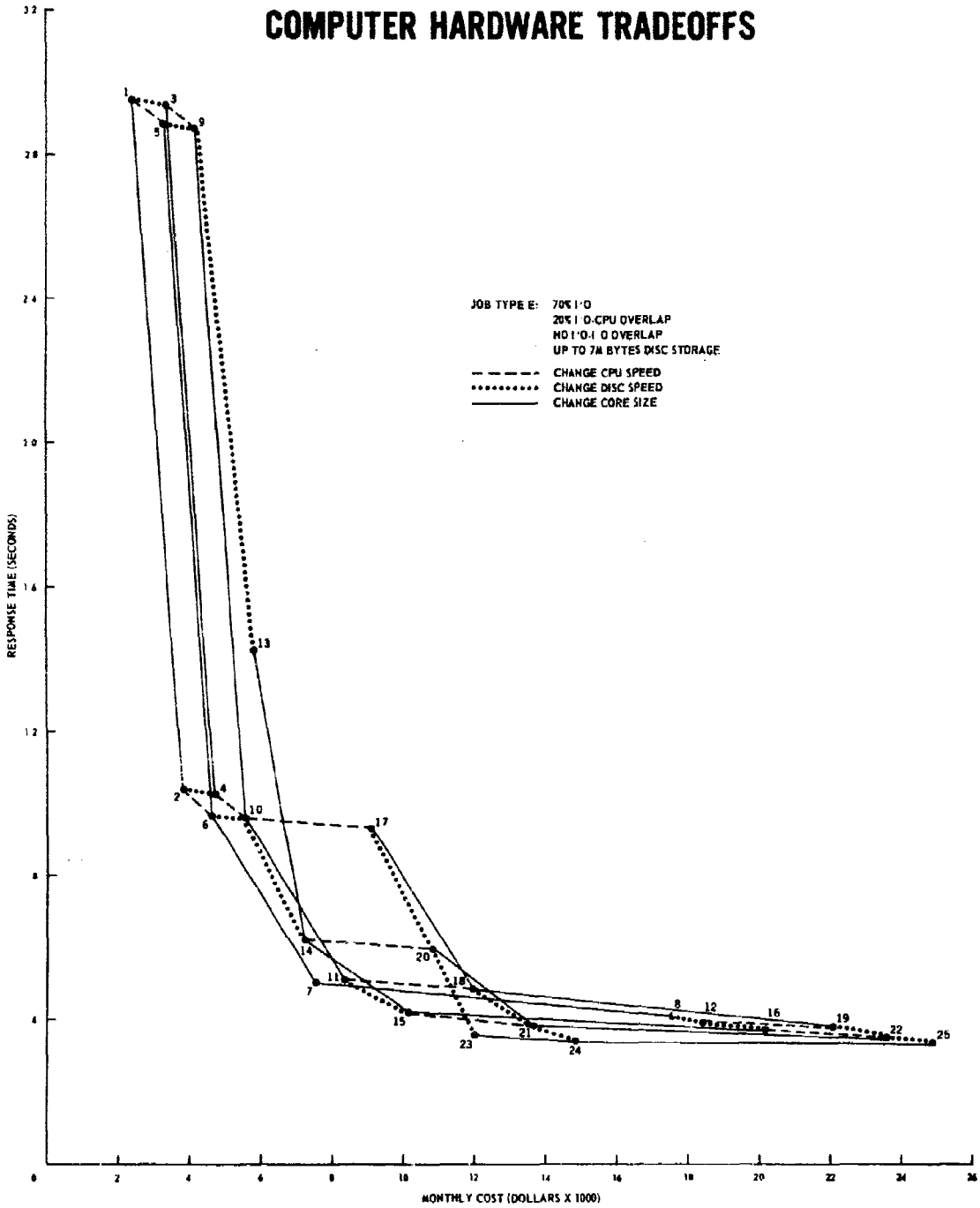


Figure 2-6. Computer Hardware Tradeoffs: Job Type E

The graphs can be used as guides to system selection. If a fixed requirement is known, the user can simply pass a straight edge from left to right along the graph until the first point that meets the requirement is encountered. For example, if a requirement for Job Type D (Figure 2-5) is a response time of no more than 0.4 seconds, the first point encountered is configuration 15, which would be the least costly selection that meets the requirements. Similarly, if a fixed price limit is known, a straight edge can be moved upward until a point to the left of the price limit is found. This will indicate the "best" system configuration at that cost. For Job Type D, the best system at less than $8,000/month is configuration 7, which costs $7,485/month and has a response time of 0.485 seconds. If neither requirement limits nor cost limits are set, the dominant points may be examined and further tradeoffs among them may be performed. (A dominant point on these graphs has no points both to the left and below it.) The lines on the graph indicate optimal ways in which existing systems may be upgraded or incrementally replaced. At least two approaches may be used here, depending on whether or not there is a long-range goal. If there is no long-range goal and the system analyst is concerned only with making the most cost-effective improvement to his existing system, the graph clearly indicates what the available options are and which one is likely to be the most cost effective. If the point that corresponds to an existing system is viewed as a decision point, and if the set of lines leading from that point to an improved performance value represent the possible decisions, then the best decision corresponds to the line that lies to the right of all the others when viewed from the decision point. This strategy was used to generate the chains of improvements from configuration 1, shown in Figure 2-7.

On the other hand, if a long-range goal can be established, this step-by-step approach will not ensure that the goal will be reached. Thus, occasional non-optimal steps may have to be taken to reach the goal. In this case, it is just as important to work backward from the goal as forward from the existing system to find the path that represents the best solution. Examples of the use of this strategy, in which the owner of configuration 1 wishes to incrementally improve his system until configuration 23 is attained, are shown in Figure 2-8. Note that if the look-ahead method is not used, configuration 23 is never attained; only by the use of locally non-optimal decisions may the globally optimal path be followed. Note, also, that not all the solutions are the same, either for the look-ahead or no-look-ahead case. For example, a user with a 40G computer and 2314 discs (configuration 11) should increase his core size with Job Type D to get a 40I computer (configuration 12); with Job Type C he should, instead, move to a 50G computer (configuration 18).

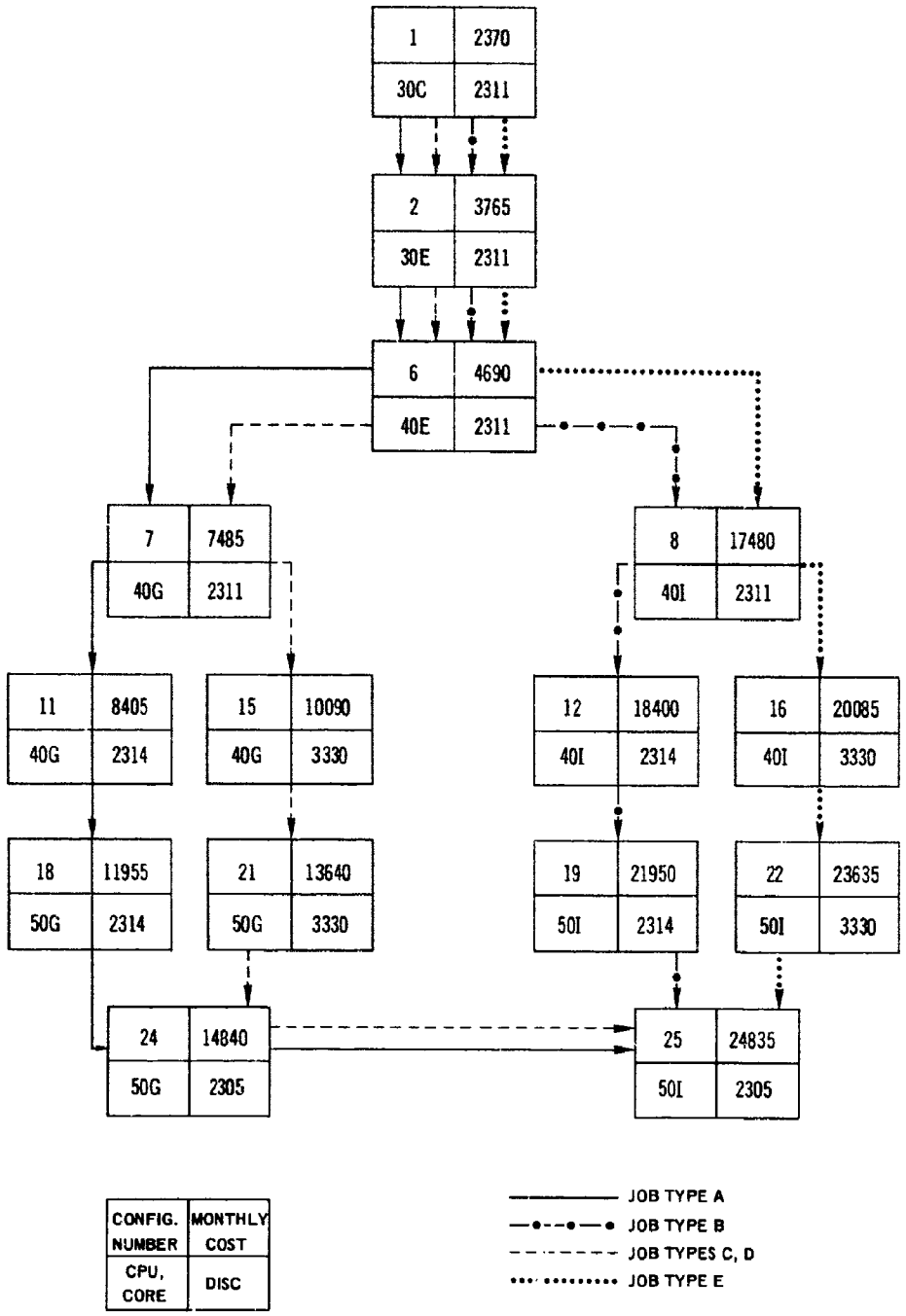


Figure 2-7. Incremental Improvement Plan Without Look-Ahead

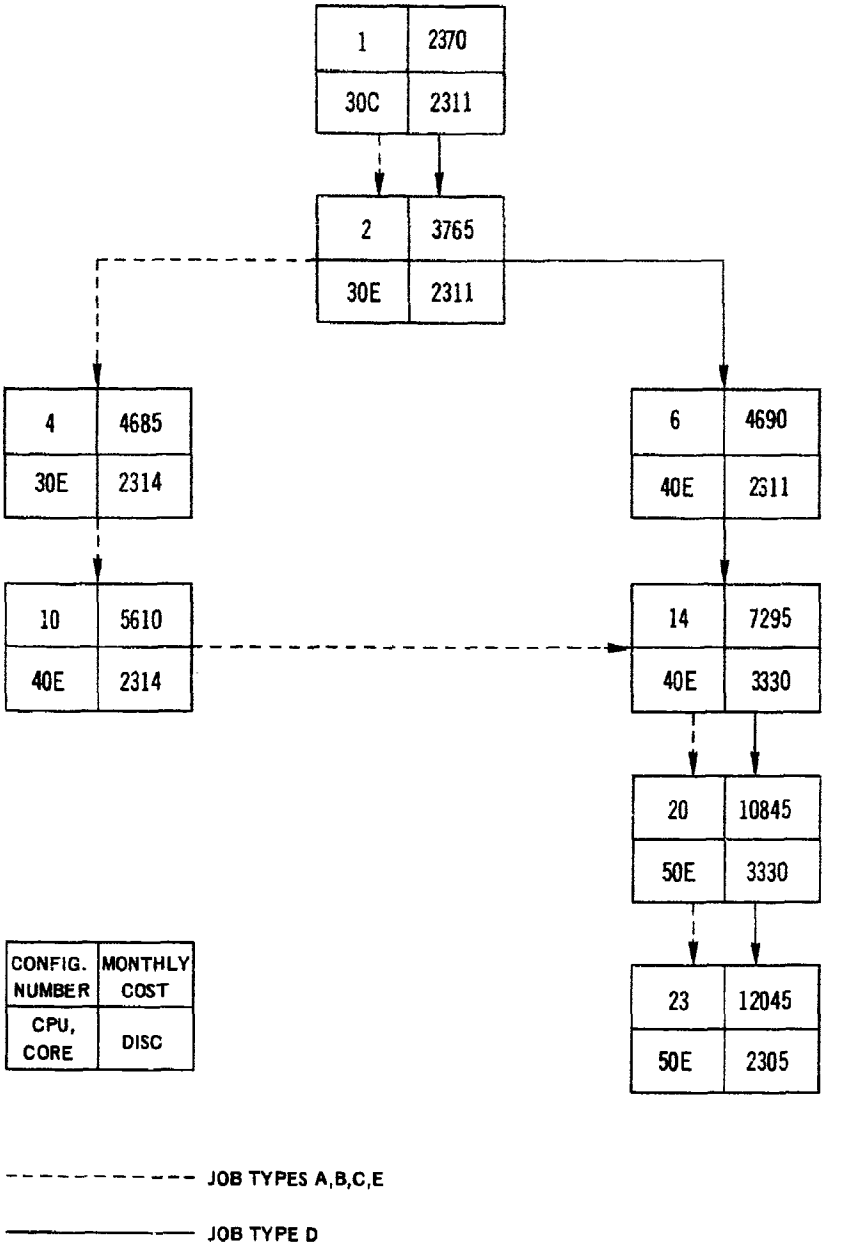


Figure 2-8. Incremental Improvement Plan With Look-Ahead

A common reason for upgrading a system is to handle increased workload as well as improve system performance. While the graphs used here represent a constant workload, they could be modified to reflect a workload that is increasing with time, perhaps by defining a new performance measure based on, say, throughput and the number of users. The results of these experiments point to the following areas in which additional useful work might be performed in the future:

- The study can be extended to include tradeoffs among more hardware parameters. The three choices considered here (CPU, core size, and disc units) represent only a small part of the spectrum of possibilities.

- Communication parameters can be traded off against computation parameters in a study of the interactions of these two on total system performance, using communication and computation hardware that is actually available today.

- For the sake of credibility, the results should be verified by a parallel study using simulation techniques. This is standard procedure when a completely new approach and new equations are being tested for the first time, as is the case here.

- A refinement of the planning techniques outlined here could be programmed on the computer, whereby the computer could select optimal paths through the "alternatives graph" and automatically guide the user toward system optimization.

2.2 SUMMARY

The basic goal of CACTOS is to advise the Department of Defense on the most cost-effective means of meeting its future data-processing needs. During the past year, through tradeoff analyses of hardware, software, and communications characteristics, CACTOS has shown that improvements can be attained through tradeoff studies that focus on throughput, response time, cost, reliability, and similar measures of performance. In particular, incremental improvements phased over time can be analyzed and projected. Results from analyzing the Marine Manpower Management System and the ARPA Network support the validity and accuracy of the CACTOS network-analysis model. The throughput model has, to date, incorporated several important hardware properties. The technology state-of-the-art studies, although not the central focus of CACTOS, have provided guidance for studying future improvements in hardware, software, and communication systems.

In the coming year, the working prototype model will be expanded to enable the storing of configurations constructed on-line, the handling of priority traffic,

and the capability of processing larger numbers of centers, lines, and data. The major emphasis will be placed on a simulation effort sensitive to both hardware and software characteristics. The simulation results will be incorporated into the modified CACTOS model. With the simulation, the model can be further expanded to perform switching and routing analyses as well as more detailed throughput studies. Because of the need for probing both existing and proposed DoD networks, CACTOS will continue to gather data and perform experiments, with frequent review by the relevant military agencies. The next year's work will include a preliminary review of the integration of existing networks for performance enhancement.

2.3 STAFF

Dr. B. P. Lientz, Principal Investigator

G. Cady, Programming Task Leader
D. Lashier
L. Waul (part time)
D. Reingold (part time)

Dr. R. L. Citrenbaum, Analysis Task Leader
Dr. L. G. Chesler
Dr. R. E. Marsten
Dr. R. B. Parente

Dr. N. E. Willmorth, Data Collection and Modeling Task Leader
D. Gunn (part time)
R. Mosier (part time)
C. Weitzman (part time)

2.4 DOCUMENTATION*

Cady, G. M. An Approach to Computer Throughput Analysis. SDC document TM-4743/006/00. July 1971.

Cady, G. M. Computer Network Models: Packets, Acknowledgments, and Other Considerations. SDC document TM-4743/011/00. August 1971.

Citrenbaum, R. L. An On-Line Model for Computation-Communication Network Analysis and Modification. SDC document TM-4743/003/00. May 1971.

Citrenbaum, R. L. CACTOS Experiment Results. SDC document TM-4743/008/00. August 1971.

Citrenbaum, R. L., and L. G. Chesler. CACTOS Experiment Specifications. SDC document TM-4743/007/00. July 1971.

Gunn, D. M. Survey of Digital Data Communications. SDC document TM-4743/005/00. May 1971.

Mosier, R. A. Memory Organization and Addressing. SDC document TM-4743/004/00. May 1971.

Weitzman, C. Aerospace Computers and Peripherals. SDC document TM-4743/010/00. July 1971.

---

*These documents are not available for public release.

Willmorth, N. E. CACTOS Overview 1970-71. SDC document TM-4743/012/00. September 1971.

Willmorth, N. E. Minimizing Network Costs. SDC document TM-4743/001/00. May 1971.

Willmorth, N. E. The Case for Distributed Intelligence. SDC document TM-4743/002/00. May 1971.

3. NATURAL COMPUTER INPUT/OUTPUT

As the power of both computer hardware and computer software increases and the price decreases, the computer becomes more omnipresent as both a tool and a necessity in our daily lives. As the expanding circle of contact between men and computers continues to increase, the disparity between man and computer in communications capability becomes more evident and less tolerable. Therefore, it behooves us to divert some of the available power of computer systems to mediate the communication gap and provide computer input and output systems that are "natural" and convenient to men. Toward this end, the Natural Computer Input/Output task is providing research and development in computer processing and semantic interpretation of natural English, hand-drawn pictorial and symbolic input, computer-generated images, speech understanding by the computer, and computer-synthesized speech.

The Natural Computer Input/Output work is divided into three projects: the CONVERSE project (the development of an English data management system), the Voice Input/Output project (speech understanding and synthesis), and the Graphic Input/Output project (recognition and utilization of hand-drawn and hand-printed input). The interdependence between the Voice Input/Output and CONVERSE projects is both obvious and natural, and--though the two projects are at different levels of attainment--communication, cooperation, and commonality of basic intent are uppermost in their direction. The Graphic Input/Output project's primary concerns are with information whose content cannot be readily conveyed by either the spoken or the written word, but rather through pictorial or notational conventions best portrayed and conveyed in two dimensions. The major emphasis of this work has been on hand-printed input and computer-generated output of mathematics, developing applications requiring mathematical notation, and extending the notational capability into other domains.

Taken as a whole, the Natural Computer Input/Output task is providing the technological basis for operational man-machine systems for which the ultimate end user will require little, if any, special training in computer science.

3.1 CONVERSE: AN ENGLISH DATA MANAGEMENT SYSTEM

The goal of this project is to realize a conversational on-line English data management system that will allow a computer user to manage and maintain complex data files of substantial size and to retrieve data from these files through the computer's processing of user-specified English sentences. Successful achievement of this goal will eliminate current on-line communication restrictions due to the severe compromises existing in today's artificial query languages--compromises between expressive power on the one hand and ease of communication on the other.

Our long-range objective is the construction of a language processor capable of carrying out sophisticated syntax-analysis, inference, and semantic-interpretation procedures in order to provide the user with a virtually unconstrained English subset, one that he can use effectively to carry out all data management tasks. This objective is being attained through the construction of successively more powerful versions of the CONVERSE prototype system.

The shorter-range objectives pursued during this contract year include: (1) the ability to recognize a substantial subset of English sentences; (2) the ability to efficiently store and search large quantities of conceptual and factual information; and (3) the implementation of user-dialog-enhancement features such as user feedback and user extensibility of the English subset. Particular emphasis has been placed on developing a versatile prototype system, one that is potentially applicable to the management of a wide variety of data file types including files of formatted, complex relational, and textual information.

In the first six months of this contract year we completed and demonstrated version zero (V0) of CONVERSE. This system was implemented using the 46-page configuration of our LISP 1.5 programming system under ADEPT on the IBM 360/67 computer. V0 could process questions of moderate complexity and produce answers from a census data base of approximately 4,000 facts about California cities. A major step forward occurred during this time when we were able to generate 85-page configurations of LISP 1.5 and use them in the construction of more powerful versions of the CONVERSE natural-language compiler and data management system.

3.1.1 Progress

During the past six months we have constructed and demonstrated versions 1 and 2 of CONVERSE. Version 1 (V1) permits English on-line question answering of a 12,000-fact data base; version 2 (V2) adds to V1 the capability to process declarative and imperative sentences, which permits a user to describe and maintain data bases in English as well as interrogate them. Both versions employ an expanded natural-language compiler that produces procedural semantic

interpretations from deep syntactic structures. The key areas in which progress has been made are: dictionary and data base development, English analysis, question answering, user feedback, English assertions and commands, and advanced language processing developments. The remainder of this section describes progress in these areas.

Dictionary and Data Base Development

Each user of CONVERSE is now provided with a dictionary comprising several hundred English function words ("of", "the", etc.) and a few basic conceptual primitive terms (e.g., "person", "thing", "relation"). With the aid of this dictionary and recently developed DECLARE commands, the CONVERSE vocabulary and concept net can be expanded, and data bases of various content and structure can be described. The number of syntactic features present in many lexical entries in the dictionary has been increased to facilitate case assignment. The dictionary file can now be compiled with any of three other content-word dictionary files containing open-class words and concept-net information associated with different data bases.

The census data-base file currently consists of 12,000 facts concerning cities in California and New York. This file was used in the demonstration of CONVERSE V1 in June. For purposes of testing the new DECLARE commands that have recently been added to the intermediate language, a smaller census data base of roughly 150 facts has been created. DECLARE commands tell the data management system how to create hierarchies of concepts in the concept net, how to assign internal proper names to objects and concepts, how to insert new objects into the universe of discourse, and how to build set extensions for classes, properties, functions, and relations.

For some time, we have been using and expanding a file of English sentences to facilitate checkout of the parser (the syntax-recognition component of the natural-language compiler). A special subset of sentences tailored to the census data base permits a quick checkout of the parser's ability to recognize sentences querying this data base. A much larger set of sentences is used to test the parser's syntactic generality. In the past year, the following categories of sentences have been expanded: declaratives, yes/no questions, informative questions (which, where, etc.), conjunctions, quantification, relatives, superlatives, complementation, and articles. A special emphasis has been placed on comparatives and is reflected in a large number of comparative sentences. The file now consists of about 500 English sentences illustrating a wide range of parsing, storage, and retrieval phenomena.

A recent extension of the dictionary-building procedures has made it possible for CONVERSE to handle information about symbolic objects that do not have short alphanumeric names. This change permits the description, storage, and retrieval of complex symbolic objects such as text strings (sentences, phrases,

document abstracts, etc.) or arbitrary LISP symbolic expressions (functions, grammar rules, subroutines, etc.). The first use of this extended dictionary facility will be to introduce the file of sentences as complex objects to be described in a new data base for CONVERSE. This sentence file has already been described in English and in CONVERSE intermediate language (IL)--i.e., sentences are sorted into applicable attribute classes such as negative, comparative, declarative, etc. Version 2 allows the user to refine any of these subclassifications, and the system is being extended so that English commands can generate actions that will result in calls upon the parser itself. In the future, therefore, the sentence data base will be usable in contexts in which a user can make statements such as "Parse all yes/no questions containing restrictive relative clauses!" This will lead to the retrieval of the appropriate subset of sentences, which, in turn, will be passed on to the parser one at a time. This sentence data base will allow us to use the system in a reflexive or "boot-strapping" fashion to further increase the generality of the English-analysis procedures.

English Analysis

The CONVERSE parser employs a bottom-to-top, right-to-left, nondeterministic algorithm to produce all possible parsings in a single pass. During the past six months the parser has been greatly enlarged in scope and capability, most importantly by the addition of a series of general structure-changing (SC) rules that efficiently carry out transformational operations on tree structures output by a series of phrase-structure-building (SB) rules. The SB rules are nondirected, and the SC rules are called from within the SB rules. This allows the two types of rules to operate in such a fashion that deep and surface structures are built up simultaneously.

The new parser was first demonstrated in CONVERSE V1 in June. Since then, the SB rules have been expanded to handle additional syntactic constructs such as quantifier floating ("Are all the cities smoggy?", "Are the cities all smoggy?"), preposed adjectives ("Is Downey as large a city as Fresno?"), and a number of additional forms of nominalization, negation, complementation, clefting, tag questions, and verb-particle constructions. Rules have also been added that permit the user to query the structure of the concept network and introduce new terms and phrases during the process of data base description.

The SC rules create deep structures in the form of verbal predicates followed by series of actant arguments. This form facilitates the later application of semantic rules to create IL expressions. These rules have been expanded to create underlying structures out of the wide variety of surface structures present in relativization, clefting, informative questions, yes/no questions, and complements. In addition to simplifying the semantic-interpretation process, the SC rules generalize the syntactic component to enable the system to process

a larger number of syntactic paraphrases--for example, some of those for a simple question having to do with whether or not the population of a certain city is of a particular value:

1. Is Downey's population 60,000?
2. Is the population for Downey 60,000?
3. The population of Downey is 60,000 isn't it?
4. Is the population of the city of Downey equal to 60,000?
5. The city of Downey has a population of 60,000 doesn't it?
6. Is Downey's population equal to 60,000?
7. Does Downey have a population of 60,000?
8. Is Downey a city that has a population of 60,000?

Because this query is of a simple form, the possibilities for paraphrase are limited. More complex queries would have many more possible paraphrases, and it is especially in these situations that the SC rules play an important part in effecting paraphrase reduction.

The power of the grammar has been increased by the addition of a system of grammatical case in the deep structure of sentences. Case specifies the semantic relationships between a sentence's syntactic elements. This year we have concentrated on case as it applies to verb phrases, with particular emphasis on prepositions and question words (who, what, how, etc.). Case analysis of nominalized phrases will be implemented next year. Initially, a detailed study was made on all of the prepositions used in verb phrases in a representative sample of English sentences. Semantic criteria were used to classify the prepositions, and the set of cases was derived from these classes. Out of this set we selected nine cases for initial implementation in CONVERSE:

CASE NAME	PRINCIPAL RELATIONSHIP
1) Domain	logical subject
2) Neutral	direct object
3) Goal	location toward which action is directed
4) Benefactive	entity that benefits from a performed action
5) Location	where an action is performed
6) Time	when an action is performed
7) Manner	the way in which an action is peformed
8) Means	the instrument or agent with which an action is performed
9) Source	location where the action originates

While prepositions are important in identifying case roles, not all sentence constituents with case functions have prepositions. Our treatment of case also includes several adverbial constituents in sentences (temporals, locationals, manner, etc.), so that we handle in a unified framework what requires several different mechanisms in other systems.*

The following examples illustrate the assignment and use of case information:

The assertion: "The firm sold five truckloads of goods to the state." maps into the following data structure format: [SELL (DOMAIN = the firm), (NEUTRAL = five truckoads of goods), (GOAL = the state)].

This form of data structure is especially convenient for interrogation from our derived deep syntactic structures.

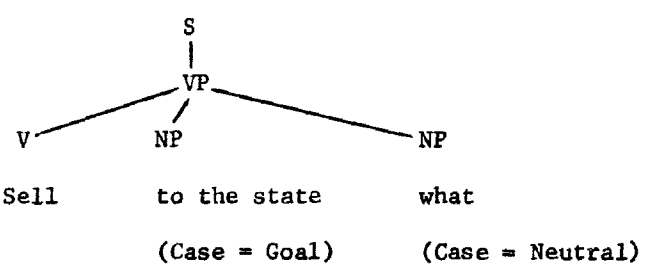
Two possible queries of this structure are:

1. What was sold to the state?

2. To whom did the firm sell five truckloads of goods?

The deep syntactic structures of these sentences, including case information, are:

(1)



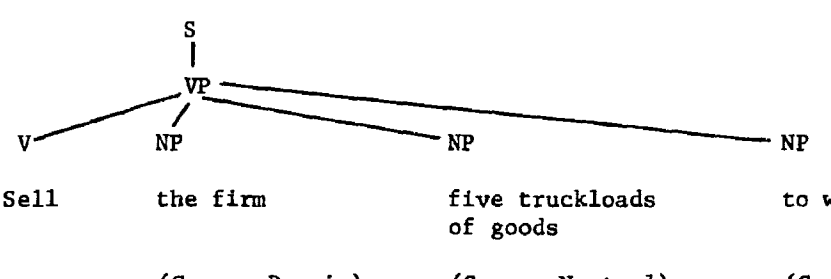
S = Sentence

VP = Verb Phrase

V = Verb

NP = Noun Phrase

(2)



*Woods, William A. Augmented Transition Networks for Natural Language Analysis. Report No. CS-1 Computation Laboratory, Harvard University. December 1969.

Winograd, Terry. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. MAC TR-84, Project MAC MIT. February 1971.

In (1), case information identifies the Neutral entry in the data base as the part of the relation being queried; in (2), it is the Goal.

A second set of examples assumes a data base containing information on airline schedules. Information is stored in the form:

Flights depart from <places> for <places> at times.

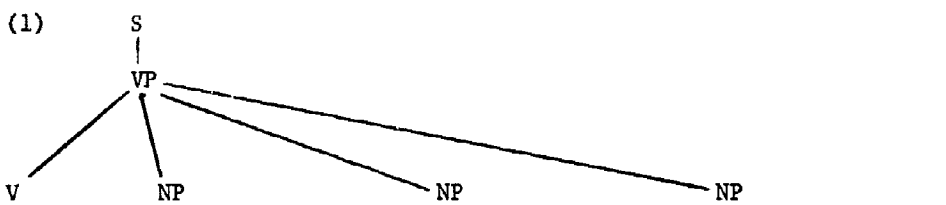
(where <places> are cities or airports)

Typical queries might be:

1. How many flights depart from L. A. before 7:00 p.m.?
2. At what time does American Airlines Flight #27 depart from Chicago for Philadelphia?

The syntactic deep structures are:

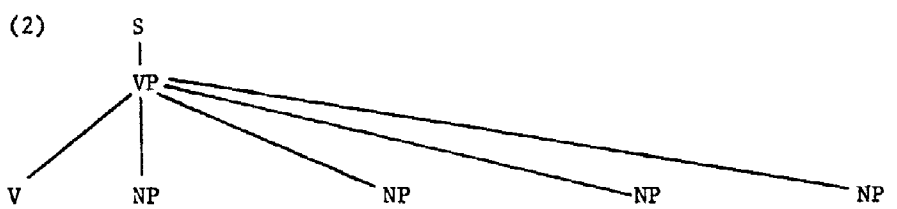
(1)



depart how many flights from LA before 7:00 p.m.

(Case = Domain) (Case = Source) (Case = Time)

(2)



depart A.A. Flt. #27 from Chicago for Phil. at what time

(Case = Domain) (Case = Source) (Case = Goal) (Case = Time)

Our implementation of case provides some distinct advantages in the parser. First of all, case forms a natural link between a sentence's deep syntactic structure and its semantics because the semantic rules make use of the case

information to translate an English sentence to an IL equivalent that operates on the data base. There is a direct correspondence between data base entries and the case categories of any verb in the data base. This correspondence is used not only in querying information already in the data base but also in determining how to store new information contained in English sentences.

Two principal improvements have been made in the semantic processing: first, the power to block unwanted syntactic parsings has been increased; second, input from declarative and imperative sentences can now be converted into commands to store, delete, and search information in the data base. The latter improvement is significant because it permits all three of the major types of English sentences to be processed and translated into calls upon the four major data management commands.

An initial version of an English sentence generator has been completed. It translates IL statements and data management answer expressions into stylized but readable English sentences that enhance on-line communication by providing the user with appropriate English feedback messages and, in certain cases, English answers to questions.

Question Answering

Our experience with question answering from the 12,000-fact data base has been quite satisfactory. Almost all questions are answered in a few seconds. Depending on the form and content of the original question, the system responds with one of three basic types of answers: a single-word answer, such as "yes", "no", a number, or "none"; a tabular array of data in report format; and, in some circumstances, a complete English sentence. The sentence format is resorted to by an algorithm that decides when the English sentence form is helpful in expressing a complete answer to the question. Since imperative sentences are now recognized (see below, "Assertions and Commands"), many questions can be stated as commands (e.g., "List the population of Downey!", "Determine the land area of each city in the Western states!").

User Feedback

A user will have confidence in the question-answering system only if it gives him an informative response to any reasonably well-formed English input sentence, whether or not the sentence can be completely matched to the contents of the data base. User feedback is most essential when the system detects, and is unable to interpret or resolve, anomalies or ambiguities in the user's query. These anomalies and ambiguities may be lexical, syntactic, or semantic. Thus, a user may use in a sentence words that are not defined in the CONVERSE vocabulary, syntactic patterns that are not currently recognizable by the parser, or a combination of terms that are in the vocabulary but cannot be matched to the contents of the data base. There may also be some unresolved instances of ambiguity that require a clarifying response from the user. CONVERSE deals with these situations straightforwardly. When ambiguity is

detected and unresolved, several IL semantic interpretations are usually generated. They are then back-translated into English and printed out as alternative search requests for the user to review, so that he can choose the one that is closest to his intended meaning. For example, if the user enters the simple but lexically ambiguous query: "What is New York's population?", the system returns two alternative queries: "What is the population of New York City?" and "What is the population of New York State?". The user selects the query he wants and obtains the answer directly. When a question could be misinterpreted because of the use of certain function words, as in the disjunctive "Did Downey or Lakewood have a land area less than 10?", the system factors the query into two separate requests and provides the two responses: "It is not the case that Downey has a land area less than 10." and "It is the case that Lakewood has a land area less than 10.". In CONVERSE V2, additional feedback facilities will ensure that users supply sufficient information to adequately characterize or describe a new data base.

Assertions and Commands

A major extension in implementing CONVERSE V2 has been the addition of rules for translating declarative and imperative sentences into calls on the four basic CONVERSE IL commands: DECLARE, STORE, FORGET, and RETURN.

Declarative sentences can be used to state assertions where new terms or phrases are introduced through the use of quotation marks. This permits the user to introduce new concepts into the system in order to characterize or extend data base descriptions. Assertions of new relations and facts using the existing vocabulary can also be stated. Simple examples of the former are: "County is a place.", "City is a place.", "Smoggy is a property.","Los Angeles is a city.", and "Los Angeles is a county.". Examples of the latter are: "No counties are cities.", "Some cities are smoggy.", and "The population of the city of Los Angeles is 7,500,000." Imperative sentences can be used to invoke commands that define new concepts in terms of concepts derivable by the system or that call for the retrieval of data.

A mechanism called "generalized define" has recently been added to the system. It allows a user to extend the system by increasing its working vocabulary, enriching its concept net, and defining functions that increase its data-manipulation capability. The user types "Define <new concept> as <old concept>!" to the parsing mechanism and the old concept is parsed to give the new concept its semantic meaning in the system. The most trivial use of this facility is to introduce synonyms for previously defined words. A more interesting use is to define a new concept in terms of the semantic interpretation of the old concept. The most powerful use of the capability is to introduce new operators that can then be used in questions to derive implicit data from the explicitly stored facts.

Advanced Development

Extended Intermediate Language. We have made considerable progress in the design of an extended intermediate language (IL2) with greater expressive power, greater compatibility with the English deep structures from which IL expressions are derived, and the ability to drive sophisticated deductive processes.* IL2 can express interconnections among sets and relations even when the interconnections involve implicit quantifiers, complex interactions among quantifiers and among quantifiers and other operators like negation, and n-ary relations where n is greater than 2. It can represent such sets as "cities north of every city in Los Angeles county" or "flights which depart from cities for cities farther east" compactly and clearly. To facilitate translation from English as well as computational interpretation of the expressions, the use of variables has been minimized in IL2.

Case relations are explicitly designated in IL2 expressions (in contrast to the use, in the predicate calculus, of position in the argument string as an implicit indicator of case). This simplifies the translation from English deep structure to IL2 given the deep structure representation which has been adopted for CONVERSE.

IL2 expressions will drive a data management system that can (1) do explicit retrieval, (2) call LISP programs to evaluate functional expressions defined in terms of other expressions, and (3) call deduction processes that make use of the general information (the relation interconnections) stored in the data base. An IL2 interpreter is now being designed. Its design is complicated by the fact that when all of the above ways of evaluating relational expressions are available for the same expression, the interpreter must try them in appropriate order and with appropriate effort limits. (Predicate-interpretation programs are therefore to be written in a language like Hewitt's PLANNER** with a TRY operator and effort parameters.)

Inference Making. The development of the deduction grapher, which will enable CONVERSE to draw inferences from general facts, is continuing. The language for specifying IL2 interpretations will be capable of driving the deduction grapher and specifying which strategies the grapher will use. (Thus, for example, the deduction grapher can be told that in evaluating the predicate "P" as in "P(a)" it should chain backward, or that it should start with some "P(b)" and chain forward, with the general statement "$(P(x) \& R(x,y)) \supset P(y)$". In this case, it is told that it should deduce from a particular general statement and is also told a possible strategy for using the general statement.)

The deduction grapher is designed to make inferences in an information retrieval context in which the task of selecting relevant premises (both general and specific facts) from a very large set of premises is at least as difficult as

---

*Kellogg, C. ILS2 Working Note 1. SDC document N-24490. 31 August 1971.

Kuhns, J. L. IL2 Working Note 2. SDC document N-24490/001/00. 16 September 1971.

Kuhns, J. L. IL2 Working Note 3. SDC document N-24490/002/00. 16 September 1971.

**Hewitt, Carl. PLANNER. MAC-M-386, Project MAC, MIT. Revised August 1970.

that of showing that the selected premises have the needed deductive relationships. The grapher proceeds by generating derivation proposals (structures of possible relevant premises) and then, if possible, filling out the details of the proposals to build a complete derivation. Proposals can be modified or new ones can be generated as required. The selected proposals represent derivations that would be valid if only truth-functional structure and first-level unification of predicate occurrences were considered. Programming done so far has been on the algorithm for generating these original proposals. The programs for filling out the proposals have been partially specified.

Analysis of Fundamental Syntactic/Semantic Relations. Definitions in dictionaries--in particular, Webster's 7th Collegiate (W7)--are being analyzed and generalized to (1) get at basic relations, particularly basic case relations, that can be efficiently incorporated in all versions of CONVERSE regardless of subject matter; (2) generate syntactic and semantic interpretation rules for productive uses of suffixes; and (3) discover the general premises that are needed for correct deductions over many different domains of discourse.

The procedure focuses on the keywords of the major defining formulas in W7--for example, the word "state" as used in the definition of nouns ending in "-ion", "-ment", "-ity", etc. After a (computer-aided) survey of some 45,000 formulaic definitions in W7, an initial set of 50 notions was selected. Conceptual analyses have been prepared so far for "quality", "state", "process", "action", "act", "cause", and "person", and also for seven notions ("will", "purpose", "deliberation", "goal", "agent", "behavior", and "effect") that have no direct counterparts among the keywords of defining formulas but that underlie case or thematic relations. Further formalization of these analyses via translation into IL2 will be undertaken.

3.1.2 Summary

At the beginning of the current contract year, initial versions of a natural-language compiler and its associated data management system were implemented. The utility and capability for expansion of each program were limited by the severe restrictions of core-memory working space within the LISP 1.5 system on ADEPT/50. CONVERSE V0 was implemented within these memory constraints and demonstrated during the first half of the contract year. V0 could translate questions of moderate complexity into intermediate-language procedures by means of surface-structure rules and the use of a small concept network. These procedures could then be passed to the data management system for evaluation, and tabular answers could be returned. In pushing this demonstration milestone to a successful conclusion, we managed to solve a number of problems in program/program and program/disc-file communication, but core-memory constraints prohibited the expansion necessary to incorporate further improvements in the prototype.

With the advent of ADEPT/67 and its increased core memory we have been able to implement CONVERSE versions V1 and V2. A major extension of the grammatical-analysis component was achieved in V1 by the addition of a series of transformational structure-changing rules. These rules both generalize syntax analysis and simplify the production of well-formed semantic interpretations. Other important features of V1 include English user feedback, generation of English answers where appropriate, and exercise of the data management system with a data base of medium size and complexity.

V2 adds to CONVERSE the ability to handle and process declarative and imperative sentences. It also incorporates features that permit a user to describe and maintain data bases of diverse content and structure by means of ordinary English sentences. These initial user-extensibility features represent an important step toward the achievement of an on-line system with which people can conveniently and effectively communicate with data files of substantial size and expanding content. These features will receive increasing attention as future on-line experiments are carried out. The addition of case assignment in V2 has expanded sentence analysis beyond that available in V1. The case-assignment procedure uses general linguistic knowledge about the interrelations among English question words, prepositions, and verbs to identify the basic semantic relationships that underlie the syntactic constituents of deep structure.

Further versions of CONVERSE will grow out of our efforts in on-line experimentation with different data bases and such advanced developments as deductive inference, morphological and thematic analysis, IL2, and predictive linguistic constraints to support speech recognition. The present CONVERSE software has been designed to allow for such future growth. Efficient, functional I/O routines now exist for use of auxiliary disc memory storage for (1) binary code (functions and grammar rules), (2) symbolic expressions (concept-net entries and complex symbolic objects), and (3) arrays (set extensions and dictionary entries).

During the year progress has been made in developing a closer liaison with the Voice I/O project and a better understanding of the nature of the critical interface problems to be solved before an eventual vocal CONVERSE can be constructed. Steps have been taken to direct much of our advanced work in semantics, syntax, and morphological analysis toward the solution of these interface problems. A close tie between the projects will be achieved when a common data base is selected. This is a high-priority objective, to be attained early in next year's research. In addition, our work on an advanced formal language for expressing data management functions (IL2) will be closely associated with the objective and experimental work to be carried out by the Network Data Sharing project (Section 4.1). Our increasing association with both of these projects, added to our own experimental efforts, leads us to expect increasing activity with other researchers through the resources of the ARPA Network.

3.2 VOICE INPUT/OUTPUT

The long-term goal of the Voice Input/Output project is the operation of SDC's CONVERSE system with vocal speech input and output. CONVERSE (see Section 3.1) is a natural-language question-answering system that employs a user-extensible subset of English and aims, eventually, at employing a virtually unconstrained subset of English for data base management. In order to achieve a vocal CONVERSE, we will have to attack and solve almost all of the outstanding research problems regarding the recognition of speech by a computer—including the recognition of continuous speech (as opposed to the recognition of single words), the ability to handle large vocabularies, and the development of scanning processes and system integration techniques that allow parsing and disambiguation of noisy input. Because of its difficulties, the task of solving these problems must be approached in increments. The first two increments are (1) a reimplementation of the Vicens-Reddy speech recognition system from the Stanford University PDP-10 to the SDC IBM 360/67 and the Raytheon 704 and (2) the implementation of a Vocal Data Management System (VDMS) that employs a highly constrained subset of spoken English.

VDMS has been selected as a test and development vehicle for tackling continuous speech. Its major thrust is called Predictive Linguistic Constraints (PLC). In present natural-language and compiler systems, symbolic rules describe a symbolic input, and various types of sophisticated pattern-matching techniques are used to establish validity and transform the input into an object-level language. Experience of investigators has shown that this approach is not appropriate for understanding speech utterances; selecting a word from the input stream and using it as a key into a symbol table or concept net does not work. Instead, PLC predicts highly plausible components in the input. The match criterion may be relaxed because a reasonably good match merely confirms a hypothesis. Of course, a PLC system must preserve options to change assumptions or readjust the parsing to data based upon further information. Figure 3-1 is an overview of the system.

3.2.1 Progress

During the past six months, the implementation of virtually all of the hardware and software tools necessary to support full-scale speech-understanding research was completed. A graphics display package for waveform analysis has been developed for the Raytheon 704 computer. Because the package is a collection of callable FORTRAN IV subroutines, it can be used in a variety of applications, such as displaying the speech signal and various parameters extracted from it. A disc-partitioning system has been implemented on the 704 so that multiple overlay routines can communicate without physical intervention. Other utilities that are now operational include a mini-LISP System, digital-to-analog (DAC) and analog-to-digital (ADC) conversion systems, data formatters that allow the exchange of tapes between the IBM 360/67 and the Raytheon 704, a program (VOICEBOX) that controls the audio environment, and MUSIC V, a program

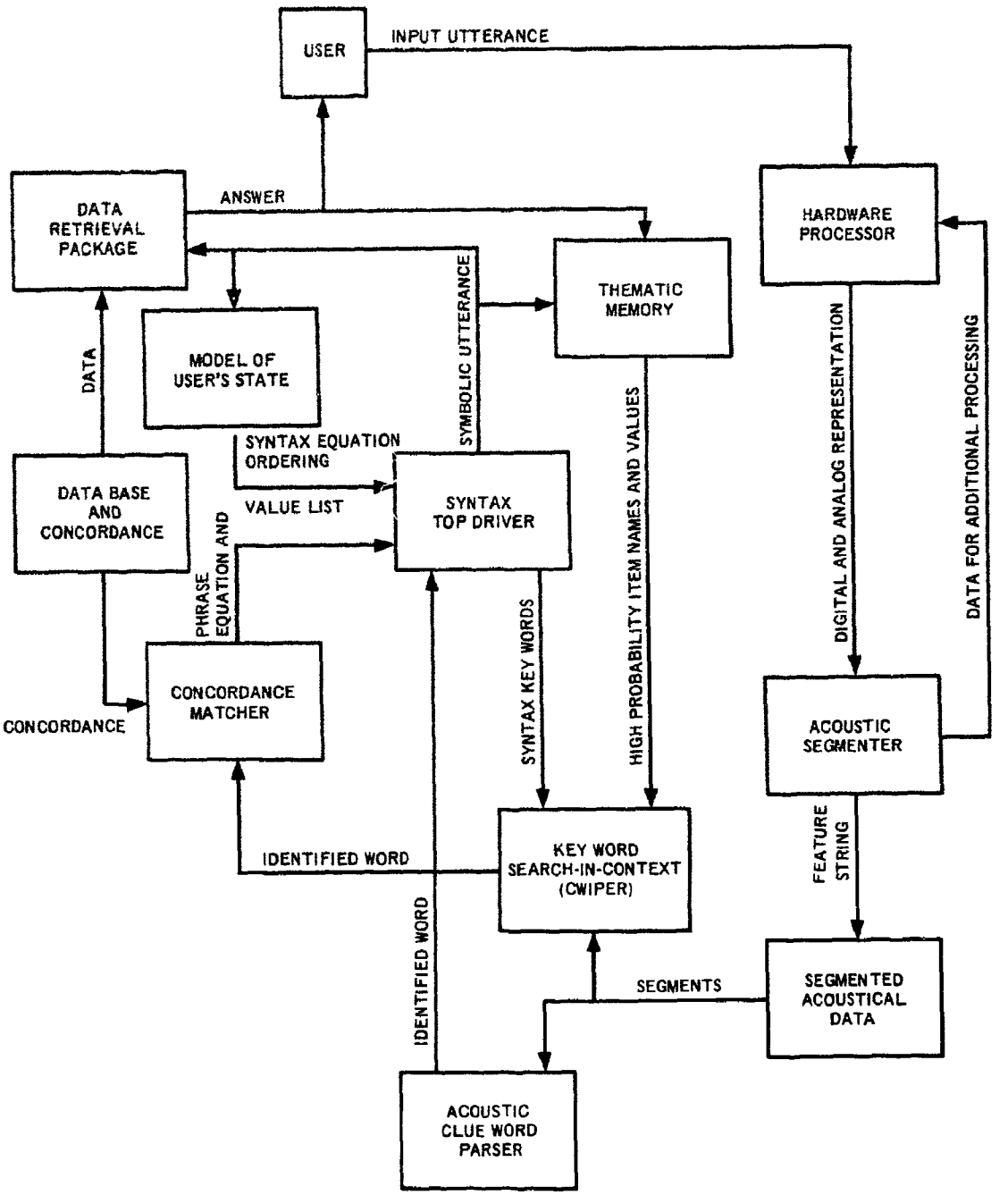


Figure 3-1. Initial PLC Model for VDMS

that will be used for speech synthesis. Utility routines for mathematical support of acoustic processing include Fast Fourier Transforms (FFTs), CHIRP-Z transforms, a CEPSTRUM analyzer, and a group of digital filters. Various algorithms for each of these routines have been compared for accuracy, computation time, and core requirements. A formant tracker has been constructed and is undergoing detailed analysis. Several all-integer versions of the FFTs are being designed and constructed.

CWIPER Implementation

CWIPER, the acoustic processing portion of the planned VDMS, is based upon the speech-understanding system implemented by Vicens and Reddy at Stanford University.* The Vicens-Reddy system is unique in the sense that it approaches the problem of speech recognition as a whole, rather than treating particular aspects of the problem as in previous attempts. For example, where earlier systems treated only segmentation of speech into phoneme groups, or detected phonemes in a given context, the Vicens-Reddy system processes the incoming speech signal, applies heuristics to segment the signal and to identify phoneme-like units, and then uses the total phonemic pattern to recognize an entry in the lexicon. The segmentation portions are identical. CWIPER will differ from the Vicens-Reddy system in that it will use different algorithm and pattern-matching techniques that allow searching for words (or phrases) in a sample of continuous speech.

To construct CWIPER, the Vicens-Reddy system has been reimplemented on the IBM 360/67. The software has successfully built a lexicon and recognized a set of 20 speech samples comprising five distinct words utter by two different speakers. Detailed program documentation has been written, and an extensive analysis of the heuristics in the Vicens-Reddy system has been undertaken, with the results documented in the program description. The analysis and the documentation will allow easier modification of and extension to the system in the future.

Hardware Development

The Voice Laboratory hardware that supports the CWIPER and Vicens-Reddy systems has been designed, constructed, checked out, and integrated with the Raytheon 704 ADC and DAC systems. It includes positive and negative peak-detection and threshold-crossing counters. Digital interfaces allow program control of the entire subsystem. The recording booth has been wired and a two-level pluggable patch panel has been installed to allow modular use of the Laboratory's audio facilities.

---

*Vicens, P., Aspects of Speech Recognition by Computer, Stanford University, Memo AI-85 (CS 127), 1969, 210 pp.

Reddy, D. R., An Approach to Computer Speech Recognition by Direct Analysis of the Speech Wave, Stanford University, Memo AI-43 (CS 49), 1966, 144 pp.

VOICEBOX

VOICEBOX, a program that operates interactively on the Raytheon 704, has been implemented to control the audio environment. A partial list of its capabilities is as follows:

- Record speech samples from the microphone or tape recorder.
- Play back all or part of a speech sample, either once or repetitively.
- Scale the speech sample.
- Pass the sample through one or more of the filters attached to the Raytheon 704.
- Play back all or part of the filtered signal, either once or repetitively.
- Determine the parametric input to CWIPER for the speech sample.
- Symbolically display the speech sample, filter contributions, or CWIPER parameters.
- Construct binary "save" tapes of recording sessions.

VOICEBOX also has a macro definition and execution facility to allow the on-line user to catalog often-repeated sequences of tasks.

MUSIC V

MUSIC V, a sound-synthesis program originally developed by Mathews at Bell Labs, has been made to operate on the IBM 360/67 and the Raytheon 704, and some short (approximately 10-second) samples of music have been synthesized on the 704. MUSIC V will be used to synthesize speech segments of approximately one to two seconds to enable us to verify the assumptions made by the speech-recognition system.

Work in Progress

In several areas, work has started but not reached culmination. The CWIPER software is being moved from the IBM 360/67 to the Raytheon 704. This does not appear to be a major task because the FORTRAN discrepancies are minor and well understood. The 704 CWIPER will be used for CWIPER verification and data-base-building tasks to be undertaken during the next contract year.

The design of true root means square (TRMS) hardware has begun. This hardware will be used for additional parameter input to CWIPER and will allow vastly improved fricative detection and classification. Another hardware activity underway is the design of the Raytheon 704-IMP interface to move the 704 into the ARPA Network next year.

Version I of a system to aid equipment maintenance procedures is now being checked out. The system vocally directs the user to set a group of switches and toggles, and then asks for the status of various fault lights. The user answers the questions (currently by typing the answers on a teletypwriter terminal, but later, in Version II, by speaking into a microphone). After interpreting the answer, the computer asks the next question. This sequence is repeated until the faulty component(s) is located and replaced. (The same question-answer sequencing is used in many computer-assisted instruction systems.) Version II will use the 704 CWIPER for vocal input.

3.2.2 Summary

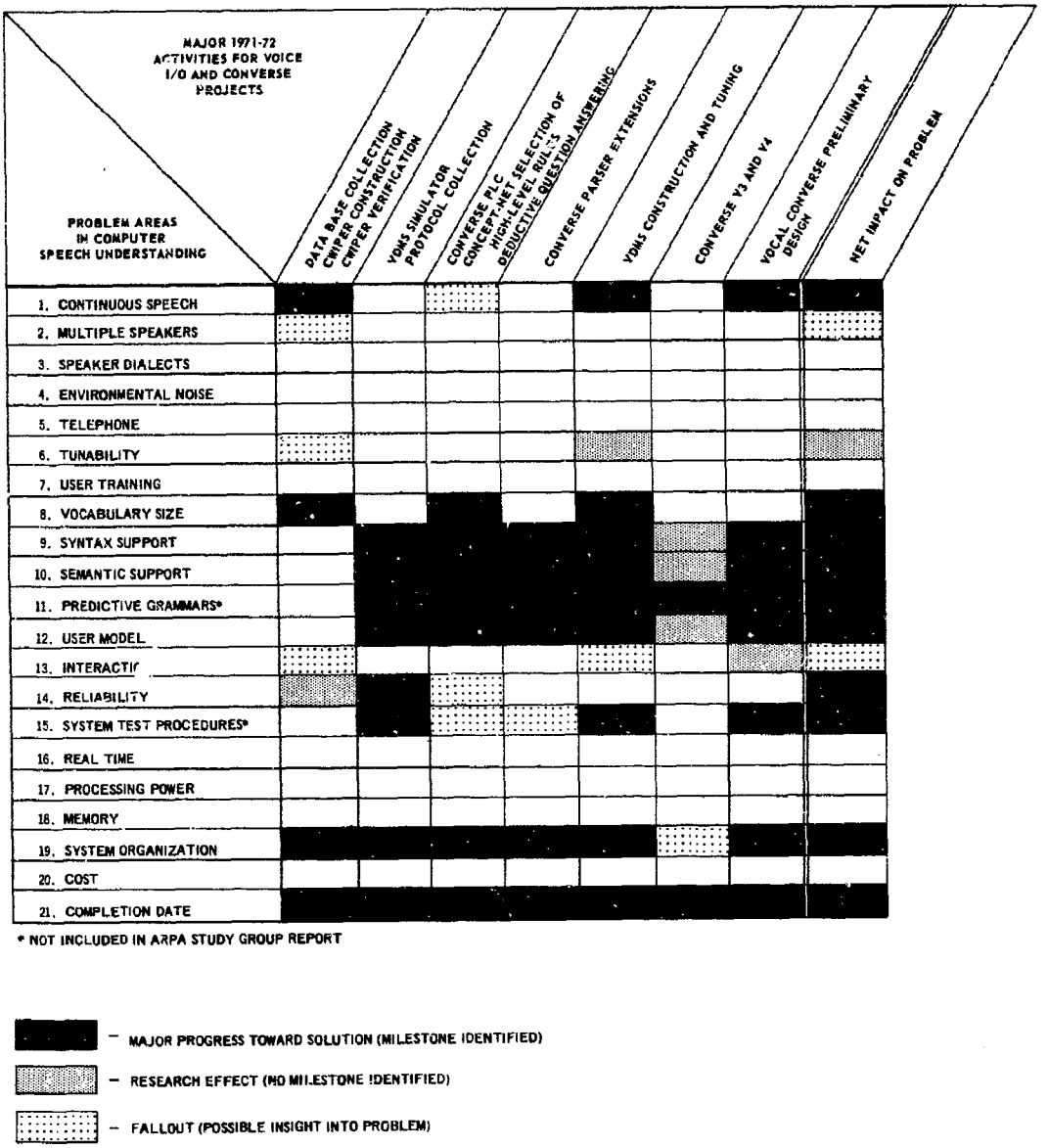
The Voice Input/Output project, now completing its first full year, is pursuing speech understanding by computer on the basis of the hypothesis that both the human's "inquiry state" and the words and phrases he uses can be dynamically modeled and predicted to greatly increase the likelihood that the computer will recognize and interpret his spoken utterances. During the past year, the project has developed the tools that will allow the design and development of a speech-understanding system that accepts continuous-speech input. A high-quality audio-recording environment has been provided that allows precise control of speech sampling. Computer interface hardware and software have been developed. Several software modules for recording and producing acoustic waveforms have become operational; they include CWIPER, MUSIC V, and VOICEBOX. Appropriate mathematical analytical tools have been incorporated into the system.

In its Final Report on Speech-Understanding Systems,* an ad hoc study group formed at the request of ARPA's Information Processing Techniques office and consisting of ARPA/IPT research contractors (including SDC's Voice Input/Output Project Leader) formulated a list of 19 problem areas in which technical breakthroughs must be achieved before flexible natural-language speech-understanding systems will be a reality. This list, plus two additions of our own, is shown in Table 3-1, a matrix that identifies the future milestones for the Voice Input/Output and CONVERSE projects. As the matrix indicates, we expect to achieve substantial progress toward solutions in 11 of the 21 areas and some insight in two more, which suggests that there is considerable promise of achieving a vocal CONVERSE within the next five years.

---

*Speech-Understanding Systems: Final Report of a Study Group. Published for the Study Group by Computer Science Department, Carnegie-Mellon University, Pittsburgh. May 1971.

Table 3-1. Impact of Voice I/O-CONVERSE Research on Problems Facing Speech Understanding by Computer



3.3 GRAPHIC INPUT/OUTPUT

Enchancing communication between man and computer by providing functions and facilities that are compatible with, or equivalent to, techniques used in visual man-to-man communication is the principal goal of the Graphic Input/Output (I/O) project. Our concern has been to develop methods of graphical input and output that will permit a user to carry on a dialog with a computer in the language and notation of his discipline or problem domain. The functional entities required for such a dialog are a data-input tablet (e.g., the RAND Tablet), an interactive CRT display operable as a terminal in a time-sharing system, and a character-recognition program that will accept the symbols used in the notational expressions.*

The near-term project goal is to develop programming systems that utilize two-dimensional notation. Our initial efforts are to use mathematics, the most ubiquitous scientific notation for numeric and symbolic programming, and to develop a programming system that uses flowcharts as its source language. The work is based upon previously developed programs that accept two-dimensional mathematical expressions hand-drawn on the input tablet, extract the explicit and implicit information they contain, and transform that information into representations amenable to existing processing techniques.

3.3.1 Progress

In this reporting period we completed the first version of a system called The Assistant Mathematician (TAM). TAM uses ordinary mathematics to specify numerical computation. TAM accepts much of the commonly used mathematical notation to provide a casual user, such as an engineer or a scientist, with a powerful, flexible, natural, and easy-to-use computing resource.

We have completed the initial version of a flowchart-input program that allows a user to construct a freehand flowchart on the graphics console. We have also begun work toward conducting experiments using our character recognizer with graphic-tablet data transmitted over the ARPA Network. In addition, various graphics programs, such as the Parser and Unparser, have been improved.

The Assistant Mathematician (TAM)

Our goal in constructing TAM is to create a powerful computational system that uses mathematics, the common language of technology, as its programming language. TAM is an incremental system; each statement is executed before the next statement is requested. It accepts a wide range of commonly used mathematical notation. TAM provides arithmetic manipulation using a powerful set of operators on constants, variables, and one-dimensional or two-dimensional

---

*Bernstein, M. I. Hand-Printed Input for On-Line Systems. SDC document TM-3937. April 1968.

arrays, and incorporates looping facilities, single-statement functions, and user-defined input/output. Some built-in functions, such as square root, logarithm, and the trigonometric functions, and built-in constants, such as π and e, are also provided. Considerable effort has been made in the implementation of the system to ensure that the system does what a user who is not sophisticated in the use of computers would expect it to do.

The description that follows attempts to give the reader a "feel" for the use of the system by describing a complete sequence of operations. A more detailed and formal description of the system is available.*

Figure 3-2 is a sequence of photographs showing the use of TAM to find a root of a quadratic equation using the formula

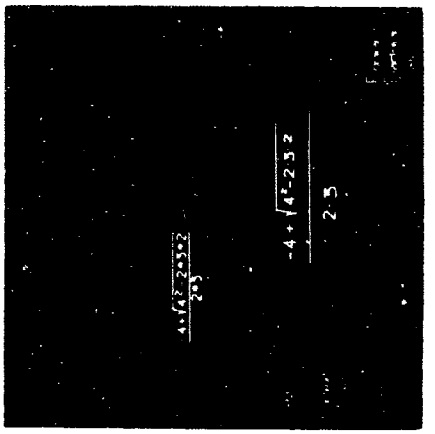
$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

for one root of the equation $aX^2+bX+c=0$. In Figure 3-2(a), the user has hand-printed the expression on the TAM console. In Figure 3-2(b), the expression has been processed by the character recognizer and the Parser. The character recognizer has replaced each hand-printed character with a computer-generated character of the same size, aspect ratio, and position so that the user can verify character recognition, while the Parser has operated on the characters to produce a linear-string form of the two-dimensional mathematical expression. This linear form has been converted back to two-dimensional form by the Unparser and is displayed so that the user can verify the results of linearization. In Figure 3-2(c), the user has, through use of the button labeled TAM, requested that the system execute the expression; the result is displayed at the top of the screen.

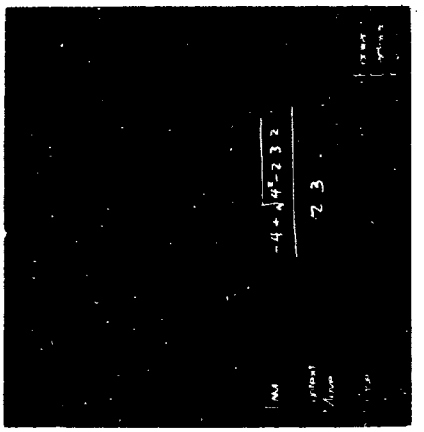
Figure 3-3 shows the use of TAM for matrix arithmetic. (Photographs of the hand printed input are omitted from here on.) In Figure 3-3(a), a matrix has been defined; in 3-3(b), the value has been stored in TAM. Figures 3-3(c) and 3-3(d) show the matrix-inversion operation, denoted in the usual way. Figures 3-3(e) and 3-3(f) show matrix multiplication (the C matrix was stored previously).

The sequence of photographs in Figure 3-4 illustrates function definition. In Figure 3-4(a), the first-order approximation of the orbital velocity of a satellite as a function of height has been input. Selecting the TAM button

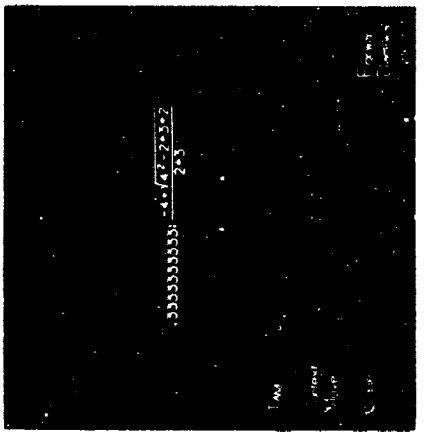
*Bebb, Joan. The Assistant Mathematician (TAM). SDC document TM-4790/000/00. October 1971.



(b)

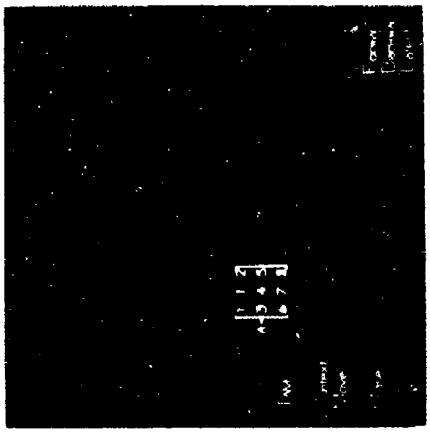


(a)

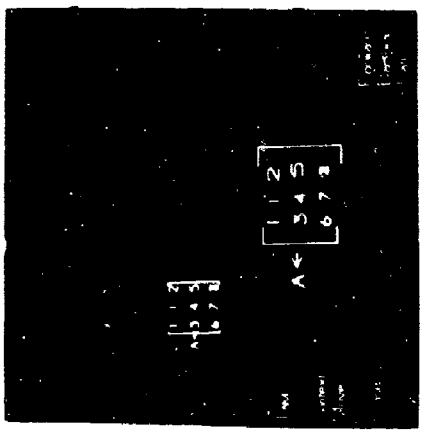


(c)

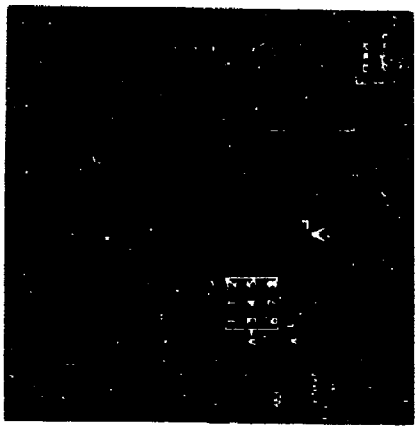
Figure 3-2. Quadratic Formula



(b)

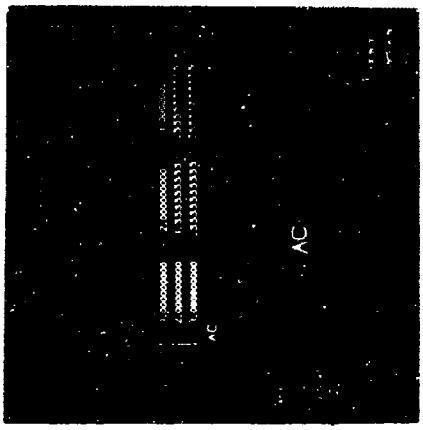


(a)

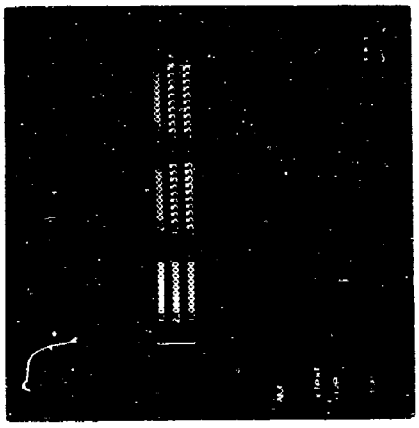


(c)

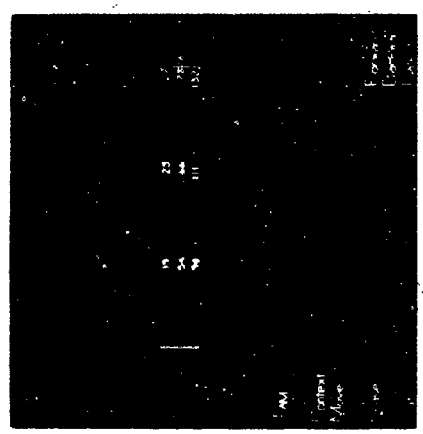
Figure 3-3. Matrix Arithmetic



(e)



(d)



(f)

Figure 3-3. Matrix Arithmetic (Cont'd)

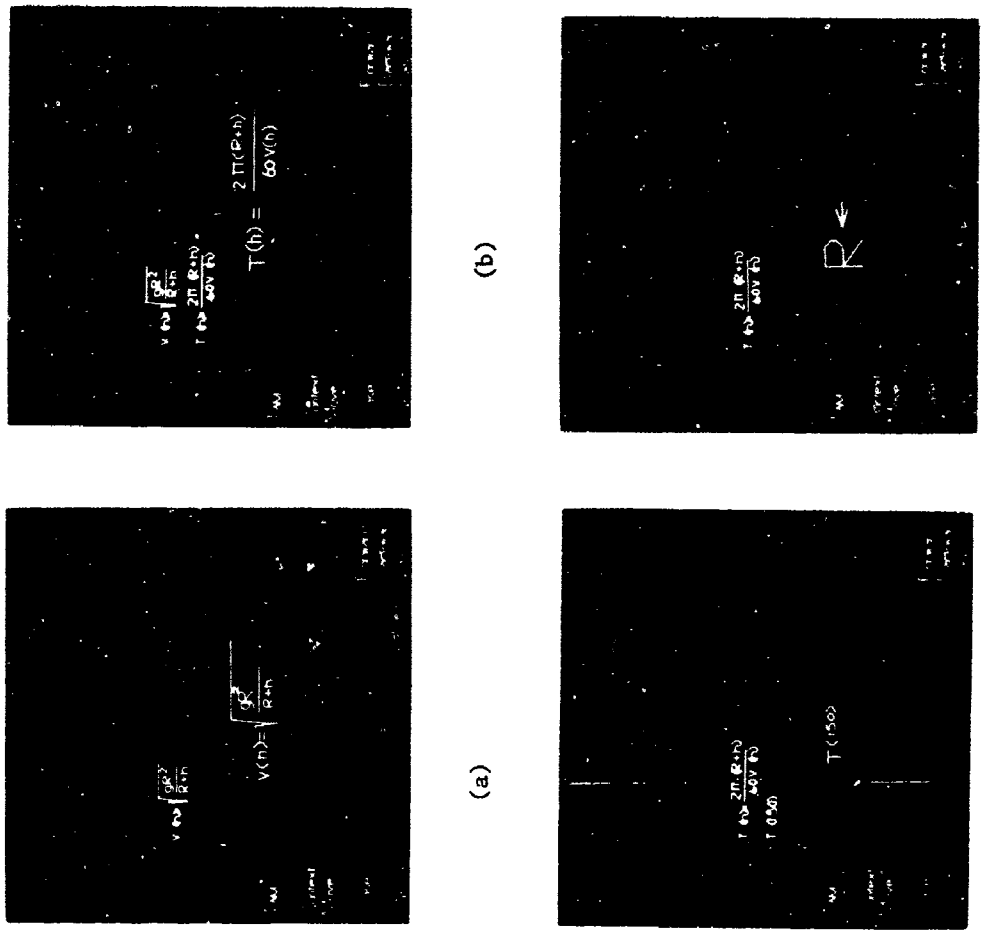


Figure 3-4. Function Definition and Use

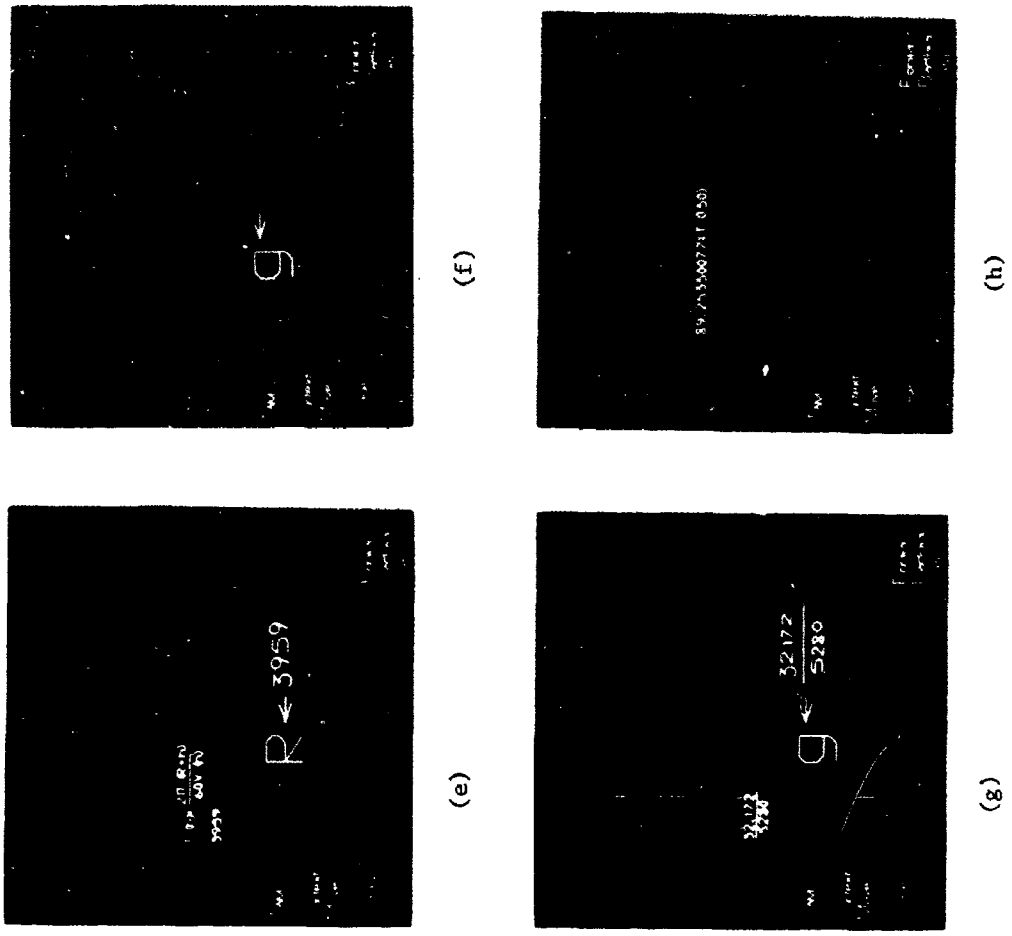


Figure 3-4. Function Definition and Use (Cont'd)

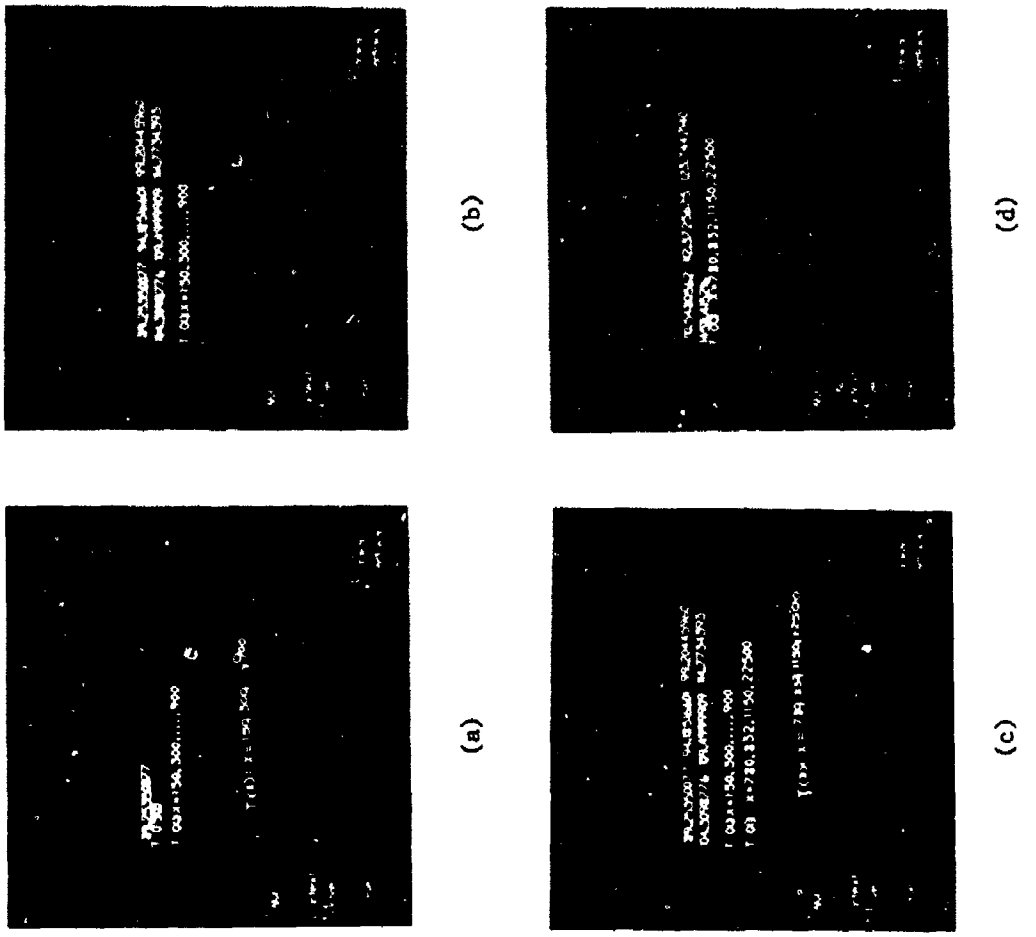


Figure 3-5. Iteration Statements

stores this for later use (Figure 3-4(b)). Similarly, in Figures 3-4(c) 3-4(d), the orbital time as a function of altitude is defined and stored. In Figure 3-4(c), the orbital time of a satellite at an altitude of 150 miles has been requested. TAM recognizes that the constants "g" and "R" in the functions have not yet been defined, and requests their definitions (Figures 3-4(d), (e), (f), and (g)). When all needed constants are defined, TAM computes the answer (Figure 3-4(h)).

An iteration capability is also avilable in TAM. In Figure 3-5(a), TAM has been requested to print the orbital times for altitudes of 150, 300, 450, 600, ..., 900 miles, with the result given in Figure 3-5(b). An alternative form of the iteration statement, shown in Figure 3-5(c), gives orbital times for altitudes of 780, 832, 1150, and 22,500, shown in Figure 3-5(d).

TAM also incorporates an editing system to aid the user in composing and altering expressions. The user may delete, change, or move groups of characters as necessary to obtain the expression he desires. Previously used expressions are saved and may be recalled as necessary.

TAM has been operational, in various stages of development, for four months. During this time, it has been used by a number of people not connected with the project. Their responses to TAM's capabilities have been enthusiastic. We feel that TAM is an important contribution to the systems and techniques needed for truly natural man-machine communication.

Flowchart Input

A flowchart, a graphical representation of a computer program or routine, provides considerable aid in constructing and debugging a program. The overall aim of the flowchart-programming system is to allow direct entry and use of the flowchart for computer programming. The programs described here allow entry of flowchart structures. These programs are the initial part of a complete flowchart programming system.

A detailed description of the flowchart programming system is available (see Section 3.5). Briefly, a flowchart consists in circles and boxes connected by lines indicating the direction in which information flows among them. Boxes contain either executable statements or decision statements, the distinction being made by the number of exiting flow lines: a processing box has one exiting flow line, a decision box two. Circles indicate remote connectors and switches. A remote connector to a subsequent page of a flowchart is defined by a circle with only an entering flow line; its use on the subsequent page is specified by a circle with only an exiting flow line. A circle with an entering flow and multiple exiting flow lines defines a switch. Both boxes and circles may have only one entering flow line; subsequent flow lines entering a previously entered box or circle should be connected to the entering flow line. Flow lines may intersect each other, but they may not overlap a circle or box.

To use the flowchart-input system, the user must supply two dictionaries for use by the character recognizer. One must consist of boxes and circles, and is used for the construction of the flowchart itself. The other should contain all of the numeric, alphabetic, Greek, and special characters that the user needs to provide text and header information for the flowchart. In particular, the second dictionary should contain all of the characters the user will need for the programming language statements to be placed within the boxes and circles.

Figure 3-6(a) shows the initial state of the flowchart-input system as it is ready for construction of a flowchart. Each flowchart consists of a single page--in this case, page number 1. The user must supply the name of the routine and its type, procedure, or function. Figure 3-6(b) shows the results after this header information is supplied.

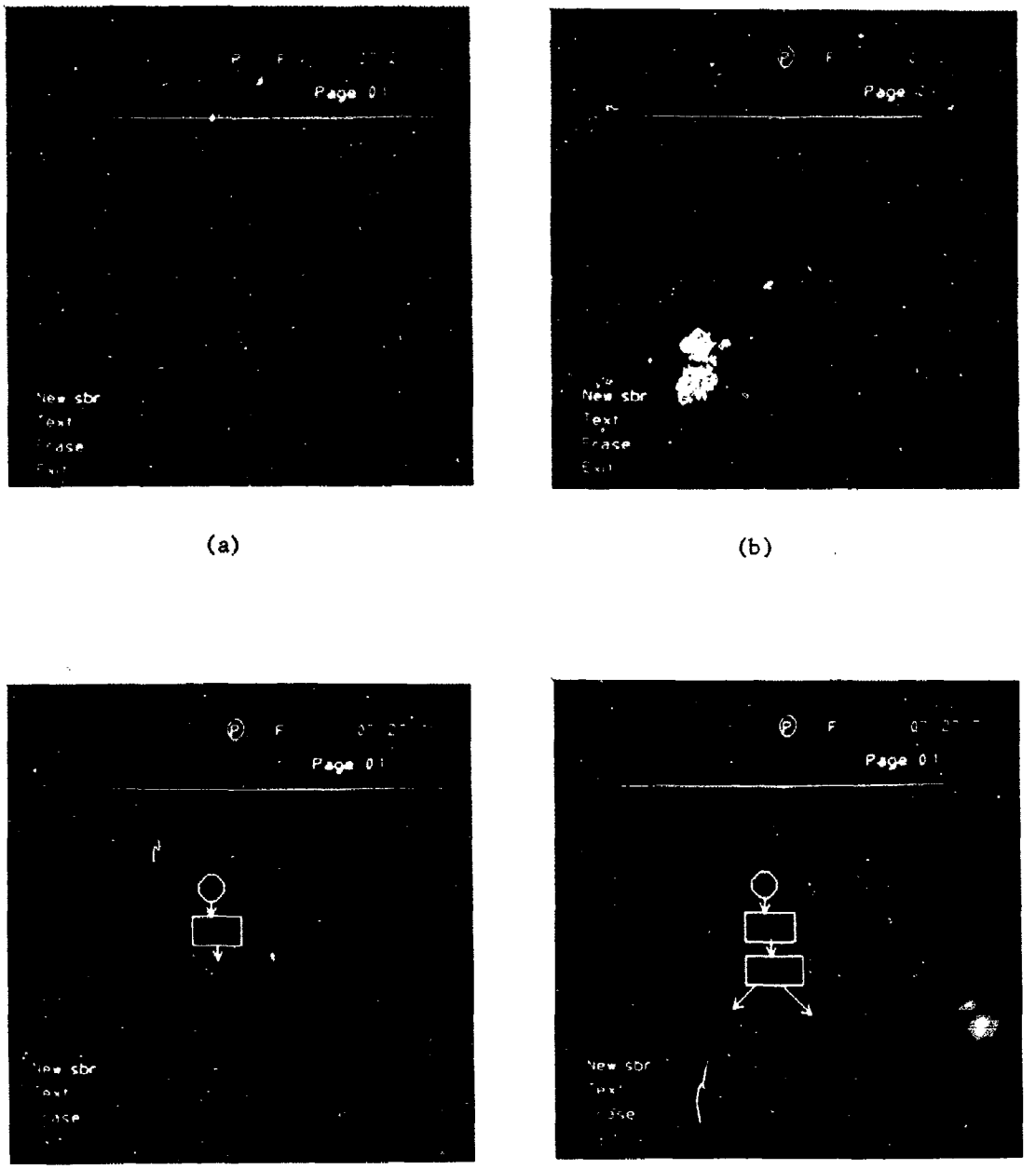
After the header information is completed, the user may start flowcharting by inputting boxes, circles, and lines. Strokes recognized as boxes and circles are replaced by system-generated boxes and circles at the same position and of approximately the same size. A stroke is considered to be a flow line if an arrowhead exists on one end. The direction of a flow line is determined by the placement of the arrowhead regardless of the direction in which the line was drawn. Each flow line is approximated to the nearest 45° angle, the horizontal line being at zero degrees. Flow lines that intersect boxes and circles are illegal and are ignored.

Figures 3-6(c), (d), (e), and (f) show the construction of a flowchart. In Figure 3-6(c), a remote connector has been defined and a processing box has been used. In Figure 3-6(d), a decision box has been added; in 3-6(e), a switch has been used. In Figure 3-6(f), interconnecting lines have been added, showing how flow lines are connected to provide multiple entrance to a box or circle.

The user inputs textual information by touching the TEXT light button. Text is entered in circles or beside flow lines, as shown in Figure 3-6(g). If a box is selected after the TEXT button has been used, an asterisk appears in the upper right corner so that the user can verify selection of the box. This is shown in Figure 3-6(h).

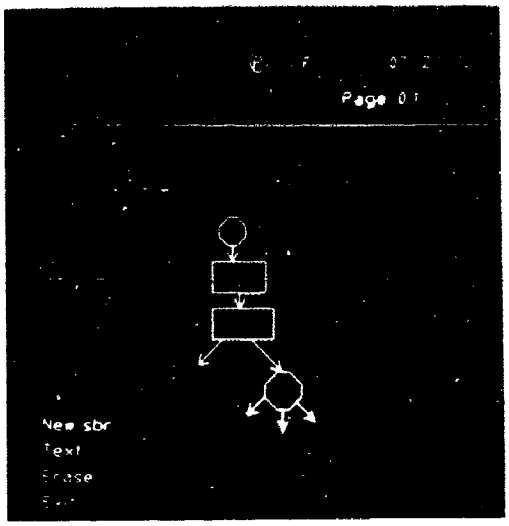
The user erases elements by a scrubbing motion of the light pen. A box or circle is erased by a scrub in its center; when a box or circle is erased, its exiting flow lines are automatically deleted. The user erases a line by scrubbing its arrowhead. The user terminates the program by touching the EXIT button.

The flowchart-input program is an important first step toward the development of a complete flowchart programming system. When this program is coupled to a flowchart interpreter, a compiler, and a numeric processing facility such as TAM, a user will have a powerful programming facility that operates directly

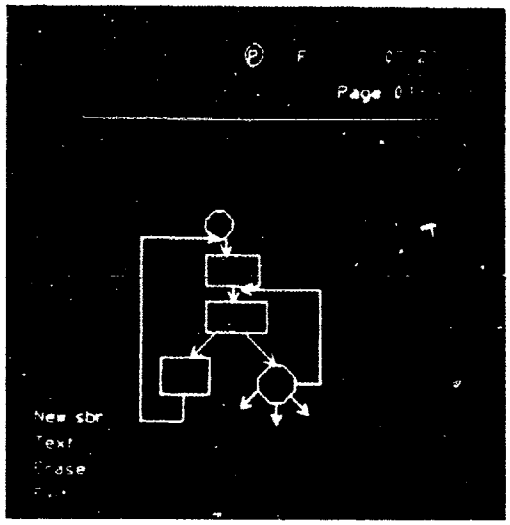


NOT REPRODUCIBLE

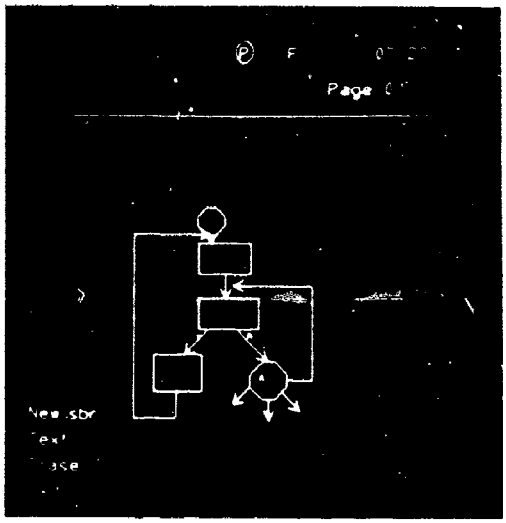
Figure 3-6. Flowchart-Input System



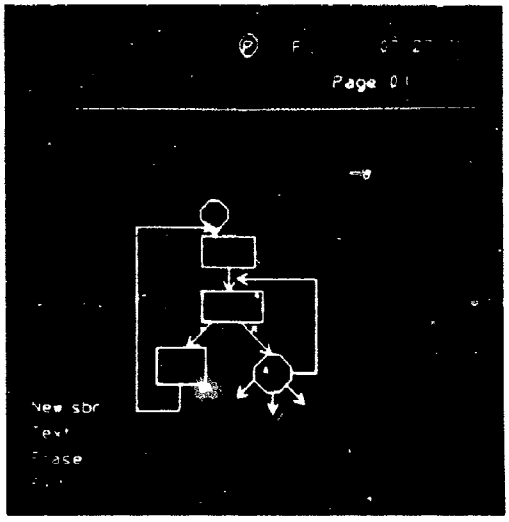
(e)



(f)



(g)



(h)

Figure 3-6. Flowchart-Input System (Cont'd)

in a flowchart language. This will allow him to create, debug, and test his program in a small fraction of the time presently required by conventional means.

ARPA Network Graphics

At the first ARPA Network Graphics meeting, which was hosted by MIT on July 18-20, 1971, the SDC Graphic I/O project and MIT Dynamic Modeling and Computer Graphics (J. C. R. Licklider) defined an initial graphics experiment for the ARPA Network. It involves the use of SDC's character recognizer with data generated at a data tablet in the DMCG facility, the purpose being to study the feasibility of remote character recognition--in particular, the interaction times involved. The experiment should also suggest ways in which the Network could be used to compare the character recognizers available at its various nodes. We are now writing the programs that will interface our character recognizer to our Network Control Program, and we expect the experiment to be finished during October 1971.

3.3.2 Summary

The work of the Graphic I/O project during the past year can be divided into two areas: (1) conversion and refinement of various graphic components (character recognizer, Parser, Unparser) and the development of a new component (flowchart input), and (2) the synthesis of the components into a complete system (TAM). From a long-range viewpoint, the synthesis effort is the more important of the two. In particular, by building TAM, we have obtained valuable information on both the operational problems of constructing interactive graphics systems efficiently and the problems of processing the language of mathematics.

TAM is constructed from modules written in several different languages. The graphic components are written in SDC's L^6 assembly language; the interpreter, which performs the necessary analysis and computation, is written in the META compiler and translator language; the linkage between these components is handled by FORTRAN programs. We have developed techniques for easily managing and linking these modules together. By using special languages for each subtask in TAM, we gain flexibility and efficiency in altering its capabilities. This makes TAM a powerful vehicle for experimentation with the language of mathematics.

The drawbacks of conventional mathematics as a computer language are that it is ambiguous and context dependent. Various symbols have different meanings depending on context; for example, + refers to the addition operation in conventional algebra and to the OR operation in Boolean algebra, i can mean the $\sqrt{-1}$ or be a variable (j also is used as $\sqrt{-1}$) . These drawbacks can be ameliorated by introducing new symbols or declarations to specify the nature of symbols, but that solution runs counter to the concept of natural computer communication; after all, the user, within the context in which he is working, uses methematics in a completely consistent manner. An important reason for developing TAM was

to begin exploring the tradeoffs between constraints on the naturalness of the language and the problem-solving domain over which the language is consistent. The development of TAM has led to the notion, to be explored in the next contract period, of defining a problem-solving context in which the user operates and incorporating that definition into the system in order to reduce ambiguity and take advantage of the context-dependency of mathematics. The system will then be of maximum assistance to the user while requiring minimum alteration of his customary problem-solving behavior. The development of this concept should provide a major breakthrough in man-machine communication.

3.4 STAFF

Natural Computer Input/Output Staff

T. G. Williams, Head

CONVERSE Project

C. H. Kellogg, Principal Investigator
J. H. Burger
T. C. Diller
K. J. Fogt
J. C. Olney, L. Travis — Consultants

Voice I/O Project

J. A. Barnett, Principal Investigator
C. A. Kalinowski
Iris Kameny (part time)
L. M. Molho
H. B. Ritea
R. DeCrescent Consultant (part time)
R. Bobrow (part time)

Graphic I/O Project

T. G. Williams, Principal Investigator
Joan Bebb (part time)
Jean Igawa
D. R. Lashier
J. McGahey
Jean Saylor

3.5 DOCUMENTATION*

Bebb, Joan. TAM. SDC document TM-4790. October 1971.

Bernstein, M. I. On-Line, Interactive Parsing and Programming. SDC document TM-4582. August 1970.

Kameny, Iris, and H. Barry Ritea. Analysis and Development of the Vicens-Reddy Speech Recognition System: Table of Contents for Document Series TM-4652. SDC document TM-4652/000/01. In press.

Kameny, Iris, and H. Barry Ritea. Introduction and Overview of the Vicens-Reddy Speech Recognition System. SDC document TM-4652/001/00. December 1970.

*The documents listed here are available for public release.

Kameny, Iris, and H. Barry Ritea. Description and Analysis of the Vicens-Reddy Preprocessing and Segmentation Algorithms. SDC document TM-4652/200/00. December 1970.

Kameny, Iris, and H. Barry Ritea. Description and Analysis of the Vicens-Reddy Recognition Algorithms. SDC document TM-4652/300/00. March 1971.

Kameny, Iris. The Lexicon Design for the IBM 360/67. SDC document TM-4652/400/00. May 1971.

Kameny, Iris, and H. Barry Ritea. Description of the Vicens-Reddy Lexicon Candidate Selection Algorithms. SDC document TM-4652/500/00. In press.

4. SYSTEMS RESEARCH

The systems research projects are developing new hardware and software tools and techniques useful to ARPA's computer-science research community. These projects presently include the ARPA Networks development activity and the Graph-Meta analysis work, each of which is concerned with both the design and the implementation aspects of the overall system to support computer-assisted military planning. The Graph-Meta research is based on a rigorous analytical foundation in graph theory, while the Networks effort is more applications oriented, involving HOST-to-HOST protocol development and studies of distributed data-base systems in a computer network. Both projects have been quite active during the reporting period, and have developed Network-related facilities for experimental use by the other ARPA-sponsored SDC research projects.

4.1 NETWORKS

The goals of the Networks project are to make SDC's ADEPT Time-Sharing System an operating part of the ARPA Network and to explore ways in which it can both contribute resources to the ARPA community and benefit from the services the community makes available to us. As part of our Network effort, we are also investigating the "distributed data base" problem, and have considered a number of possible ways of integrating dissimilar data management systems within a computer network. Because of the long lead time needed for the development of a feasible, practical approach, this effort is being pursued in parallel with the HOST-to-HOST protocol implementation, so that we can utilize the ARPA Network for data-sharing experiments.

The ADEPT system runs on an IBM 360/67 computer and utilizes a Honeywell DDP-516 peripheral computer as an interactive I/O controller. The ARPA Interface Message Processor (IMP) is also connected to the DDP-516, which collects messages and passes them between the Network and the IBM 360/67. The subsystem of ADEPT that interfaces with the Network, called HOSTOSS (HOST Operating SubSystem), has the following features:

1. Interprocess communication. Programs running under ADEPT can communicate with each other and with programs running elsewhere in the Network. All users can simultaneously have multiple connections. (Presently, there is an overall limit of 32 connections for ADEPT, but this can be increased, if necessary, by enlarging some system tables.)

2. Ability to log in on remote systems. This has been accomplished by writing a user-level program incorporating the TELNET specifications (RFC #158)* and using interprocess communications. Our initial version of TELNET has been coded in LISP.

3. Ability to make ADEPT available to remote users. The software provides up to five job entries to remote users in addition to the ten local job entries. Initially, however, we may administratively limit the external ARPA community to one or two jobs.

---

* References to RFCs refer to the "Request for Comments" series of informal ARPA Network documentation.

4.1.1 Progress

HOST-to-HOST Protocol Implementation

Version 2 of HOSTOSS, which includes the new protocol features specified in RFC #107, is now debugged and working. The interprocess communication feature has been tested via communication between two LISP systems running under ADEPT and communication with test facilities at The RAND Corporation.

The TELNET program, which was designed to facilitate access to foreign (outside SDC) systems, has been debugged and tested, and was used to checkout Version 2 of HOSTOSS. Our TELNET program conforms to the specifications of RFC #158 and includes a self-explaining capability to aid beginning users and to help people who use it only occasionally to remember details they may forget.

The LOGGER works and uses the initial connection protocol (ICP) defined in RFC #156. The LOGGER and ICP have been tested by RAND and by logging SDC's TELNET program into ADEPT via the Network. In both cases we successfully logged in and loaded a program.

Distributed Data Base Study

A study of three approaches to data sharing on computer networks (SDC document TM-4760) brought us to the conclusion that the "integrated" approach is worth pursuing. The main advantage of this approach is that it facilitates the integration of existing data management systems and their associated data in such a way that data can be shared on a computer network. The integrated approach has two primary components.

1. A common data management language to be used by all users who wish to share data on the computer network. The language should have the power to define data (including data structure and data relations); to update, modify, and retrieve the data; and to add to and delete existing data.

2. Interfaces that translate requests in the common language to requests in the languages of existing data management systems. For the purpose of having a fail-safe system, it is desirable to locate the interfaces at the nodes where their associated data management systems are. However, it is possible to locate all the interfaces at a single node.

We chose to concentrate our efforts on studying the interfaces by using the natural-language compiler of the CONVERSE English data management system developed at SDC (see Section 3.1). CONVERSE accepts requests in English

and represents them in a formal intermediate language (IL). IL is the common language that will translate requests into the different data management languages. We chose to experiment initially with the languages of DS/2 (a proprietary SDC data management system), DM-1 (which will be available at RADC), and FFS-NIPS (a data management system used in military agencies).

Data management languages strongly depend on the logical structure of data. IL depends on a relational data structure, in which one can describe relations between elements of sets. The function of an interface is to translate a request in IL, described in terms of a relational data structure, into a request (or a series of requests) in terms of the data base structure of the target data management system. Two major questions currently being investigated are:

1. Is the relational data structure assumed in IL general enough to represent a variety of system data structures, such as hierarchical (tree) structures and network structures? If not, what data structure should a common language be based on?

2. How can a request in IL be translated into a data management language based on a different data structure?

Coordination with Other Projects

The personnel of the Networks project offer their consulting services to SDC's other ARPA research projects. We have worked with the Graphic I/O project to use the ARPA Network to communicate graphic-tablet data between programs at SDC and MIT-DMCG. (See Section 3.2.1 of this report.) We have also consulted with our systems programmers to set up a background job to run under ADEPT that will handle the MAILBOX protocol discussed in RFC #196. (See section 5.2.1.)

We have pursued HOST-to-HOST protocol developments and other Network coordination efforts through attendance at the quarterly Network Working Group meeting and as a member of the TELNET and data management committees. We also participated in SRI's meeting concerning TNLS, which is the on-line system to be used by the Network Information Center.

4.1.2 Summary

HOST-to-HOST Protocol Implementation

The ADEPT time-sharing system running on the IBM 360/67 computer has been integrated into the ARPA Network. Interprocess communication, TELNET, initial connection protocol, and the LOGGER have been debugged and tested, both internally and by RAND. During this contract year the Network component of

ADEPT, called HOSTOSS/360, was coded and debugged. In addition to HOSTOSS/360, which is coded in IBM 360 assembly language, additional development of HOSTOSS/516 continued, network primitives were added to the LISP system, and TELNET was coded in LISP.

Distributed Data Bases

During the past year we have identified the problems associated with data sharing on computer networks and have studied several alternative approaches to solving them. We have chosen an evolutionary approach that permits the use of existing data management systems as well as facilitating data sharing. We believe that this approach will promote the acceptance of the value of data sharing and eventually evolve into a system in which the best data management systems will be used the most.

We have chosen to concentrate on the problem of interface translators by using the intermediate language of CONVERSE as a common network data-sharing language. We are now identifying the problems involved in actually implementing the translators.

4.2 GRAPH-META

The purpose of our Graph-Meta research, on which this is the final report, was threefold. First, we wanted to explore the theoretical bases for designing compilers that would accept a program written not only by a professional programmer but by engineers or scientists with only rudimentary expertise in programming and, through the compilation process, optimize it--that is, produce program code whose efficiency would approach, if not equal, the efficiency of the same program written by an expert programmer. Second, we wanted to build into an optimizing compiler certain analysis capabilities that would result in tighter code than even the most skilled programmer is capable of achieving--capabilities drawing on the power of the computer to examine the complex interrelationships among segments of a large system program. Third, and consequently, we wanted to extend the GENERATOR language, developed earlier by SDC researchers, to make optimizations of these types possible. During the past two years, this research has progressed from the extension of purely syntax-directed compiling techniques, through code generation with local optimization, to the theoretical foundations for global optimization using graph-theoretic techniques. A treatise on the mathematical theory of global program optimization has been completed, extensions have been added to the GENERATOR language, and an experimental optimizing FORTRAN IV compiler has been written to validate the theoretical work.

4.2.1 Progress

During the past six months, we have focused on a number of problems, not fully resolved by other investigators, involving the solution of Boolean equations relating the availability of certain computations on entrance to certain blocks of code within a computer program. A major problem concerns strongly connected subgraphs, or loops, of the program that have multiple entry points--i.e., loops that can be entered from several parts of the program without first going through a common block of code.

Investigators at the National Science Foundation have detailed a technique of solving these equations by a non-unique transformation of the program graph into any one of a family of equivalent graphs such that all loops must be entered through a node that is unique to each loop. Such a graph is called a "split graph" because numerous nodes are split into identical copies, each placed into an image of the loop. Hence, in the split graph, at least one loop that had multiple entry points in the original graph is reduced to a loop having a single entry point. This type of transformation has been described primarily through diagrams; no well-defined mathematical algorithm has been forthcoming that would produce either a unique split graph for a given program graph or a choice function to evaluate alternative split graphs produced from a given graph. As a result, the construction of an optimizing compiler that takes full advantage of the mathematical theory of optimization has not been possible. Instead, the process has been to not recognize these parts of code as loops, and instead to perform only more localized optimization on them.

We have found that by applying a Boolean distributive lattice algebra to the prime cycles of a graph (that is, to those loops in the program that contain no proper subloops), numerous mappings of the graph into split graphs are possible. Once we have identified all possible split graphs, the remaining problem is to determine which of the split graphs is the most preferable. We have chosen one distinct type: that which can be partitioned into a maximum number of non-trivial intervals.

Not all existing optimizing compilers produce correct, safe optimizations. The reason for this may have to do with the system of Boolean equations relating the availability of various computations on entrance to blocks of code. The graph-theoretic solution is dependent upon the time at which the substitutions into simultaneous equations are performed. This problem does not occur in the solution of systems of simultaneous equations in algebraic fields in general, but the solution process does involve the minimization and maximization of two independent sets of equations that are assumed to have different initial conditions but that tend toward the common limit of an infite series of approximations. It is apparent that because we are working with a Boolean lattice algebra, and because each variable attains the value of 0 or 1, if at some time a premature substitution is made, a value that should ultimately reach a value of 0 could be promoted to 1, or vice versa. Since the limit of the series is achieved through a monotonically increasing or decreasing series, it is clear that such a mistake would make a correct solution impossible. Our understanding of the errors found in various optimizing compilers indicates that they have not taken full cognizance of this hazard. Hence, due to a lack of pessimism on the part of other investigators, it is possible that a number of computations will be assumed to be available within a certain block of code when, in fact, they are not.

We were made aware of these hazards through a random simulation of the analysis process using random program data. We have successfully implemented a solution of a series of redundancy equations for arbitrary graphs, splitting nodes where necessary. One optimization we consider desirable involves identifying a computation that necessarily occurs in every sucessor block of code exiting a given node, d, but that does not occur within d itself. Under appropriate conditions, introducing that computation within d would render it redundant with each of the successor blocks in the sense that its computation in d would lead to its being available on entrance to each successor block. The optimization does, of course, require a simple store of the computation with a special variable. If the computation occurs only once in each successor block, no execution time is saved by the optimization, but there is a saving in core allocation, making the program somewhat smaller than it would otherwise be. This process is referred to as "hoisting an expression," and we have rigorously described the configurations of code that permit it. We have also investigated the possibilities of an analogous process, called "sinking an expression," in which computations are moved forward in code, rather than backward; the theorems that describe these processes are listed in Chapter 9 of the textbook as Theorems 9.11 and 9.13.

We have paid considerable attention to guaranteeing a program's correctness, as well as its logical and computational integrity, as it undergoes optimization. A program that has been processed by a nonoptimizing compiler and that runs successfully should continue to operate correctly after it is processed by an optimizing compiler. Hence, we have made a thorough study of the criteria for identifying computations that should not be moved because moving them might produce a divide exception or program interruption that does not occur with the existing code. The result is a pessimistic optimizing compiler that maintains safety. We have determined that certain types of optimization, though they may result in loss of precision or the possibility of overflow or underflow in dealing with multiple-precision and floating-point numbers, have a high probability of being safe and correct. If a programmer is not concerned with the possible loss of precision or does not expect certain operations to produce the other exceptions, certain applications of the destributive, commutative, and associative laws of algebra can be applied to his program, causing it to run more rapidly.

Other guarantees we must honor for the programmer concern the formal specification of the programming language itself. For example, since all subroutines in a FORTRAN program must be capable of being independently compiled, it is not possible to make a thorough analysis of the data flow involved on either side of the invocation of a subroutine. Even though the entire subroutine and its effects on data, including data in COMMON, can be considered at compile time, a portion of the program might be recompiled at a later time, thereby seriously affecting certain of the optimizations that were performed.*

It is also necessary to consider the frequency with which any piece of code is expected to be executed in the flow of the program. Those parts of the program that are most frequently executed need to receive the most meticulous optimization, while those parts that are rarely executed, or are executed

---

* For example, if the computation a × b is performed immediately prior to a call on a subroutine and is again performed immediately after returning from the subroutine and either a or b is a global (common) variable, it is possible that the function itself modifies either a or b, in which case the second computation of a × b must be recomputed. If it is known that the function will have absolutly no effect on a × b, then recomputing a × b introduces an additional computation to the program that would not otherwise be needed. However, there is no guarantee that the function will not be recompiled in the future in such a manner that a × b would be affected by it. Therefore, a price is paid in the optimization that need not be paid where the language specification is altered.

only once, need not be optimized at all, since the optimization may have an insignificant effect on the program's overall execution time. Since it is difficult to ascertain the execution frequency of code, we have assumed that code in loops is executed more frequently than code that is not in loops, and that nested loops are executed more frequently than the loops that contain them. These assumptions can, obviously, fail--it is possible that code in a loop may never be executed.

If a computation in a loop remains constant throughout the loop's execution, it is desirable to provide a variable, outside the loop, that contains the computation. As a result, the computation is available to the loop without further need for recomputation. If this is done, however, and if the non-executed code remains in the loop, we have lengthened execution time rather than shortened it, since the computation that would not otherwise be performed must now be performed prior to entry into the loop.

We have studied methods for determining the relative execution frequencies of blocks of code within loops using Markov processes,* and we believe that with this method it may in the future be possible, provided that we have necessary information from a programmer and data from previous executions of his program, to perform a total program optimization.

The Optimizing FORTRAN IV Compiler

The first pass of the optimizing compiler scans syntax and allocates storage. Its output can be fed either to the second pass for optimization or to the third pass for code generation without optimization. The FORTRAN language allows storage allocation and overlay through the use of both labeled and unlabeled COMMON; the GENERATOR language proved to be well suited for programming this type of storage allocation in the first pass. The routine that does this was coded in only 3/4 of a page, while the corresponding code in the CDC FORTRAN, which was itself coded in FORTRAN, took more than six pages.

The second pass is the most important for our research because it does the optimization. It begins by breaking the code into blocks of consecutive instructions so that control comes in through the top of the block, passes straight through, and exits from the bottom. These blocks become nodes of a graph

* We found that the stochastic analysis of execution frequency is highly dependent on branching probabilities, and that a minor change in just one of the transition probabilities can result in an enormous distortion of certain execution frequencies.

that represents the flow of control of the FORTRAN program; this is done for the program and for each subroutine. The graph in Figure 4-1 represents the flow of control of a program. Each node of the graph is a block of code. Control enters the top of the block, goes straight through, and exits from the bottom. The graph may be thought of as a function from the nodes to their successors. It may be represented as a list of pairs; the first term of each pair is a node and the second term is a list of its successors.

Related nodes of the graph are grouped into what is called an interval. The intervals then become the nodes of a derived graph. The derived graph can be further reduced by grouping its nodes into intervals, forming another derived graph. The process continues until a graph is derived that has only one node. During this process, it is sometimes necessary to split nodes--i.e., to reproduce a node in more than one place in the graph without altering the flow of control. In doing this, the second pass follows the lattice-algebra algorithm that was formulated in the theoretical portion of this project.

A data-flow analysis is performed on these layers of derived graphs. Some optimizations, such as the elimination of common subexpressions, can then be performed on the basic blocks of the program, making use only of this data-flow information. Other optimizations, such as removing invariant subexpressions from loops and reduction in strength of operators in loops (i.e., substituting a less time-consuming computation, such as addition, for a more time-consuming one, such as multiplication), require going through the layers of derived graphs while the optimization is taking place. Finally, register allocation is optimized.

The third pass generates code. Although it is basically similar to other compilers that have been written in the GENERATOR language, it has capabilities not found in them. One is a new register-allocation scheme, written entirely in the GENERATOR language, that makes it unnecessary to transfer intermediate results from machine registers to temporary core storage while an expression is being evaluated.

Extensions of the GENERATOR Language for Global Program Optimization

Floating-point arithmetic and several types of arrays have been added to the GENERATOR language to facilitate global optimization. The floating-point arithmetic is used to evaluate constant expressions at compile time. One-dimensional Boolean arrays are used in the data-flow analysis to hold such information as which expressions are available on exit from a block or which are busy on entrance to a block. A two-dimensional integer array is used as a distance matrix in computing the prime cycles of a graph. (This is part of the node-splitting algorithm mentioned earlier.)

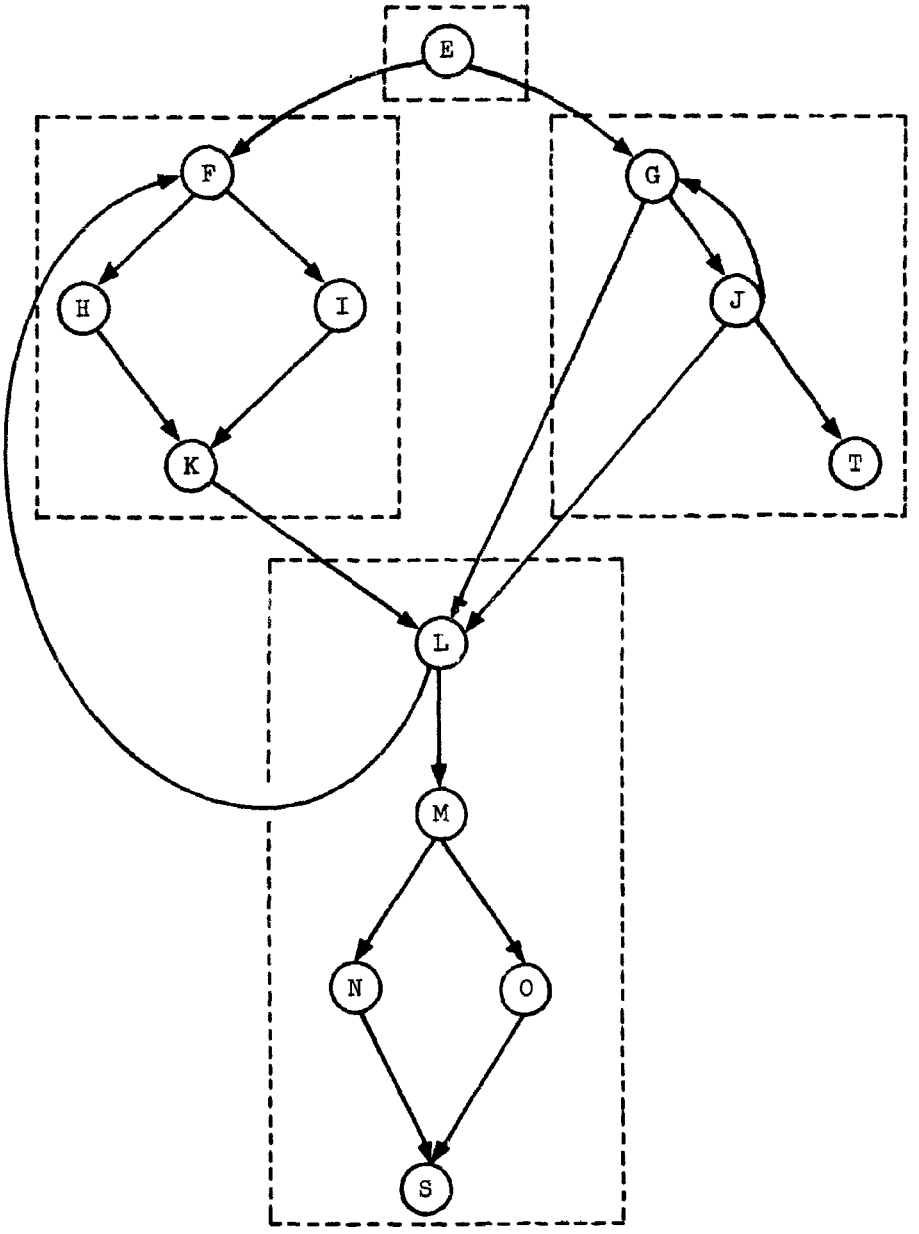


Figure 4-1. Program Control Flow Graph

We will illustrate the use of the GENERATOR language to encode optimization algorithms by describing a few functions that involve the flow-of-control graph. The first two, LOOK and PREDECESSORS, are elementary operations on the graph. They are followed by definitions of the functions INTERVAL, PARTITION, and CONDENSE, which group nodes together to form a simpler graph. Each time this collapsing operation is performed in an optimizing compiler, the various optimization functions have been applied over the intervals.

The function LOOK is used to find the value of any function represented in list structure. In GENERATOR language, it is defined as:

```
LOOK($[A,B], X) => .FIRST(B .SUCH A=X)
```

The function LOOK has two parameters: a list of pairs [A, B] and an argument X. It searches the list until a pair is found whose first term, A, equals the argument X, then returns the corresponding B. If the graph of Figure 4-1 is represented by a list of pairs in the variable GRAPH, then the successors of (for example) the node F may be computed by LOOK(GRAPH, 'F'); the value of this expression is the list ['H', 'I']. Because finding the value of a function that has been enumerated as a list structure occurs often, there is a special notation for this--GRAPH:[X]--which produces the same value as LOOK(GRAPH, X).

Although most operations work in terms of the successors of a node, there are times when predecessors are needed as well. This is more difficult because the structure representing the graph has been organized to facilitate the finding of successors. To find predecessors of the node in X, the following function is used:

```
PREDECESSORS($[A, B], X) => $(A .SUCH XεB)
```

Because this function is seldom used, no special notation has been provided for it.

A control-flow graph is partitioned into subgraphs called intervals. The first node of an interval, called its "head," is the node to which control passes. This greatly reduces the analysis required in the optimization process, since the interior nodes of the interval may be reached only by first executing the code in the head.

The following GENERATOR function is the algorithm for finding an interval, given a graph and a node to be used as the head:

```
INTERVAL(GRAPH, H) => $Z
     .WHERE $Z := [H]; X := NIL;
     .LOOP UNTIL X == $Z:
          X := $Z;
          $W := MAKESET([$-[GRAPH:[Z]]-]) ¯ $Z;
          $Z := [-[X]-, ~[$(W .SUCH PREDECESSORS(GRAPH, W)≤≤X)]-] .END
```

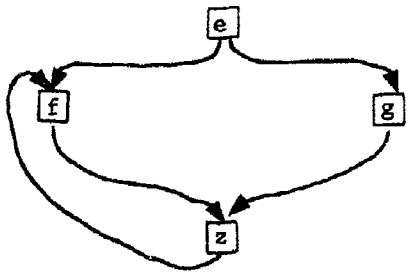
GRAPH, again, is the graph, represented by a list of pairs, and H is the interval head. The list of nodes that is returned by the function is called $Z. The algorithm begins by initializing the list $Z to the list containing only the head. The variable X is initialized to the empty list. Each time the loop is executed, new nodes will be added to $Z. The variable X will contain the previous value of $Z, and the loop will continue until no new nodes are added--i.e., until X contains the same nodes as $X, X==$Z (though possibly in a different order). $W is set to the list of successors of nodes in $Z that are not themselves in $Z. The function MAKESET removes duplicates from a list. The ¯ symbol used as an infix operator yields the difference of two sets; only those members of $W whose predecessors are already in $Z can be added to $Z. Subset is represented by the relational infix operator ≤≤.

Now we are ready to partition the graph GRAPH.

```
PARTITION(GRAPH) => P
     .WHERE P := NIL;
     $[A, $B] := GRAPH; E := $A; $Z := [E.1];
     .LOOP UNTIL $Z = NIL:
          $X := $INTERVAL(GRAPH, Z);
          P := [~[P]-, -[$X]-];
          $Q := [$-[X]-];
          E := E ¯ $Q;
          $Z := MAKESET([$-[GRAPH:[Q]]-]) ** E .END;
     .IF E ≠ NIL
          .THEN .SPEAK 'UNREACHABLE BLOCKS -- '; PRINT(E) .END
```

The value P, which is returned, is a list of intervals. Each node of GRAPH that can be reached will appear in exactly one of the intervals; nodes that cannot be reached will not appear in the partition. Within the loop, E will

represent the nodes of the graph that have not yet been included in the partition P. $Z will contain a list of interval heads whose intervals have not yet been included in P. Inside the loop, $X is set to all the intervals whose heads are in $Z; these intervals are then added to the partition P. $Q is set to the list of nodes that have just been added to P. In the next statement these nodes are removed from E, which contains the list of all nodes not yet included in P. In the last statement of the loop, a new set of interval heads is created, consisting of only those successors of the nodes just added to P that are not yet in P. (The infix operator ** represents intersection of sets.) Any nodes remaining in E at the end of the loop cannot be reached and are not included in the partition. The partition can then be formed into the following graph:



The function that performs this can be expressed in GENERATOR language as:

```
/CONDENSE($$P, GRAPH) => $[$P.1, $++GRAPH:[P]**H ¬ [$P.1]]
        .WHERE H := $($P.1)
```

The argument $$P is the partition and the original graph is called GRAPH. The variable H is set to the interval heads, which are used for the nodes of the derived graph. The expression $++GRAPH:[P] forms the union of the successors of the node P.

The Optimization Text

The treatise entitled A Mathematical Theory of Global Program Analysis has been completed and published as an SDC document series. The first three chapters have been approved by ARPA for unlimited publication, and the remaining chapters are in the process of being submitted for approval. Once the necessary approvals have been received, a publisher will be sought. The complete table of contents is as follows:

4.2.2 Summary

A treatise on the mathematical theory of global program optimization has been completed. In the course of its completion, several new theoretical results were obtained, the most important of which is a node-splitting algorithm based on lattice algebra. Other results involved a general algorithm for hoisting, an area of optimization that had been investigated only superficially by

other workers. New features, including floating-point arithmetic and Boolean and integer arrays, were added to the GENERATOR language to facilitate optimization. An experimental optimizing FORTRAN IV compiler was written, which proves that the GENERATOR language with its new features is a tool well suited for the production of practical optimizing compilers based on the mathematical techniques of global program optimization as described in the treatise.

4.3 STAFF

Systems Research Staff

E. Book, G. D. Cole, Managers

Networks Project

Dr. R. E. Long, Principal Investigator

Dr. A. Shoshani
A. S. Landsberg
Janet Troxel, NIC Station Agent (part time)

Graph-Meta Project

D. V. Schorre, Principal Investigator

M. Schaefer

4.4 DOCUMENTATION*

Long, R. E. ARPA Network Project--Status and Goals. SDC document N-(L)-24451. February 1971.

Schorre, D. V. GRAPH META. SDC document N-(L)-24448. February 1971.

Shoshani, A. Distributed Data Base Study. SDC document N-(L)-24452. February 1971.

Shoshani, A. Three Approaches to Data Sharing on Computer Networks. SDC document TM-4760.

*Documents in the SDC "N" series are internal working papers and are not available for public release.

5. INTERACTIVE SYSTEMS

Interactive systems projects reported here include Problem Solving and Learning by Man-Machine Teams, Time-Sharing, and LISP Extensions. Although they are part of the larger task of Systems Research covered in Section 4, they are reported separately here because they all deal with man-machine interfaces, whereas Section 4 tasks focus on internal, machine-machine problems. Furthermore, two of the projects--Time-Sharing and LISP Extensions--are "shadow activities" in that they are mandatory supporting efforts to the more visible research reported earlier.

Because the operation of Gaku, a man-machine problem-solving system, is dependent upon the man-machine communication language, the major effort during the contract period has been devoted to implementing the User Adaptive Language (UAL) on the ADEPT time-sharing system, using LISP 1.5 as its source language. More than 80 percent of UAL's features are now implemented. Updated documentation of UAL is available (Hormann, et al., SDC document TM-4539/000/01).

Our ADEPT Time-Sharing System supports the bulk of our research program, and, although no research effort is specifically expended for its expansion, development work is carried on to accommodate the research objectives of SDC's other ARPA research projects, most notably Networks, CONVERSE, and Problem Solving. Developments embodied in ADEPT Releases 8.8 and 8.9 include a near-doubling of user-program memory to 85 pages (approximately 348,000 bytes) of core, completion of the Object Sub-System control mechanism, development of a combined hardware/software communications front end for the Programmable Controller, and further efforts to improve system reliability.

5.1 PROBLEM SOLVING AND LEARNING BY MAN-MACHINE TEAMS

The primary objectives of this project are to explore ways in which a man and an appropriately programmed computer can augment each other's capabilities and to develop techniques by which those augmented capabilities can be used effectively in a variety of decision-making and problem-solving situations. Teaming man with an "adaptive" machine that can be made to "co-evolve" with him through interaction is expected to be needed especially in those complex problem situations for which complete analyses and detailed decision making in advance are not feasible. Such situations arise mainly from the combinatorial complexity and uncertainty of possible events and their impacts and from the incompleteness and impreciseness of information in changing problem situations.

5.1.1 Progress

Since the development of Gaku is dependent upon man-machine interaction, the major effort during the past six months was spent on implementing and documenting the User Adaptive Language (UAL) on the ADEPT time-sharing system, using LISP 1.5 as its source language. UAL is used for two purposes: as a programming language, it is used for initial Gaku implementation; in its extended form, it is used for user-Gaku interaction in problem solving and for designer-Gaku interaction in system modification.

More than 80 percent of UAL's features have now been implemented. Major efforts have been directed at (1) refining the current version of UAL, (2) documenting its design and user-oriented features, (3) conducting detailed analyses of experimental data gathered from the earlier Shimoku experiments (supported in part by ARPA and in part by the Office of Naval Research) and interpreting the results, and (4) continuing conceptual work in man-machine techniques in problem solving, planning, and evaluative judgmental processes.

A report, published jointly by SDC and the UCLA School of Architecture and Urban Planning, summarizes earlier work in artificial intelligence research, machine-aided problem solving, and the Gaku design that has evolved, and presents detailed accounts of the Shimoku experiments. Some highlights of the experimental results are summarized below. Working exclusively with the "objective game records" kept by the computer of the subjects' actions, it was possible to group their problem-solving procedures into three main types: Incremental (INC), Master Planned (MP), and Adaptive Master Planned (AMP). INC procedures resulted in final scores ranging from -64 to 110. All MP and AMP scores were over 110, MP peaking at 185 and AMP at 241 points. Thus, task

performances describable as generated by different types of procedures had detectable consequences in the effectiveness with which elements of the experimental problem situation could be handled to attain the stated objective.

- The INC procedure can be characterized as a short-range, piecemeal attack on a problem with a collection of rules of thumb that suggests ways to modify the execution of the basic incremental approach. Attempts to execute incremental procedures while taking note of many interrelated elements and events seriously overloaded even highly intelligent subjects and led to oversimplification and disorganization in weaker subjects. If the rules of thumb were appropriate and their application well managed, higher-range scores were attained by the INC group.

- In contrast, the group that used MP procedures prepared a totally preplanned goal for conditions of the environment in the form of a structure diagram and proceeded to construct the desired conditions according to an easily operationalized, nearly "algorithmic" prescription. Little adaptation to the given conditions of the environment was attempted, and the imposition of a master plan reduced the cognitive load on the subjects considerably. MP subjects, however, while considering their games "optimal," did not score as high as the AMP subjects.

- The AMP subjects employed strong features of both INC and MP procedures to prepare and use (a) rules of thumb to guide on-line decision-action processing adaptively and (b) a preplanned, but flexible, structural scheme. Adaptability and flexibility in both planning and in plan execution seem to set the AMP procedures apart from the others. The given conditions in the environment were not ignored, as they were by most MP subjects; instead, many elements of the given environmental conditions were capitalized on toward producing the desired goal conditions.

A very general description of the experimental findings is that the study showed (a) how subjects' different understanding of the "real nature" of the same problem depended on their search for problem representations; (b) how the representations selected imposed different information-processing demands; and (c) how these representations permitted different procedures (INC, MP, and AMP types) for formulating solution steps--which, in turn, determined different degrees of success.

5.1.2 Summary

To explore and develop effective techniques for man-machine cooperative problem solving, efforts during the course of this research have been concentrated on the following four interrelated groups of questions:

1. What machine capabilities, including adaptivity, are needed to effectively couple human and machine functions? What can be preprogrammed and what must be left to man-machine interaction?

2. What are some of the more apparent weaknesses and strengths of human cognitive processes? What cognitive limits can be extended, and what weaknesses can be fortified, by man-machine techniques? How can man's special faculties, such as intuition, imagination, inductive reasoning, and pattern recognition be promoted and put to good use in man-machine partnerships?

3. What characteristics of problem situations are especially in need of man-machine synergistic work, and from which can substantial payoffs be expected?

4. What type of language is needed for man-machine communication toward man-machine synergy--to enable the man to express evolving concepts and problem-solving methods dynamically--i.e., to facilitate interaction even at the problem-conceptualization and definition stage as well as at the exploratory and intuition-guided stage of problem solving?

Previous models of Gaku, a man-machine system used as a vehicle for exploring these questions, were geared mainly to autonomous evolvement of problem-solving capabilities. The new design of Gaku is especially geared to provide assistance to team planning and problem-solving groups, with structured coordination that permits higher-level planners to maintain broad and comprehensive views of problem situations while exploiting the detailed knowledge and specialized skills available in the lower echelons of problem-solving and planning organizations. Three major efforts involved were (1) to gain a deeper understanding of human intellectual behavior in complex problem-solving situations, aiming toward extending and fortifying human capabilities; (2) to develop a man-machine interactive language that will better enable man to express evolving concepts and problem-solving methods dynamically; and (3) to examine a variety of real-world problems that may be amenable to a man-machine approach.

5.2 TIME-SHARING

The ADEPT Time-Sharing System Executive functions as the operating system for SDC's ARPA research projects and serves as an experimental basis for systems research in time-sharing, networking, natural input/output, and natural English data management. The time-sharing research activities center on a combined hardware/software communications controller consisting of a Honeywell DDP-516 computer and several specially developed interfaces. Connected to the IBM 360/67, which supports the primary requirements of the ADEPT system, the DDP-516 handles the high-overhead interactive device functions such as display refresh, interrupt handling, and terminal I/O buffering.

During the past six months, the Time-Sharing project was concerned with the hardware/software systems design and implementation related to the continued support of the ADEPT user community and with providing several enhancements to the system capabilities. These changes, motivated by new user requirements, have required that a careful balance be maintained between system flexibility and system reliability. Several such features that have contributed to this balance are discussed in the following section.

5.2.1 Progress

Communications Multiplexor

The DDP-516 Programmable Controller handles all of the interactive terminals of the ADEPT system, including display devices and teletypewriter terminals. The latter devices were initially limited to a maximum of 15-character/second operation, which allowed software bit-sampling of terminal input/output instead of the more expensive hardware approach. However, accommodating higher-speed teletypewriter devices with software resulted in excessively high overhead in the Programmable Controller and limited its ability to handle other tasks effectively. A cost-effective solution to this problem was found through the use of a modified communications multiplexor that forms the characters and transfers them directly to the core memory of the DDP-516 via a synchronous single-line controller. This system, now in use with our ADEPT system, provides the capability of supporting terminal rates of 10, 30, or 60 characters/second while, at the same time, freeing a significant amount of processor and storage resources. The estimated processor overhead has been reduced from more than 50% to approximately 25%, and the storage requirement has been reduced by more than 2,000 bytes.

We originally intended to use some of the freed DDP-516 resources to implement a memory-protect system to prevent user-oriented programs in the DDP-516 from damaging the executive code. The apparent need for such a system arose from the existence of several "non-supervisory" DDP-516 programs, such as those utilized by the Graphic I/O and Networks projects. However, the occurrence of system failures due to such programs has been reduced to the point that

implementing the memory-protect system became unnecessary. This was brought about by the development of appropriate change-control procedures and the extended "shakedown" period during the development of the multiplexor.

ARPA Network Interface

Five ghost-job table entries were added to the system to support remote ARPA-Network users wishing to operate under ADEPT. These jobs have been pre-armed by our system-initialization routine to trap all object program and system supervisor calls. These jobs differ from an ADEPT interactive job in that they may not read or write a terminal directly and are pre-armed for hardware interrupts; the processing of these interrupts is under the control of HOSTOSS rather than the ADEPT Executive.

An operator's subjob table entry was added to the system to support the MAILBOX Protocol (see Network Working Group RFC #196). The MAILBOX Protocol will permit the Network Information Center to deliver and receive messages and documents from remote sites without having to know the details of path-name conventions and file-system commands at each site. An ADEPT operator's sub-job receives its commands from the operator through a shared-page facility and can write the operator's terminal but cannot read it directly. An operator's command, /LISTEN, was added to the system to cause the subjob to wake up, log in, and issue a supervisor call for command filter activation; as a consequence, the command string, /MAILBOX, is processed by the Extended Executive. This will cause a LOAD and GO command for the LISP program named, MAILBOX, to be processed. The appropriate protocol for the MAILBOX program will be incorporated into this program during the next contract period.

Extended User Capability

The planning phase for extending the number of interactive ADEPT users from 10 to 15 has been completed and will be implemented early in the next contract period. This task will be more difficult to implement than our ghost-job facility because it will require four system modules, involving both the DDP-516 and the 360/67, rather than one 360/67 module. Also, interactive jobs require more table space than non-conversational jobs, and resident routines may be shifted to upper core or to swap storage to meet this requirement. As a consequence, new system-tape fabrication will be required.

Extension to ADEPT F-Level Assembler

A library facility was added to the ADEPT interactive F-level assembler. This assembler is used primarily by ADEPT system programmers during time-sharing hours; the library facility will allow system support personnel to share commonly used system macros during the assembly process. The macro library is created and placed on auxiliary storage through an object program called LIBRARY, and the OS-360 Macro library is available to ADEPT programmers via this facility.

System Reliability

Efforts to improve the software and hardware reliability of the ADEPT-67 system have continued during this period. Mean time between failures (MTBF) for the last six months has varied between three and fifteen hours. The changes in the MTBF can be attributed to new user support functions incorporated into executive releases and to the random nature of hardware and software errors in a dynamic research-support system such as ADEPT.

5.2.2 Summary

The ADEPT time-sharing system was significantly modified to support our changing system environment and user needs. The first of these changes was the transfer from a dedicated IBM 360/50H to a block of time on the SDC 360/67I to meet the constraints of a smaller operational budget, while providing additional core memory to support large-program users. The I-core (512K-byte) memory of the 360/67 allowed us to extend the allowable user-program space from 46 pages to 85 pages, an extension that was readily exploited by several users.

Other system changes were required by the Networks project. ADEPT release 8.9 incorporated these changes and has been in regular operational use since April 1971. Building this ARPA-Network interface involved redesigning our earlier batch-monitor job (an operator's subjob) so that it would support an executive Network Control Program and an object process that runs in the supervisory state. Five pre-armed ghost jobs were added for remote Network users, and one operator's subjob was added to support the MAILBOX protocol. Other changes involved adding a supervisor-state call (SVC) and an executive, internal LOAD and GO call to the system.

A third area of change involved replacing the software bit-sampling routines of our DDP-516 Programmable Controller with a more optimal and cost-effective combination of hardware and software. An inexpensive communications multiplexor was utilized for hardware character reassembly, with subsequent software processing remaining much as before. This change significantly reduced the processor and memory requirements of the communications controller, particularly in the case of handling the higher-speed terminals that are currently available. The system is presently supporting ASCII terminals operating at speeds of 10, 30, and 60 characters/second.

Other, less significant, areas of change have included enhancements to system performance, maintainability, and reliability in an effort to better support ADEPT users.

5.3 LISP EXTENSIONS

The SDC LISP 1.5 system is a proprietary SDC product written in 1968.* The system operates under the TS/DMS time-sharing system executive on IBM 360 computers. SDC has made the LISP system available (on a no-cost basis) to ARPA for many years. It is the basic product that was modified for operation under the ADEPT system. The principal users of the LISP 1.5 system are the ARPA-sponsored CONVERSE and Problem Solving and Learning by Man-Machine Teams projects (Sections 3.1 and 5.1, respectively).

5.3.1 Progress

During the past six months, efforts were focused on three extensions and refinements of the system as it was configured at the end of the first six months of the contract year:

1. The program reference space was expanded by 3/4 page (768 reference words) to gain additional core space. Gaining additional core in this manner proved to be simpler and less costly than writing a Core Image Generator, as had originally been planned.

2. An I/O feature was added that allows LISP, via the Teletype or any other terminal, to read and write any of the 256 EBCDIC characters regardless of the character set of the device.

3. A provision was made to permit the evaluation of LISP symbolic code (function EVAL) without using the LISP compiler, thus permitting the removal of the compiler/assembler code (about 3506 words in core) by means of a mechanism, developed earlier in the contract year, that allows portions of LISP's binary program space to be unloaded onto disc, giving more space in core for data.

With these extensions, LISP is now capable of supporting all of the present and anticipated demands of its users, and no further extension work is planned.

5.3.2 Summary

During the past year, significant improvements were made in the SDC LISP 1.5 system's capability of supporting the increasing demands that have been made upon it by the SDC ARPA research projects that are its principal users. The most significant improvement during the year was the incorporation into the

---

*Barnett, J. A., and R. E. Long. The SDC LISP 1.5 System for IBM 360 Computers. SDC document SP-3043. January 1968.

system of a GROW feature that permits the system to expand from 46 to 85 pages, thus quadrupling the amount of data space available to users (from 11.5 to 47 pages). Additional data space was made available through a mechanism that allows portions of the system's program space to be swapped between core and disc. Improvements were also made in the capabilities and user conventions of the system's interaction, via the ADEPT Executive, with peripheral input/output devices, and in the system's editing capabilities. We believe that the system is now fully capable of supporting the increased requirements that will be made upon it during the next contract year, when major advances are planned in the CONVERSE, Voice I/O, and Networks project activities.

5.4 STAFF

Problem Solving and Learning by Man-Machine Teams Project

Aiko M. Hormann, Principal Investigator
A. Leal

Time-Sharing Project*

R. R. Linde, Leader

B. D. Gold
D. M. Gunn
Patricia Kribs
R. H. Larson
A. Tschekaloff

LISP Extensions Project*

J. F. Burger, Leader

J. A. Barnett
A. Leal
Dr. R. E. Long

5.5 DOCUMENTATION**

Hormann, Aiko M. "A Man-Machine Synergistic Approach to Planning and Creative Problem Solving: Part I." International Journal of Man-Machine Studies, Vol. 3, No. 2 (1971).

Hormann, Aiko M. Machine-Aided Value Judgements Using Fuzzy-Set Techniques. SDC document SP-3590. March 1971.

Hormann, Aiko M. "A Man-Machine Synergistic Approach to Planning and Creative Problem Solving: Part I," International Journal of Man-Machine Studies, Vol. 3, No. 2 (1971).

Hormann, Aiko M. "A Man-Machine Synergistic Approach to Planning and Creative Problem Solving: Part II," International Journal of Man-Machine Studies, Vol. 3, No. 3 (1971).

Hormann, Aiko M., A. Leal and D. Crandell. User Adaptive Language (UAL): A Step Toward Man-Machine Synergism. SDC document TM-4539/000/01.

---

*The nature of this task is such that it does not require full-time activity, but the part-time services of a number of people. Only the principals are noted here.

**Documents in the SDC "N" series are internal working papers and are not available for public release.

Hormann, Aiko M., S. Kaufman-Diamond and C. Martin Cinto. Problem Solving and Learning by Man-Machine Teams (Final Technical Summary Report to The Office of Naval Research). SDC document TM-4771.

Landis, D. ADEPT Performance Statistics--Calculation, Interpretation, and History. SDC document N-24469. April 1971.

Larson, R. UTIL User's Guide. SDC document N-24276/302. July 1971.

Linde, Richard R. ADEPT Extensions Schedule. SDC document N-(L)-24453. February 1971.

Linde, Richard R. Program Specifications for the I Core Version of ADEPT. SDC document N-(L)-24461. March 1971.

Peng, Te-Fu. The Correction of the Attention Bit Problem in the 516/360 Coupler. SDC document N-(L)-24465. March 1971.