

A STUDY OF INFORMATION IN MULTIPLE-COMPUTER AND
MULTIPLE-CONSOLE DATA PROCESSING SYSTEMS

K. B. Irani
I. S. Uppal
J. W. Boyse
et al

The University of Michigan

Approved for public release;
distribution unlimited.


FOREWORD

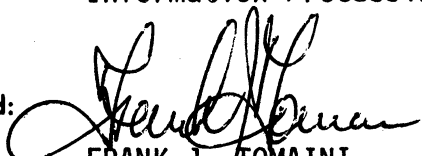
This report was prepared by Messrs. K.B. Irani, I.S. Uppal, J.W. Boyse, D.M. Coleman, D.L. Hinshaw, G.A. McClain, L.S. Randall, and A.M. Woolf of the University of Michigan, Systems Engineering Laboratory, Ann Arbor, Michigan, under contract F30602-69-C-0214, Job Order Number 55810000. Rome Air Development Center Project Engineer was Rocco F. Iuorno (ISIS).

The period of time covered in this report is April 1970 to March 1971. Contractor's identification number is Annual Report No. 4.

This report has been reviewed by the Information Office, OI, and is releasable to the National Technical Information Service.

This Technical Report has been reviewed and is approved.

Approved: 
ROCCO F. IUORNO
Project Engineer
Information Processing Branch

Approved: 
FRANK J. TOMAINI
Chief, Information Processing Branch
Information Sciences Division

ABSTRACT

This report documents the achievements from April 1970 to March 1971 of continuing research into the development and application of mathematical techniques for the analysis and optimization of multiple-computer, multiple-user systems. A summary of the theoretical investigations conducted, the major conclusions reached, and some typical applications are included.

TABLE OF CONTENTS

<u>Section</u>		<u>Page</u>
1	INTRODUCTION	1
	1.1 Contract Objectives	1
	1.2 Contract Requirements	1
	1.3 Progress Toward Contract Objectives	3
2	MESSAGE PROCESSING AND COMMUNICATION SYSTEMS	6
	2.1 Introduction	6
	2.2 Some Problems in Computer Message Processing and C ommunication System Design	9
	2.3 A CMPC System Synthesis P rocedure	22
	2.4 Example	28
	2.5 Conclusion	45
3	MULTIPROGRAMMED AND MULTIPROCESSOR COMPUTER SYSTEMS	51
	3.1 Multiprogrammed Systems Using Storage Hierarchies	52
	3.1.1 The Nature of the Problem	55
	3.1.2 Objectives	56
	3.1.3 The Model	57
	3.1.4 An Example	66
	3.2 Optimum Task Scheduling	72
	3.2.1 Markov Renewal Decision Processes	72
	3.2.2 Data Collection and Analysis	80
	3.3 Selection of Optimal Sets with Application to Computer Programming	89
	3.3.1 Selection of Optimal Program Pages in Multiprogramming System	90
	3.3.2 Paging	91
	3.3.3 Model	92
	3.3.4 Solution Technique for Grahpical Partitioning	92

<u>Section</u>	<u>Page</u>
3.3.4.1 Optimal Systems	92
3.3.4.2 Knowing Optimal Procedures	93
3.3.4.3 Heuristic Procedures	93
3.4 Multiprocessor Scheduling	96
3.4.1 Multiprocessor Scheduling - Assignment and Scheduling	99
3.4.1.1 Hard Real-Time Environment	103
3.4.1.2 Soft Real-Time Environment	109
3.4.2.2 Scheduling Parallel Processes - A Zero- One Programming Approach	111
3.4.2.1 Model and Notation	113
3.4.2.2 Assumptions and Limitations	114
3.4.2.3 Formulation of the Optimization Problem	116
3.4.2.4 Objective Functions	116
3.4.2.5 Constraints	119
 4 CENTRAL PROCESSOR DESIGN	 125
4.1 Data Path Optimization	125
4.1.1 Example	126
4.1.2 Model for the Study	130
4.1.2.1 Model Language	132
4.1.2.2 Model Architecture	133
4.1.2.3 Hardware Unit Library	134
4.1.2.4 Algorithm Library	135
4.1.2.5 Model of the Data Path	135
4.1.2.6 Optimization Criteria	136
4.1.2.7 Generality of the Model	137
4.1.3 Progress Toward a Solution	137
4.2 Microprogram Control	139
4.2.1 The General Design Method	139
4.2.2 PTL an Intermediate Language	144
4.2.3 Optimization	149
 5 DATA STRUCTURES AND THEIR REPRESENTATION	 151
5.1 Computer Memory Data Representation	151
5.1.1 Development of a Model	153
5.1.2 Optimization	158
5.2 Computer Graphics Systems	160

<u>Section</u>	<u>Page</u>
5.2.1 The General Approach	161
5.2.1.1 Topological Structure	163
5.2.1.2 The Picture Generator	170
5.2.1.3 Representation of the Topological Structure	172
5.2.1.4 The Optimum Implementation	172

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
2.1	CMPC System Configuration	8
2.2	Offices and Branches of a Large Banking System	11
2.3	File Request Origin Probabilities for Two Example Files	13
2.4	Variation in Processing Cost, Communication Cost and Total System Cost with Degree of Centralization of File Assignments	15
2.5	Comparison of Point-to-Point and Multi-Point Segments	15
2.6	Illustration of Changes in Edge Traffics and Total Traffic When File is Moved from One Node to an Adjacent Node	
2.7	Average Message Delay in a Communication Channel	21
2.8	Feasible Values of Channel Capacity in Two Channel Communication System with Performance Constraint	21
2.9	A Methodology for Computer Message Processing and Communication Design System	23
2.10	Modified Topologies Resulting from Tree Topology by Deletion of One Edge and Insertion of Another	24
2.11	CMPC System Design Methodology	25
2.12	Terminal Locations and File Parameters for Design Examples	29
2.13	Line Cost Function	30
2.14	CMPC System Designs Selected by Each Successive Improvement Step	31,32,33
2.15	Non-Optimal System Design Example	36
2.16	System Design Example with Network Topology Selection	37
2.17	System Design Example with Optimal File Assignments and Network Topology Selection	38
2.18	System Design Example with Optimal File Assignment, Tree Segmentation, and Network Topology Selection	39
2.19	System Design with Optimal File Assignment, Optimal Edge Capacity Allocation and Network Topology Selection	40

<u>Figure</u>		<u>Page</u>
2.20	System Design Example with Optimal File Assignment, Tree Segmentation, Optimal Segment Capacity Allocation, and Network Topology Selection	41
2.21	Comparison of Several CMPC System Design Procedures	44
3.1	System Analysis Model	59
3.2	System Model	59
3.3	Detailed System Model, Part 1	60
3.4	Detailed System Model, Part 2	61
3.5	System Diagram	67
3.6	Performance Versus Number of Programs	68
3.7	Queue Length Versus Number of Programs	68
3.8	Performance Versus User Program Size	69
3.9	Optimal Number of Programs Versus Program Size	69
3.2.1	Cumulative Distribution of Time (t) from Page Demand Until Page Available	82
3.2.2	Cumulative Distribution of CPU Time (t) Used Between I/O During Fortran Compilations	83
3.2.3	Cumulative Distribution of Time (t) Used Per I/O Request During Fortran Compilations	84
3.2.4	Cumulative Distribution of CPU Time (t) Used Between Page Faults During Fortran Compilations	85
3.2.5	Cumulative Distribution of CPU Time (t) Used Between Page Faults as a Function of the Number of Pages in Core for Fortran Compilations	85
3.4.1	A Typical Task Set	104
3.4.2	Initial Configuration of Task List	105
3.4.3	An Example Graph and Schedule	108
4.1	CPU Data Path I	127
4.2	CPU Data Path II	128
4.3	CPU Data Path III	129
4.4	Overall Structure of Model	131
5.2.1	Interactive Display Program as Described by W. R. Sutherland	162
5.2.2	A Picture and a Representation of Its Topology	164
5.2.3	Examples of Pictures for which the Topology of Each Picture Facilitates Response to Control Language Inputs	167

1. INTRODUCTION

1.1 Contract Objective

This effort is for applied research in the area of mathematical techniques for analyzing multiple computer, multiple console, real-time on-line data processing systems, and for analytical techniques and hypothesis to assist system designers and users in determining the optimum configuration, most complete utilization, and most efficient scheduling of this type of system.

The main objective of this work is to make it possible, through development and application of new mathematical techniques, to more optimally design and control computer systems. A computer system consists of a collection of electronic data processing machines, data transmission channels, and multiple user terminals, organized to efficiently service the computational needs of a geographically or functionally diverse population of users. Such systems permit: remote communication and manipulation of shared data bases; cooperative operations between user and computer (symbiosis); an immediate access to a high-capability facility for problem solving and data manipulation.

1.2 Contract Requirements

- 1) Exploration of Queueing Theory to enable the analysis of more general models of computer utilities and their subsystems.
Emphasis shall be concentrated on numerical techniques, and

shall include extension of earlier work on quasi-birth-and-death (QBD) models and the Recursive Queue Analyzer.

- 2) Collection and analysis of statistical data from existing systems to determine the validity of the mathematical models developed, and to isolate problem areas in need of attention. The techniques of computer data collection shall be studied.
- 3) Application of new mathematical techniques in conjunction with those previously available, to the analysis and optimization of hardware and software configurations of general purpose computers. These techniques shall be applied to typical systems in order to test the analytical methods and provide specific analyses/recommendations concerning the effectiveness of these systems.
- 4) Continued development of general design guidelines for time-shared computer systems with distributed processing capabilities. Distributed processing has become economically feasible because of rapidly decreasing small computer costs. This task shall extend previous investigations of remote display terminal structures.
- 5) Continued development of mathematical models for the optimal structuring of communication networks associated with computing systems.

- 6) Continued development of optimal design of storage systems and data base structures.
- 7) Continued exploration of Discrete Optimization Theory and Graph Theory in relation to application concerning the scheduling of programs in multi-processor systems. Also to investigate the use of these theories in relation to problems of program organization.
- 8) Development of new conclusions and rules which can be used by persons performing initial designs of real-time computer systems having a large number of user consoles. Such rules shall allow system designers to more rapidly choose the type of hardware/software system needed to fit a particular organization or problem.
- 9) Application of statistical analysis to data collected from various computing systems in order to gain an understanding of user demand structures and their effects on systems performance. A search for other theoretical approaches to the analysis of multiple computer systems shall be pursued.

1.3 Progress Toward Contract Objectives

The following sections details the progress made during the second year of this contract. Section 2 reports progress in the area of design of message processing and communications systems. In Section 3 we report research into the application of optimization theory to the problems

of program organization and program scheduling. Section 4 reports on efforts to apply mathematical techniques to the analysis and optimization of the hardware configuration of the central processor. Finally Section 5 concentrates on data structures and their representation in the computer memory.

During the past year Systems Engineering Laboratory Technical Reports Nos. 48[1] and 51[2] were published. The other four reports [3,4,5,6] are in the final stages of completion.

References for Section 1

- 1) V.K.M. Whitney, "A Study of Optimal File Assignments and Communication Network Configuration in Remote - Access Computer Message Processing and Communication System", SEL Tech. Report No. 48, The University of Michigan, Ann Arbor, Jan. 1971.
- 2) J. H. Jackson, "Optimum Implementation of Topological Structure for Interactive Computer Displays," SEL Tech. Report No. 51, the University of Michigan, Ann Arbor, Jan. 1971.
- 3) A. M. Woolf, "Analysis and Optimization of Multiprogrammed Computer Systems Using Storage Hierarchies", SEL Tech. Report No. 53, The University of Michigan, Ann Arbor, April, 1971.
- 4) J.W. Boyse, "Solutions of Markov Renewal Decision Process with Application to Computer System Scheduling", SEL Tech. Report No. 52, The University of Michigan, Ann Arbor, to be published.
- 5) L. S. Randall, "A Relational Model and its Optimization for the Representation of Structured Data Within a Random-Access Computer Memory", SEL Tech. Report No. 54, The University of Michigan, Ann Arbor, to be published.
- 6) D. Coleman, "On Binding Groups - A Quadratic Programming Approach in Zero/One Variables with Applications," SEL Tech. Report No. 56, The University of Michigan, Ann Arbor, to be published.

2. MESSAGE PROCESSING AND COMMUNICATION SYSTEMS

In this section we report research into the solution of several problems involved in the optimal design of computerized message processing and communication systems. The class of systems studied includes airline and hotel reservation systems, and time-shared computer systems. After a brief description of the general characteristics of the class of systems to which this research is pertinent in Section 2.1, the various subproblems encountered in the design of computer message processing and communication systems are presented in Section 2.2. The various subproblems and their solution techniques are then integrated into a comprehensive design procedure illustrated in Section 2.3. An example of a complete system design using the techniques of the previous sections is given in Section 2.4, and finally a summary of the specific contributions of this research is presented in Section 2.5. The details of the research reported in this section can be found in the report entitled, "A Study of Optimal File Assignment and Communication Network Configuration in Remote Access Computer Message Processing and Communication Systems" [5] , which was published during the last year.

2.1 Introduction

This work is concerned with the design of large on-line real-time computer communication systems. Both the size and the number of

such systems are rapidly increasing today. Three such systems are the Barclays Bank Ltd. on-line banking system serving 4,500,000 customers through more than 2500 terminals [1] , the American Airlines SABRE reservation system [2] , and the General Electric Corporation nationwide order processing system [3] .

The computer-communication systems studied will be composed of the four basic sub-systems illustrated in Figure 2.1., the user terminal system, an on-line communication system, a message processing system, and a file system. The user terminals are connected by a large telecommunication network to one or more processing facilities. Each processing facility may have its own data files or fetch the necessary data located at the other processing sites through a telecommunication network. Certain properties are common to all of these systems:

- Many users at widely separated geographic locations randomly requesting records of a sizable data base.
- A large data base consisting of files of homogeneous records.
- One or more processing facilities which select, process, and answer the user requests for information.
- An on-line real-time communication system linking the user to the processing facilities.
- Quantitative measures of system performance such as average message delay time, average line utilization, and

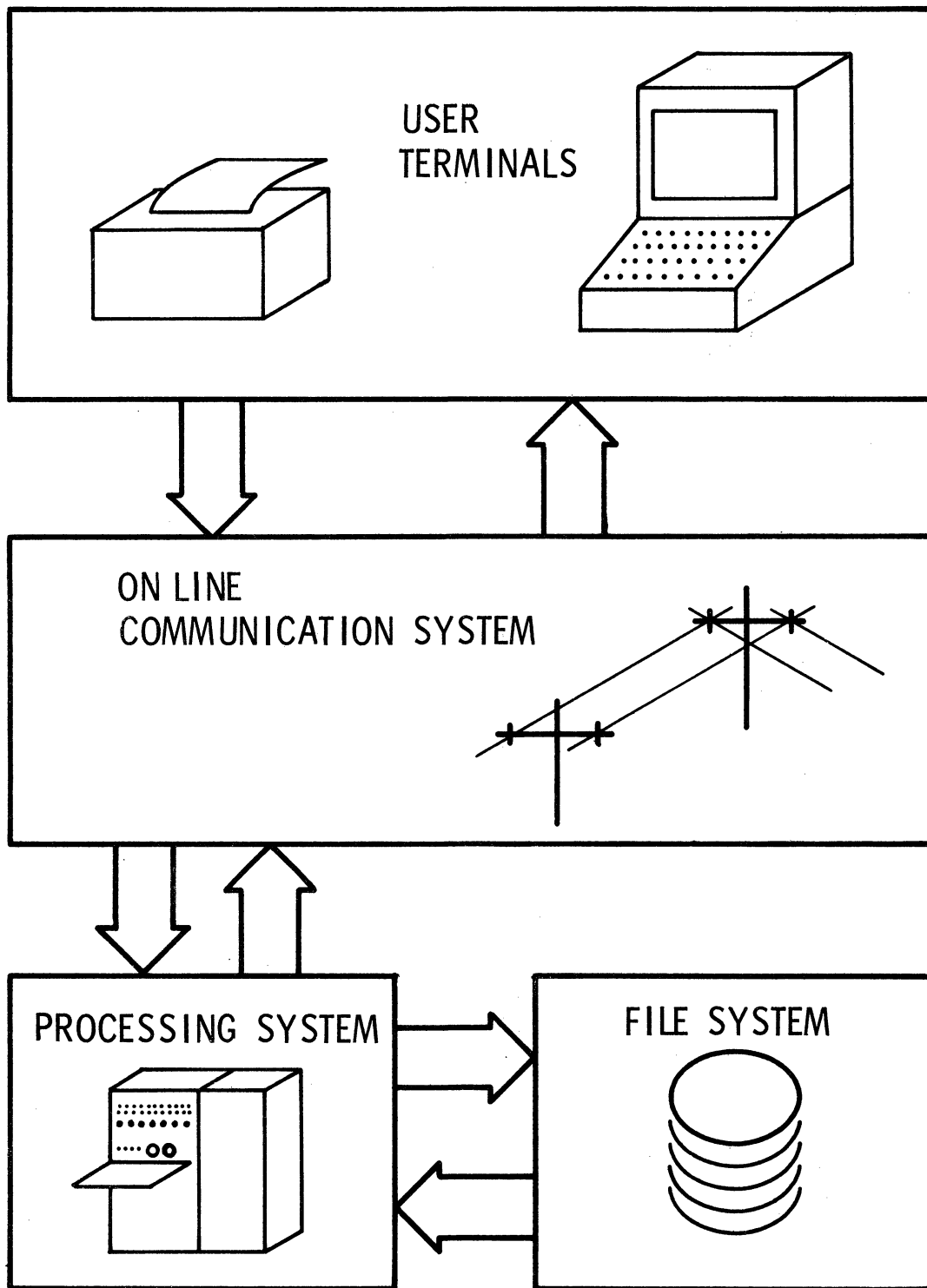


Figure 2.1 CMPC System Configuration

maximum hourly throughput.

- Performance constraints on the system, such as the requirement that the average message delay not exceed 3 seconds.

The goal of this research is the development of a comprehensive realistic optimal design procedure for these computerized message processing and communication (CMPC) systems. While this ambitious goal has not been completely attained, considerable progress has been achieved. A mathematical model of CMPC systems has been constructed and studied to identify the important design variables and to isolate specific problems for analysis and solution. Several of these specific design problems have been solved and the solution techniques organized into a unified system design procedure.

2.2 Some Problems in Computer Message Processing and Communication System Design

Since the installation of a large computer system requires a great initial investment of capital and time as well as substantial operating expense, it is desirable to design the least expensive system which will satisfy the system specifications and performance constraints. In this section, several problems arising in the design of these large systems will be discussed.

At the commencement of a system design certain information will be assumed known. The location of the users, the records of the

system data base, the probabilities that individual users' requests will be for specific records, and the rates at which users request records from the data base must be known before the design procedure begins.

In addition to this factual data, certain decisions must be fixed before the design can proceed. A measure of system performance — mean message delay, for example — and the value corresponding to the worst acceptable system performance must be selected. Often, additional constraints on the types of equipment to be used in the system are decided before the system begins. Many such possible constraints will become evident in the following description of the system design procedure.

Designing a communication system linking thousands of users to files containing billions of records is a very difficult problem. The first step of the design procedure is to reduce the problem to a more manageable size. The users of the system are gathered into a moderate number of terminal user groups, with one user location in each group serving as a communications terminal. Each user can communicate with his terminal, and each terminal can communicate with the other terminals. Users in distinct terminal user groups may communicate only by using their terminals as relay stations. The partitioning of 230 users into 11 terminal user groups for a banking system example is illustrated in Figure 2.2. Since the selection of terminal locations and the users connected to each is often a political or managerial decision rather than a technical decision, and since the user population of a system may change rapidly, a stable basis for system de-

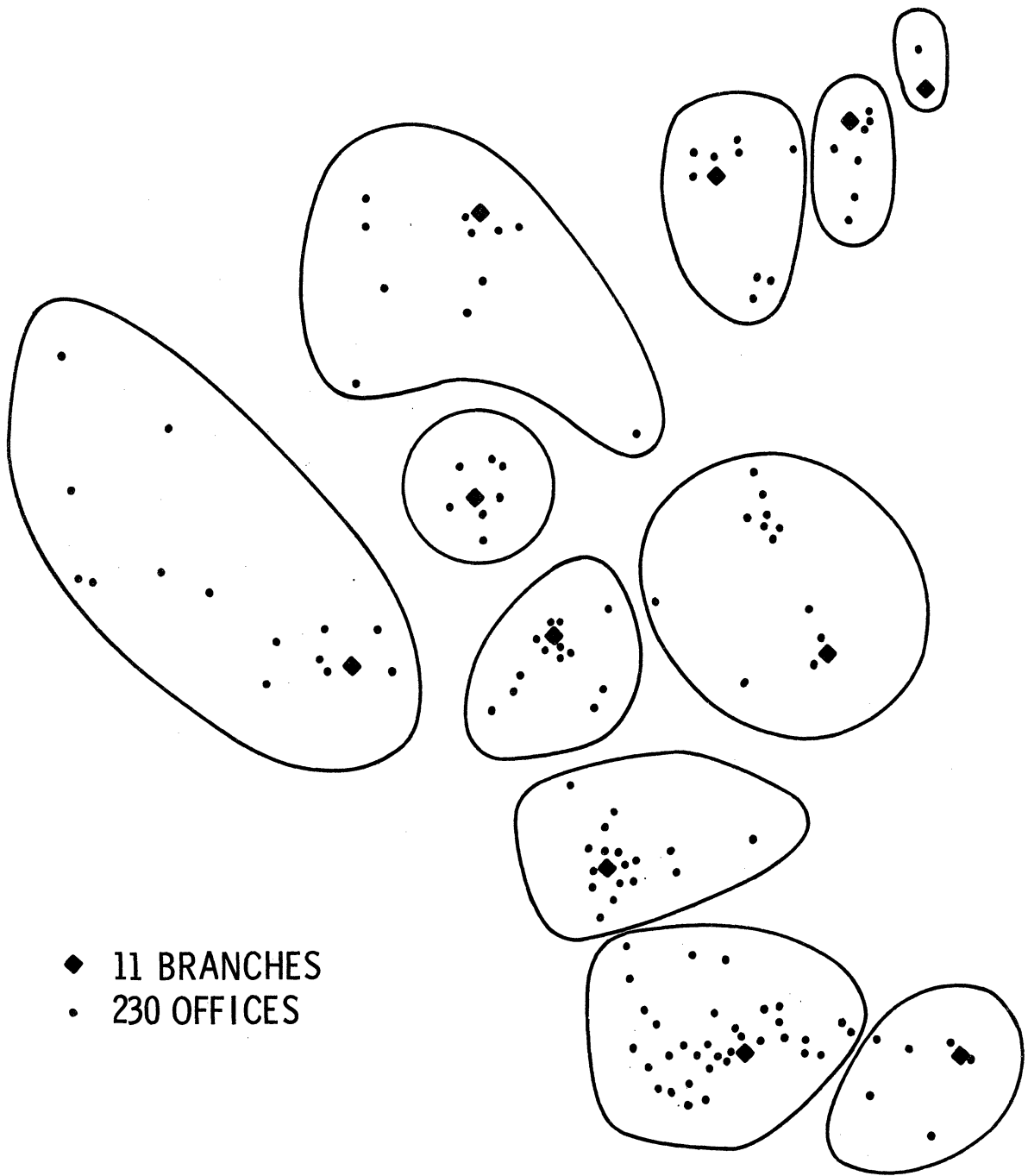


Figure 2.2 Offices and Branches of a Large Banking System (from {4})

sign has been assumed by fixing in advance the terminal locations.

A second reduction in system complexity results from partitioning the records of the data base into individual files of similar records. All records of a single file are assumed equally likely to be requested by the users at a specific terminal. Of course, the likelihoods of a record's being requested by users at different terminals may be different. Sometimes the grouping of files will be done by fiat or will be obvious. In the banking system, for example, each file could contain the savings account records for customers of a particular branch.

The file access request origin probabilities for two files are illustrated in Figure 2.3. At each of the twelve terminals is a number specifying the probability that a request for a record of that file originated with a user at that terminal. For file A nearly all requests are from users in the west; for file B most requests are from the east.

A major problem in CMPC system design is the selection of the proper sites at which to store the files of the system data base. If the cost of communication for a file is proportional to the product of the number of requests made by the distance those requests must travel, we can calculate the communication cost induced by locating the example files at each of the terminals. When cost is the product of traffic rate by distance, the least costly location for file A is terminal 4 (cost 1.92) and for file B is terminal 10 (cost 1.30). If, however, it is necessary to locate both files at the same terminal, then the optimal location is terminal 6 with cost 3.70. Note that

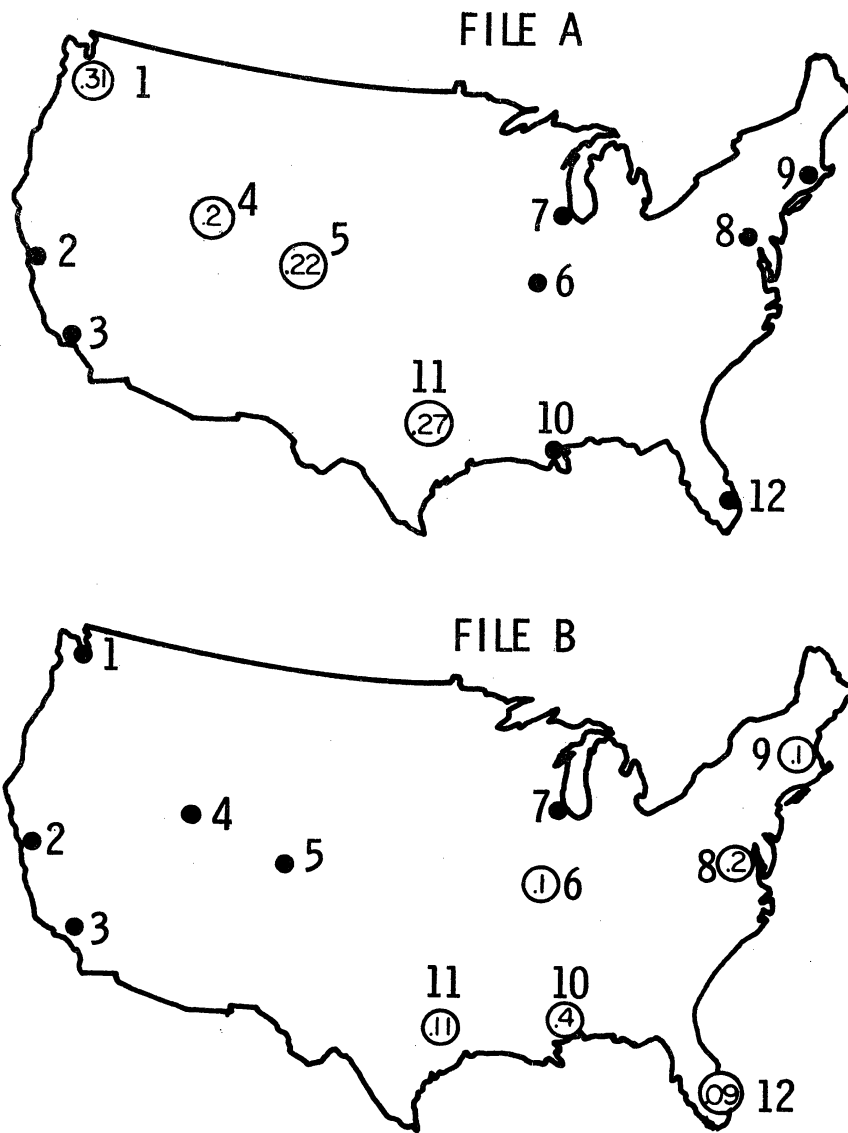


Figure 2.3 File Request Origin Probabilities for Two Example Files

the communication cost for locating both files at the same terminal is greater than the cost of locating each file at its optimal site since then both are located at nonoptimal sites. This example illustrates a problem in file site assignment represented more generally in Figure 2.4. There it is pointed out that while a centralized file site assignment results in lowest storage or processing cost, a decentralized file site assignment results in lowest communication cost. Hence neither a fully centralized nor a fully decentralized file site assignment results in minimal total system cost.

It is appropriate to be concerned with this problem of file site assignment, even though most existing computer communication systems are centralized. The number of multiple location computer systems is rapidly increasing; as the communication facilities become more readily available and less expensive, this increase will continue. Careful consideration of the potential savings resulting from partial or full decentralizations should be taken in the early phases of system design. If indeed, the results of such a study for a particular system indicate that a centralized configuration is most economical, that does not necessarily indicate that a decentralized configuration should never have been considered in the first place. Even though they are later rejected, as many options should be left open to the system designer as possible.

Although the problem of file assignment is important for the design of all message processing and communication systems, the other problems studied in this work relate more directly to systems in which the commun-

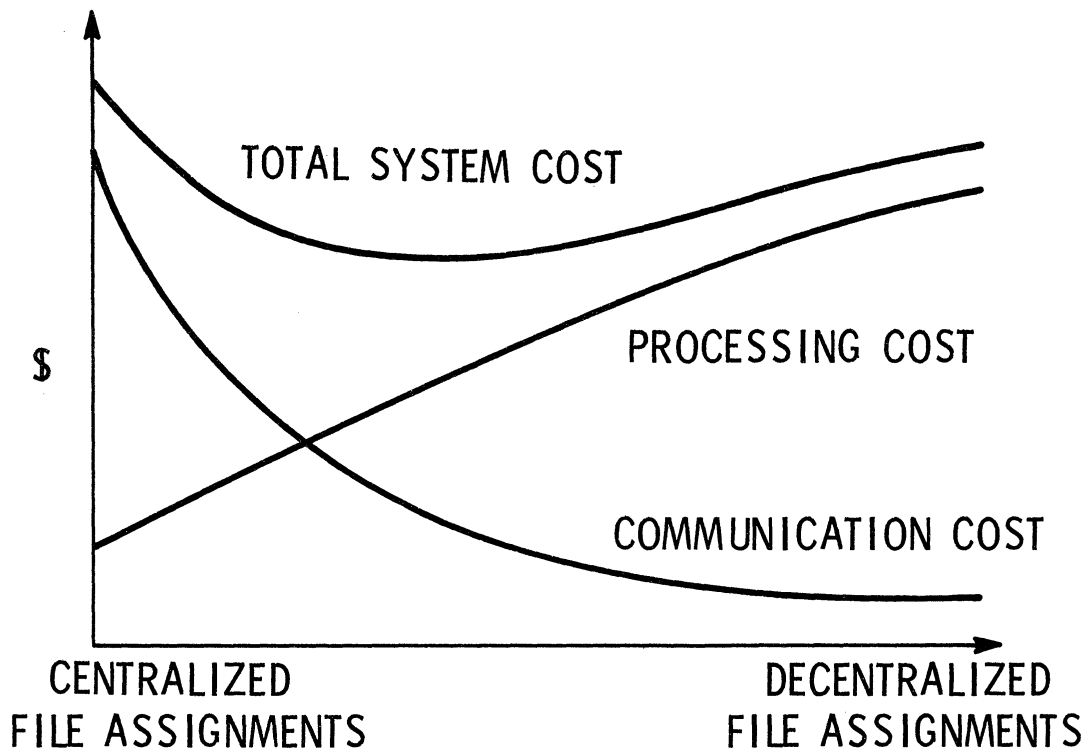


Figure 2.4 Variation in Processing Cost, Communication Cost, and Total System Cost with Degree of Centralization of File Assignments

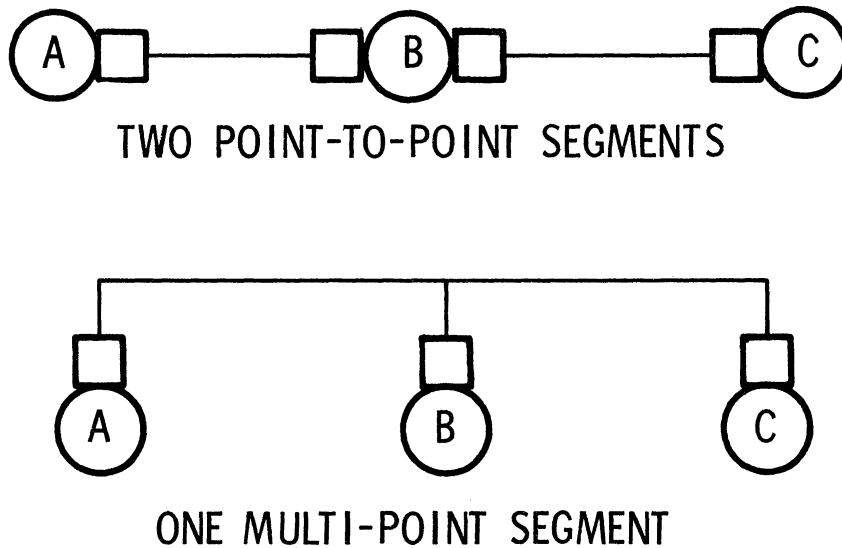


Figure 2.5 Comparison of Point-to-Point and Multi-Point Segments

ication facility is composed of leased private lines configured with a tree topology. This restriction of attention to tree networks deserves a word of justification. Although network reliability is an important consideration in system design, the additional cost of a redundant topology may not be justifiable. The many corporate communication networks operating today with a tree topology indicate a conscious decision that network reliability is not worth the additional cost. But even if redundancy is essential, the design of tree networks is important for cost minimization. One of the largest airline reservation systems has a communication network consisting of a basic tree network used during normal operation together with a minimal spanning tree used as a backup system when one of the links of the basic system fails. Further considerations explored in this research suggest that the random nature of message traffic is most economically handled with as few channels as possible. Hence, in many situations where there are alternate routes for message traffic, the system will be least expensive when only one route is used and the other closed down completely.

The cost of a private leased line network consists of two basic components, the line costs and the line interface costs. The line cost depends on the lengths of the lines leased and their bandwidth or maximum information transfer rate. The line interface costs depend on the actual transfer rate or capacity at which the lines are operated. By using different line interfaces, the same physical line may be operated at a variety of

capacities. Since higher capacity interfaces cost more, the system designer must choose interfaces to minimize cost but still satisfying the performance constraint.

The non-centralized leased line networks for inter-terminal communication facility have also been studied. The notion of multi-point segments is formulized and methods of finding segments which satisfy specified performance constraints are introduced. Multi-point segmentation is a technique used to reduce the number of line interfaces required for a communication facility. The basic technique is illustrated in Figure 2.5. In the upper diagram terminals A, B, and C communicate with the aid of two point-to-point communication lines which require four line interfaces. Traffic from terminal A to terminal C must be relayed through terminal B. In the lower diagram the three terminals are joined with a single multi-point line requiring only three line interfaces. In this case, each pair of terminals may communicate directly, but only one pair at a time. With certain traffic patterns, multi-point lines may be significantly less expensive than point-to-point line configurations.

When network traffics are low, interface costs are sufficiently smaller than line costs that the least expensive line configuration, the minimal spanning tree, is the optimal network topology. For higher traffics, the interface costs increase, and the selection of the optimal topology becomes a much more difficult problem. Methods for selecting inexpensive networks subject to a performance constraint are studied. These will be explained

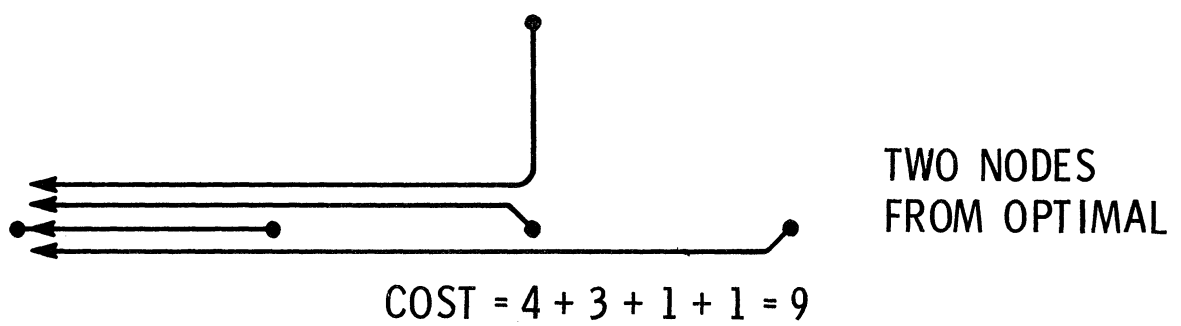
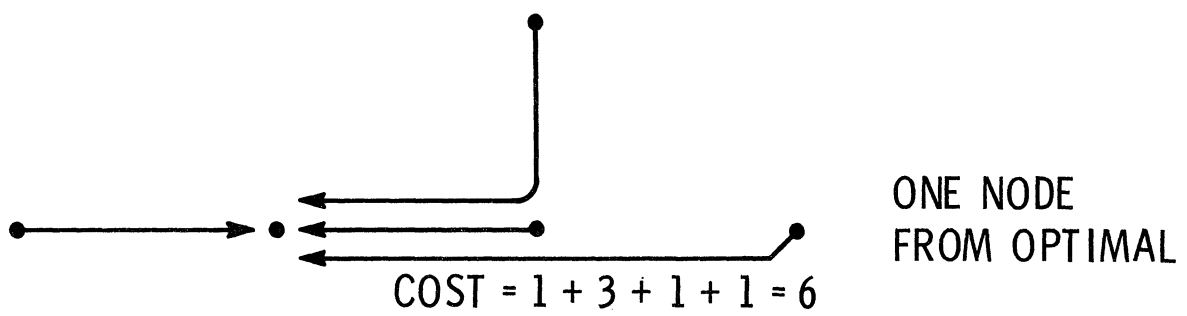
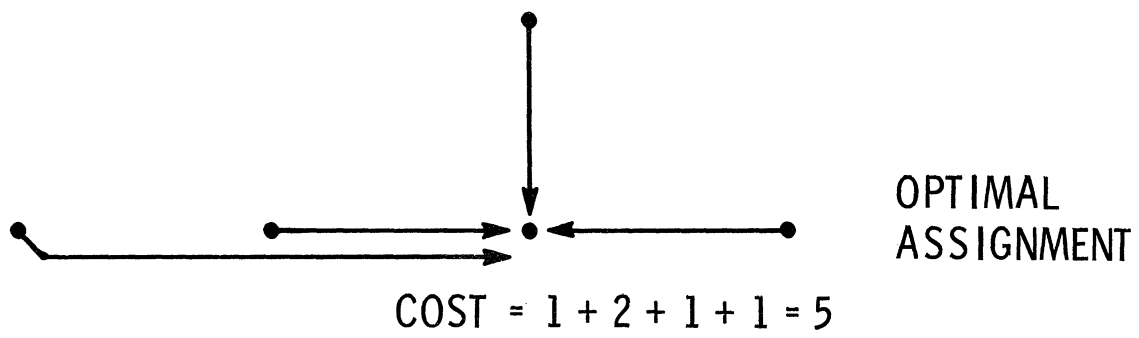


Figure 2.6 Illustration of Changes in Edge Traffics and Total Traffic When File is Moved from One Node to an Adjacent Node

after a discussion of file assignment on trees and channel capacity allocation.

For a fixed tree topology, efficient methods of assigning files to optimal nodes have been developed. A single procedure is valid for a wide variety of possible communication cost functions. The essential idea underlying this assignment procedure is illustrated in Figure 2.6. In that Figure are shown the individual edge traffics and total edge traffic for a file with requests equally likely to come from any terminal assigned to or located at three different nodes. As the file is moved away from the optimal (lowest total traffic) terminal node, the traffics in certain edges of the tree increase, but none may decrease. Hence any cost function which is an increasing function of each of the edge traffics must increase too. In this manner, assigning files to minimize total traffics also minimizes any increasing function of the edge traffics. Many possible cost functions satisfy this condition, including some which accurately incorporate the stochastic nature of message flow.

In the design of communication systems which must satisfy performance constraints on the delay of individual messages, such as the requirement that the average message delay be three seconds or less, it is impossible to ignore the randomness of message arrivals and lengths. Rather it is necessary to allocate additional communication channel capacity to reduce the expected additional communication channel capacity to reduce the expected delays, and to provide storage for messages when several arrive almost simultaneously. A graph of the reduction in message delay with an

increase in channel capacity is shown in Figure 2. 7. As the capacity is increased, the delay decreases, but a point of diminishing returns is reached.

A number of other methods for reducing message delay besides additional capacity are also explored. Here we are concerned with the design of the message switching computers which would be placed at the nodes of a network of store-and-forward communication links. Borrowing heavily from the literature of queueing theory developed for scheduling jobs in computer central processors, a number of interesting message delay reducing schemes are evaluated. In general these message switching computers result in more efficient utilization of communication lines than simple fixed sub-channel multiplexers.

Another method used to reduce total capacity allocation in communication channels is possible when the measure of network performance is an average of performance measures on channels of the network. Such averaged measures are often appropriate for evaluating the performance of CMPC inter-terminal networks. Averaging channel responses allows the delay in one channel to be large while the delay in another is small, while keeping the average at an acceptable level. This is useful since the effectiveness of additional capacity and the cost of additional capacity vary from channel to channel. This is illustrated in Figure 2. 8 for a two channel case. In that figure is shown a curve of pairs of possible channel capacities which yield the same performance for the system. Since the cost of capacity is

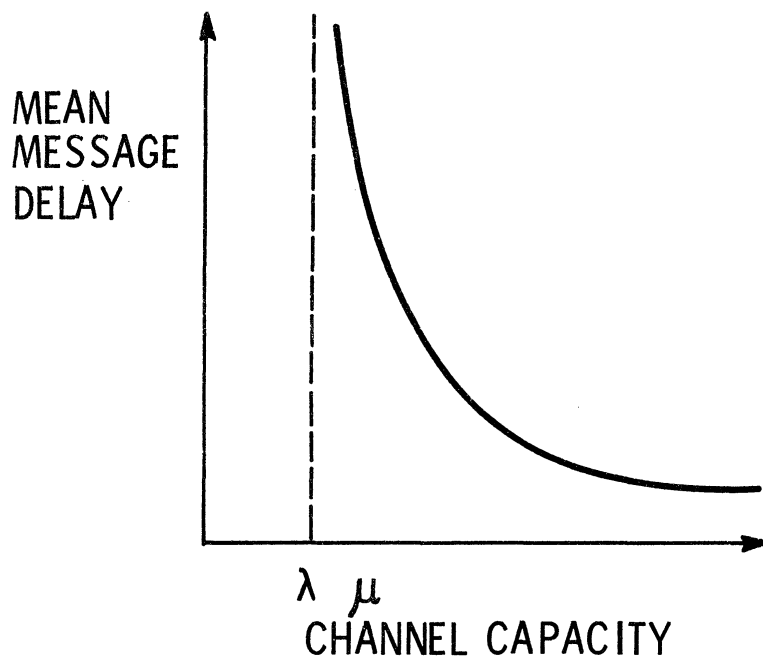


Figure 2.7 Average Message Delay in a Communication Channel

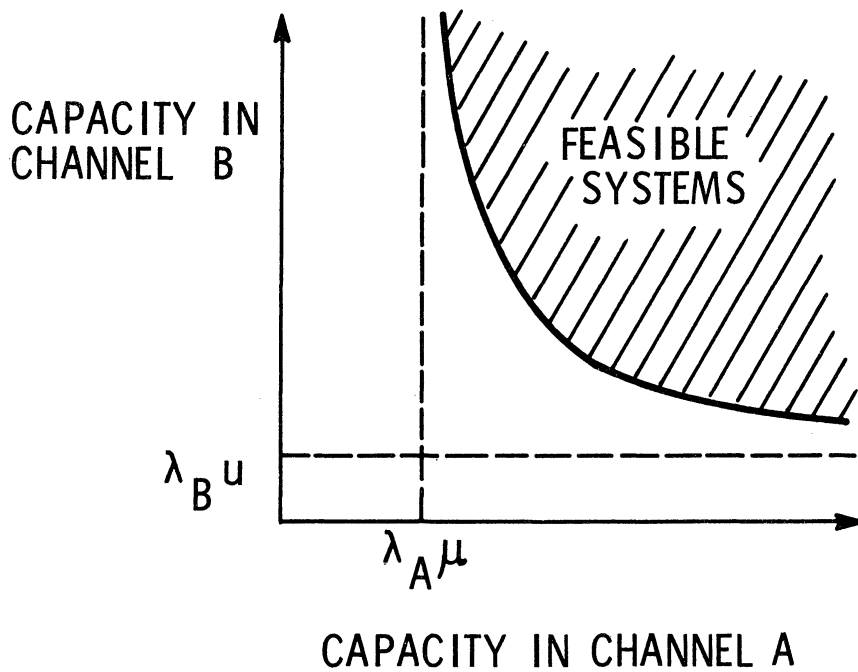


Figure 2.8 Feasible Values of Channel Capacity in Two Channel Communication System with Performance Constraint

likely to be different in the two channels, it is necessary to find the least expensive point on the curve of constraint performance. In a simple two channel case, the selection of the most economical system satisfying the performance constraint is easy. For systems with many channels the problem is much more difficult. Several of these optimal channel capacity allocation problems have been solved,

2.3 A CMPC System Synthesis Procedure

All of the design problems discussed so far are parts of a general design procedure for CMPC systems. This system design procedure separates a computationally infeasible problem into sub-problems amenable to optimal or efficient sub-optimal solutions and combines the solution of those sub-problems into a solution to the original problem. The design procedure follows the methodology of Figure 2.9. An initial configuration consisting of a star tree of leased lines, files assigned optimally, and channel capacity chosen optimally is selected. Then other configurations based on topologies which can be obtained from the base topology by the replacement of a single edge are constructed and the optimal system cost for each of these modified systems is calculated. A typical tree topology and several one edge modifications are illustrated in Figure 2.10. That modified topology which results in greatest reduction in system cost is selected as the new base topology. This iterative process of topology improvement continues until no further feasible cost reductions are possible.

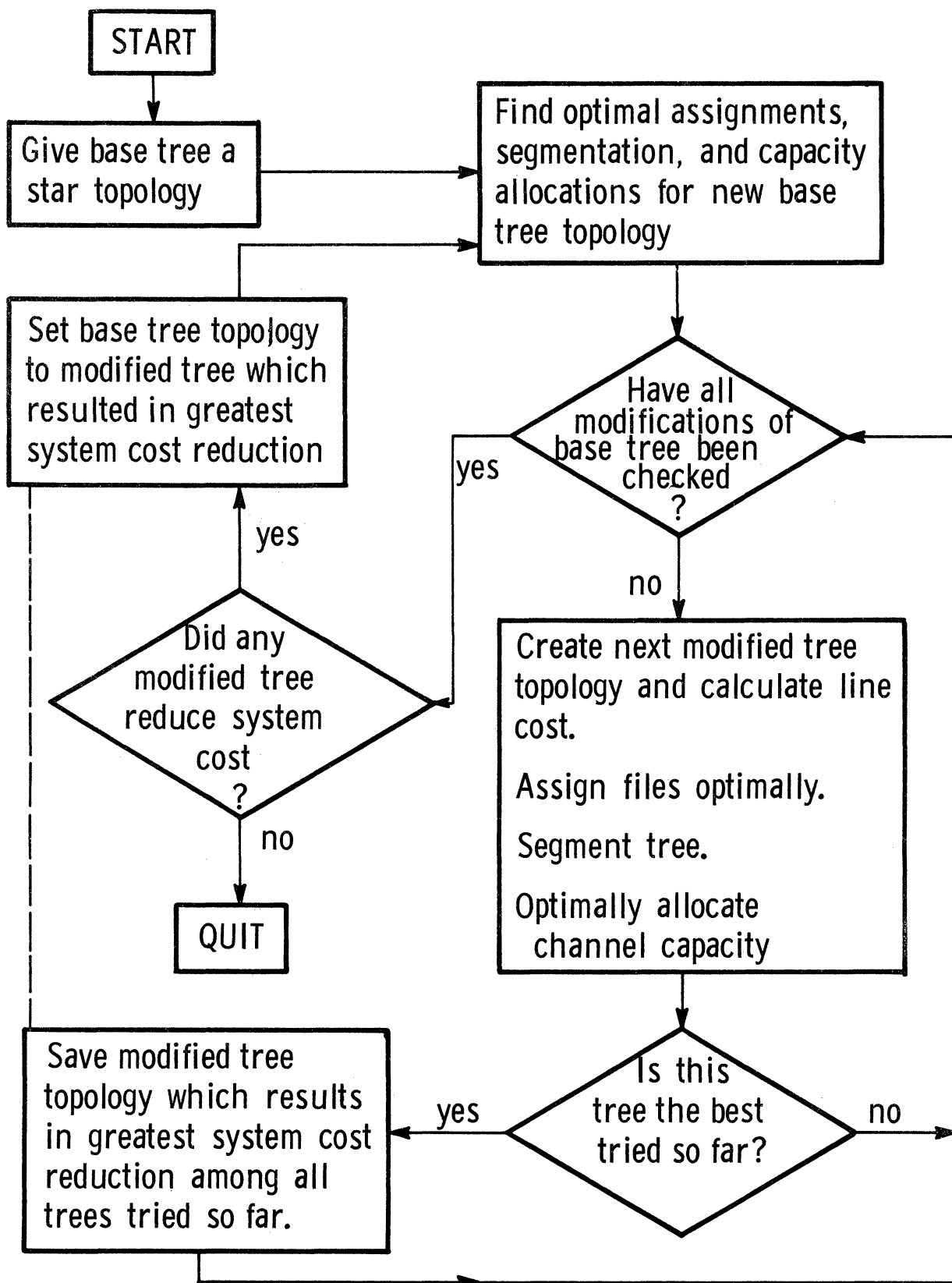


Figure 2.9 A Methodology for Computer Message Processing and Communication Design System

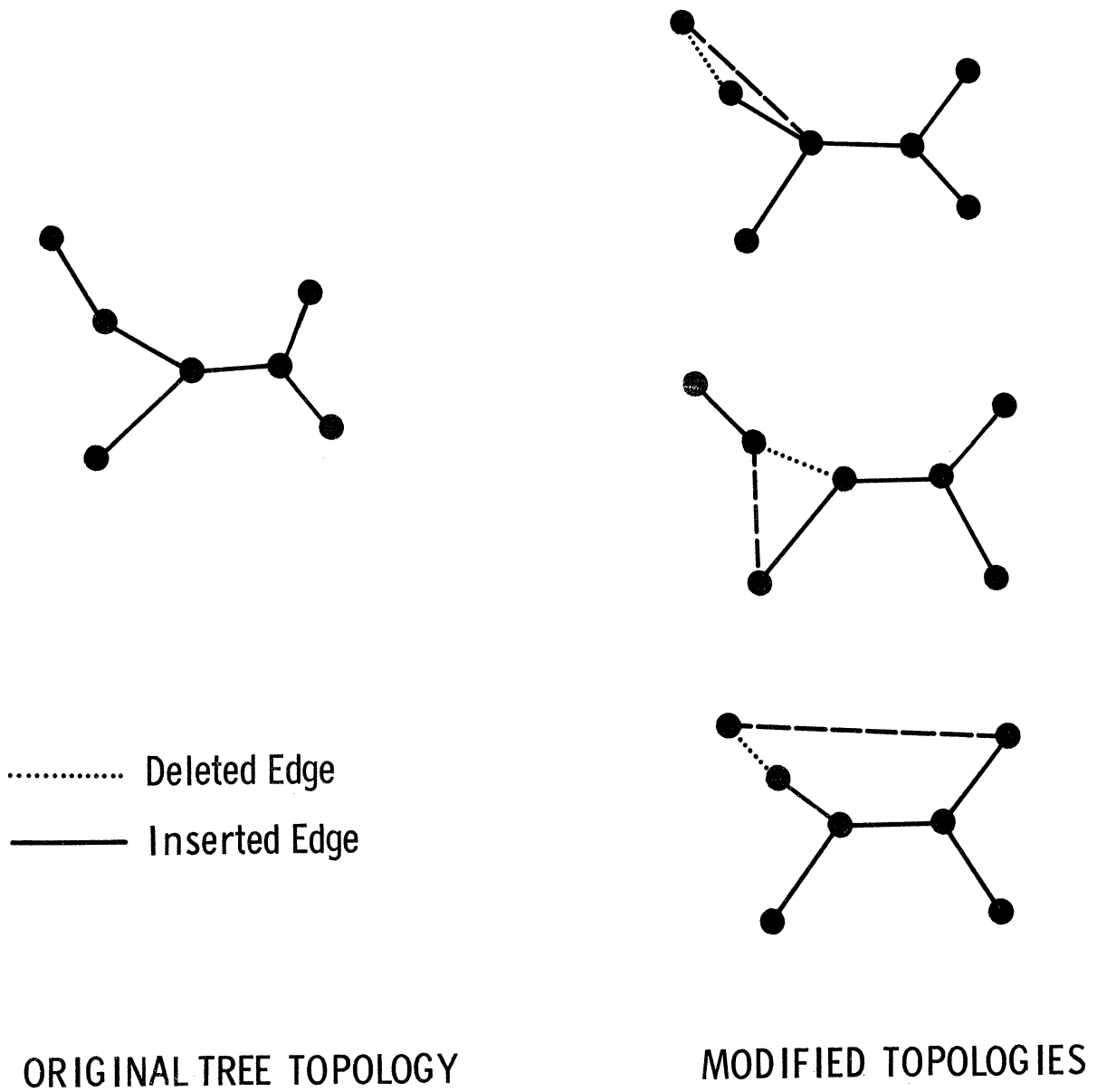


Figure 2.10 Modified Topologies Resulting from Tree Topology by Deletion of One Edge and Insertion of Another

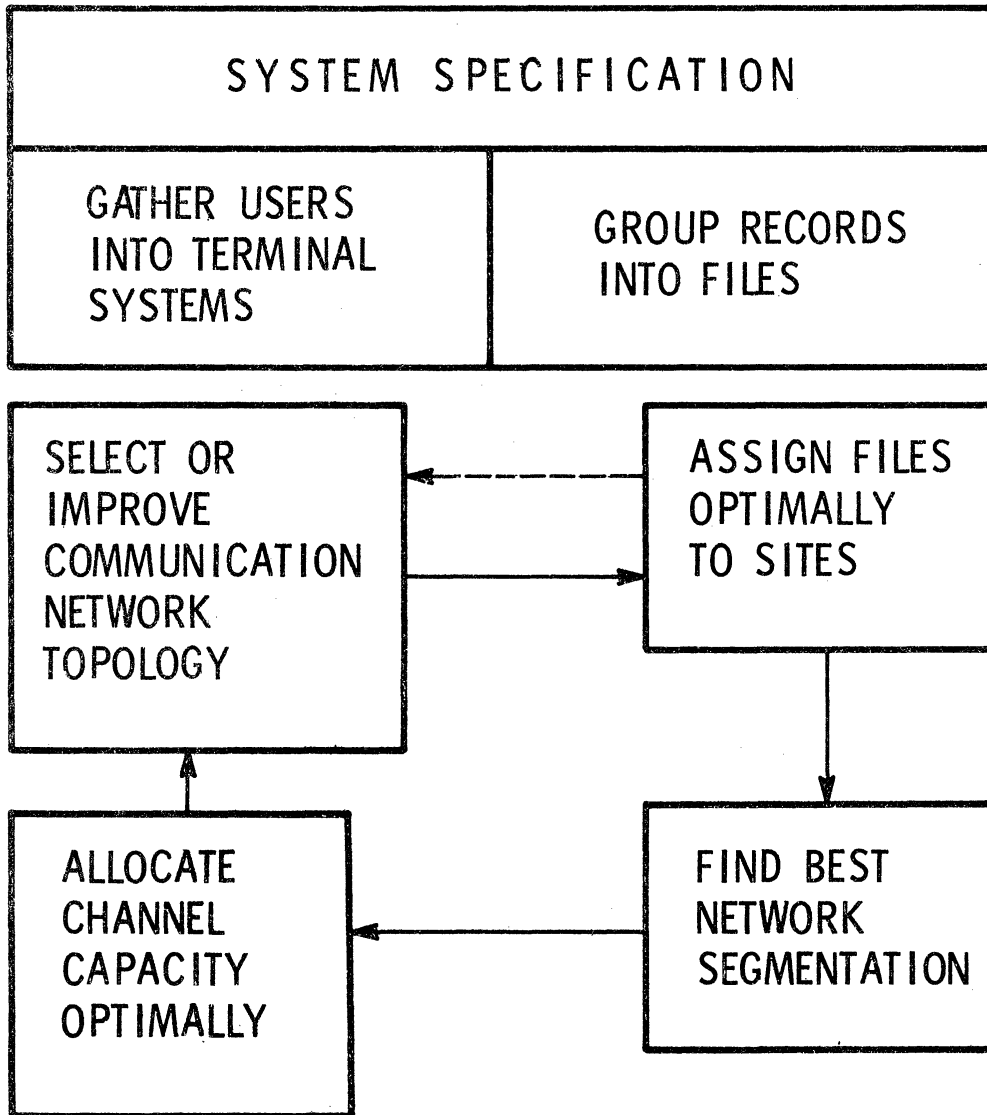


Figure 2.11 CMPC System Design Methodology

The heuristic nature of the solution procedure and the lack of comparable procedures make the evaluation of this total system design procedure difficult. Several points of non-optimality are evident, but the degree of non-optimality introduced is difficult to estimate or bound. Since the procedure follows steps similar to those of a human system designer, and performs some of these steps optimally, it can be expected to produce better solutions than seat-of-the-pants design procedures. In general, as CMPC systems become larger and more complex, the ability of the human designer to cope with the design problems will decrease. The availability of comprehensive procedures such as the one developed here, which handle large systems as well as small systems can contribute to the development and feasibility analysis of such large systems.

A sequence of steps used by a CMPC system designer is shown in Figure 2. 11. The system design may be conveniently separated into two basic activities, the system specification and the system design itself.

The activities that comprise system specification usually involve non-technical inputs or constraints. For our purposes, according to the problem types studied, these basic inputs and constraints include the terminal systems, the file system, and the performance constraint. Other real inputs are the cost functions for the various communication and processing system structures allowable in the system design.

The problem of gathering the users into terminal systems and locating the terminals for each group of users is a difficult and interesting one,

which has not been addressed in this work. When complete freedom is given to the designer, this problem may be formalized and an optimal solution sought. More often, however, the user groups and terminal locations are specified by non-technical considerations, and therefore are not appropriate topics for research of academic interest.

Another important phase of system specification is the construction and organization of the file system whose records are to be made accessible by the CMPC system to the users. Here too, decisions are often made administratively and not subject to engineering design considerations. It should be noted, however, that were better design tools more widely available, there would be more dependence on the system analysts in these decisions.

Once the terminal systems and file systems have been fixed, the design of the inter-terminal CMPC system may begin. At this point the design procedure specified earlier begins. This system design procedure attempts to find the jointly optimal choice of file site assignment, communication network topology (restricted to trees), tree network segmentation, and segment channel capacity allocation while guaranteeing an acceptable measure of system performance. This problem is very difficult. Straight-forward design techniques are computationally infeasible for even the fastest computers.

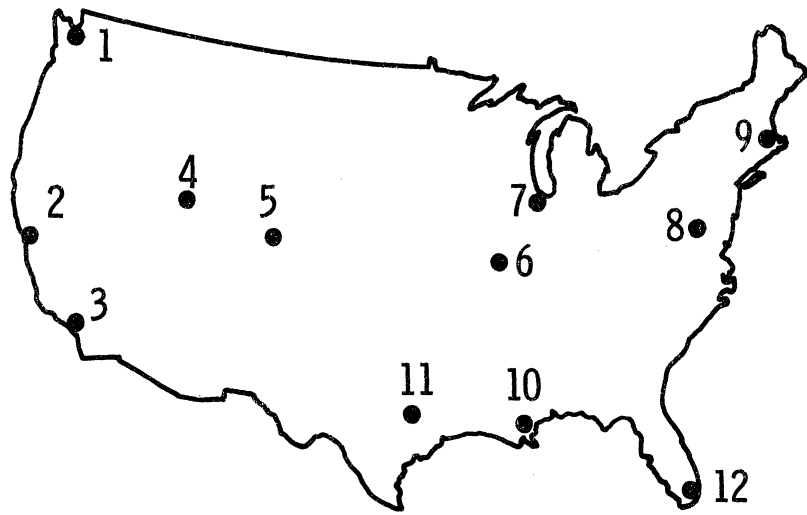
The validation of such a comprehensive and complex design procedures is a difficult problem, because of the lack of theoretical guidelines or solutions known to be optimal. This total design procedure serves both as a

useful design tool and as a frame work illustrating the relationship among the subproblems discussed earlier. Each step of the procedure is important in the overall design process. One evidence of this, explored in the example of the next section, is that more costly solutions are obtained when one or more of the optimization steps is deleted. These studies also relate the procedure to manual design techniques, since often system designers omit one or more of these steps attempting to achieve computational feasibility. In particular, the optimal file assignment and optimal segment channel capacity allocation are often entirely ignored. In other cases where these problems are not ignored entirely, only partially optimal or very crude design procedures may be used for their solution.

2.4 Example

In this section a CMPC system design for a specific problem is presented. Several other system designs are also presented to illustrate the effect of ignoring some of the problem variables. Although the entire design procedure is of interest because of the complex problems it solves, the basic emphasis of this section will be relating the separate steps of the general optimization procedure, and showing their effect on the total system design. With other cost functions than used in this example, the relative importance of the steps of the design procedure might be different. Where possible, however, general guidelines will be drawn.

The system to be designed in this section represents a small airline reservation system. The terminal locations and files are specified by the



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	.17	.31													
2	.42		.30	.21	.20										
3	.41		.41			.31									
4		.20	.09	.10	.30										
5		.22	.20	.19		.29	.40	.33							
6							.35		.29	.10					
7				.22	.21				.26		.35	.85	.55		
8								.44		.20	.65			.50	.41
9				.28						.20		.15		.29	.11
10						.21		.23		.30			.45	.21	
11		.27							.24	.11					
12					.29	.19	.25			.09					.38
R	1.0	.90	.60	1.8	.36	.50	.55	.64	.20	2.2	.80	.08	.58	.70	.60
E	10	7	5	11	3	5	5	7	2	11	10		7	10	6

Figure 2.12 Terminal Locations and File Parameters for Design Example

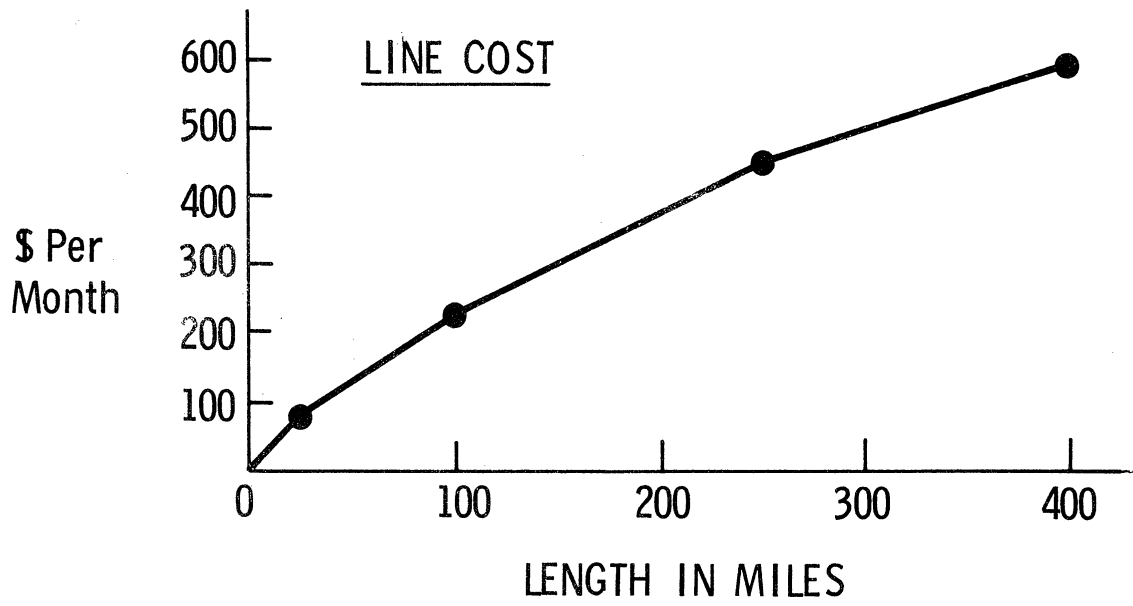
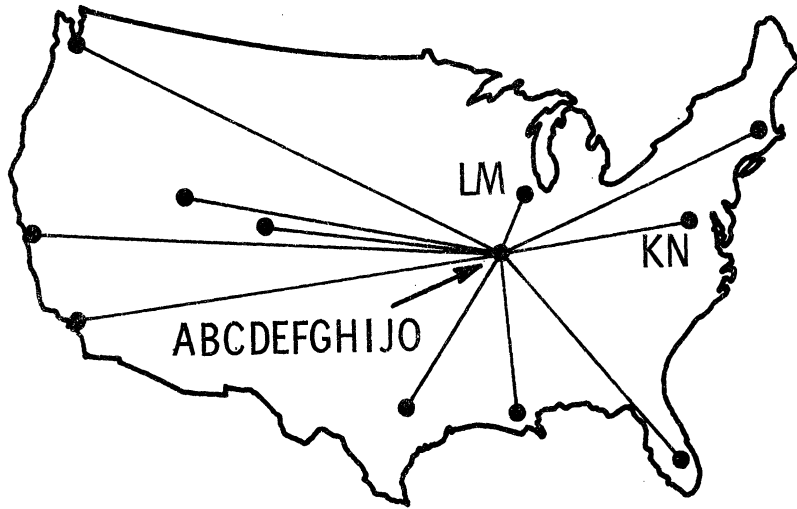


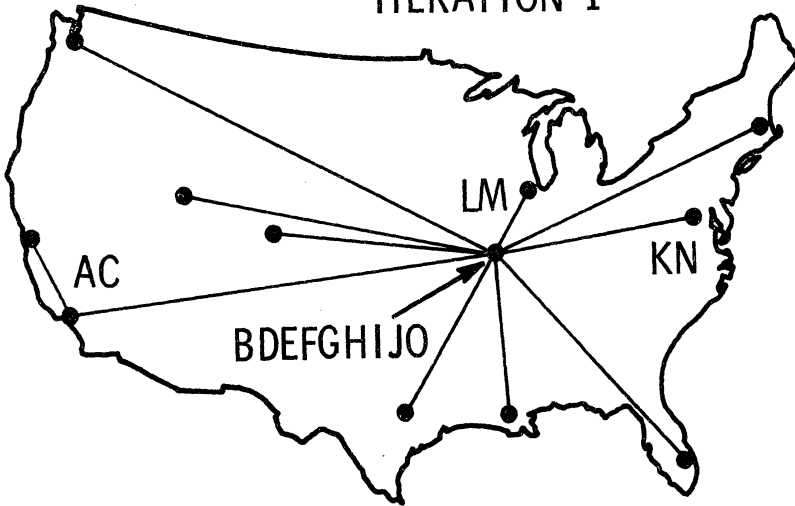
Figure 2.13 Line Cost Function

INITIAL CONFIGURATION



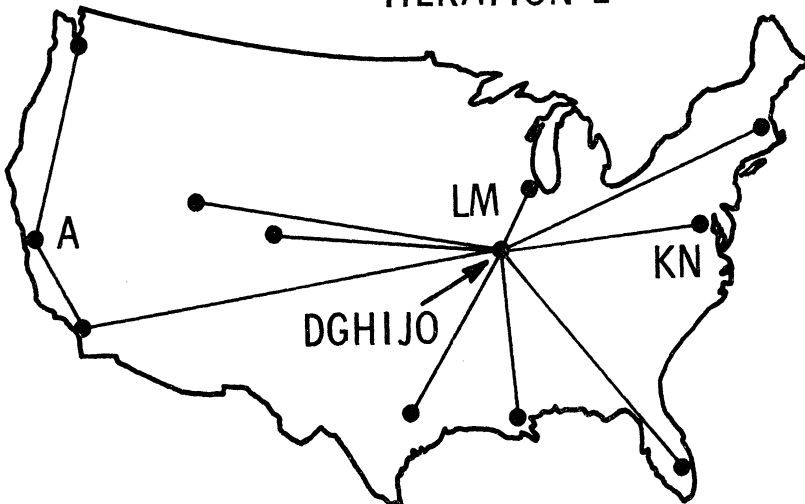
LINES	13501
MODEMS	1800
SEGMENTS	<u>400</u>
COST	15701

ITERATION 1



LINES	12331
MODEMS	2100
SEGMENTS	<u>400</u>
COST	14831

ITERATION 2



LINES	11372
MODEMS	2100
SEGMENTS	<u>400</u>
COST	13872

Figure 2.14 (Continued on the following page)

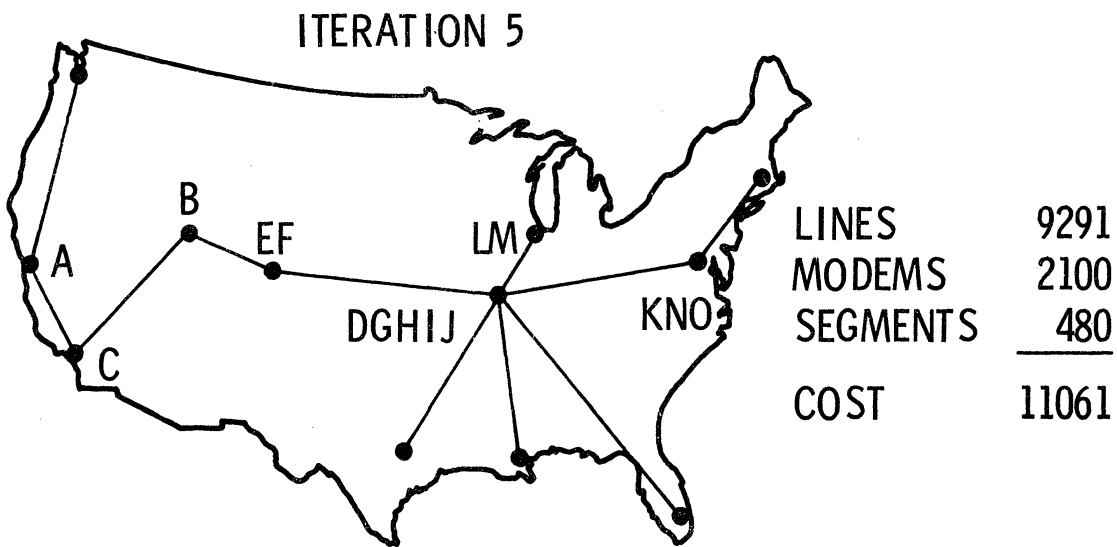
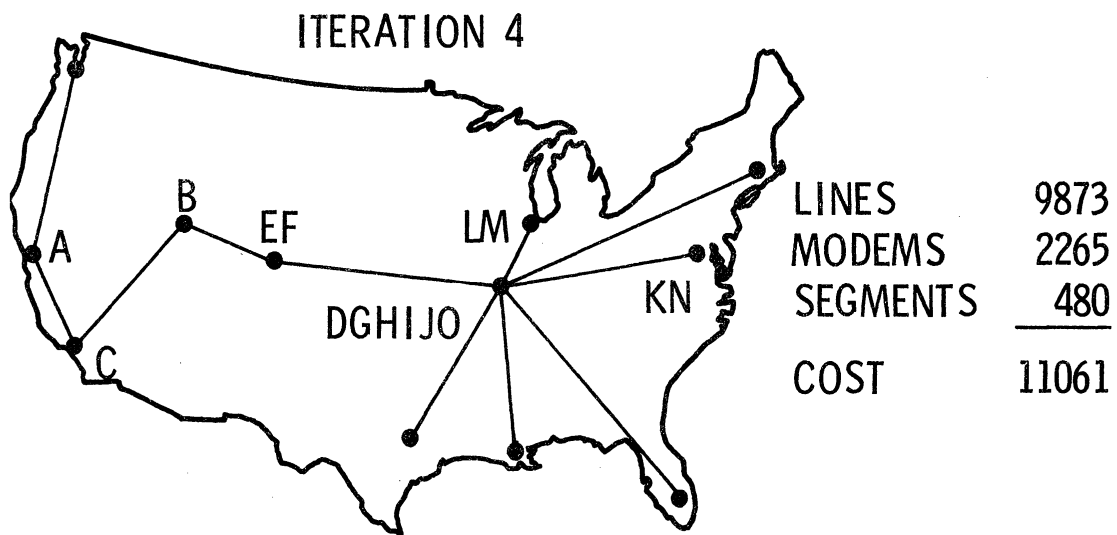
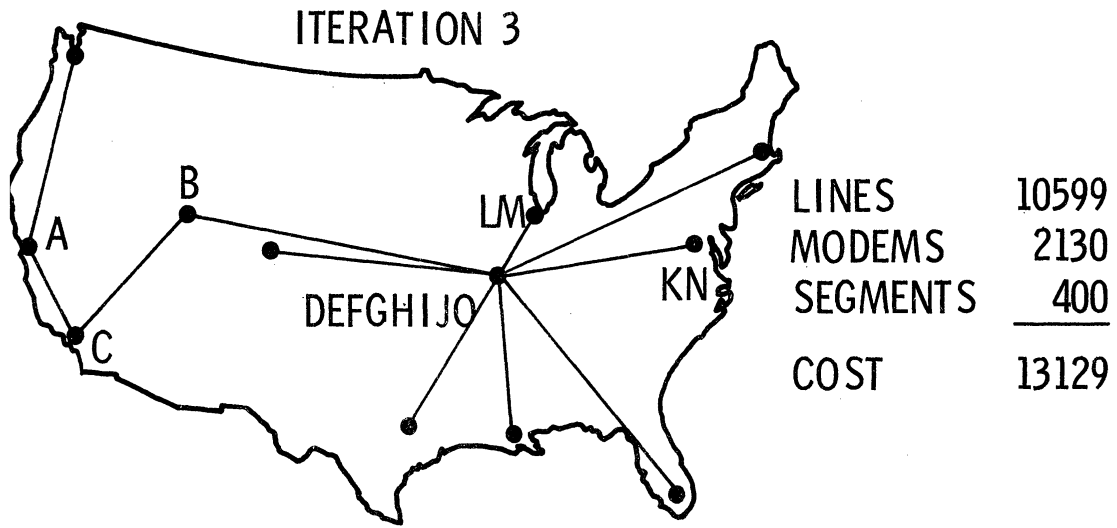


Figure 2.14 (Continued on following page)

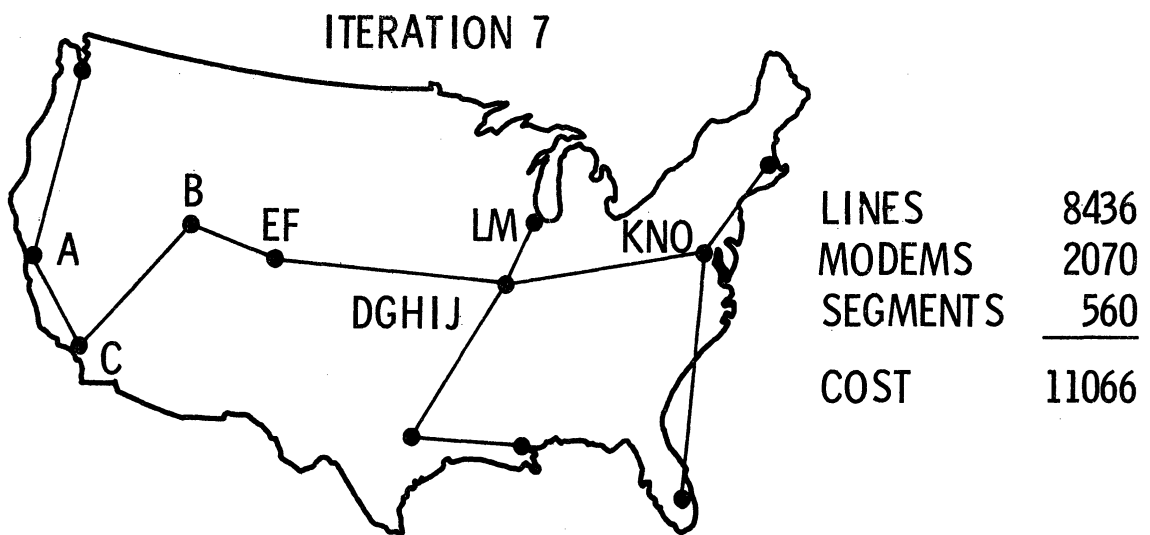
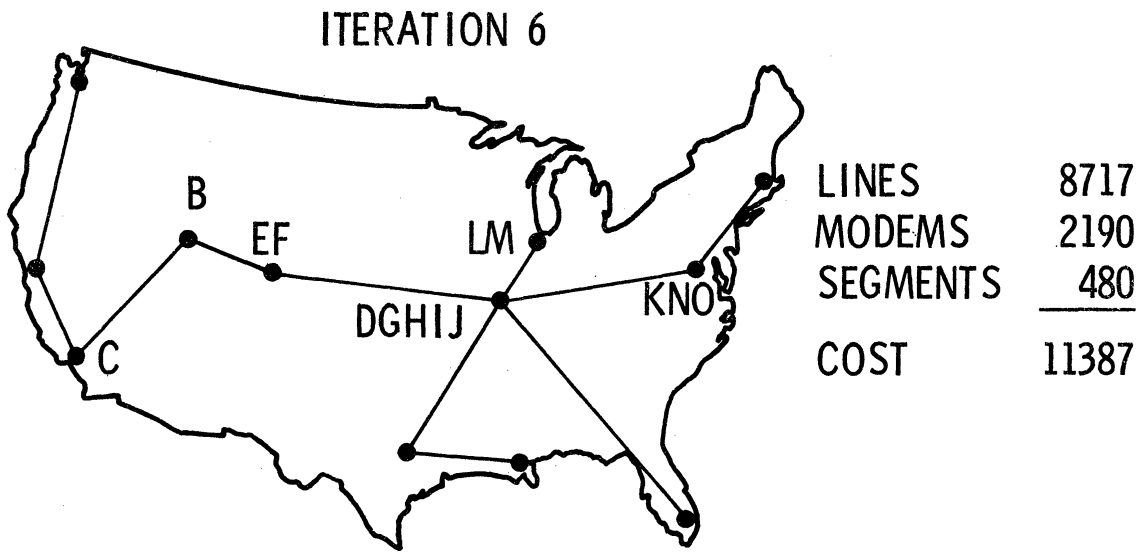


Figure 2.14 CMPC System Designs Selected by Each Successive Improvement Step

parameters of Figure 2. 12. The inter-terminal communication system will consist of store-and-forward concentrators joined by multipoint segments. Line interface speeds are available only in the values 500, 1000, 1500, 3000, 3500, 4000 and 5000 bps. These values correspond to higher actual transmission rates with redundancy for error correction and line control taken into account in the lower TRIB values given. Modems for these lines cost respectively 15, 30, 45, 90, 105, 120, 150 dollars/month rent. In addition, each segment requires a single line control unit leasing for \$80/month. Line costs are given by the graph of Figure 2. 13. No consideration of processing cost is given in this example, because maintaining message switching computers at each terminal accounts for the fixed cost, and the processing and file costs are assumed to be independent of file locations. The three variable costs of the problem are the line cost, the modem cost and the segment control cost.

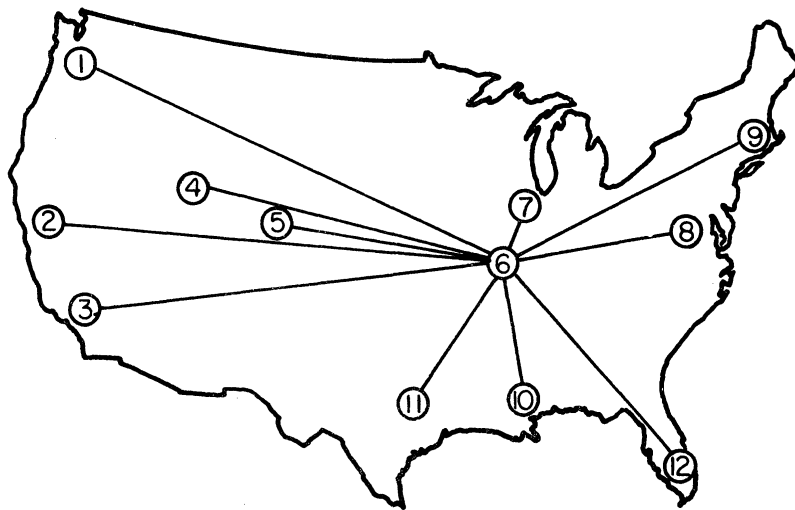
The initial configuration and the system design after each iterative improvement step with these costs constraints are shown in Figure 2. 14. At each step of the procedure all possible single edge replacements are examined with optimal file assignments on the new tree, segmentation to reduce network traffic, and optimal segment capacity allocation. The tree yielding maximal cost reduction is chosen and the next iteration begun. The final system design is illustrated in Figure 2. 20 with all segment traffics and capacities indicated. The basic pattern of the iterative improvement is to reduce the line cost as much as possible; as the line cost decreases,

however, network traffics increase, and modem costs increase. In the example, modem costs and segment costs both increase with the reduction in line costs. The procedure terminates when the next line cost reduction exceeds the increase in modem cost, or when no additional line cost reductions can be made.

This solution will be compared to five alternatives, exhibited in Figures 2. 15 through 2. 20. In each of these five designs one or more steps of the procedure has been omitted or replaced by a non-optimal design technique.

Figure 2. 15 exhibits the least sophisticated design for the MPC system example. In this system, all files are located at a single terminal, modems are chosen to have the same transmission rate (the lowest possible), each link is a separate segment, and no attempt to minimize line cost other than the selection of the central node is made. The solution cost is high, but the design is simple enough to be performed manually without the aid of computers.

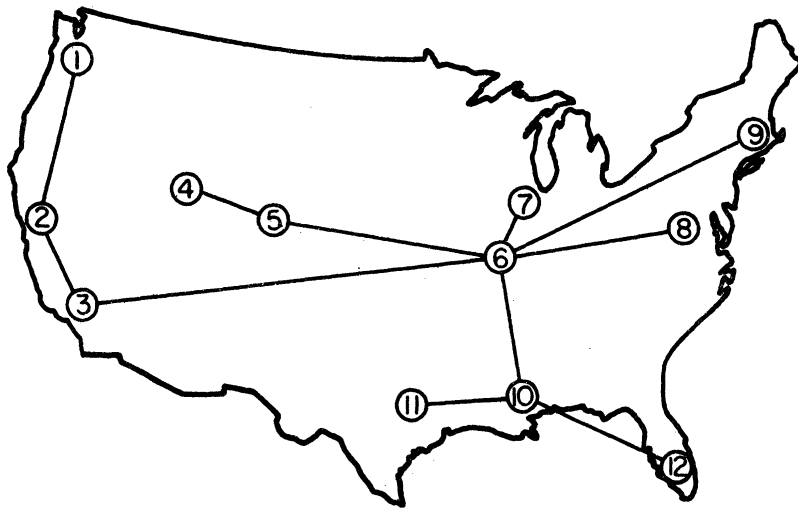
Figure 2. 16 exhibits the system design when there is an iterative tree improvement selection, but no additional attempts at optimization are made. All files are located at the single best node, and each link is an individual segment. The tree improvement continues to reduce total link cost until a constraint on the maximal link traffic (necessary because modem capacity is limited) is broken.



SEGMENT NUMBER	NODE SET	TRAFFIC	CAPACITY	MODEM COST
1	1, 6	673	2000	120
2	2, 6	1575	3000	180
3	3, 6	1216	2500	150
4	4, 6	783	2000	120
5	5, 6	1849	3000	180
6	7, 6	1833	3000	180
7	8, 6	2837	4000	240
8	9, 6	1900	3000	180
9	10, 6	1976	3000	180
10	11, 6	799	2000	120
11	12, 6	1144	2500	150

MODEM COST 1800
 SEGMENT COST 880
 LINE COST 13501
 TOTAL COST 16181

Figure 2.15 Non-Optimal System Design Example

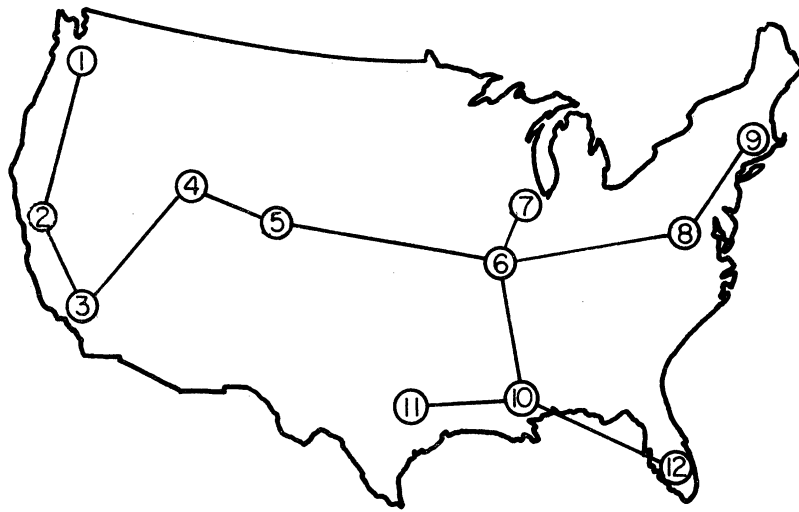


SEGMENT NUMBER	NODE SET	TRAFFIC	CAPACITY	COST
1	1, 2	673	4000	240
2	2, 3	2248	4000	240
3	3, 6	3465	4000	240
4	4, 5	783	4000	240
5	5, 6	2623	4000	240
6	6, 7	1833	4000	240
7	8, 6	2837	4000	240
8	9, 6	1900	4000	240
9	10, 6	3920	4000	240
10	10, 11	799	4000	240
11	10, 12	1144	4000	240

MODEM COST 2640
 SEGMENT COST 880
 LINE COST 9661

 TOTAL COST 13181

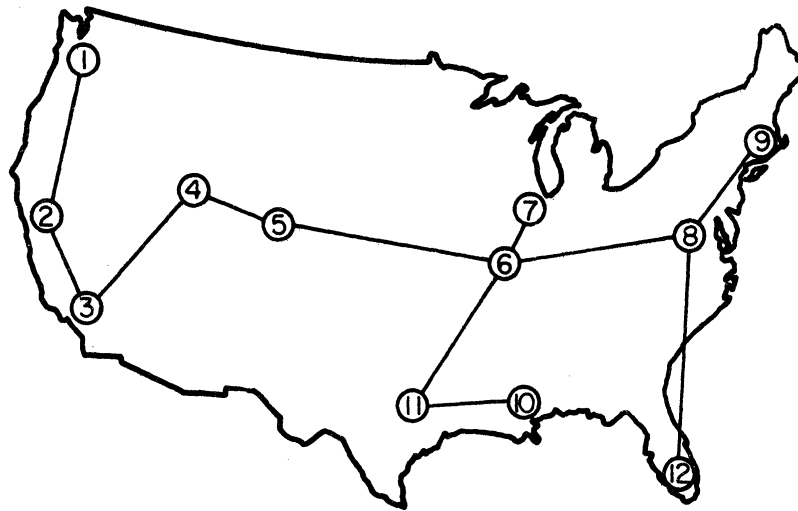
Figure 2.16 System Design Example with Network Topology Selection



SEGMENT NUMBER	NODE SET	TRAFFIC	CAPACITY	MODEM COST
1	1, 2	673	4000	240
2	2, 3	1978	4000	240
3	3, 4	1587	4000	240
4	4, 5	2181	4000	240
5	5, 6	2926	4000	240
6	6, 7	1662	4000	240
7	6, 8	3602	4000	240
8	8, 9	1900	4000	240
9	10, 6	3920	4000	240
10	10, 12	1144	4000	240
11	10, 11	799	4000	240

MODEM	COST	2640
SEGMENT	COST	880
LINE	COST	<u>8306</u>
TOTAL	COST	11826

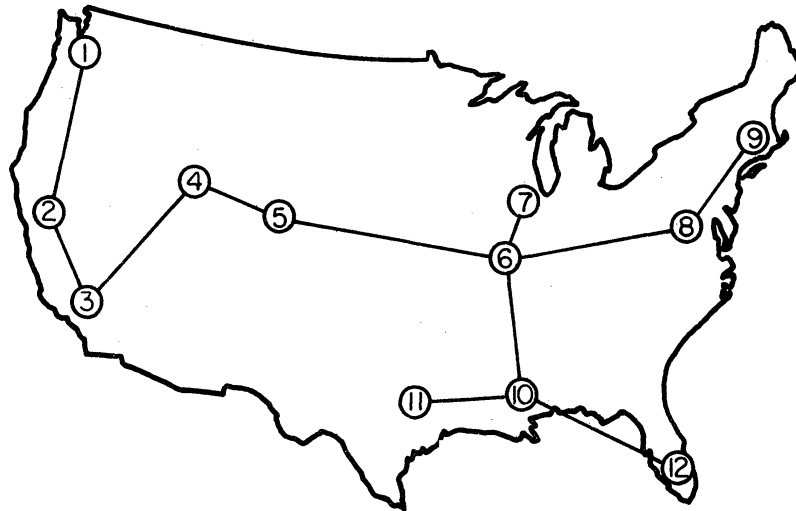
Figure 2.17 System Design Example with Optimal File Assignments and Network Topology Selection



SEGMENT NUMBER	NODE SET	TRAFFIC	CAPACITY	MODEM COST
1	1, 2, 3, 4	2727	5000	600
2	4, 5, 6	3905	5000	450
3	6, 7	1622	5000	300
4	6, 8	4062	5000	300
5	8, 9	1900	5000	300
6	12, 8	1144	5000	300
7	10, 11, 6	2776	5000	450

MODEM COST	2700
SEGMENT COST	560
LINE COST	<u>8436</u>
TOTAL COST	11696

Figure 2.18 System Design Example with Optimal File Assignment, Tree Segmentation, and Network Topology Selection

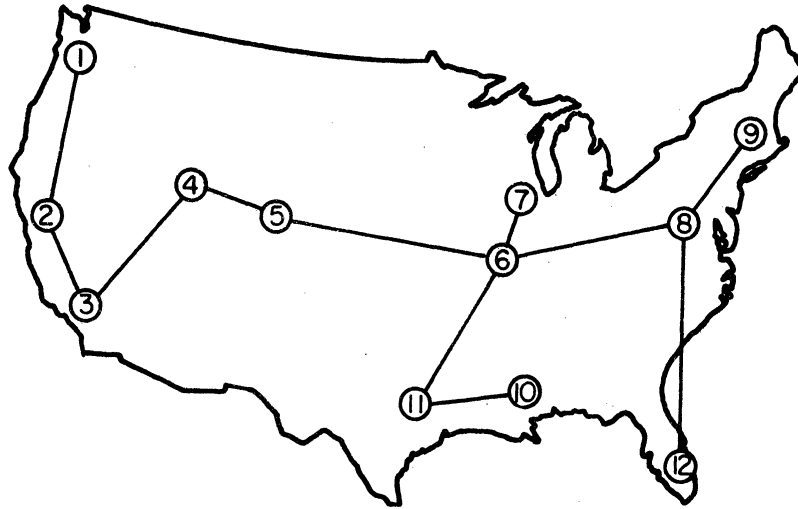


SEGMENT NUMBER	NODE SET	TRAFFIC	CAPACITY	MODEM
1	1,2	673	1000	60
2	2,3	1978	3000	180
3	3,4	1587	3000	180
4	4,5	2181	3000	180
5	5,6	2926	3500	210
6	7,6	1662	3000	180
7	8,6	3602	5000	300
8	9,8	1900	3000	180
9	10,5	3920	5000	300
10	11,10	759	1500	90
11	12,10	1144	1500	90

MODEM COST 1950
 SEGMENT COST 880
 LINE COST 8306

 TOTAL COST 11136

Figure 2.19 System Design with Optimal File Assignment
 Optimal Edge Capacity Allocation and Network
 Topology Selection



SEGMENT NUMBER	NODE SET	TRAFFIC	CAPACITY	MODEM COST
1	1, 2, 3, 4	2727	3500	420
2	4, 5, 6	3905	5000	450
3	7, 6	1622	3000	180
4	11, 10, 6	2776	4000	360
5	8, 6	4062	5000	300
6	9, 8	1900	3000	180
7	12, 8	1144	3000	180

MODEM COST	2070
SEGMENT COST	560
LINE COST	<u>8436</u>
TOTAL COST	11066

Figure 2.20 System Design Example with Optimal File Assignment, Tree Segmentation, Optimal Segment Capacity Allocation, and Network Topology Selection

The system design of Figure 2. 17 improves the solution of Figure 2. 16 by assigning each file to its optimal site. This reduces traffic in the links so that additional iterative steps reduce total line cost as well. No attempts at segmentation or optimal channel capacity allocation are made.

The system design of Figure 2. 18 considers tree segmentation to reduce system traffic, resulting in a further reduction in total system cost. It can be seen, by comparing Figure 2. 18 with 2. 17 that considerations of tree segmentation may alter the final system topology. Hence it is important to consider the segmentation of each possible tree improvement, rather than simply adding the segmentation procedure to the system design obtained by selecting a topology on the basis of line lengths alone

The system design procedure whose final design is exhibited in Figure 2. 18 adds optimal segment capacity allocation to the procedure resulting in Figure 2. 19, but does not perform a segmentation on each tree alteration in the iterative improvement step. In this example, adding this optimal capacity allocation does not change the final system topology, because the total cost of the modems is considerably less than the cost of the topology alterations. For systems with higher traffics, capacity allocation might also affect the system topology.

Figure 2. 20 is the result of the most comprehensive design procedure of this section. Each new tree in the iterative improvement steps is given optimal file assignments, a segmentation to reduce system traffics, and optimal segment capacity allocation.

These six design procedures are compared for the airline example in Figure 2. 21, showing the cost savings and percentages resulting from the consideration of more design variables in the system selection procedures. Although these exact reductions cannot be expected for another system, there are some general conclusions to be drawn for systems using the same cost functions, but possibly different file and terminal systems. It is clear that segmentation does not result in such a large reduction as does optimal channel capacity allocation. This would be reversed if the fixed cost associated with individual system segments were higher. Optimal assignment also resulted in an improvement in the system cost, both by its effect on feasible topologies and for the actual reduction in link traffics (and total modem capacity).

It would be unfair to claim now that the comprehensive procedure has been shown universally valid, or even useful. Rather this example is intended to demonstrate that many different system designs can be evaluate quickly and the least costly chosen, that the procedure of Figure 2. 9 provides a convenient framework for evaluating various types of systems, and that for this example the most general design was significantly less costly than the most restricted design. Note that system design tools such as this procedure are most useful for the very large systems which are the most difficult to analyze by hand, and that a small percentage improvement may represent a very considerable cost savings in a large and expensive system.

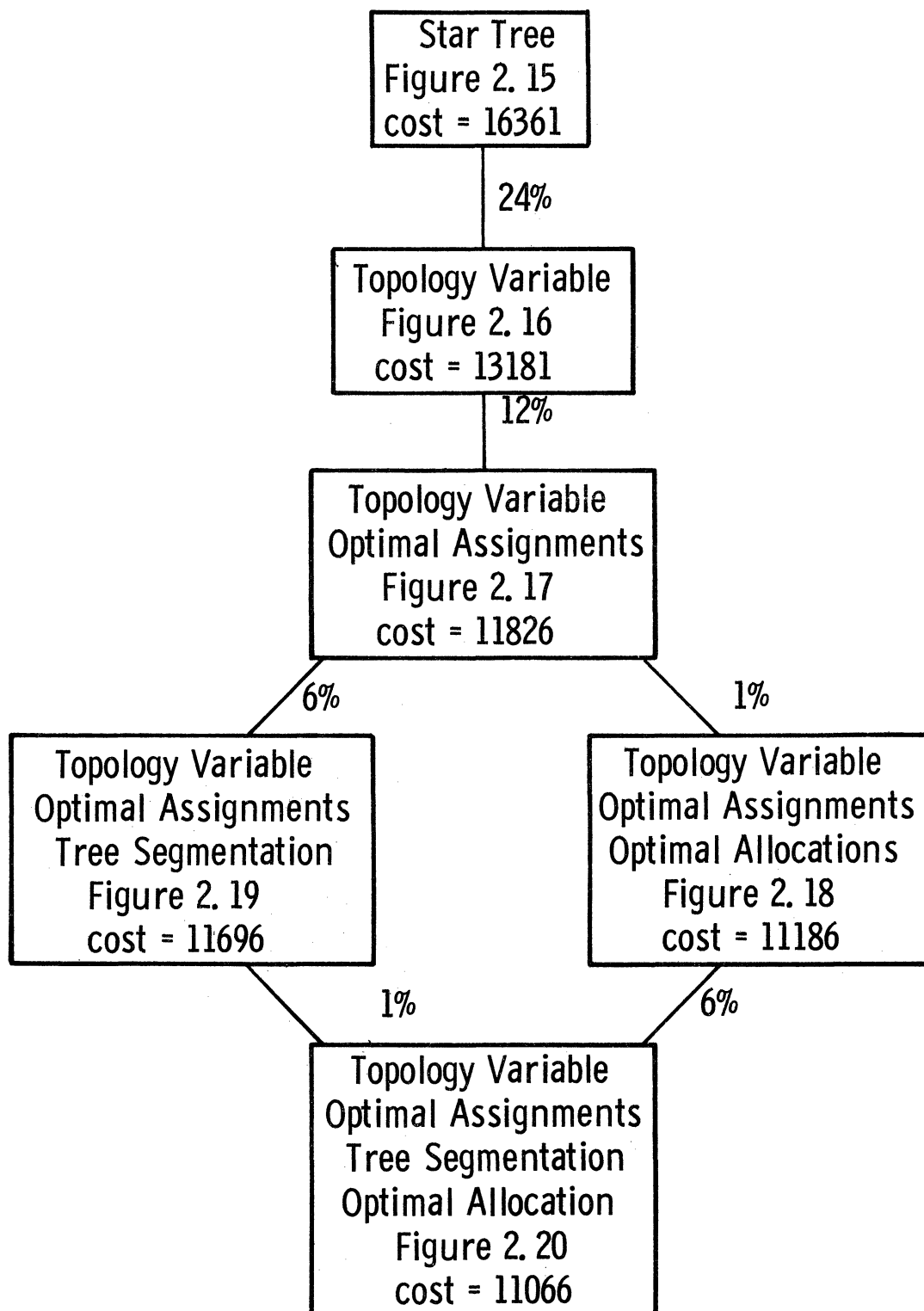


Figure 2.21 Comparison of Several CMPC System Design Procedures

2.5 Conclusion

A precise mathematical model of the systems being studied is formulated to draw attention to the fundamental subsystems and to clarify the basic independent variables of the design processes. Systems considered must have a large data base organized into record files, widely distributed users, an on-line communication network joining users to data base sites, a quantitative performance measure and specified performance constraints. The most general system model studied uses a queuing model for individual communication channels, with the channel interconnections specified by a weighted linear graph. The data base is assumed to be organized into files of homogeneous records, and the processing costs are specified by a single function of the number of messages processed.

One major problem area studied is the determination of the optimal number and locations of sites for the system files. Several basic properties of file site assignment on linear graphs are demonstrated and used in efficient procedures for file assignment.

When the file locations are specified, the communication requirements of the system are known. A second major problem area considered is the optimal design of communication channels and networks of these channels. A thorough study of stochastic message transmission channels has been undertaken. The effects of channel capacity with one or more channels, message length distributions, message retransmission order, and departures from the model assumptions on message delay are studied, both analytically and

by simulation. Several channel and network performance measures are defined and compared. The problem of the optimal allocation of channel capacity among the channels of a network is solved for several important system models. A complete set of guidelines is given to aid the designer of communication systems and channels.

The third major problem area studied is the optimal design of communication network topologies. Several previously available techniques for the design of centralized networks are critically evaluated. New techniques for the design of centralized and non-centralized networks are presented. These new procedures more faithfully model communication networks than the other procedures by including more realistic cost considerations, and yield more general solutions.

Finally the solution procedures of these problems are integrated into a systematic design procedure for a general class of computer message processing and communication systems. An example of a complete system design is given.

Some specific contributions of this research, arranged in no particular order, are listed below.

1. A comprehensive model of on-line Message Processing and Communication Systems has been developed which explicitly considers each of the following
 - a) Communication network line topology
 - b) Communication network tree segmentation into multi-point

segments

- c) Optimal allocation of communication channel capacity to minimize cost while maintaining acceptable system performance
- d) Variation in inquiry origins for separate files and its effects on optimal file assignment
- e) Processing cost for messages and storage cost for files
- f) The stochastic nature of message requests
- g) One or more copies of each file

The model allows the introduction of very general cost structures in each of the basic sub-systems.

2. A procedure for the design of Computer Message Processing and Communication systems has been devised which allows effective consideration of each of the design variables included in the CMPC model. While not necessarily yielding optimal system designs, this procedure yields designs of lower cost and greater generality than those generated by other currently available design procedures.
3. Hakimi's theorem [4] that optimal assignments of files lie at nodes of a linear graph has been extended with a simpler proof to a much more general class of communication cost functions.
4. Optimal assignments of files on a tree graph are shown to be independent of the edge traffics. Hence, optimal assignment to minimize induced traffic also minimizes a wide class of other

cost functions, namely those monotone non-decreasing in each edge traffic.

5. A careful derivation of the mean message delay in a network of Poisson queues has been given. This exhibits clearly the underlying assumptions necessary to the use of this model for communication networks.
6. Efficient procedures for allocating channel capacity in a Poisson message network have been developed and its validity proved carefully. These procedures use a variety of response functions, both continuous and discrete.
7. A careful study of store-and forward message transmission channels has been made. The effects of message length distribution, retransmission order, buffer size, and channel capacity allocation upon mean message delay have been studied by numeric, simulation, and analytic techniques. A comprehensive set of channel design guidelines is presented.
8. The concept of segmentation of a tree and its relationship to non-centralized network design has been formalized and studied. Efficient procedures for the selection of good segmentations of a fixed tree have been developed.
9. Procedures for the synthesis of centralized networks of multi-point and multiplexed segments have been developed which yield designs superior to any procedures currently available.

10. A method of calculating the mean waiting time of and mean number in system for Poisson S-server queues with non-homogeneous service rates has been devised. This method uses a combination of analytic and numeric techniques.

References For Section 2

- 1) "Look Ahead" Datamation, April 1968, p. 53.
- 2) James Martin, Design of Real-Time Computer Systems, Prentice-Hall, Inc. , Englewood Cliffs, N. J. , 1967, p. 629.
- 3) C. DeGabrielle, "Design Criteria for an On-Line Nationwide Order Processing System", pp. 71-75 in Disk File Applications, American Data Processing, Inc. , 1964, Detroit, Michigan.
- 4) S. L. Hakimi, "Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems", Operations Research vol. 13, no. 3, 1965.
- 5) V. K. M. Whitney, "A Study of Optimal File Assignment and Communication Network Configuration in Remote-Access Computer Message Processing and Communications Systems", SEL Technical Report no. 48, The University of Michigan, Ann Arbor, Sept. 1970.

3. MULTIPROGRAMMED AND MULTIPROCESSOR COMPUTER SYSTEMS

In this section we discuss research into problems related to the operational optimization of multiprogrammed and multiprocessor computer systems. We first present (3.1) a very comprehensive mathematical model of complex computing systems using storage hierarchies and multiprogramming order to predict and control the behavior of such systems. This model has been implemented in the form of a fast and highly interactive computer program which leads easily to an incremental optimization of the computer systems under study by providing the user with comprehensive performance measures as model parameters are varied. The research outlined in section 3.2 uses queuing theoretic approach to investigate optimal task scheduling rules in heavily loaded multiprogrammed computer systems. The effects of reentrant procedures and sharing of information in such systems is also investigated. Section 3.3 is concerned with the problem of breaking up programs into pages in such a way as to minimize the number of page faults. Using an algorithm developed for the solution of a particular quadratic programming problem efficient solution technique for the above partitioning problem is presented. Finally the scheduling of programs in a multiprocessor computer system is investigated in section 3.4.

3.1 Multiprogrammed Systems Using Storage Hierarchies

The design and application of large computing systems involves considerable risk. One can draw a parallel between those who have designed and applied today's computing systems and those who, in the early days of flight, strapped wings to their backs and jumped from the roofs of barns. Neither had adequate methods of predicting system performance. The work reported here attempts to contribute to computer system technology through the development of improved methods of predicting system performance.

Throughout the development of computing systems there has been an effort to make a given hardware technology perform better as a system. The primary emphasis in the early days of development was on hardware technology. However, as computing systems have progressed, there has been a continued and growing interest in the design of system which makes the best and most efficient use of a given hardware technology.

One important method of achieving efficient use of a given hardware technology has been the use of storage hierarchies. Storage hierarchies are not unique to computer systems. Your pocket, your desk or dresser drawer and your basement represent a storage hierarchy. The basic idea is simply to store those items used most frequently in as easily accessible location. Those items seldom used may be stored

in less accessible and correspondingly less costly locations.

The storage hierarchy was first implemented in a computing system developed at the University of Manchester in England in 1949. As computer systems have progressed, two things have happened.

First, the complexity of the hierarchy itself has increased. We can easily locate systems with 5, 6 or more different kinds of storage hardware capability. For instance, registers, cores, drums, disks, data cells, magnetic tapes, punched cards, and punched paper tape. A second and very important change has occurred in the management responsibility of the hierarchy.

The gains achieved by the storage hierarchy have imposed considerable burden upon the programmer. At the outset the programmer was responsible for deciding what information should be stored on what device at each instant of time. In addition the programmer was responsible for carrying out the necessary operations required to move the information about as required. Lastly and very important, the programmer was responsible for keeping track of how and where everything was stored as it was moved about in the system. As we have progressed and hierarchies have become more complex the operating systems and hardware have assumed these responsibilities in varying degrees. In some case all responsibilities for management of a hierarchy have been assumed by the operating system and/or hardware and the hierarchy is invisible to the programmer.

Another area of computer systems development has been in the use of multiprogramming. A computer is often limited by its slowest (or most overworked) component, often the I/O device. One of the methods used to alleviate this problem has been multiprogramming or the practice of working with more than one user program at a time.

One of the effects of multiprogramming is to provide the system with a more balanced workload. If a system processes one program at a time, it will find some programs using a great deal of CPU resources and leaving the I/O devices idle, while others leave the CPU idle and do voluminous I/O. On the other hand, if a system is processing 10 or 20 programs at once, it is unlikely that the system will see this group of programs exhibit the wide variations in resource demand exhibited by the individual programs in the group. In other words system loading is more consistent and predictable when the sample space grows larger.

The second effect of multiprogramming is to introduce parallel paths in the workload as seen by the computing system. If a computing system must follow a single thread of execution, a delay in any part of the system holds up the entire system. Multiprogramming is one way of providing the necessary parallel paths of execution, which if properly used, can increase resource utilization.

The difficulty with multiprogramming is complexity. The problems of resource scheduling are difficult. The problems of protecting one user from another, protecting the operating system itself, and charging for resource

use become most complex. However, the rewards for efficient resource utilization can be great. There has been considerable motivation for complexity in the effort to better utilize computing system resources. This has been added to by the response requirements of time-sharing and real-time systems.

This need for complexity has placed the modern computing system outside the capability of man's unaided intuition.

3.1.1 The Nature of the Problem

In very few words the problem is that of predicting and controlling the behavior of complex computing systems using storage hierarchies and multiprogramming.

A computing system using a storage hierarchy generally involves 2 or more different types of storage devices. Different device types have widely varying performance characteristics. Drums, cores and disks all behave differently under load. A system's performance may be determined by a complex balance of workload throughout the system of devices, or may be determined by the performance characteristics of a single overworked device.

User programs can be big or small. They can access data in a serial or random manner. They can do large quantities of I/O or almost none at all.

The data paths and routing of information influences the load seen by storage devices. The logical record sizes throughout the system also influence the load seen by storage devices and in turn their response times.

The number of programs running in a multiprogrammed system effects the storage allocations of user programs. This in turn influences the demands placed on the system by user programs.

All of these factors and many others combine to create an enormously complex and remarkably difficult analysis problem. It is simply difficult to determine what a given system will and will not do in a given circumstance. It is even more difficult to design such systems especially if some kind of optimal or near optimal design is required.

3.1.2. Objectives

The specific objective of this research is to develop and demonstrate a mathematical model of a computing system. The computing systems in question here fall into the class of those systems using storage hierarchies of 2, 3 or more levels and multiprogramming. The model developed exhibits the following characteristics:

1. The model includes the effects of user program behavior, operating system characteristics and hardware performance.
2. The model is versatile and easily applied to a wide range

of system configurations. The model is useful as a tool for the investigation of computing systems in general as well as applicable to the detailed investigation of a particular system.

3. The model is useful as a tool for both analysis and optimization.
4. The results obtained from the model approach the accuracy and realism of those obtained from simulation models.

3.1.3. The Model

At this point we will attempt a broad preview of the model which has been developed.

The overall model for computer system analysis is shown in Figure 3.1. On the left we see 3 categories of independent variables. First we have the user program description. This includes independent variables such as the size of the user programs and other characteristic of the user program behavior. Next there is storage device description. This includes such items as drum RPM, disk seek time and variables relevant to storage device performance. Last we have data traffic dependencies and architecture. This includes CPU performance characteristics, logical record size, data transfer timing dependencies and other global system characteristics. As an output for the model we show performance. In the narrow sense,

performance is defined as the mean rate at which the collection of user programs running on the system make reference or access to data and instructions. In the broad sense the performance also includes many details such as mean queue lengths at various devices in the storage system and CPU utilization.

Figure 3.1 shows the model as seen by the designer when being used for analysis. Figure 3.2 shows the model in slightly greater detail and from a different point of view. It is this form of the model that we will be most concerned with. We see that the independent variables on the left have not changed. However, the model shown here does not give us the performance (mean user program access rate) directly but rather tells us if an assumed performance is greater than or less than a given system's capability. For analysis we will carry out a simple search to find the performance of a given system. The primary advantage of this particular approach to the problem occurs in optimization where systems are compared in a search for a system configuration with the greatest performance or mean user program access rate.

Figure 3.3 and 3.4 show a detailed breakdown of the model of Figure 3.2. Beginning at the left of Figure 3.3 we see the User Program Model. The independent variables supplied to this model fall into two classes. The user program description consists of 5 independent

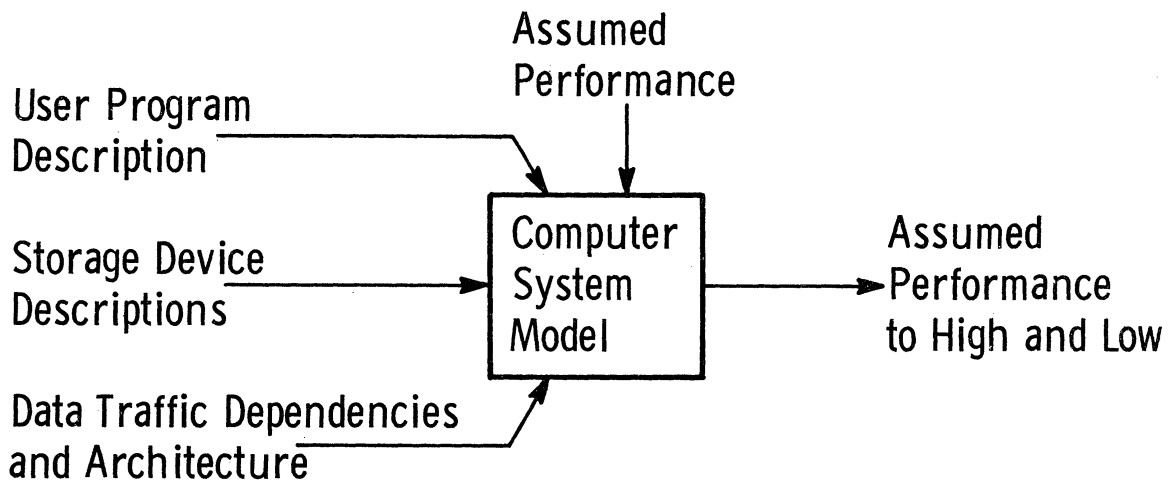
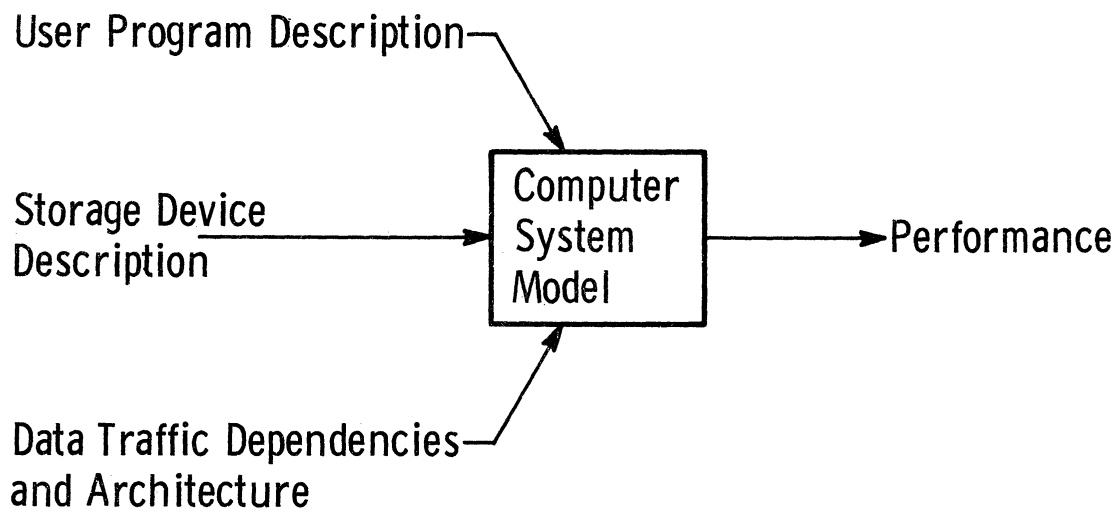


Figure 3.2 System Model

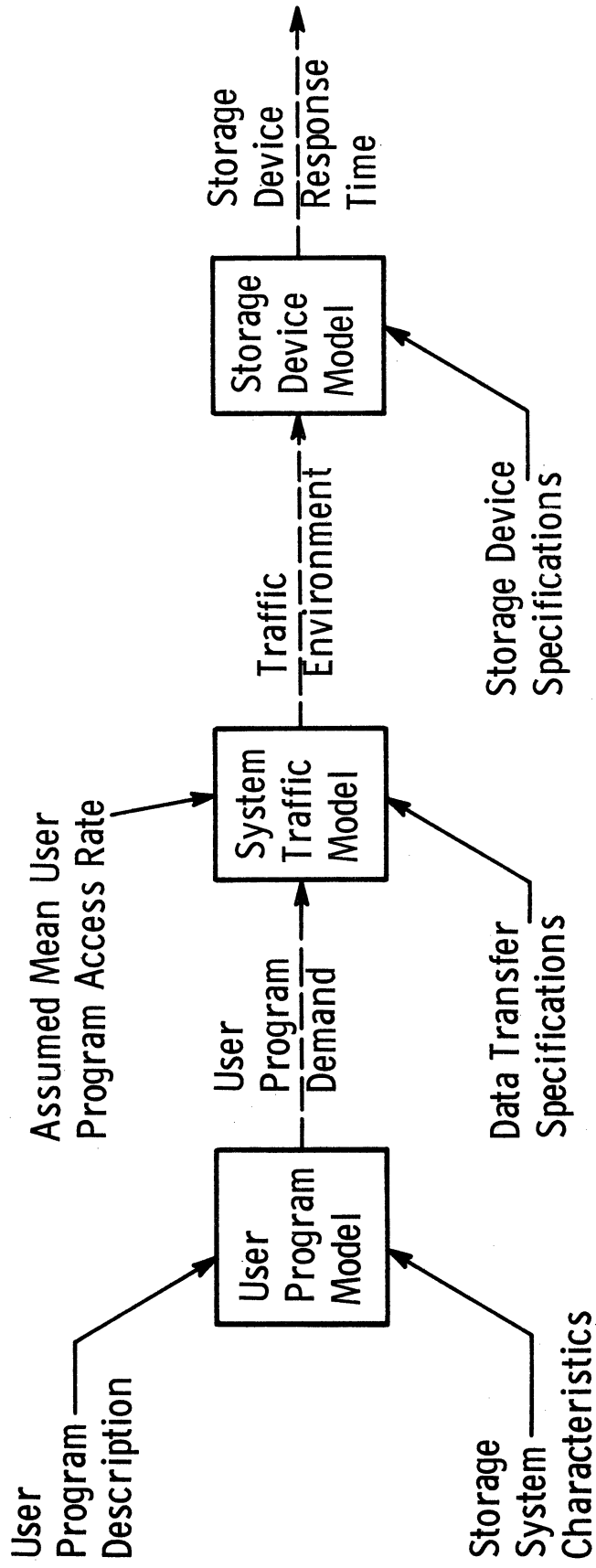


Figure 3.3 Detailed System Model, Part 1

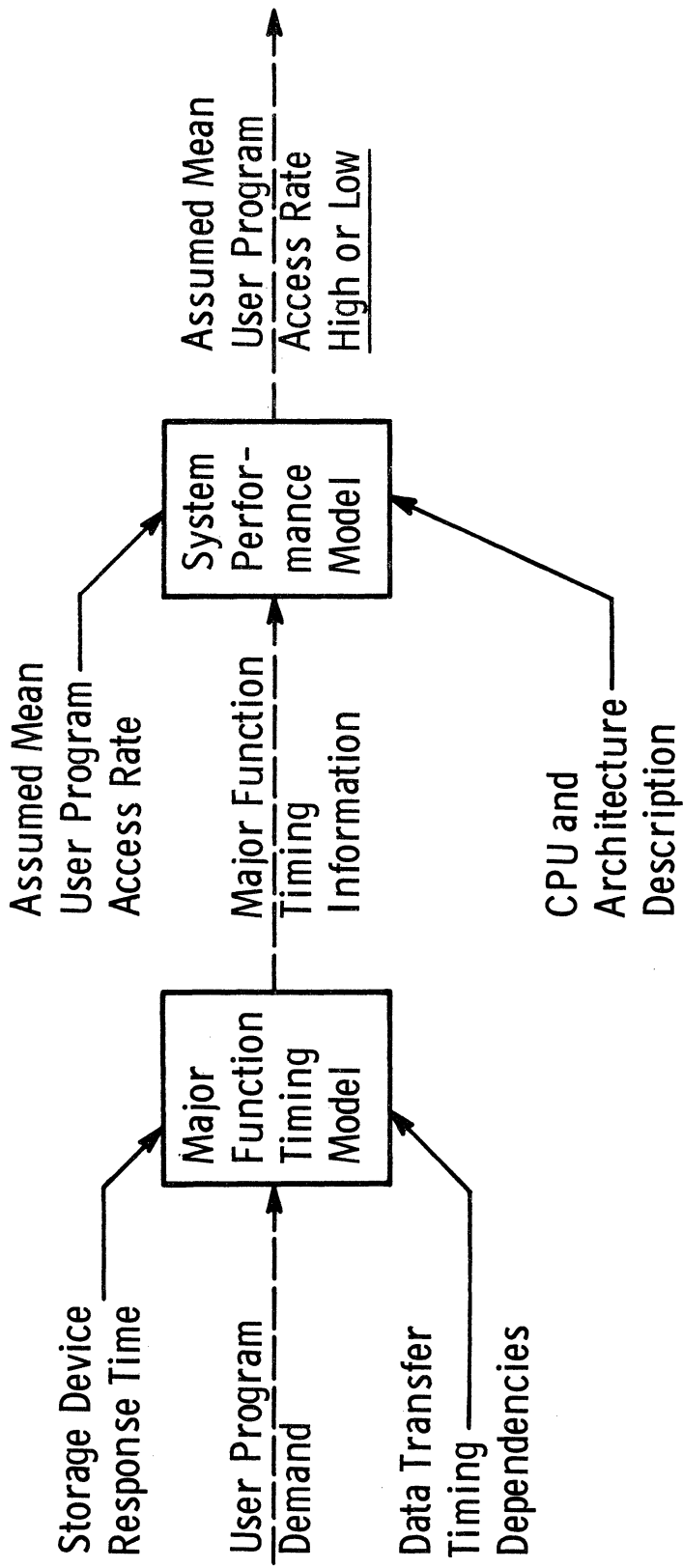


Figure 3.4 Detailed System Model, Part 2

variables which describe the characteristics of the user program. The storage system characteristics consist of the storage allocation at each level in the hierarchy and the logical record size at each level. The user program model, given this information, determines what fraction of a user program's accesses will be directed to each level in the hierarchy of storage. This is referred to and shown in the Figure as user program demand. For example, in a two level system with a core and a drum, this would be the fraction of user program references or accesses to storage which are satisfied at the core and the fraction which require drum activity.

The next block in Figure 3.3 shows the System Traffic Model. This model has as its independent variables the assumed mean user program access rate (assumed performance), the user program demand and the data transfer specifications. The data transfer specifications describe what happens when a user program accesses a given level in the hierarchy. We are treating the user program access to a given level as a cause and the activity generated in the system as a result of that access as the effect. The effect takes the form of data transfers in the system. The data transfer specifications describe the data transfer which occur as a result of user program accesses to some level in the hierarchy. Using this information along with the mean user program access rate and the fraction of user program

accesses to each level (user program demand) the System Traffic Model generates a description of the traffic environment at each level in the hierarchy. The traffic environment is given in terms of the mean rate at which records are being read and written at each level, the mean record size being read and written at each level and the maximum number of requests for service that can be generated by the system for each level.

We arrive now at the Storage Device Model at the right hand side of Figure 3.3. The storage device is actually a collection of models any one of which can be used to represent the hardware at any level in the storage hierarchy. The device models give us the mean time required to read or write a record of a given size at a given level in the hierarchy. The model independent variables are the traffic environment seen by the level in question and device specification relevant to the device type at that level.

Models for core or random access devices, drums, disks and data cells have been developed. The models are not particularly unique or special in any real sense. They are based on finite Markov chains and require a numerical solution. The models were designed to provide generality of application, realistic results and rapid solution.

Backing away from the details of Figure 3.3 we can review what goes in and what comes out of this portion of the system model. We

provide as independent variables a description of user program behavior and some storage system characteristics. From this we learn how the user program behaves in the system. Next assuming a mean user program access rate and given the data transfers which occur at a result of user program behavior we compute the traffic flows at each level in the system. Given this and a description of the hardware at each level we compute the mean time required to read or write records of various sizes at each level. The principal result of the portion of the model shown in Figure 3.3 is simply the mean response time for each of the levels in the hierarchy.

We will now turn to Figure 3.4 where the second portion to the system model is shown. On the left we have the Major Function Timing Model. This model is responsible for timing information such as the mean time that a user program remains ineligible for execution following an access to some lower level or storage. The independent variables of this model are the storage device response times, the user program demand and the data transfer timing dependencies. The data transfer timing dependencies require some explanation.

Imagine a system having a core and a drum. If a user program references some piece of data that happens to be on the drum a record or page will be read from the drum and written in core. There may also be a transfer from the core to the drum in order to make room

for the new information coming into core. Here we see the possibility for 4 reads or writes. There is the operation of reading a record from the drum and writing that record in core and reading a record from core and writing it on the drum. However, in a typical system the only delay experienced by the executing program is that of reading the record from the drum. The other reads and writes do occur and do contribute to the congestion in the system but are generally carried out in such a way as to avoid a direct delay in execution. The data transfer timing dependencies specify precisely which reads and writes contribute directly to delays in program execution.

Finally on the right of Figure 3.4 we have the system performance model. This model has as its independent variables the assumed mean user program access rate, the major function timing information and certain items of a CPU and architecture description. This submodel determines if in fact the assumed mean user program access rate is too high or too low. This completes the model giving us the final result as shown in Figure 3.2.

This description of the model has been necessarily incomplete and simplified somewhat to aid in explanation. The detailed description with examples will be available soon in Systems Engineering Laboratory Report No. 53, Analysis and Optimization of Multiprogrammed Computer Systems Using Storage Hierarchies, by A. M. Woolf.

The model has been implemented in the form of a highly interactive computer program which provides the user with an animated view of the system performance as model parameters are varied. There are roughly 50 independent variables in the model. (The number of independent variables is dependent on the hardware configuration.)

3.1.4 An Example

A typical system is shown in Figure 3.5. The first level of storage, Level 1, in this system is dedicated to the executing program as is the code in the IBM 360/85. The space in the second and third levels of storage are shared by all the programs being multiprogrammed. In this case the third level of the storage hierarchy is a secondary level. This means that an access to this level causes the executing program to lose the CPU until the necessary data transfers are complete. There are many other parameters relevant to the performance of this system and included in the model which will be omitted in this brief discussion.

Typical results of analysis and optimization are shown in Figure 3.6, 3.7, 3.8 and 3.9. In Figure 3.6 we show the performance of the system as a function of the number of programs being multiprogrammed. Figure 3.7 shows the behavior of the queue length at the drum under the same conditions.

In Figure 3.8 we show the performance of the system as a function of user program size. The lower curve in the figure indicates the system performance when 13 programs are being multiprogrammed. The

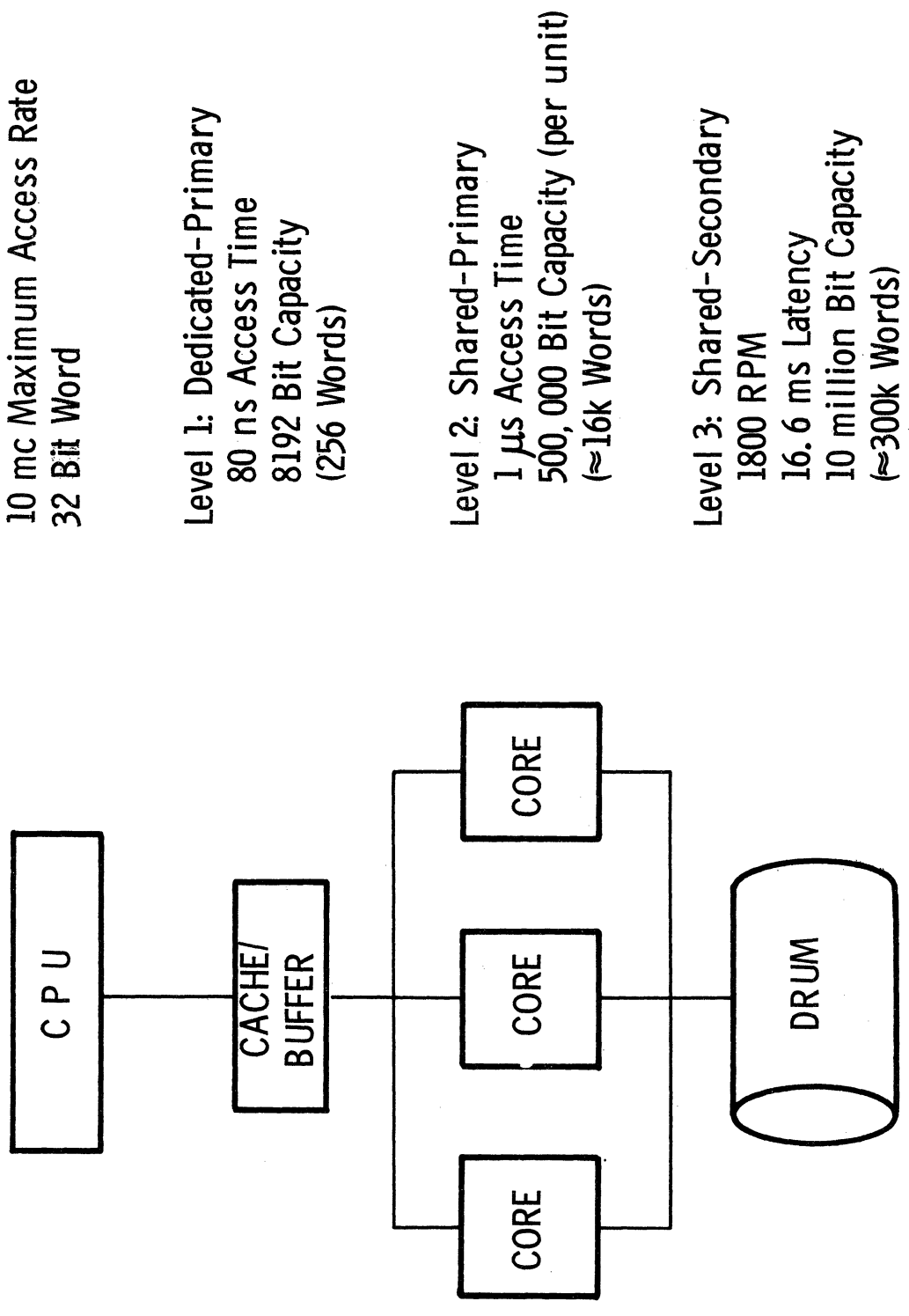


Figure 3.5 System Diagram

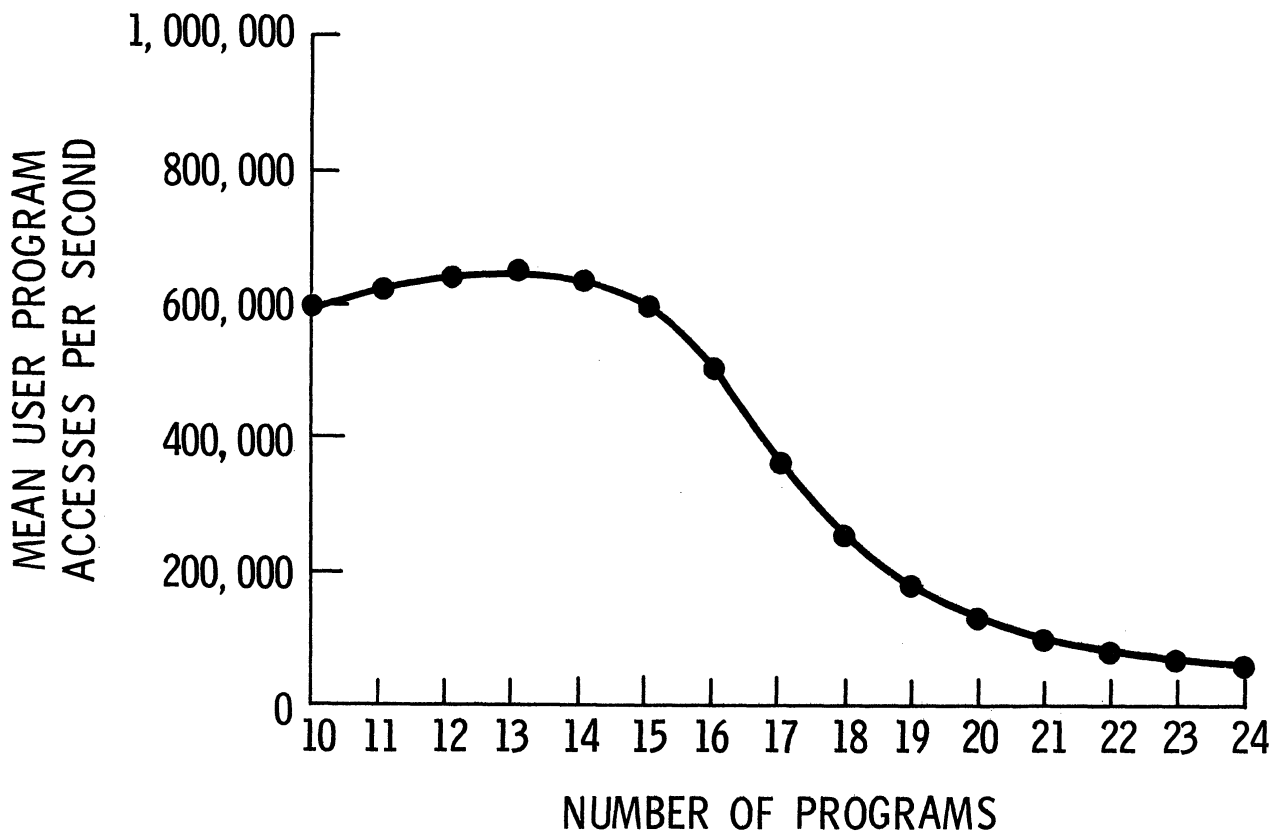


Figure 3.6 Performance Versus Number of Programs

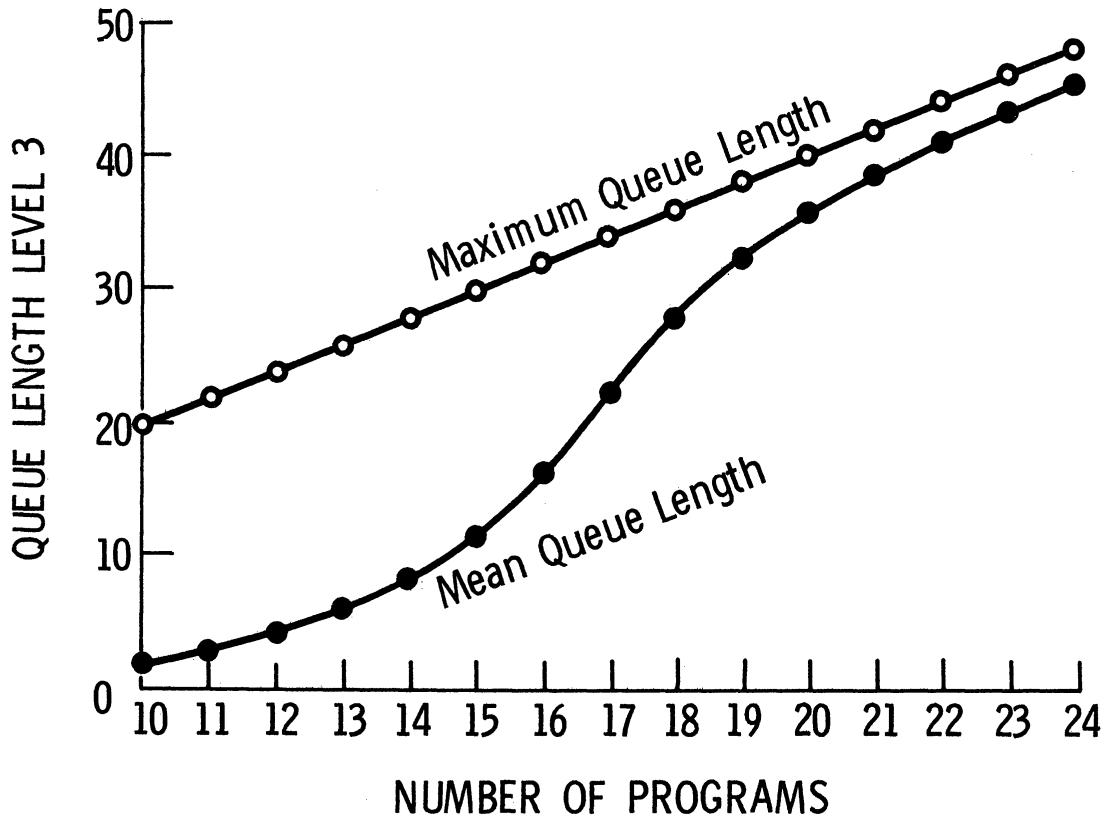


Figure 3.7 Queue Length Versus Number of Programs

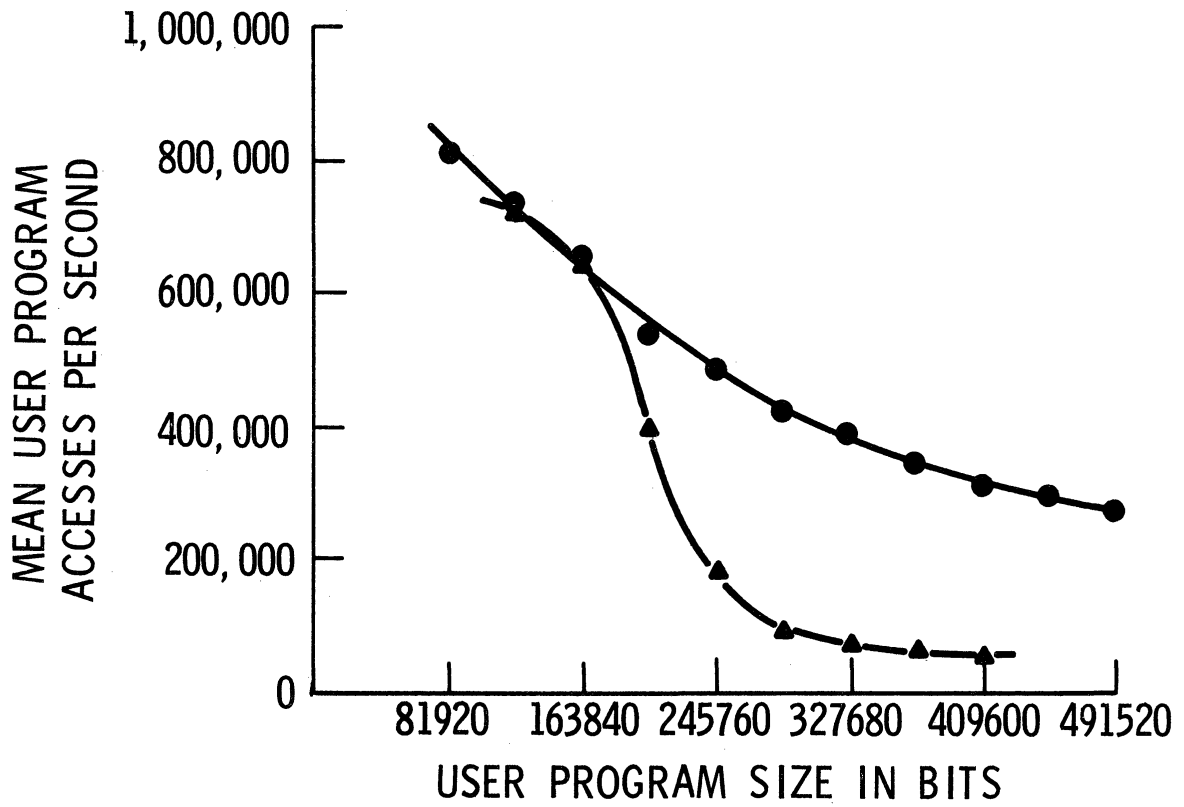


Figure 3.8 Performance Versus User Program Size

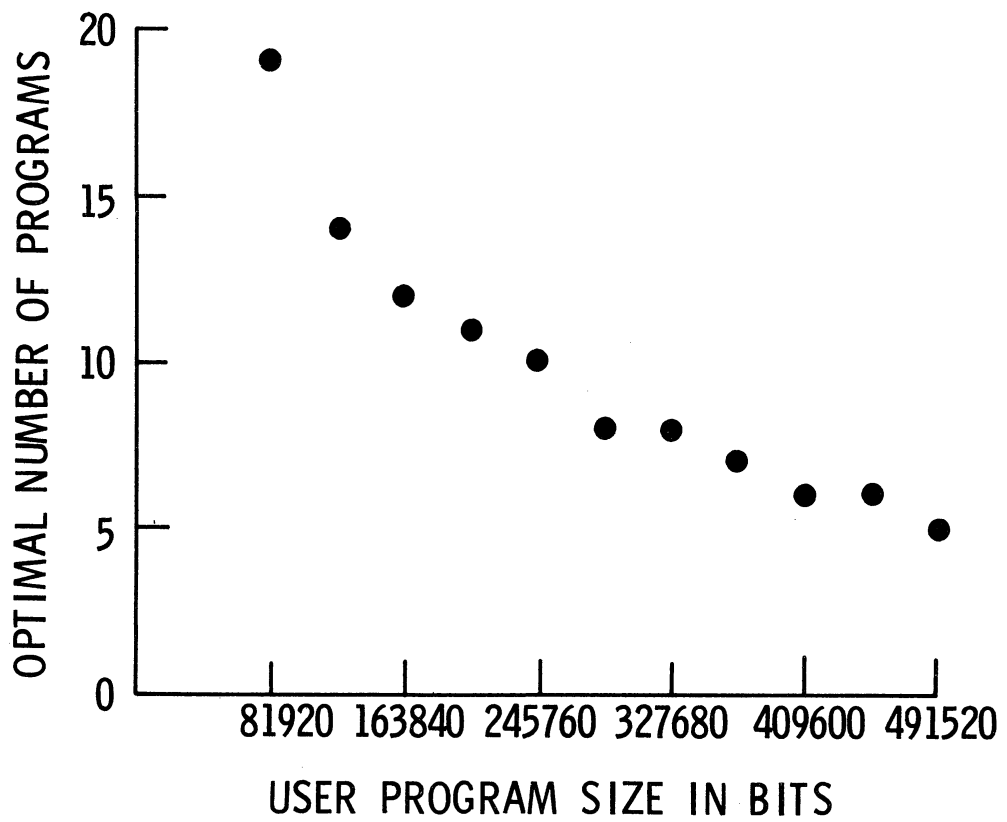


Figure 3.9 Optimal Number of Programs Versus Program Size

upper curve shows the performance when an optimal number of program of the given size are run. Figure 3.9 shows the optimal number of program for each program size.

The results shown here are only a small sample of those possible and are intended only to serve as an example of the kinds of results obtainable.

The solution of the combined system model involves numerical techniques. A complete solution requires approximately 1 second, based on IBM 360/67. When the model is used in the context of optimization, an evaluation can be obtained in approximately 50 ms. In practical terms this means that in the case of analysis the printing of results is more expensive than the analysis itself. In the case of optimization over 1000 systems can be examined in 1 minute or over 60,000 per hour.

The capability to explore the system, by modifying parameters and observing changes in performance, is extremely useful in identifying the important performance controlling characteristics of the system. This capability to explore a system is the result of 3 factors:

1. Low cost of solution
2. Model flexibility
3. An interactive implementation of the model.

The CPU time required for analysis and/or optimization is small. In the case of analysis and the simpler optimizations the cost of solving for the result is generally less than the cost of printing the result. The interactive nature of the implementing program is such that the user spends much more time deciding what changes to make than he does actually making the changes.

The approach taken in the design of the model deserves some discussion. The model was developed specifically for use in the form of a computer program. This has affected a number of important model design decisions.

The model is modular. The most important aspect of this modularity is the ease with which diverse mathematical methods can be applied to different portions of the problem. In choosing a method for modeling storage hardware there was no need to use the techniques employed to model the user program behavior. In each case the method used was developed for the specific application with little regard for methods used elsewhere. This modularity of method, which would lead to a uselessly cumbersome model outside the context of computer implementation, was instrumental in providing the realism and speed of solution.

If there is to be one characteristic of the mathematical model identified as contributing most to its worth as a tool for modeling computer systems, it would have to be this modularity of method.

3.2 Optimum Task Scheduling

The research outlined here is concerned with determining task scheduling rules which will maximize the throughput (task completions per unit time) in heavily loaded multiprogrammed computer systems. A queueing model has been developed to simulate the behavior of the system as a function of scheduling policies chosen. This model is documented more fully in the previous annual report [15] and in the report which presents the full results of this study. [16] In brief, we are interested in investigating system throughput as a function of policies chosen and as a function of whether or not reentrant procedure or other sharable information is made use of in the system.

This model is Markovian and is formulated as a Markov renewal decision process (MRDP) so that optimization of this decision process gives optimum task scheduling rules for both core memory and the central processors. Two problems arise out in the attempt to get useful results from this model. First, is the problem of finding a suitable solution technique for the problem. Second, is the problem of obtaining values for the parameters of the model and justification for assumptions made in the model. These two problems are presented in more detail in the following two sections and some results are given.

3.2.1 Markov Renewal Decision Processes

This section provides the preliminaries needed to understand

techniques for the solution of a Markov-renewal decision processes (MRDP's) first formulated by Jewell [1]. These processes are a generalization of the Markov decision processes described by Bellman [2] and Howard [3]. Briefly, MRDP's are Markov-renewal processes with an finite number of states in which the parameters of the process (p_{ij} and $F_{ij}(t)$) are dependent on a controller which makes one of a finite number of choices at each transition. With each decision made by the controller are associated transition dependent rewards and the solution to the optimization problem consists of finding the set of decisions which maximizes the expected return as determined by the rewards.

When occupying state i ($i = 1, \dots, N$), we choose one of a finite number of alternatives $k(i)$ ($k(i) = 1, 2, \dots, K(i)$). For $k(i) = k$ we have associated a transition probability p_{ij}^k , and distribution on transition interval, $F_{ij}^k(t)$. The reward under alternative k for making a transition from state i to state j is $R_{ij}^k(t|\tau)$ where $t \leq \tau$ is the elapsed time since the beginning of the transition interval which is of length τ . Define

$$R_{ij}^k(\tau) = R_{ij}^k(\tau|\tau)$$

Then

$$q_i^k = \sum_{j=1}^N p_{ij}^k \int_0^{\infty} R_{ij}^k(\tau) d F_{ij}^k(\tau)$$

is the expected one transition return starting in state i and using alternative k .

Note that if the transition interval is fixed and of length T then we have the special case

$$q_i^k = \sum_{j=1}^N P_{ij}^k r_{ij}^k \quad (2)$$

where $r_{ij}^k = R_{ij}^k(T)$. This is the case considered by Howard [3] and for this situation the more general rewards reduce to the above.

We may define two **distinct** objectives for the controller.

- (1) Maximize the expected return per transition (which we call the transition-optimal problem).
- (2) Maximize the expected return per unit time (which we call the time - optimal problem).

Bellman's "Principle of Optimality" [4] may be used to derive recurrence relations for these processes. These recurrence relations will specify the alternatives to be chosen at each transition in order to maximize expected returns. The recurrence relations for the two problems are

- (1) The transition optimal problem. The recurrence relation is

$$v_i^k(n) = \max_k \left[q_i^k + \sum_{j=1}^N p_{ij}^k v_j^k(n-1) \right] \quad (3)$$

where

$v_i^k(n)$ is the expected return given that we start in state i and the process makes n transitions before terminating.

$v_i^k(0)$ is the boundary reward received for being in state i at the termination of the process.

(2) The time-optimal problem. The recurrence relation is

$$w_i(t) = \max_k \sum_{j=1}^N p_{ij}^k \left\{ \int_t^\infty dF_{ij}^k(\tau) [R_{ij}^k(t|\tau) + S_{ij}^k(t, \tau)] + \int_0^t dF_{ij}^k(\tau) [R_{ij}^k(\tau) + w_j(t - \tau)] \right\} \quad (4)$$

$w_i(t)$ is the expected return given that we start in state i and terminate the process after time interval t .

$w_i(0)$ is the boundary reward received for being in state i at termination of the process.

$S_{ij}^k(t, \tau)$ is a boundary reward received when the process terminates. It is a function of the two states involved in the transition which is interrupted at the time of termination and of the total time and elapsed time of this interrupted transition.

The first term in the recurrence relation represents the expected return if no transitions are made before the process terminates. The second term represents the return from one transition of time $\tau < t$ plus the expected return in the remaining time, $t - \tau$. For convenience we rewrite the recurrence relation as follows:

$$w_i(t) = \max_k \left\{ \sigma_i^k(t) + \sum_{j=1}^N p_{ij}^k \int_0^t w_j(t-\tau) dF_{ij}^k(\tau) \right\} \quad (5)$$

$$\sigma_i^k(t) = q_i^k + \sum_{j=1}^N p_{ij}^k \left\{ \int_t^\infty dF_{ij}^k(\tau) [S_{ij}^k(t, \tau) - R_{ij}^k(\tau) + R_{ij}^k(t|\tau)] \right\} \quad (6)$$

For general distributions on the transition interval it is clear that Equation (5) cannot be solved. We may, however, approximate $F_{ij}^k(t)$ with a lattice distribution and obtain a solution in this manner.

The recurrence relations given for the transition optimal and time optimal problems formulated above provide a practical method of solution for these problems providing the number of iterations does not grow too large. In many practical problems, however, this is not the case and in fact we want to know the behavior of the recurrence relations as $n \rightarrow \infty$ and $t \rightarrow \infty$ in Equations (3) and (5) respectively. This in fact is the problem we must solve to obtain solutions in the scheduling model and the research is devoted to this end. Howard [3] and Jewell [1] obtain solutions to these problems but their solution method requires matrix inversion and for large models the computation required may be excessive. Therefore we would like a method which avoids this.

Define a stationary policy, α , as an N-tuple which specifies a particular alternative in each of the N states. Then we may write recurrence relations for the expected return when following stationary policy α . They are

$$V_i^\alpha(n) = q_i^\alpha + \sum_{j=1}^N p_{ij}^\alpha v_j^\alpha(n-1) \quad (7)$$

$$w_i^\alpha(t) = \sigma_i^\alpha(t) + \sum_{j=1}^N p_{ij}^\alpha \int_0^t w_j^\alpha(t-\tau) dF_{ij}^\alpha(\tau) \quad (8)$$

Under a stationary policy these recurrence relations may be shown [1] to have asymptotic forms

$$v_i^\alpha(n) \approx g^\alpha n + v^\alpha \quad \text{as } n \rightarrow \infty \quad (9)$$

$$w_i^\alpha(t) \approx f^\alpha t + w_i^\alpha \quad \text{as } t \rightarrow \infty \quad (10)$$

Because it has been shown [5] that there exists an optimal stationary policy for each of the infinite stage and infinite time optimization problems, it is clear that our objective is to find that policy, α , which maximizes g^α or f^α in Equations (9) or (10), respectively. g^α is called the gain and f^α the gain rate using policy α .

The following theorem proves the result in Equation (9) along with some other related results which are useful in obtaining solutions to MRDP's.

Theorem 2.1

If we are given a recurrence relation

$$v_i(n) = q_i + \sum_{j=1}^N p_{ij} v_j(n-1) \quad (11)$$

$$v_i(0) = c_i \quad (12)$$

where the $N \times N$ stochastic matrix $P = (p_{ij})$ is acyclic and has only one closed communicating class of states so there exists a vector of limiting state of probabilities $\pi = (\pi_1, \dots, \pi_N)$, we have the following results.

$$\lim_{n \rightarrow \infty} [v_i(n) - v_i(n-1)] = g, \quad (13)$$

a constant independent of i .

$$v_i(n) \sim gn + v_i \quad \text{as } n \rightarrow \infty \quad (14)$$

$$g = \sum_{i=1}^N \pi_i q_i \quad (15)$$

Proof

$$\text{Define } g_i(n) = v_i(n) - v_i(n-1)$$

$$\text{and } G(n) = \begin{pmatrix} g_1(n) \\ \vdots \\ g_n(n) \end{pmatrix}$$

$$\text{then } G(n) = PG(n-1) \quad (16)$$

$$G(n) = P^{n-1} G(1) \quad (17)$$

$$\text{Since } \lim_{n \rightarrow \infty} P^n = \begin{pmatrix} \pi \\ \vdots \\ \pi \end{pmatrix}$$

$$\text{we obtain } \lim_{n \rightarrow \infty} g_i(n) = \pi_1 g_1(1) + \dots + \pi_n g_n(1) = g \quad (18)$$

It follows immediately that

$$v_i(n) \sim gn + v_i \text{ as } n \rightarrow \infty.$$

To derive Equation (15) define vectors

$$v(n) = \begin{pmatrix} v_1(n) \\ \vdots \\ v_n(n) \end{pmatrix} ; \quad Q = \begin{pmatrix} q_1 \\ \vdots \\ q_n \end{pmatrix}$$

$$\text{Then } V(n) = Q + PV(n-1) \quad (19)$$

$$V(n) = Q + PQ + P^2 V(n-2)$$

$$\vdots$$

$$V(n) = Q + PQ + P^2 Q + \dots + P^{n-1} Q + P^n V(0) \quad (20)$$

$$G(n) = P^{n-1} Q + (P^n - P^{n-1}) V(0) \quad (21)$$

$$\lim_{n \rightarrow \infty} G(n) = \begin{pmatrix} \pi \\ \vdots \\ \pi \end{pmatrix} Q$$

$$\text{or } g = \sum_{i=1}^N \pi_i q_i$$

3.2.2 Data Collection and Analysis

This section gives the background for the data collection and analysis carried out on the multiprogrammed, time-shared system at The University of Michigan and some results from that effort.

The computing system consists of an IBM System/360 Model 67 with two central processing units and 384 pages of core storage (1024 words per page). This is a virtually memory machine operated in a page-on-demand mode with paging to two drums (and possibly disks). The system runs under UMMPS (University of Michigan Multi-Programming System) which is described in more detail by Alexander [[6] and Pinkerton [7, 8] . In addition to the supervisor there exists a reentrant program, MTS (Michigan Terminal System), which controls remote terminal users and the batch streams.

Code has been added to the system which allows data collection to be initiated at the operator's console. The specific items which may be collected are described by Pinkerton [7, 8] . Events which occur along with the time they occur are placed in buffers which are written onto tape when they are full. For example, each time the CPU changes hands, the time is recorded along with the identification of the job relinquishing the CPU and the job receiving the CPU. Pinkerton has shown that the process of collecting data disturbs the system operation a negligible amount so no cognizance need be taken of this when interpreting results.

Data were collected October 10, 1970 about 3:00 p.m. This time was chosen in order to observe the system with as heavy a load as possible. During the data collection period (approximately 22 minutes) there were, on the average, about 50 terminal user and two batch streams active. The CPU time used may be placed in four broad categories. MTS batch and MTS terminal CPU time consists of time used by jobs in the batch stream and terminal users, respectively. Non-MTS CPU time is time used by system routines which cannot reasonably be charged to a particular job. Finally the CPU may be idle. The number of seconds spent by the CPU in each of these four categories is shown in Table 3.2.1. The average length of time a job held a CPU was 5.7 ms.

Table 3.2.1
CPU Use During Data Collection (seconds)

Non-MTS	601
MTS batch	341
MTS Terminal	1571
Idle	129

We now present some results of analysis of the collected data. Figure 3.2.1 shows the cumulative distribution for the elapsed time between occurrence of a page fault and satisfaction of the page request. The particular curve shown here is for page fault during runs of a Fortran compiler, but the distribution is independent of the type of task. In the following we present more results for a Fortran compiler.

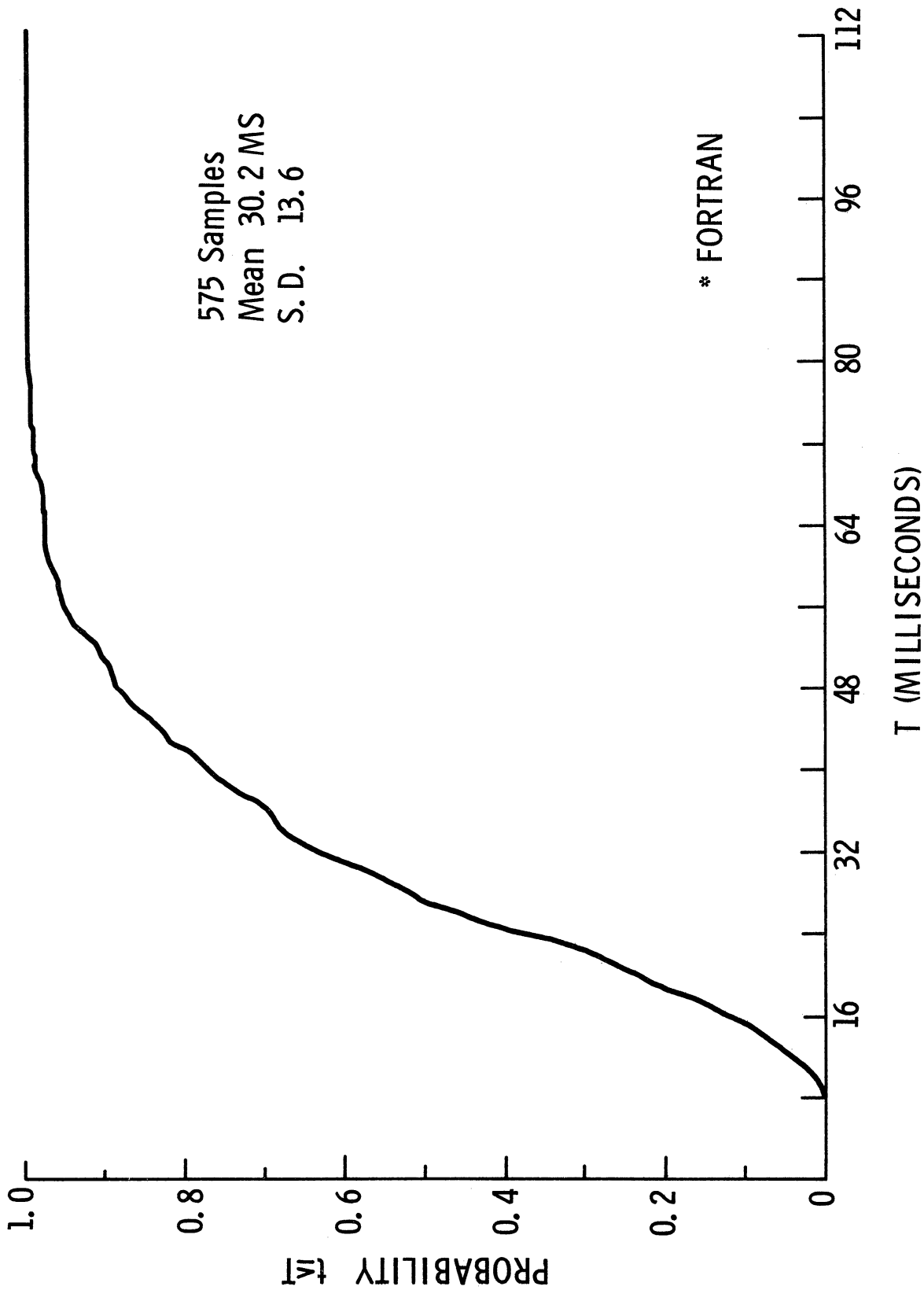


Figure 3.2.1 Cumulative Distribution of Time (t) From Page Demand Until Page Available

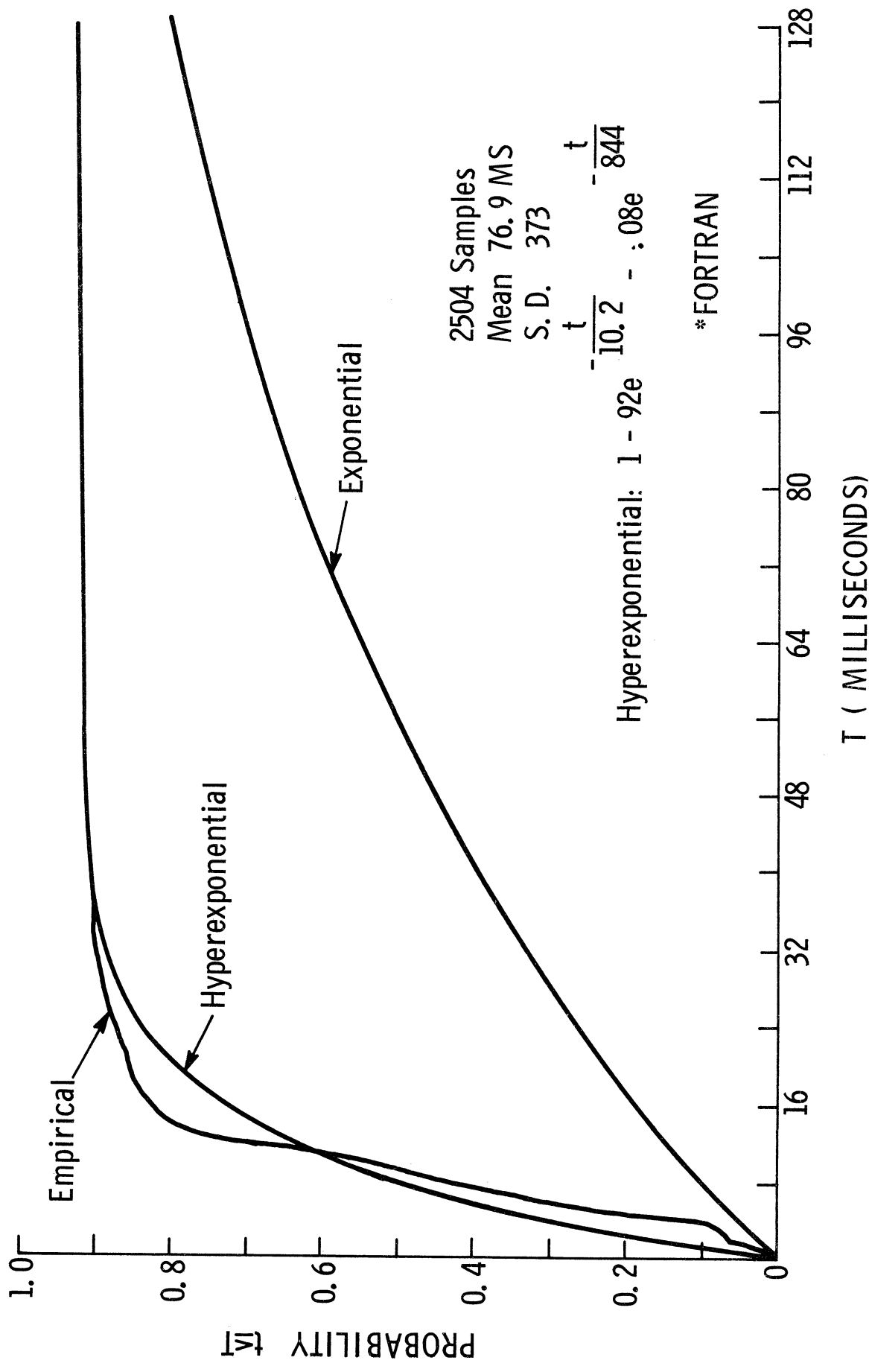


Figure 3.2.2 Cumulative Distribution of CPU Time (t) Used Between I/O During Fortran Compilations

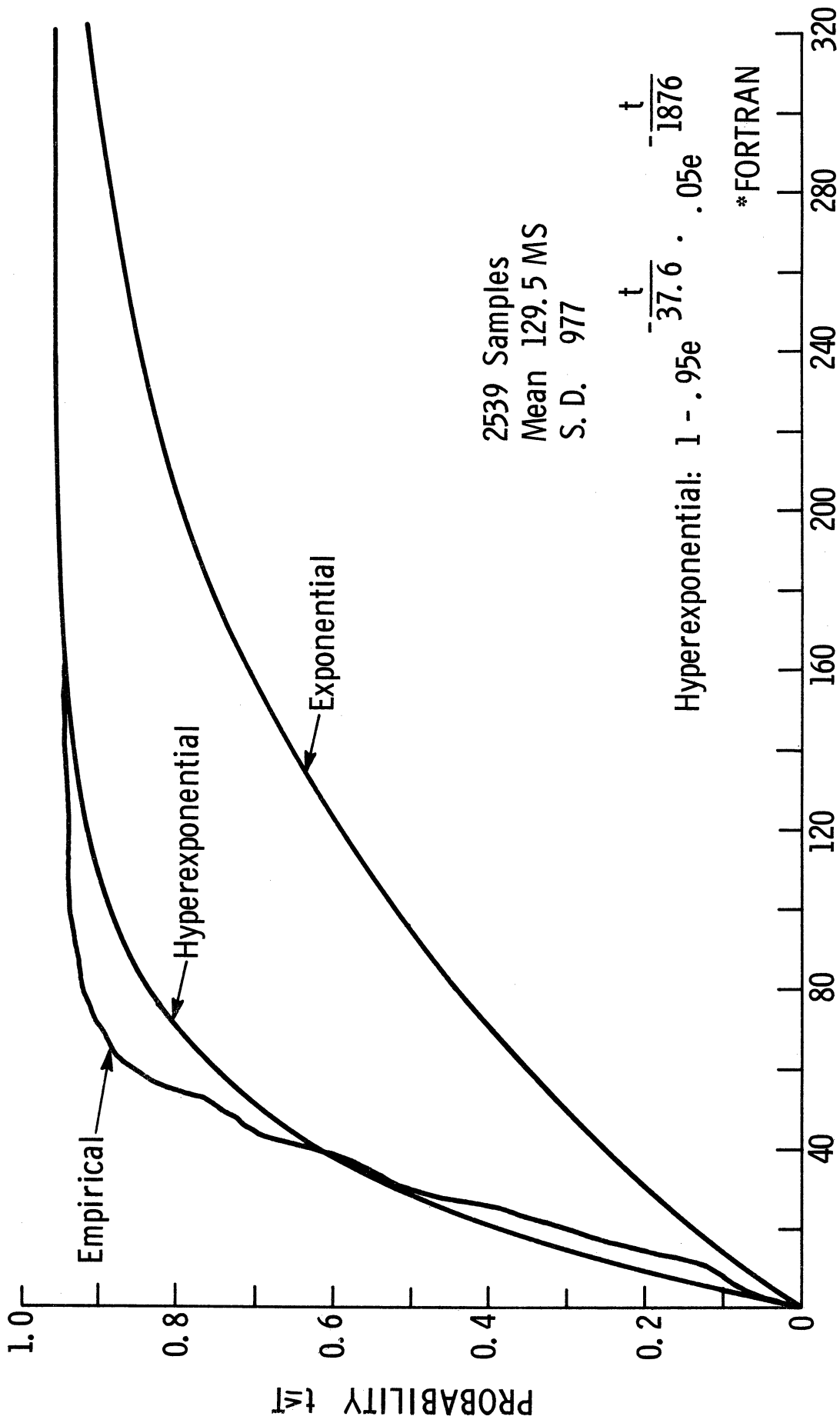


Figure 3.2.3 Cumulative Distribution of Time (t) Used Per I/O Request During Fortran Compilations

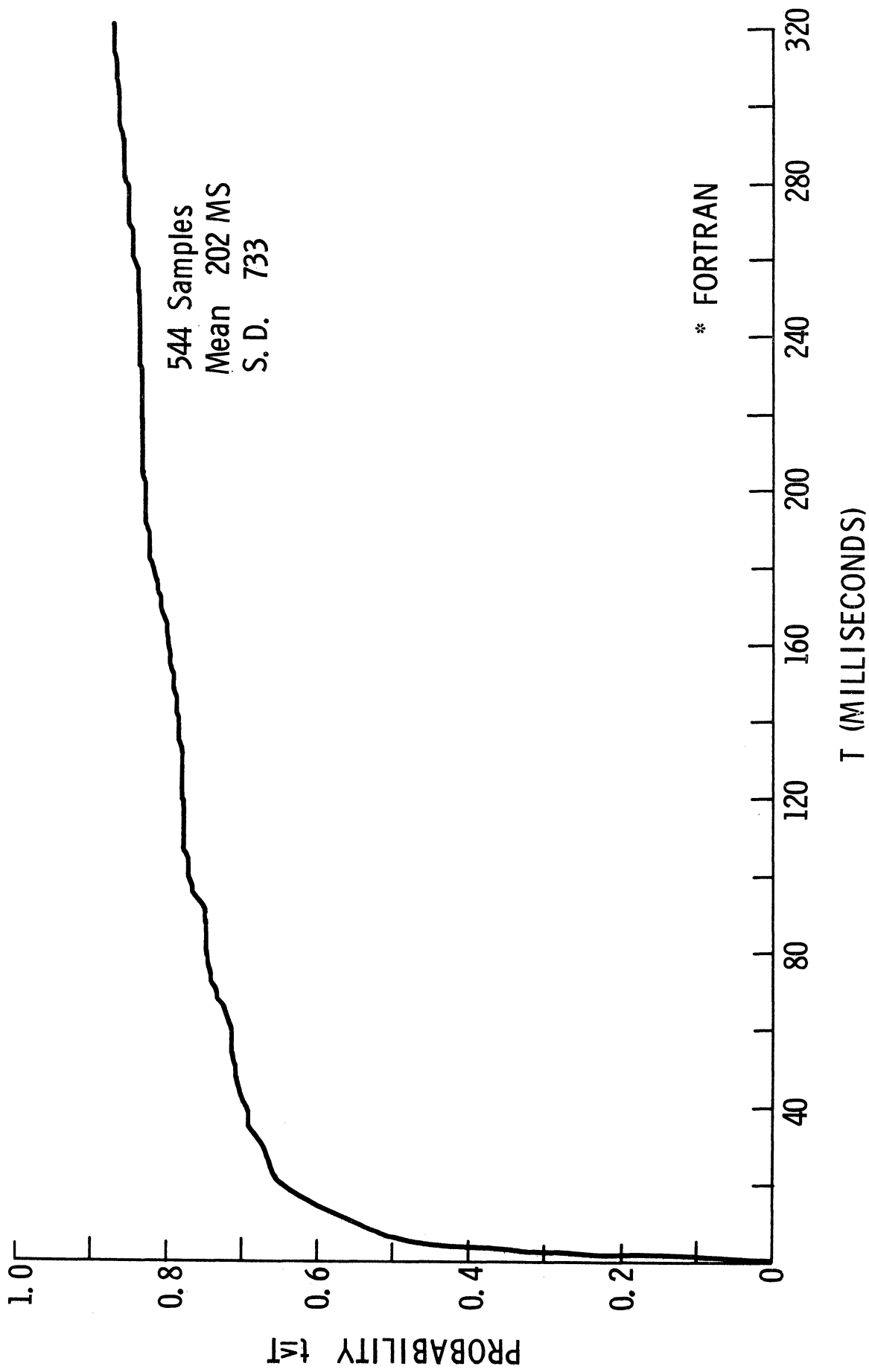


Figure 3.2.4 Cumulative Distribution of CPU Time (t) Used Between Page Faults During Fortran Compilations

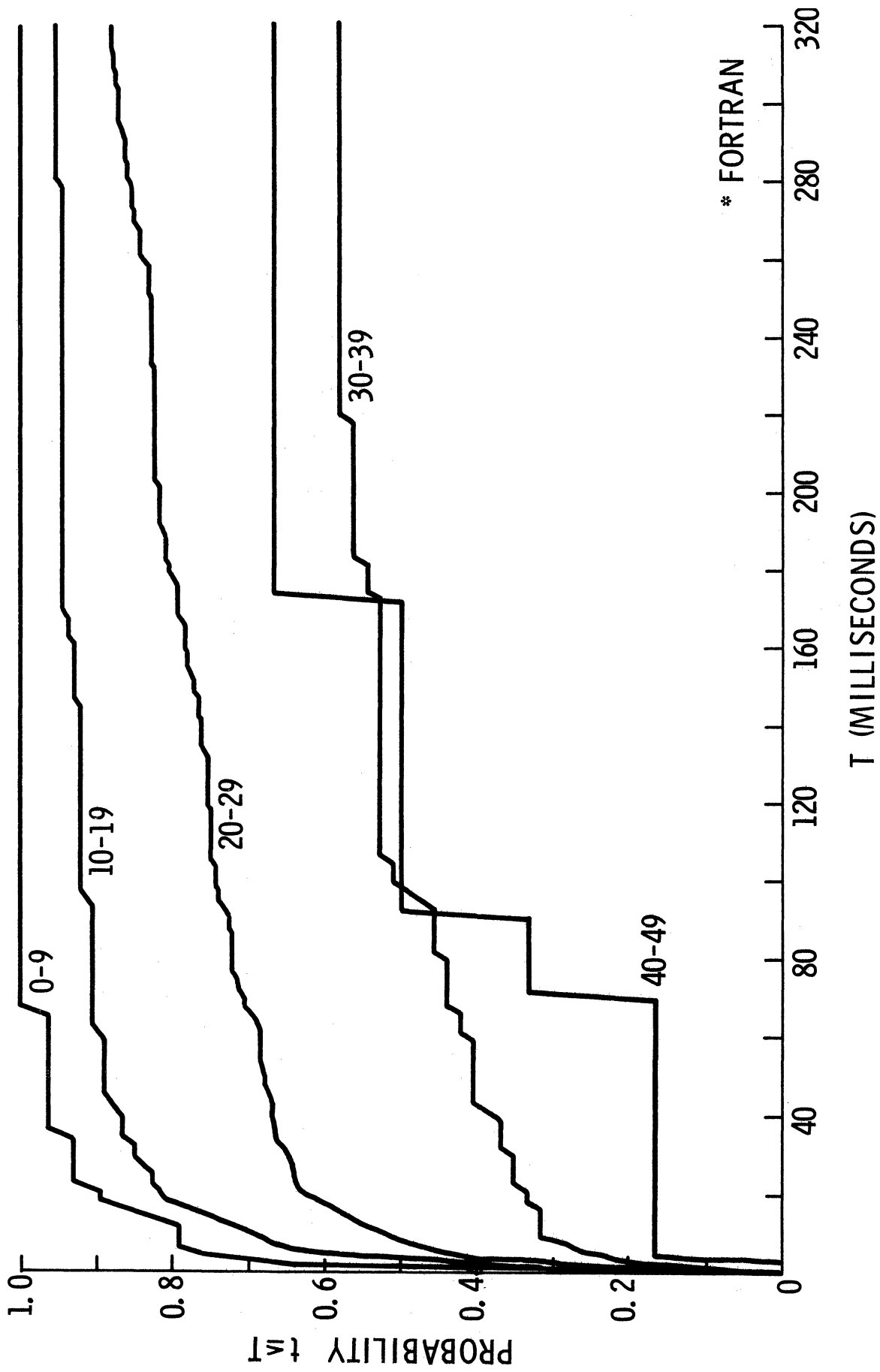


Figure 3.2.5 Cumulative Distribution of CPU Time (t) Used Between Page Faults as a Function of the Number of Pages in Core for Fortran Compilations

This compiler is by far the most heavily used program run by users of MTS. It is the IBM Fortran IV (G) compiler with a modified interface to allow it to run with the operating system (UMMPS) used at The University of Michigan. There were 41 requests (loads) for this compiler during the 22 minute data collection period. The mean elapsed loading time for those 41 samples was 5.06 seconds with a standard deviation of 1.75. Data obtained during completed executions of this compiler are presented in Table 3.2.2. This table presents means and standard deviations for elapsed execution time, CPU time, I/O time, and page wait time. The time the tasks are eligible to use the CPU (ready) may be obtained by subtracting CPU time, I/O time and page wait time from the elapsed running time. The mean number of I/O operations and the mean number of page waits per run are also given in this table.

In addition the table shows virtual memory and real memory assigned to tasks during execution. The virtual memory consists of the memory assigned the executing task plus three pages of tables used by the system to keep track of the job status.

Cumulative distributions are given for the length of time the task uses the CPU between I/O requests and for the length of time the task uses the CPU between page faults. Also distributions are given for the time required to carry out requested I/O operations (time from I/O request until request satisfied and task again eligible to use a CPU).

In Figure 3. 2. 2 we show the distribution of lengths of CPU intervals between input-output requests for the compiler. In addition, two theoretical distributions with the same mean as the empirical distribution are shown. It is clear that the hyperexponential distribution fits the data far better than the exponential distribution. Figure 3. 2. 3 gives the distribution of lengths of I/O operations during Fortran compilations and again we see that a hyperexponential distribution fits the data far better than an exponential distribution.

Table 3. 2. 2
Fortran Execution Data

Item	Samples	Mean	S. D.
Elapsed Time Per Run (sec)	40	19. 1	27. 0
CPU Time Per Run	40	4. 90	5. 37
I/O Time Per Run	40	8. 13	16. 3
Page Wait Time Per Run	40	. 434	. 758
Number I/O Requests Per Runs	40	63. 1	71. 9
Number Page Waits Per Run	40	14. 4	24. 1
Virtual Pages Per CPU Interval	2504	29	7
Real Pages Per CPU Interval	2504	24	7
Virtual Pages Per I/O Interval	2539	30	9
Real Pages Per I/O Interval	2539	25	8

Figures 3.2.4 and 3.2.5 give the distribution on the CPU time used between page faults. Figure 3.2.5 is a breakdown of the data in Figure 4 by mean number of pages in core during the period of CPU use between page faults. Pertinent data for this figure are given in Table 3.2.3.

Table 3.2.3
Data for Figure 5

Pages in Core	Samples	Mean	S.D.
0-9	29	7.54	13.6
10-19	127	43.6	136
20-29	325	205	775
30-39	57	620	1179
40-49	6	415	492

3.3 Selection of Optimal Sets with Application to Computer Programming

In the design of large scale systems it is frequently useful to formulate a number of engineering problems as optimization problems in zero/one variables.

Much effort has been directed towards developing algorithms for solution of these problems which are readily programmed and do not require unreasonable amounts of computation and storage.

Linear programming has been one of the more successful techniques developed to deal with problems of this nature. For example, special algorithms have been developed for efficient solution of the chemical

transportation and assignment problems.

In this work we develop algorithms for solution of a particular quadratic programming defined as follows:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i,j} c_{ij} x_i x_j - \sum_j b_j x_j \\ \text{Subject to} \quad & \sum_i a_i^k x_i \leq \mu^k \quad k=1, 2, \dots, m \end{aligned}$$

where $x_i \in \{0, 1\}$, $c_{ii} = 0$ for all i , $c_{ij} \geq 0$, μ^k, a_i^k, b_j are real numbers for $k=1, 2, \dots, m$ and all

For the case of $m \leq 2$, an algorithm utilizing generalized Lagrange multipliers [13] was developed for solution of the above problem. Using this approach techniques were developed for solution of problems containing a relatively large number of variables which did not require a prohibitive amount of computation time or storage.

This problem can be shown to have a wide range of applicability in both operations research and computer systems design.

In the following system we discuss a particular problem area where we have found application of quadratic programming to be useful.

3.3.1 Selection of Optimal Program Pages in Multiprogramming System

Multiprogramming, as the term implies, means having several programs occupy high speed memory simultaneously. Problems of effectively using storage in a multiprogrammed mode are appropriately called problems of storage allocation. When we consider these problems in

the context of the early days of computing, they were relatively straight forward.

In early batch systems where programs were run one at a time, each program had the entire high speed memory (core) available. Problems arose when a program was larger than the available core. In these cases the programmer had to improvise by "segmenting" his program (instructions and data), and controlling the "overlying" of segments. Segmenting referred to dividing the program into parts (segments), and overlying was the process of bringing unused parts of a program into core from auxiliary storage as they were needed during execution.

When the problem of segmenting a program was given to the operating system instead of the programmer, we had what was called an automatic segmenting system; the problem of segmenting a program in an optimum manner, e. g. , so as to minimize the number of overlays, became known as the classical overlay problem or the problem of program segmentation. In some systems, excessive overlying could cause a serious decrease in the system's operating efficiency.

In multiprogramming systems overlying or segmenting is intended to increase the size of effective memory. A summary of techniques for overlying can be found in [9].

3.3.2 Paging

When high speed memory is divided into fixed size blocks called page frames, and programs and data divided into similar fixed sized blocks

called logical pages the process of mapping the logical pages to page frames is called paging. The quadratic programming algorithm was used as a tool for determination of the logical pages of programs. Comparative results with alternative strategies and some applications can be found in [36] .

3.3.3 Model

Another natural model for the study of the optimal selection of program pages or "pagination" is the linear graph. Ramamoorthy [10] was the first to formulate the problem of pagination as a graph partitioning problem.

Basically, the model for optimal selection of program pages is as follows. Each vertex of a graph represents a block of instructions or data; the directed arcs between blocks represent transfers of control between blocks of instructions. Usually, undirected areas are used to indicate a block of instructions referencing data. Since the arc is undirected, control is to remain in the instruction block.

The abstract statement of the problem in graph theoretic terms is, given a graph G with weighted arcs, partition the graph into disjoint subsets of vertices such that no subset of vertices is larger than some maximum size, and the sum of the weights of the arcs between subsets is minimized.

3.3.4 Solution Technique for Graphical Partitioning

3.3.4.1 Optimal Systems

It is possible to formulate the graphical partitioning problem in terms of an integer linear program with a large number of constant equations to insure the feasibility of the subsets of vertices selected.

However, for problems of practical significance, i. e. , the number of vertices greater than about 20, the methods of integer linear programming are not sufficiently powerful to give solutions in a reasonable amount of time. For example, the number of ways in which a set of 25 objects can be partitioned into 10 subsets is 1, 203, 163, 392, 175, 387, 500. Now given that many partitions will be infeasible due to size constraints on the vertices, there will remain an inordinate number of cases even after those infeasible partitions are deleted.

3.3.4.2 Known Optimal Procedures

There have been reported in the literature a number of optimal procedures for special cases of the graphical partitioning problem. One algorithm using backtrack programming has been used to solve the general partitioning problem when the number of vertices is small [11] .

Other algorithms for optimal solution suitable for large graphs usually require that graph be tree structured see [12], [14]. Some extensions of these methods have been made when the graph is acyclic (directed but with no cycles). These extensions require that the partition is not disjoint, i. e. , certain vertices are duplicated and appear in more than one block of the partition.

3.3.4.3 Heuristic Procedures

In view of the fact that most practical problems involve graphs with the numbers of vertices too large for exact methods, evaluations of existing heuristics methods and a heuristic based upon quadratic programming were

made.

The quadratic programming problem was used as vehicle for selecting block of a partition as follows:

Selecting Maximum Weights Blocks

Let c_{ij} be weight of arc (i, j) and a_j be the size of vertex j , then we can select a maximum weight block of vertices by solving,

$$\text{Maximum} \quad \sum_{i, j} c_{ij} x_i x_j$$

$$\text{Subject to} \quad \sum_i a_i x_i \leq \mu$$

(A)

$$x_i \in \{0, 1\}, \quad \text{where}$$

$$x_i = 1, \text{ vertex is selected}$$

$$= 0, \text{ otherwise;}$$

μ represents the maximum size of a block of vertices. Successive solution of the above problem gives a partitioning of the graph.

Selecting Block Which Minimizes Inter-arc Weights

An alternative strategy for partitioning is to select the set of vertices for a block of the partition which minimizes the total weight of the arcs between the selected vertices and the vertices which are not selected. This can be accomplished by solving the following:

Maximize

$$\sum_{i,j} c_{ij} x_i x_j + \sum c_{ij} (1-x_i)(1-x_j)$$

Subject to

$$\sum_i a_i x_i \leq \mu$$

(B)

$$\sum_i x_i \geq 1$$

In this case the second constraint is used to avoid the nontrivial solution $x_i = 0$ for all i . Successive solution of this problem also gives a partition of the graph. In comparing this method of partitioning with the first strategy, it was found that the results were in general better using problem (B). Typical solution times for problem (B) are shown in table 3.3.1, where N represents the number of vertices of the graph and M represents the number of arcs.

Table 3.3.1

N	M	Time (sec.)
4	4	.012
8	10	.47
14	17	3.42
25	50	2.44
46	75	4.80

3.4 Multiprocessor Scheduling

The basic characteristic of a multiprocessing system is the existence of several processors which can operate independently and have access to a common central memory. In cases where sufficient processing power is not available from one processor, a system of two or more processors closely connected through shared I/O and core storage may provide sufficient capability. The efficient utilization of such a system can be effective in decreasing the computation times of many programs. This is especially important for real-time applications but also lengthy computations, when the results are needed more quickly than they can be provided by a single processor. The stringent reliability and availability requirements of certain applications including real-time process control, air traffic control or air reservation systems, and radar control can be met through the means of multiprocessing. The scope of this research is limited to the operational and scheduling aspects of the multiprocessors computer systems, and to the problem of making optimal use of a multiprocessing system in reducing the computation times and meeting the requirements of parallel processable and real-time programs which are available for analysis and scheduling prior to execution. The reliability aspects of such systems will not be of much concern.

In a typical multiprocessor computer system, the interaction between the program and the system's demand environment is of

prime importance. The usual kind of demand environment of a multi-processor computer system is statistical in nature and the system's response to user's requests are constrained to occur within reasonable time. Small variations in service are tolerated. The typical time-sharing systems behaves in this manner and following Manacher [17] , we call such an environment a soft real-time environment. We will also be concerned with another kind of environment, one in which the external world presents the system with a set of rigid constraints or timing bounds. Certain times before a program or a sub-program cannot be started and certain times before which it must be completed. The starting time constraint could result from lack of data for processing before a certain time. When the external world is characterized in this fashion, it is called a hard real-time environment. The reason for considering this added complexity is that this type of environment is the one which has provided the main impetus for the development of the computer under consideration. Multiprocessor systems can be envisioned as ones which possess a pool of computing resources, and which when applied to a single program with sufficient degree of parallelism can reduce its run time significantly and at the same time the surplus resources can be shared with other programs.

The advent of multiprocessing systems, with the added complexity of parallel processing, has naturally introduced a new level of complexity or better, a number of new research areas, in virtually all aspects of computer

systems. A number of significant abstract and concrete theories, at different levels of detail and sophistication, have been proposed which might serve as foundations for research in parallelism. Investigations into program parallelism are going into two directions. The first is the introduction of programming facilities that encourage parallelism to be explicitly specified and used at the coding level. The other is the implementation of algorithms at the compiling level to investigate and detect potential implicit parallelism in sequentially coded programs. Algorithms have been proposed to recognize parallelism between program blocks (macro-parallelism) or within a single or a collection of program statements (micro-parallelism). Programming techniques which constrain problem solutions to a sequential order have merely hidden any inherent parallelism in these solutions. Either this inherent parallelism can be detected through algorithmic techniques or new programming techniques can be introduced which relax the sequential constraints thereby allowing the programmer to re-introduce the inherent parallelism in a problem solution and use it to develop algorithms which are faster. However, to exploit this parallelism a general sequencing mechanism has to be developed which assigns program blocks to the processing units and controls the interaction between them.

3.4.1 Multiprocessor Scheduling - Assignment and Sequencing

Assignment and scheduling problems arise whenever a set of objects must be processed by a set of available facilities, and where the goal is to distribute the objects to the facilities in a most effective manner, i. e. , the questions are: "Which object should be processed by which facility and in what order?" and "In what order should each facility process the objects assigned to it?". Such problems arise in particular, in the area of operations research where they are called assignment problems, job-shop sequencing problems, and assembly line balancing problems. Such problems also arise in making optimal use of a multi-processing system.

Historically, assignment and sequencing problems were first formulated for solving job-shop problems. Application to multi-programming and multiprocessing has been considered only recently, and most of such applications are attempts to use formulations and solution procedures developed earlier for the job-shop sequencing problems. Another related problem is that of project scheduling which allows multiple resource constraints. A single project is to be undertaken, consisting of a large number of individual jobs. Every job has a set of immediate predecessors which must be completed before it can be started. Thus, some pairs of jobs can progress simultaneously, while other pairs must be done serially. How can the progress of the project be monitored so that the due date of the project is met, if it is

possible to do so? Here the principle tool is the "critical path analysis" of the project graph. [18]

A comprehensive review of the job-shop sequencing up to 1960 is given by Sisson [19] . Later work can be found in the survey paper by Elmaghraby [20] . The books Industrial Scheduling [21] and Theory of Scheduling [22] contain extensive discussion of this problem; they list respectively, over 100 and over 200 references to works on scheduling. A specific scheduling problem can usually be described by the following types of information:

1. The jobs and operations to be performed. Using our terminology, a job is identified with a computation and an operation with a process. The partial ordering of the operations that comprise the job, the processing times of the operations, and the machines or resources required to process each operation are also given as part of the problem description . Problems differ in the number of jobs that are to be processed and the manner in which jobs arrive.
2. The number and types of machines (resources) that comprise the shop.
3. Disciplines that control the manner in which assignments can be made and the flow pattern between the machines .
4. The criteria by which a schedule will be evaluated.

Among the current methods being used to attack scheduling problems are loading rules, heuristic rules, integer linear programming, complete or partial enumeration techniques, and finally stochastic queueing theoretic and simulation techniques. Loading and heuristic rules specify a discipline to decide which of the large number of jobs waiting to be processed should be worked on next. Some of these rules may choose to work on the job that arrived first, the job that arrived last, the job that can be completed most quickly, or the job that has the maximum number of uncompleted operations still to be performed. Selection procedure may even be completely random.

The job-shop scheduling problem can be modeled as an integer programming problem. In order of increasing simplicity and compactness the models are due to Bowman [23], Wagner [24] and Manne [25]. None of these authors claim that the formulations are computationally practical but they are theoretical formulations suitable for further study, especially in light of recent developments in integer programming. The other approach to the problem that is guaranteed to give an optimal answer is complete enumeration, which has been studied by B. Giffier and G. L. Thompson [26]. The enumerative procedure has been improved by some of the recent work of Schrage [28]. Schrage also presents a branch and bound method for implicitly enumerating all schedules and determining the optimum based on his enumerating scheme.

Unfortunately there are not extensive theoretic results available

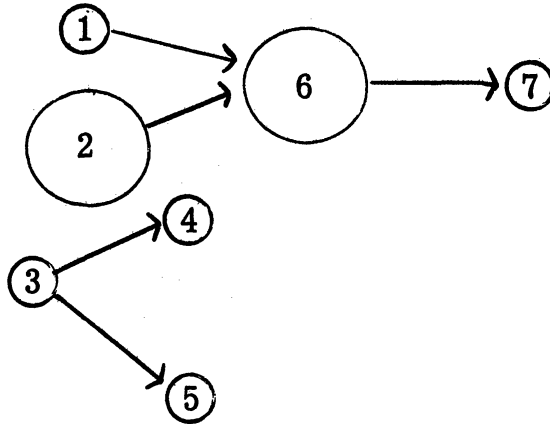
for queueing networks in which there are arrangements of multiple-servers. The most general result is the decomposition theorem due to Jackson [27] . In essence, the Jackson result is that (1) if the input to the shop is Poisson, (2) if the routing of jobs is determined by a probability transition matrix, (3) if the processing rates are exponential and, (4) if the dispatching rule is independent of a job's routing and processing times, then the network of queues may be treated as an aggregation of independent queues. A case of tandem queues, or queues in a series has been studied by Kleinrock [29] . Multiple channel or parallel channel queues having a number of identical servers working in a parallel are an important special case of the generalized birth-death process with Poisson input. Some interesting variations of parallel channel models in which there are special purpose processors, processors with different service rates and so-called cooperative processors have also appeared in literature [30] .

So far we have discussed the scheduling problem in a very general manner without any special emphasis to the multi-processor computer systems. In the remainder of this section we shall examine a few studies which are directly relevant to such systems. The major point of departure in such studies is the relative ease of preemption and the preempt resume disciplines are especially important. We shall sub-divide our discussion depending upon the kind of the demand environment of the system.

3.4.1.1 Hard Real-Time Environment

In hard real-time environment the processes constituting a job have a set of rigid timing bounds associated with them. Each process may have a definite starting and completion time. There is also the implicit assumption that the programs in such an environment are available for analysis and measurements prior to execution. Many of the real-time and production run computations fall into this category.

Properties of one of the simplest control structure applicable to such an environment have been studied by Graham [31] and Manacher [17]. Their essential idea is as follows. Programs are divided into segments called tasks (process); each task is executed by a single processor. A program consists of one or more tasks; these tasks are linked together by a precedence relation which specifies the essential ordering among tasks. They further require that there are no loops of tasks; that is all programming loops occur within tasks and not over tasks. The precedence structure is therefore that of a partially ordered set. Figure 3.4.1 is an example of such a task set. Nodes correspond to the task and the directed arcs correspond to the precedence relation. Thus task 6 can be started only after tasks 1 and 2 have been completed. Their control structure consists of a task list which is shown as a table in Figure 3.4.2.



A Typical Task Set

Figure 3.4.1

Corresponding to each task in the task set, there is an entry in the table containing four fields. The A field, called the absolute enable bit field tells, at any time, which tasks are ready for execution. The B field, or enable decrement field, tells how many of the tasks just prior to the task in question will have to be done. The C field, or successor field, is a list of tasks which immediately follow the task in question. Finally, the D field is a pointer to the program in storage corresponding to the task.

A	B	C	D
1	0	6	Pointer to T ₁
1	0	6	Pointer to T ₂
1	0	4,5	Pointer to T ₃
0	1	-	Pointer to T ₄
0	1	-	Pointer to T ₅
0	2	-	Pointer to T ₆
0	1	-	Pointer to T ₇

Initial Configuration of Task List

Figure 3.4.2

The procedure for maintaining control is quite simple. A free processor enters the list and scans it from the top down, looking for the first entry with the A field bit 'up' denoting readiness of the corresponding task for execution. It then 'lowers' that bit and enters the program specified in the D field. Upon completion of a task, it looks in the C field of the task. The B field of every task mentioned in this C field is decremented by 1, so that track of the antecedent programs yet to be done is kept. When the B field of any entry reaches 0, the flag in the A field is put up, denoting that the task so flagged is ready for execution. Programs are terminated when no flag in the A field of any task is up, and no processor is busy.

Note that the procedure outlined above would work for an arbitrary precedence relation and even if the execution times of the various tasks are unknown.

Changing the order of the entries in a task list effects the run time of the program. No algorithm is given to find the optimal task list (in the program execution time sense) but a bound on the variation in the total program run time over arbitrary variations in the list order has been given by Graham [31] . For n processors, the run time can vary by a factor of $(2 - \frac{1}{n})$.

Varying the list order is not the only way which can cause a peculiar increase in total run -time. Graham considers four types of changes each of which are capable of such increases:

1. Changing the order of tasks in the list.
2. Removing some of the precedence relations.
3. Changing the number of processors, either increasing or decreasing it.
4. Shortening the run-time of some tasks.

Graham's contribution is a theorem which states that given two runs R and R', where R' is related to R by the application of any combination of the four conditions above, the corresponding run times ω and ω' are related by the bound

$$\frac{\omega'}{\omega} \leq 1 + \frac{n-1}{n'}$$

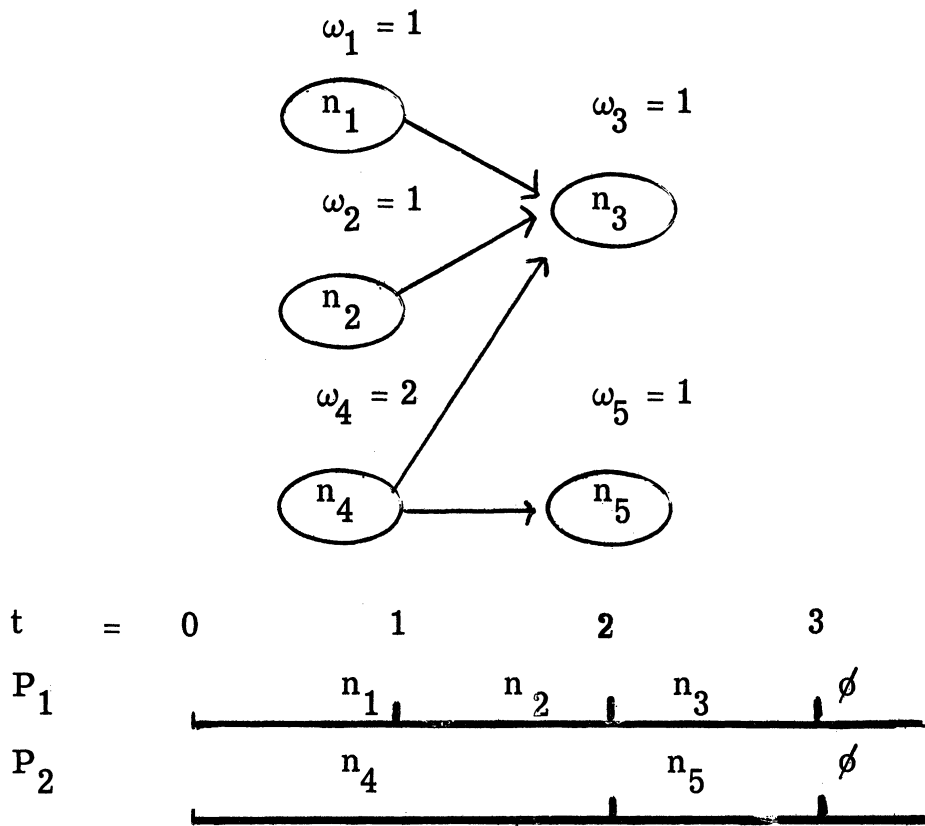
where n and n' are the number of processors in runs R and R' respectively.

This bound may be arbitrarily closely approached when any one of cases 1 through 4 is applied and it is the best possible.

Manacher [17] has considered the problem of the possible increase in the program run-time caused by shortening the run-time of some tasks. His result consists of a stability algorithm which removes the above anomaly by the addition of a modest number of additional precedence constraints.

As illustrated in Figure 3.4.1, a weighted directed graph may be associated with a task set. The nodes in the computation graph correspond to the processes and the directed arcs correspond to the precedence constraints. A weight is associated with each node of the graph and is a function of the execution time of the corresponding process. Thus, in graph theoretic terminology, the tasks sets considered above correspond to acyclic, weighted, and directed graphs. Informally, a schedule or assignment for k processors and a given computation is a description of the work done by each processor as a function of time. Of course the schedule must not violate the given precedence constraints. The simplest way of specifying a schedule is to use a Gantt chart, which consists of a time axis for each processor with intervals marked off and labeled with the name of the process being computed. An example of a computation

graph and a schedule for two processors is given in Figure 3.4.3. Symbol Φ is used to represent an idle period.



An Example Graph and Schedule
Figure 3.4.3

The general problem of efficiently constructing optimal schedule for an arbitrary number of processors (in minimum time to completion sense) and for an arbitrary acyclic computation graph whose nodes have arbitrary weights is still unsolved. Some efficient algorithms are known under weaker sets of conditions. When the computation graph is a rooted tree and the node weights are equal, a simple algorithm, due to Hu[32] , for constructing a shcedule for arbitrary number of processors is avaiable. This result has recently been extended to the case of computation trees with mutually commensurable node weights when processor preemption

is allowed (with no cost) [33] . An algorithm for constructing optimal schedules for two processors from an arbitrary acyclic computation graph with equal node weights has also been presented recently [34] . More sophisticated models describing program behavior would include, for example, conditional branching and cycles. For these more general models only heuristic scheduling techniques have proven fruitful.

3.4.1.2 Soft Real-Time Environment

In this environment, there is no *a priori* knowledge of the processing times of the incoming jobs. Usually there is no fixed priority among the jobs. With regard to the processor scheduling, all of these systems operating under this demand environment tend to have 'reasonable' response to user's requests and small variations in response are tolerated.

In spite of the current existence of the multiprocessor computer systems operating under soft real-time environment, very little work has been done on modeling and analyzing such systems. The only point of departure has been the 2-processor round-robin model studied by Coffman [35] . For the case of 2 processors, Coffman extends the RR service discipline as follows. Whenever there are two or less programs in the system these programs are allowed to run to completion or until the number in the system grows to three. As soon as there are three programs in the system a quantum is imposed simultaneously on each processor as in the

single processor case. When a quantum expires before a program completion, one of the processors is loaded with a new program after the interrupted program is returned to the end of the queue. It is assumed that the interrupted program is the one which has had the longest uninterrupted operation up to the time of the quantum expiration. Ignoring the swap and overhead times, Coffman has analyzed this model using the concept of the imbedded Markov chain with the program completions or quantum expirations as the points of regeneration. Expression for the steady state probabilities of the number in the system and the expected value of the number in the system were obtained. The analysis of the waiting times was not carried out.

In view of the tremendous investments required to implement large scale multiprocessor computer systems, the field of modeling and analyzing such systems is an active one. For the case of hard real-time environment, a generalized model is presented in the next section and the optimal scheduling problem dealing with determining when a process should be executed and how many processors should be applied to it is formulated using 0-1 programming approach. Future work will be concentrated in finding efficient solution procedures. Analysis and optimization of models of multiprocessor systems operating in soft real-time environment will be carried out. The optimal allocation of the processors to the various queues in the model will be of interest.

3. 4. 2 Scheduling Parallel Processes - A Zero-One Programming Approach

We consider systems in which all processors are identical and assume that the computations submitted to the system have been specified as a set of processes and two binary relations defined on this set. The first of these relations, called the precedes relation, is denoted by \ll and it specifies a partial order on the set of processes with the following interpretation. If p_i and p_j are processes and $p_i \ll p_j$ (read p_i is immediate predecessor of p_j or, equivalently, p_j is immediate successor of p_i), then process p_i must be completed before process p_j can be started and there is no process p_1 such that $p_i \ll p_1 \ll p_j$. The second relation, called commutativity relation, and denoted by C specifies pairs of processes which can be executed in either order but not concurrently. Commutativity is a necessary but not sufficient condition for parallel processing of two processes. There may exist, for instance, two processes which can be executed in either order but not in parallel. For example, the inverse of a matrix A can be obtained in either of the two ways shown below.

(1)

a) Obtain transpose of A

b) Obtain matrix of cofactors of the transposed matrix

c) Divide result by determinant of A

(2)

a) Obtain matrix of cofactors of A

b) Transpose matrix of cofactors

c) Divide result by determinant of A

Thus, obtaining the matrix of cofactors and the transposition operations are two distinct processes which can be executed in alternate order with the

same result. They cannot, however, be executed in parallel.

By associating a weight with each process, a function of the execution time of the corresponding process, an optimization problem of interest is to determine a schedule for the given computation (set of processes) for J processors such that the total run-time of the computation is minimized. Except for a few special cases, no general solution method is available. The assumption of the partial ordering among the set of processes implies that all conditional branching and cycles in the computation occur within the processes and not over the processes. This is reasonable if the processes represent large functional blocks so that the model is a coarse description of the computation. However, a certain amount of 'fine-structure' may not appear in the original process definition. Parallelism within a process can also exist when individual components of a compound process can be executed concurrently, and in fact, a particular process might itself be represented by a model whose characteristics are identical to those discussed above. For this reason, the model is generalized by associating with each process a set of mutually exclusive alternatives, only one of which must be performed. These alternatives specify for each process a function of the execution time when different number of processors are applied to it. It may also be noted that the processors are not the only resources of interest in a multiprocessor computer system. Due consideration should also be given to memory and other peripheral equipment. With these considerations the generalized model is presented in the following section.

3.4.2.1 Model and Notation

Consider a multiprocessor computer system with

J identical processors,

M units of memory,

and R_k amount of type k resources; $k=1, 2, \dots, K$.

We assume that a computation P submitted to the system is specified as the following tuple.

$$P = \langle P_1, P_2, P_3, \dots, P_N; \ll; C \rangle$$

where

P_i is the set of process alternatives associated with process i ,

$i=1, 2, 3, \dots, N$, and is given by the following:

$$P_i = \{ (p_{ij}, b_{ij}, a_i, e_i, d_i, m_i, \text{ and } r_{ik}) / j \in J_i \subseteq \{1, 2, 3, \dots, J\} \}$$

p_{ij} denotes the i th process when j processors are applied to it

b_{ij} = number of time periods required to complete process i with j processors

a_i = period before which process i cannot be started

e_i = earliest possible period by which process i could be completed

d_i = desired due date for process i

m_i = amount of memory required by process i

r_{ik} = amount of type k resource required by process i

\ll is the precedences relation defined on $\{P_{ig} / i=1, 2, \dots, N; \text{ and } g \in J_i\}$ with

the property that if $p_{mg} \ll p_{nk}$ athen $\forall \chi \in J_m$ and $\forall y \in J_n$ $P_{m\chi} \ll P_{ny}$

C is the commutativity relation defined on $\{p_{ig}/i=1, 2, \dots, N\}$ and $g \in J_i\}$ with the property that if $P_{mg} C P_{nk}$, then $\forall \chi \in J_m$ and $\forall y \in J_n$, $P_{mx} C P_{ny}$.

3.4.2.2 Assumptions and Limitations

- 1) No process preemption is allowed.
- 2) The execution times are mutually commensurable i.e., there exists a real number, w , such that all execution times are integer multiples of w . This is not very restrictive since we can still approximate a given set of execution times arbitrarily close.
- 3) All the cycles and conditional branching occur within the processes. This may be reasonable if the processes represent large functional blocks and the model is a coarse description of the computation. We may also note that any particular execution of a computation containing additional branches and cycles can be represented as a partially ordered set of processes.
- 4) The execution time of any process does not change if additional resources besides processors (tape, drives, memory, etc.) are devoted to it. In other words each process requires m_i and r_{ik} units of memory and type k

resource, and providing additional resources to the process does not change its processing time requirements.

The deterministic constraint b_{ij} associated with process p_{ij} can have several interpretations. The interpretation of $\{b_{ij}\}$ really determines the meaning of the schedule constructed. If b_{ij} represents the maximum execution time of the process, then, with certain precautions [17] , the schedule length may be taken as the maximum execution time of the computation. In this case minimizing the schedule length (make-span) corresponds to minimizing the maximum execution time of the computation. This interpretation is particularly useful for hard real-time problems with fixed dead lines. A second interpretation is that b_{ij} represents the mean value of the process execution-time considered as a random variable. Constructing a minimum length schedule in that case is a heuristic procedure for minimizing the mean execution time of the computation.

The scheduling problems considered here deal with determining when a process should be processed, and how many processors should be devoted to it. In the following formulation 0-1 variables are used to indicate for select periods whether or not a process is completed in those periods.

3.4.2.3. Formulation of the Optimization Problem

Determining the schedule depends upon the desired objective.

The following are considered here.

- a. Minimize total throughput time for all processes.
- b. Minimize the time by which all processes can be completed (i.e., minimize make-span).
- c. Minimize the cost of resources required to complete all processes.

Using the following 0-1 variables, the objective functions are formulated and equations are developed to insure that constraints (precedence relations, limited resources, etc.), when they are imposed, are met. Let t = time period, $t = 1, 2, \dots, \max d_i$

x_{ijt} = a variable which is 1 if process i is completed in period t when j processors are devoted to it; 0 otherwise. x_{ijt} need not be treated as a variable in all periods, since it equals 0 for $t < e_i$ and for $t > d_i$. x_t = a variable which is 1 if all processes have been completed by period t (i.e., completed in or before period $t - 1$); 0 otherwise.

3.4.2.4 Objective Functions

Processes are to be scheduled in a manner that optimizes

some measures of performance subject to certain requirements and limitations. The choice of an appropriate objective function may differ for different environments. Several common ones are selected for explicit formulation.

a. Minimizing Total Throughput Time

Individual process throughput time is defined as the elapsed time between the time that the process can be started and the process completion time. Throughput time for process i is

$$\sum_{j \in J_i} \sum_{t=e_i}^{d_i} tx_{ijt} - a_i$$

Therefore, the objective function for minimizing the sum of the throughput time for all processes is

$$\text{Minimize } z = \sum_{i=1}^N \left[\sum_{j \in J_i} \sum_{t=e_i}^{d_i} tx_{ijt} - a_i \right] \quad (3.1)$$

Minimizing throughput time for a single process is equivalent to maximizing the number of periods remaining after the process is completed. Hence, we have the equivalent objective function.

$$\text{Maximize } z' = \sum_{i=1}^N \left[d_i - \sum_{j \in J_i} \sum_{t=e_i}^{d_i} tx_{ij t} \right] \quad (3.2)$$

b. Minimizing Make-Span

An alternative objective function is to minimize the time by which all processes are completed, i. e., minimize make-span. This corresponds to

$$\text{Maximize } z = \sum_{t = \max e_i}^{\max d_i} x_t \quad (3.3)$$

c. Minimizing The Cost of Resources

Let

c_P be the cost associated with the use of one processor per unit of time,

c_M be the cost of using a unit of memory per unit of time, and

c_{R_k} be the cost of using a unit of type k resource per unit of time.

In any given period t the process i is being executed with j processors if it is completed in period ℓ (i. e., $x_{ij\ell} = 1$) where $t \leq \ell \leq t + b_{ij} - 1$. Let N_{Pt} denote the total number of processors being used in any period t. Then

$$N_{Pt} = \sum_{i=1}^N \sum_{j \in J_i} \sum_{\ell=t}^{t+b_{ij}-1} x_{ij\ell} \quad (3.4)$$

(t = min $a_i, \dots, \max d_i$).

Similarly

$$N_{Mt} = \sum_{i=1}^N \sum_{j \in J_i} \sum_{\ell=t}^{t+b_{ij}-1} m_{ij\ell} \quad (3.5)$$

(t = min $a_i, \dots, \max d_i$)

$$\text{and } N_{R_k t} = \sum_{i=\ell}^N \sum_{j \in J_i} \sum_{\ell=t}^{t+b_{ij}-1} r_{ik} x_{ij\ell} \quad (3.6)$$

(t = min a_i, ..., max d_i; k=1, 2, ..., K).

where

N_{Mt} = total amount of memory occupied by processes which are executing in any given period t.

and

$N_{R_k t}$ = total amount of type k resource being used by processes which are executing in any given period t.

From the above equations the total cost of executing all processes is given by

$$C_T = \sum_{t=\min a_i}^{\max d_i} \{C_P N_{Pt} + C_M N_{Mt} + \sum_{k=1}^K C_{R_k} N_{R_k t}\} \quad (3.7)$$

3.4.2.5 Constraints

The above objective functions are to be maximized or minimized subject to the following constraints and environmental limitations.

a. Process Completion

Out of all the alternatives associated with each process, one and only one of them must have exactly one completion period.

$$\sum_{j \in J_i} \sum_{t=e_i}^{d_i} x_{ijt} \leq 1 \quad (3.8)$$

i = 1, 2, ..., N.

b. Completion of All Processes

Definition of the variable x_t requires that x_t be zero until all the processes have been completed. This requirement can be written as

$$x_t \leq \frac{1}{N} \sum_{i=1}^N \sum_{j \in J_i} \sum_{t=e_i}^{d_i} x_{ijt} \quad (t = \max e_i, \dots, \max d_i)$$

c. Resource Constraints

In any give period, the number of processors, the amount of memory and other resources cannot exceed the amount available. Therefore,

$$N_{Pt} \leq J \quad (t = \min a_i, \dots, \max d_i) \quad (3.9)$$

$$N_{Mt} \leq M \quad (t = \min a_i, \dots, \max d_i) \quad (3.10)$$

and

$$N_{R_k t} \leq R_k \quad (t = \min a_i, \dots, \max d_i; k=1, 2, \dots, K) \quad (3.11)$$

d. Sequencing

A sequencing constraint is required when a process cannot be started until one or more other processes have been completed. For example, assume $p_{lj} \ll p_{mk}$. If t_{lj} and t_{mk} denote the completion times of processes p_{lj} and p_{mk} , then

$$t_{lj} + b_{mk} \leq t_{mk}$$

Now since $t_{lj} = \sum_{t=e_l}^{d_l} tx_{lj}$, and $t_{mk} = \sum_{t=e_m}^{d_m} tx_{mkt}$, the appropriate sequ-

encing constraint becomes

$$\sum_{t=e_\ell}^{d_\ell} x_{\ell jt} + b_{mk} \leq \sum_{t=e_m}^{d_m} x_{mkt} \quad (3. 12)$$

e. Commutativity

A commutativity constraint on the processes $p_{\ell j}$ and p_{mk} ensures that they must not be performed simultaneously, but permits them to be performed in any order. Process $p_{\ell j}$ is being performed in period t if and only if

$$\sum_{q=t}^{t+b_{\ell j}-1} x_{\ell j q} = 1,$$

and similarly for process p_{mk} . Thus the desired constraint is

$$\sum_{q=t}^{t+b_{\ell j}-1} x_{\ell j q} + \sum_{q=t}^{t+b_{mk}-1} x_{mkq} \leq 1 \quad (3. 13)$$

($t = \max\{e_\ell, e_m\}, \dots, \min\{d_\ell, d_m\}$)

Reference For Section 3

1. W. S. Jewel, "Markov Renewal Programming", Parts I and II, Operations Research, Vol. 11, 938-971 (1963).
2. R. Bellman, "A Markovian Decision Process", Journal of Mathematics and Mechanics, Vol. 6, No. 5, 1957, pp. 679-684.
3. R. A. Howard, Dynamic Programming and Markov Processes, MIT Press, Cambridge, 1960.
4. R. Bellman, Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1957.
5. D. W. Fife, "The Optimal Control of Queues, With Applications to Computer Systems", Technical Report No. 170, Cooley Electronics Laboratory, The University of Michigan, October, 1965.
6. M. T. Alexander, "Time Sharing Supervisor Programs", Computing Center, The University of Michigan, May 1969.
7. T. B. Pinkerton, "Program Behavior and Control in Virtual Storage Computer Systems", Technical Report 4, CONCOMP, The University of Michigan, April 1968.
8. T. B. Pinkerton, "The MTS Data Collection Facility", Memorandum 18, CONCOMP, The University of Michigan, June 1968.
9. R. J. Pankhurst, "Program Overlay Techniques", Comm. ACM, 11, February 1968, pp. 119-125.
10. C. V. Ramamonthy, "The Analytic Design of a Dynamic Look Ahead and Program Segmenting System for Multiprogrammed Computer", Proc. ACM 21st National Conference, August 1966, 11. 199-239.
11. Information Incorporated, "Program Paging and Operating Algorithm," Technical Documentary Report, TRGB - 793- 1, September 1968.
12. E. L. Lawler, K. N. Levitt, and J. Tunes, "Module Clustering to Minimize Delay in Digital Network", IEEE Trans. on Computers, Vol. c-18, No. 1, January 1969, pp. 47-57.
13. H. Everett, "Generalized Lagrange Multiplier Methods for Solving Problems of Optimum Allocation of Resources", Oper. Res. 11, May-June pp. 399-417.

14. B. Kerningher, "Some Graphical Partitioning Problem Related to Program Segmentation", PhD Thesis, Princeton, January 1969.
15. K. B. Irani, J. W. Boyse, et. al. "A Study of Information Flow in Multiple-Computer and Multiple-Console Data Processing Systems", RADC, Griffiss Air Force Base, New York, RADC - TR- 70-87, May 1970.
16. J. W. Boyse, "Solution of Markov Renewal Decision Processes With Application to Computer System Scheduling", Systems Engineering Laboratory Technical Report No. 52, The University of Michigan, Ann Arbor, to be published.
17. G. K. Manacher, "Production and Stabilization of Real-Time Task Schedules", JACM, vol. 14, no. 3, July 1967, pp. 439-465.
18. Andre F. Block, "Critical Path Resource Allocation for Job Shop Scheduling", Proc. of AIIE, 1968, pp. 183-187.
19. R. L. Sisson, "Method of Sequencing in Job-Shops - A Review", JORA, Jan. 1959, pp. 10-29.
20. S. E. Elmaghraby, "The Machine Sequencing Problem - Review and Extensions", Naval Research Logistics Quarterly, vol. 15, no. 2, June 1968.
21. J. F. Muth and G. L. Thompson, Industrial Scheduling, Prentice-Hall, Englewood Cliffs, New Jersey, 1963.
22. R. W. Conway, W. L. Maxwell, and L. W. Miller, Theory of Scheduling, Addison-Wesley, Reading, Mass. 1967.
23. E. H. Bowman, "The Scheduling Sequencing Problem", Operations Research, vol. 7, 1959, pp. 621-624.
24. H. M. Wagner, "An Integer Linear Programming Model for Machine Shop Scheduling", Naval Logistics Research Quarterly, vol. 6, 1959, pp. 131-140.
25. A. S. Manne, "On the Job-Shop Scheduling Problem", Operations Research, vol. 8, 1960, pp. 219-223.
26. B. Giffler and G. L. Thompson, "Algorithms for Solving Production Scheduling Problems", Operations Research, vol. 8, 1960, pp. 487-503.

27. J. R. Jackson, "Job Shop-Like Queueing Systems", Research Report 81, Management Sciences Report Project, UCLA, Jan. 1963.
28. L. Schrage, "Solving Resource-Constraint Network Problems by Implicit Enumeration - Non Preemptive Case", Operations Research, vol. 18, no. 2, 1970, pp. 263-278.
29. L. Kleinrock, "Sequential Processing Machines (S. P. M.) Analysed with a Queueing Theory Model", JACM, vol. 13, April 1966, pp. 179-193.
30. Thomas L. Saaty, Elements of Queueing Theory McGraw-Hill, New York, 1961.
31. R. L. Graham, "Bounds for Certain Multiprocessing Anomalies, Bell System Tech. Journal, vol. 45, 1966, pp. 1563-1581.
32. T. C. Hu, "Parallel Sequencing and Assembly Line Problems", Operations Research, vol. 9, no. 6, 1961, pp. 841-848.
33. R. R. Muntz and E. G. Coffman, "Pre-emptive Scheduling of Real-Time Tasks on Multiprocessor Systems", JACM, vol. 17, no. 2, 1970.
34. M. Fuji, et. al. , "Optimal Sequencing of Two Equivalent Processors", SIAM J. of App. Math. , vol. 17, no. 4, July 1969.
35. E. G. Coffman, "Stochastic Models of Multiple and Time-Shared Computer Operations", UCLA Eng. Report no. 66-39, June 1966.
36. D. Coleman, "On Binding Groups - A Quadratic Programming Approach in Zero/One Variables with Applications", SEL Tech. Report No. 56, The University of Michigan, Ann Arbor, to be published.

4. CENTRAL PROCESSOR DESIGN

This section focuses research into the design and optimization of digital computer central processors. The general study areas are:

- 1) Selection of data paths in the CPU where the term data path refers to the set of hardware logic units and the traffic between them. (Section 4.1)
- 2) The second study area is concerned with the optimal design of micro-programmed computers. A general design methodology has been formulated and is described briefly in Section 4.2.

4.1 Data Path Optimization

The problem under consideration here is that of formalizing the design of a digital computer and developing algorithms and procedures which can be applied in optimizing the design of the central processor. Specifically, it is the problem of creating the optimum data path for a central processing unit when the system architecture is given. The term data path is used here to refer to a set of hardware logic units such as registers, adders, and counters and the interconnections between them for data transfers. System architecture means a description of the computing system as it appears to the programmer and is composed of definitions of such items as data and instruction word formats, addressing and indexing structure, and the operation of each instruction. Optimization requires minimizing the total system cost subject to a constraint on the weighted average instruction execution time.

4.1.1 Example

The concepts involved here can be made clear through a simple example. Assume the system description of a central processing unit includes the definition of two registers, R1 and R2, which are accessible to the programmer and an instruction which swaps the contents of R1 and R2. Many different arrangements of hardware logic units can be used to implement this instruction and which will result in different hardware costs for the computer and different execution times for the instructions.

Figure 4.1 shows a data path having one additional register, T1, which is not accessible to the programmer and an adder. The instruction can be executed on this data path in three steps:

- 1) $R1 \rightarrow \text{ADDER} \rightarrow T1$
- 2) $R2 \rightarrow \text{ADDER} \rightarrow R1$
- 3) $T1 \rightarrow \text{ADDER} \rightarrow R2$

Figure 4.2 shows a data path which has an additional shifting unit. This allows the instructions to be executed in just two steps as follows:

- 1) $R1 \rightarrow \text{ADDER} \rightarrow T1$
- 2) $T1 \rightarrow \text{SHIFTER} \rightarrow R2$ and $R2 \rightarrow \text{ADDER} \rightarrow R1$

Finally in Figure 4.3 a new path has been introduced connecting R2 to the shift unit. This allows the instruction to be performed in just one step:

- 1) $R1 \rightarrow \text{ADDER} \rightarrow R2$ and $R2 \rightarrow \text{SHIFTER} \rightarrow R1$

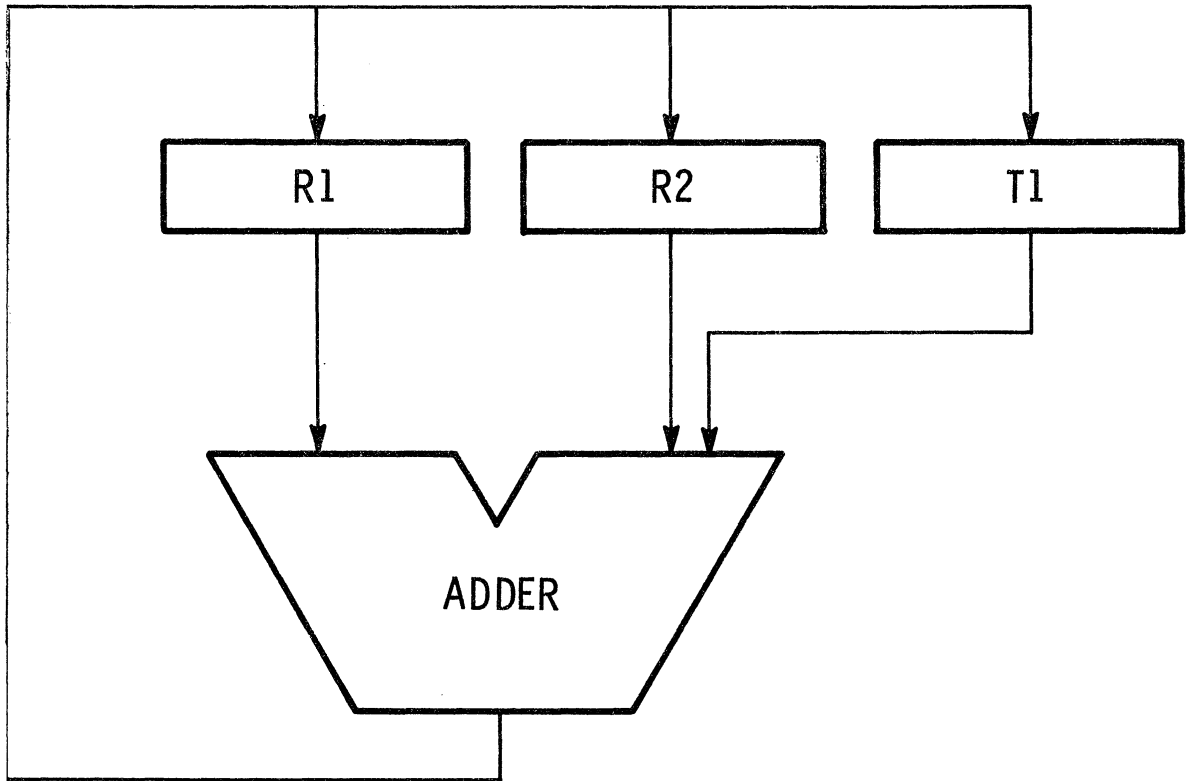


Figure 4.1 CPU Data Path I

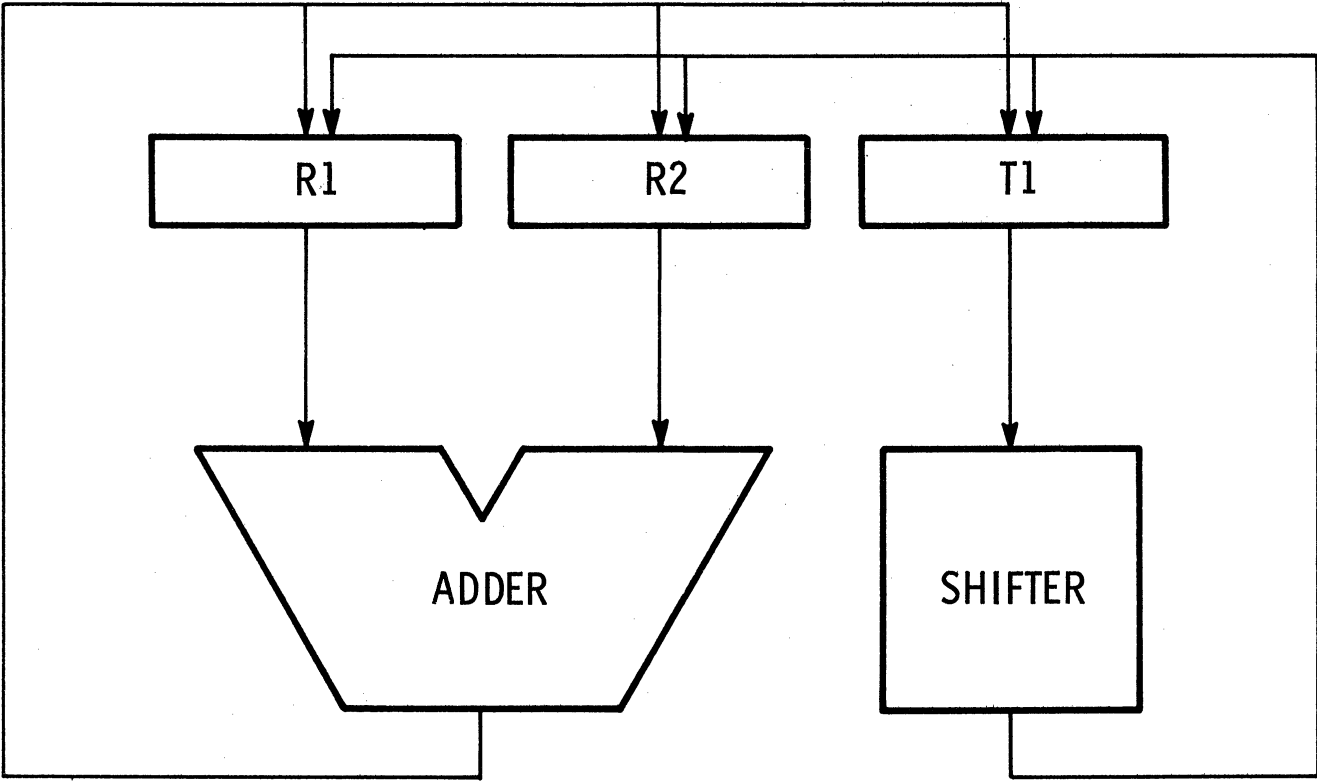


Figure 4.2 CPU Data Path II

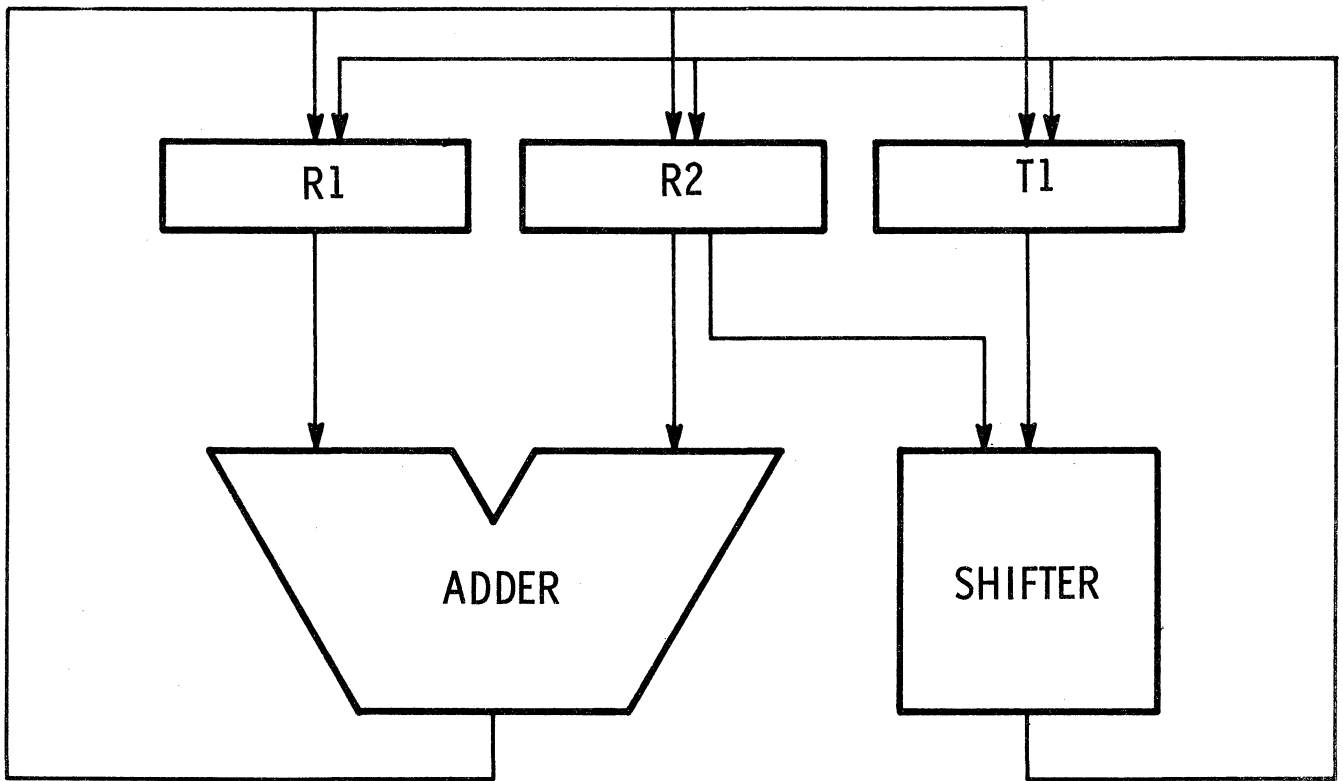


Figure 4.3 CPU Data Path III

Clearly each of these data paths has a different cost and allows different execution times for the instruction. The problem of determining which of these is optimum is solved by finding the minimum cost data path among those which satisfy the required minimum performance.

Even for a computer with a very small instruction set the number of different paths which can be used becomes large and the cost of evaluation requires a detailed design study of each one. As a result, the normal solution of this problem involves a considerable amount of intuitive judgment on the part of the designer since it is just not possible to carefully consider all of the possibilities. The objectives of the research on this problem are to gain an understanding of the relationships between the definition of the instruction and the design of the optimum data path to implement it, and to provide algorithms which will allow the computer designer to systematically explore the set of possible data paths to find the optimum one.

4.1.2 Model for the Study

The overall structure of a model being used in this study is shown in Figure 4.4. The architecture description is taken as input to the data path design process which uses a catalogue of hardware unit designs and a library of algorithms. The output consists of a description for a particular data path and the execution times for each architecture instruction on this data path.

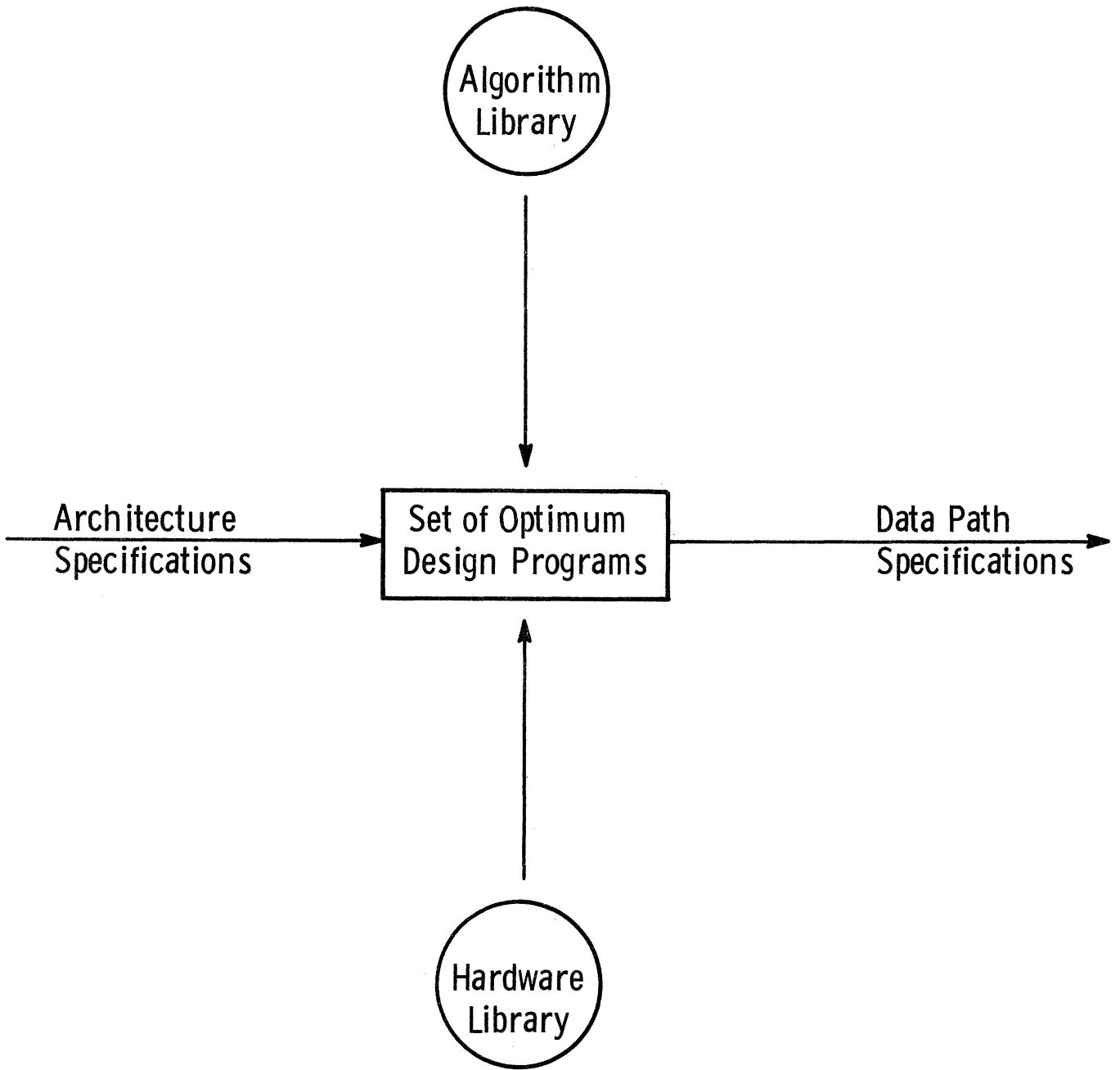


Figure 4.4 Overall Structure of Model

4.1.2.1 Model Language

In the model used in this study a language is needed to describe the transformations produced on data by the instructions, hardware units, and algorithms. That is, a language is required so that these three aspects of the model can be conveniently defined. A language which permits all three of these functions to be treated by the same grammar has been developed and will be described briefly here.

The basic unit of the language is an expression built of operator and operand names which define some new value. This expression is set equivalent to an operand name, thus forming a statement and indicating replacement of the value of this operand by the value of expression. All operators are written to the left of their operand and the operands are enclosed in parenthesis and separated by commas.

For example, if R1, R2, and R3 are the names of three programmable registers in the architecture, and + is the operator of binary addition, then the expression

$$+ (R1, R2)$$

indicates the addition of the contents of registers R1 and R2, and the statement

$$R3 \leftarrow + (R1, R2)$$

is interpreted to mean that the register, R3, is loaded with the binary sum of the contents of registers R1 and R2.

In this language a set of statements is used in describing an instruction,

algorithm, or a hardware unit transformation. With each set of statements, a partial ordering is defined to allow any desired time sequential relationship between the statements to be specified. The intention here is to require a definition of a time sequential ordering only between those statements for which it is essential and allow the remaining statements to stand in an unspecified time relationship to each other.

Although the grammar is uniform for the three portions of the model, it is necessary in each of them to restrict the set of operators and operands permitted in statements. To describe this we will define $e [X, Y]$ to be any expression in this language such that operand names are in the set X and operator names are in the set Y . Also we will write

$$Z \leftarrow e [X, Y]$$

to mean any statements such that the operand which the expression replaces is in the set Z . The sets of operand and operator names which are useful in the various model sections will be defined in the later sections of the model description.

4.1.2.2 Model Architecture

The system architecture is specified as

$$A = \langle R, P, Q, \mathcal{I}, \Phi \rangle$$

where

- R = set of programmable registers
- P = set of input busses
- Q = set of output busses
- \mathcal{I} = set of instructions
- Φ = maximum allowable value of weighted average instruction time.

Each instruction $I \in \mathcal{I}$ is defined as

$$I = \langle E, \rho, \tau, \sigma \rangle$$

where

$$E = \{R \cup Q \leftarrow e [R \cup P, S] \}$$

is a set of statements with S being the set of operators available for instruction definition.

ρ = a partial ordering on E

τ = maximum execution time for I

σ = weighting factor for I in the architecture

4.1.2.3 Hardware Unit Library

The Hardware Unit Library is a set of transformational units from which components are selected to form the data path. A unit in either the unit library or in a data path is defined as

$$U = \langle R', P', Q', F, \delta, c \rangle$$

where

R' = set of registers in the unit

P' = set of input ports to the unit

Q' = set of output ports from the unit

F = set of functions which the unit performs

δ = time delay of the unit

c = cost of the unit

A unit may be able to execute many different functions but the restriction is made that a unit can only do one function per cycle. So the selection of a function for one cycle corresponds to the selection of that unit's micro-order

for a cycle in a microprogrammed computer. Each function F is defined as

$$F = \{ Q' \leftarrow e [R' \cup P', S'] \}$$

where S' is the set of operators available for the description of hardware transformations. No partial ordering is required or permitted here since all statements in a function are assumed to be executed simultaneously.

4.1.2.4 Algorithm Library

The Algorithm Library $\mathcal{G} = \{G\}$ is a set of definitions which translate the operators in S into operators S' which are respectively the instruction and the hardware unit operators. $G_{s_i} = \{g\}$ is the set of all algorithms which translate $s_i \in S$. One such entry in the algorithm library for s_i is

$$g_{s_i} = \langle Z \leftarrow e [Z, s_i], \{ Z \leftarrow e [Z, S \cup S'] \}, \rho \rangle$$

where Z is a set of dummy operator names and ρ is again a partial ordering on the set of statements. An algorithm, then, is similar to a "macro" in the context of computer assembly languages. But the algorithm library differs from a macro library in that there may be more than one entry $g \in G$ for each operator. This allows optimization to proceed with respect to alternate algorithm choices.

4.1.2.5 Model of the Data Path

A data path, D , is simply a set of hardware units and a specification of the connections between the ports of these units.

$$D = \langle \{U\}, M \rangle$$

The inter-port connections are defined by a binary matrix, M , called the connection matrix. It has one row for each output port and one column for each input port in D . A one in the matrix designates a connecting buss from the corresponding output port to the input port.

A cycle in the data path is a transfer of information out of the set of registers, through the transformational units and back into the registers of D . The time for each individual transfer is the sum of the delays in each of the units through which it passes. For this model we assume a fixed length synchronous computer and let the duration of the cycle, T , also called the data path cycle time be the delay of the longest of such transfer path in D .

$$T = \text{Max}_{\substack{\text{all} \\ \text{paths} \\ \text{in } D}} \left[\sum_{\substack{\text{all} \\ u \\ \text{in path}}} \delta_u \right]$$

Finally the cost, C , of the data path D is the sum of the individual unit cost.

$$C = \sum_{\text{all } u \text{ in } D} c_u$$

4.1.2.6 Optimization Criteria

For any fixed data path we can find the cycle time, T , and then determine the minimum number of cycles, N_I , needed to implement each instruction I on this data path. From these we can compute the weighted average instruction execution time, Φ' , as

$$\Phi = T \sum_{I \in \mathcal{J}} N_I \sigma_I$$

An upper bound on this quantity is specified as part of the architecture definition. We will define the optimum data path to be that data path having minimum cost, C , for which $\Phi' \leq \Phi$.

4.1.2.7 Generality of the Model

The model which has been described in the previous section is obviously very general. One important facet of this generality is obtained by choosing to treat the operator sets S and S' as undefined, open-ended sets rather than to tie the model to a particular, perminate choice of operators. For a particular implementation of this model, of course, some specific set of operators would have to be selected.

Similarly the Hardware Unit Library and the Algorithm Library are open-ended and the success of an implementation of this model, that is, the ability to accurately compute an optimum data path, will vary with the richness of these libraries. But the model remains general here by not presupposing a particular set of entries.

4.1.3 Progress Toward a Solution

For some given hardware library and algorithm library there are a number of variables to be considered in arriving at the optimum data path for an architecture. There is the selection of algorithms to translate S into S' , a choice of the particular units to implement S' as well as the number of copies of each unit to be included in the data path. Also there is

the choice of an interconnecting set of busses to be made. Unfortunately this solution space is highly irregular and it is difficult to predict the overall change in Φ' produced by a small change in the data path.

For example, the deletion of one buss may significantly decrease T . But to determine the effect of on Φ' it is necessary to reevaluate N_I . For all I which use that buss N_I can not decrease, and for all other I , N_I will be unchanged. However,

$$\Phi' = T \sum_{I \in \mathcal{J}} N_I \sigma_I$$

may either increase or decrease and only an exact computation will determine the effect of that data path change.

In order to move toward an implementation of this model, the research effort has proceeded along three lines. One aspect has been to develop computer programs to implement selected portions of the model. Specifically programs to compute T and N_I have been developed in order to estimate the computer execution time needed for these computations. These two programs are at the core of the data path evaluation portion of the model and hence are basic to the solution of the general problem.

Another aspect has been to try to discover relationships between the many variables in the solution space in order to effectively reduce the number of variables that must be treated. This approach has been rather unproductive so far.

The final aspect has been to find some subset of the model for which the solution space is more regular or more restricted. Some success has

been achieved in this area and there is hope that a full implementation of the model subject to restrictions on the library entries can be realized.

4.2 Microprogram Control

The objective of the work done under this section of the contract is the optimum design of microprogram computers. A discussion of microprogramming and the inherent advantages and disadvantages associated with microprogramming were presented in last year's report. Since, last year an excellent book [1] on microprogramming by Hussin has appeared.

In the past year work has been done on three major subtasks. First, the general approach to the problem has been formulated. Second, the draft of an intermediate language has been completed. Third, an optimization model and algorithm for hardware selection has been developed. Each of the subtasks will be discussed in the following paragraphs.

4.2.1 The General Design Method

The general method of computer design we are suggesting is a two part method. This first part is the selection of the hardware components which make up the computer CPU. The second part is the preparation of the control programs which control the registers, data path, and transformation units of the CPU.

Everything evolves from the program type statements which form the intermediate language. The hardware selection is based on implementing the program types, in the most efficient fashion. What is done is to assign

to each program type a measure of its frequency of execution and its frequency of occurrence for a particular class of programs. The hardware design group specifies a base set of hardware and a group of hardware options and the cost of each. They also supply one or more (usually more) implementation methods for each program type. The different implementation methods for each program type use different sets of hardware options. What is done is to develop a cost performance curve. The curve specifies the fastest average program type execution time which can be achieved for a particular dollar expenditure on hardware. This allows the optimum selection of hardware options if a fixed amount of money has been allocated for hardware expenditures, or if a particular performance is required it gives the minimum amount of money which is required. It must be remembered that the performance figure is based on the estimated frequency of occurrence and the estimated frequency of execution.

The second part of the design problem is the development of the microprograms to control the CPU once the hardware options have been selected. The control program can be developed in two ways. The intermediate language can be implemented via microprograms and thus become the machine language, or, main programs can be described in terms of the intermediate language and the main program can be implemented directly via micro-code. The optimal selection of hardware assumed that programs written in the intermediate language are turned into micro-code. That is why the estimated number of occurrences of each program type is required as one

of the parameter input to the optimization program. What then must be developed is a micro-code compiler. The compiler will take programs written in the intermediate language and compile them into micro-code. The inputs into the micro-code compiler will be a description of the CPU, program types, and program types implementation. The output will be the micro-code required to emulate the program.

A description of the traditional method of computer design will be presented and compared with the computer design method presented in this report. Because the computer field is changing very rapidly and because there are many different groups of individuals designing computers, there are exceptions to what we described as the traditional approach to computer design. However, most of the computer systems in existence today were designed in a manner similar to what will be called the traditional approach.

The two methods arise from different hardware environments. The traditional approach developed when instructions were implemented via hardware circuitry. The approach we are advocating assumes instructions are implemented via micro-program control. Thus the cost constraints imposed upon the two methods are different.

The computer design methods differ in the choice and implementation of instruction sets. In the traditional approach, the computer is designed to implement a limited set of machine or assembly language instructions. The instruction set usually includes arithmetic, logical, branching, and I/O instructions. The implementation of more complicated operations

(for instance list manipulation or square root) is usually accomplished with a series of machine operations. The hardware implementation (assembly language) of more complicated instructions occur only if the instruction has extraordinarily high usage. The most obvious example of such "special" instructions are floating point instructions.

The more complicated instructions were implemented infrequently for two reasons. First, the cost of implementation was prohibitive. Second, the requirements of the user is not well known. The problem is one of measuring or in some fashion asserting what the user wishes to do. The user's real desires are often masked because he has been forced to modify his methods to conform to a limited instruction set.

In the method which we are suggesting, the first step is to try and determine what the user would truly wish to do. This decision is arrived at by studying higher level languages. Since higher level languages are designed specifically to allow a user to solve his problems in a natural fashion, the languages should represent a good measure of what people would like to do. In addition, attention will be paid to formal studies of compilers, sorting routines, graphic displays, etc. to determine what operations are required to implement these functions. The net result is an intermediate language which consists of a set of program types. Each program type is a function which a large set of users would like to have available. The set of program types covers the set of operations typically offered by an assembly language and much more.

The set of program types is not expected to be the ideal set or the final set of program types. It is expected that the set of program types will evolve and grow through feedback from users. Since program types are implemented via microprogram control, the modification and addition of program types should not be as difficult as it would be if they were implemented via hardware control.

The two approaches evolve as follows. In the traditional approach, a machine language is developed and a computer is designed to efficiently implement the machine language. However, the machine language is not powerful in the sense of performing in one instruction most operations common to higher level languages, or compilers, or executive systems. Therefore, each higher level language must be translated by a compiler unique to the computers assembly language into the assembly language and then into machine code.

In the method we are suggesting a powerful intermediate language is first developed. This language is developed with the user in mind and has many operations common to higher level language, executive systems, compilers, etc. Then a program is developed which generates micro-code from programs written in the intermediate language. Thus, the program types of the intermediate language are implemented efficiently. The higher level languages are then described in terms of the intermediate language. The process of describing the intermediate language is simplified because of the power of the intermediate language.

With the traditional approach the assembly language is defined, and the higher level languages are defined. However, there is often a mismatch between the higher level languages and the assembly language, and thus a mismatch between the user and the computer. In our approach the intermediate language is chosen. This language represents those things which a large set of users wish to do. Then the computer is designed to efficiently implement the intermediate language. The intermediate language is well matched to the higher level languages and thus there is a good match between the user and the computer.

4.2.2 PTL an Intermediate Language

The use of the program type language (PTL) was spelled out in the preceding paragraphs. The major features of the language will be discussed and one group of PTL operations will be listed.

What should be the major features of the language? The language should be machine independent. The language should be easy to use. The language should emphasize the basic but often powerful operations required by the user.

The language should be machine independent. The program type language is machine independent in the following sense. There are no references to registers, storage, or particular arithmetic, or I/O units. All references to data is made through named data quantities. However, in another sense the language is not machine independent. The language is not written for parallel or pipeline machines. There are no instructions

which operate on vectors in one operation or make use of parallelism. Thus, the language is designed for the traditional CPU architecture.

The language should be easy to use. The language should emphasize the basic but often powerful operations required by the user. What is a convenient, easy to use language means different things to different people and depends heavily upon what problems are being solved. Fortran is easy to use if mathematical equations of the form $(x^2 \cdot y)/2$ are being written. Fortran is more difficult to use if equations of the form $\sum_{L=1}^x s_{i_k} \cdot e^{x_i}$ are being written. Fortran is difficult to use if one wishes to interchange the 3rd and 4th bits of an integer format work in the computer memory.

Many of the instructions of PTL have evolved by studying special languages, or special problems and trying to select the basic operations required for each.

The string manipulation instructions have evolved from a study of the SNOBOL language. The instructions are the basic instructions required for string manipulation. They allow a search for a pattern match, deletion, insertion, replacement, and concatenation of strings. The instruction set is not as complete as SNOBOL and hence not as useful. Such SNOBOL operations as alternation and conditional value functions are not present in PTL. However, the functions performed by alternation and conditional value instructions can be accomplished by using two or three PTL instructions. Thus, the PTL language is useful for string manipulation since the string manipulation operations performed in SNOBOL can be

5. GET S4 FROM S1 AT S2 THROUGH S3 LABEL S

The string of characters in string S1 between the character positions specified by S2 and S3 is placed into string S4. If S2 and S3 points outside of S1 a branch to LABEL is executed.

EXAMPLE S1 = 'ABCD'

S2 = 2

S3 = 4

after GET S4 S1 S2 S3

execution S4 = 'BCD'

6. DELETE S2 THROUGH S3 FROM S1 LABEL S

The string of character between the character positions specified by S2 and S3 is deleted from string S1. If S2 or S3 points outside of S1 a branch to LABEL is executed.

EXAMPLE S1 = 'ABCD'

S2 = 2

S3 = 4

after DEL S2 S3 S1

execution S1 = 'A'

7. REPLACE BY S4 BETWEEN S2 AND S3 IN S1 LABEL C,S

The string S4 replaces the characters in string S1 between S2 and S3. If S2 or S3 points outside S1 a branch to LABEL is taken.

```

EXAMPLE      S1 = 'ABCDE'

              S2 = 3

              S3 = 5

              S4 = '2'

after        REP S4 S2 S3 S1 LABEL

execution    S1 = 'AB2'

```

4.2.3 Optimization

One of the objectives of this study is the optimal selection of the central processing unit components. Therefore, a cost performance model of the central processing unit was devised. We view the CPU as a collection of hardware units controlled by a microprogram. The functions to be performed by the CPU consist of a set of program types. Each program type is expected to be a frequently used computer program.

The hardware configuration is modeled as a base hardware unit and a set of optimal hardware units. Each program type will have one or more implementations. A program type implementation is a sequence of micro-instructions which will cause the CPU hardware to perform the desired function. Each different program type implementation will use different hardware options and will have a different execution speed and a different hardware cost. Thus, the speed and cost of a program type implementation can be computed and the cost performance curve developed.

References for Section 4

1. Samir S. Husson, Microprogramming Principles and Practices, Prentice Hall, 1970.

5. DATA STRUCTURES AND THEIR REPRESENTATION

The research reported in this section is concerned with data structure and their representation within the computer memory. Section 5.1 approaches this problem in a very general manner and a generalized computer representation which is capable of representing any given collection of data is developed. The next section focuses on a particular application, namely interactive computer graphics.

5.1 Computer Memory Data Representations

The solution of any problem on a digital computer requires that certain pieces of information or data be stored within the memory of the machine. Assuming that we know what data are to be stored, we are faced with the problem of determining a representation within the computer for these items and the relationships among them.

Intuitively, we would like to utilize the "best" representation possible. The definition of what constitutes a "best" representation is, unfortunately, very difficult to formulate, but it is generally agreed that the storage required to implement a given representation and the time required to perform certain operations upon that representation are important factors in determining the "goodness" of the representation.

Under many computer operating systems the cost of solving a given problem (i. e. , running a given program) is based largely upon the CPU time required to achieve the solution, with the only constraint placed upon storage being that the program and data representation must fit into the

available storage (usually, quite substantial). This is strictly true in a mono-programmed environment and generally true in a multi-programmed, paged environment (although in this case additional emphasis may be placed upon the amount of storage required). For this reason we are concerned with determining that representation which minimizes the time required to perform certain operations upon the data associated with the solution of a given problem, subject (possibly) to some restriction on the amount of storage available.

The first step in this study was the development of a generalized computer representation which is capable of representing any given collection of data and of being reduced to the form of any other computer representation, and a mathematical description for this generalized computer representation.

The next step was to develop measures of the storage required to implement the generalized representation and the time required to perform certain primitive operations upon the representation as functions of parameters contained in the mathematical description.

Finally, an optimization technique was developed and applied to these cost functions, resulting in the eventual specification of minimum cost representations.

The details of this work may be found in [1] and the following sections will simply indicate in general terms some of the concepts involved.

5.1.1 Development of a Model

A propositional function defined on the Cartesian product $A \times B$ of two sets A and B is an expression denoted by

$$P(x, y)$$

which has the property that $P(a, b)$, where a and b are substituted for the variables x and y respectively in $P(x, y)$, is true or false for any ordered pair $(a, b) \in A \times B$.

For example, if A is the set of all cities in the United States and B is the set of all states, then

$$P(x, y) = \text{'x is located in y'}$$

is a propositional function of $A \times B$. In particular,

$$P(\text{Ann Arbor, Michigan}) = \text{'Ann Arbor is located in Michigan'}$$

$$P(\text{Detroit, Ohio}) = \text{'Detroit is located in Ohio'}$$

are true and false, respectively.

The expression $P(x, y)$ by itself is called an open sentence in two variables or, simply, an open sentence.

A relation r consists of the following:

- (1) a set A
- (2) a set B
- (3) an open sentence $P(x, y)$ in which $P(a, b)$ is either true or false for any ordered pair (a, b) belonging to $A \times B$.

Thus, r is called a relation from A to B and is denoted by

$$r = (A, B, P(x, y))$$

Furthermore, if $P(a, b)$ is true, this fact is denoted by

$$a r b$$

which is read "a is related to b". On the other hand, if $P(a, b)$ is not true, this fact is denoted by

$$a \not r b$$

which is read "a is not related to b".

Let $r = (A, B, P(x, y))$ be a relation. The solution set R of the relation r consists of the elements (a, b) in $A \times B$ for which $P(a, b)$ is true. In other words,

$$R = \{(a, b) \mid a \in A, b \in B, P(a, b) \text{ is true}\}$$

Notice that R , the solution set of relation r from A to B , is a subset of $A \times B$.

Let R be any subset of $A \times B$. Then one can define a relation $r = (A, B, P(x, y))$ where $P(x, y)$ reads

"The ordered pair (x, y) belongs to R ".

The solution set of this relation r is the original set R . Thus to every relation $r = (A, B, P(x, y))$ there corresponds a unique solution set R which is a subset of $A \times B$, and to every subset R of $A \times B$ there corresponds a relation $r = (A, B, P(x, y))$ for which R is the solution set. Since this one-to-one correspondence exists between relations $r = (A, B, P(x, y))$ and subsets R of $A \times B$, a relation can be redefined by the following:

A relation r from A to B is a subset of $A \times B$.

Although this definition may seem somewhat artificial, it has the advantage that the undefined concepts "open sentence" and "variable" are not used.

The intrinsic structure of any collection of n data items may then be described in the following manner.

Let the set Δ consist of the n data items in question:

$$\Delta = \{d_i \mid i=1, 2, \dots, n\}$$

where d_i is the i^{th} data item in the collection.

Let the set \mathcal{P} (capital rho) consist of all relations of interest in Δ (i. e., from Δ to Δ):

$$\mathcal{P} = \{r_j \mid j=1, 2, \dots, m\}$$

where r_j is the j^{th} relation and m is an upper bound on the number of different relations.

Corresponding to each datum $d_i \in \Delta$ there exists some set \mathcal{P}_i (which may be empty) of relations such that

- (1) $\mathcal{P}_i \subset \mathcal{P}$, and
- (2) for each relation $r_j \in \mathcal{P}_i$ there exists at least one datum $d_k \in \Delta$ satisfying $d_i r_j d_k$.

For specific values of i, j , and k , where $d_i \in \Delta$, $r_j \in \mathcal{P}_i$, and $d_k \in \Delta$, $d_i r_j d_k$ is called a relation instance. d_i is called the source of the relation instance, d_k is called the target of the relation instance, and r_j is called the relation symbol of the relation instance.

In order to minimize notational confusion it will be convenient to introduce some set Π which consists of n "indicators", one for each data item in Δ :

$$\Pi = \{p_k \mid k=1, 2, \dots, n\}$$

where p_k is an indicator corresponding to the k^{th} data item $d_k \in \Delta$. As a result, sets Δ and Π are isomorphic.

Henceforth all relation instances will draw upon set Π for targets instead of upon set Δ which will serve to supply sources only. Thus, all relation instances will be of the form

$$d_i r_j p_k$$

where $d_i \in \Delta$, $r_j \in \mathcal{P}$ and $p_k \in \Pi$.

Now each $r_j \in \mathcal{P}$ can be considered a relation from Δ to Π .

At this point a number of entities, which will be used to characterize our generalized computer representation for the intrinsic structure of a collection of data, will be defined.

Consider the sets Δ , \mathcal{P} , and Π .

Corresponding to every source-relation pair $(d_i r_j)$ where $d_i \in \Delta$ and $r_j \in \mathcal{P}_i \subset \mathcal{P}$ there exists some set of targets $\Pi_k^2 \subset \Pi$ where $k \in \{1, 2, \dots, n_2\}$ such that for every $\pi \in \Pi_k^2$, $d_i r_j \pi$.

Let Π be partitioned into a number $n_1 \leq n$ of mutually exclusive subsets Π_j^1 where $j=1, 2, \dots, n_1$ such that the following conditions are satisfied:

- (1) Each Π_k^2 where $k \in \{1, 2, \dots, n_2\}$ can be constructed exactly by the union of a number (≥ 1) of these sets Π_j^1 .
- (2) No two of these sets $\Pi_{j_1}^1$ and $\Pi_{j_2}^1$ where $j_1 \in \{1, 2, \dots, n_1\}$, $j_2 \in \{1, 2, \dots, n_1\}$, and $j_1 \neq j_2$ can be combined into a single set without violating condition (1).

Let each source-relation pair $(d_i r_j)$ be designated by σ_{ij} .

Let set \sum_k where $k \in \{1, 2, \dots, n_2\}$ be the union of all σ_{ij} for which $d_i r_j \pi$ for every $\pi \in \Pi_k^2$. That is, \sum_k consists of all source-relation pairs having Π_k^2 as target set.

Let set π_j^3 where $j \in \{1, 2, \dots, n_1\}$ consist of all those sets Π_k^2 of which Π_j^1 is a constituent (subset). That is, π_j^3 indicates all those sets Π_k^2 where $k \in \{1, 2, \dots, n_2\}$ of which Π_j^1 is a subset.

Corresponding to each data item $d_i \in \Delta$ there exists some set P_i^2 of relation-target set pairs $(\rho \Pi_k^2)$ defined as follows:

$$P_i^2 = \{(\rho \Pi_k^2) \mid \rho \in P_i \text{ \& } \forall \pi \in \Pi_k^2, d_i \rho \pi\}$$

That is, for each $\rho \in P_i$ there exists some $\Pi_k^2 \subset \Pi$ and, hence, some $(\rho \Pi_k^2) \in P_i^2$ such that $d_i \rho \pi$ for all $\pi \in \Pi_k^2$. Thus, sets P_i and P_i^2 are isomorphic for every $i \in \{1, 2, \dots, n\}$.

Let each relation-target set combination $(r_j \Pi_k^2)$ be designated by γ_{jk} .

Let set $\Delta_{jk} \subset \Delta$ be that set of sources δ such that $\delta r_j \pi$ for every $\pi \in \Pi_k^2$ (i. e., Δ_{jk} is that set of sources for which γ_{jk} is a relation-target set combination).

Let set P_i^1 where $i \in \{1, 2, \dots, n_3\}$ be the union of all γ_{jk} for which Δ_{jk} is the same. (Obviously, all P_i^1 are mutually exclusive since each γ_{jk} is unique). Each Δ_{jk} is put into correspondence with its P_i^1 and hereafter may be designated by Δ_i .

Then each P_i^2 is composed of the union of all P_j^1 for which $d_i \in \Delta_j$.

Finally, let set $\Delta_i^* \subset \Delta$ be that set of sources δ such that set P_i^2 is the set of relation-target set pairs for every $\delta \in \Delta_i^*$. That is, every $\delta \in \Delta_i^*$ has identically the same set of relation-target set pairs.

In our model for a generalized representation of a given set of data each of the sets defined above is represented by a ring, where each element of the corresponding ring and the set itself is represented by the head of the ring.

This model is further characterized by some 39 decision variables (0-1 variables which indicate the presence or absence of some entity within the model). These decision variables indicate whether or not each of the sets defined above is to be explicitly represented, if it is to be so represented in what form, etc. It is this set of variables which is used by the optimization process to determine the optimal representation for the given set of data.

5.1.2 Optimization

As indicated in the previous section our model is characterized by 39 decision variables which indicates that there are approximately 10^{12} possible states which the model may assume. Fortunately, there are a number of

constraints which reduce this number somewhat and by making certain other assumptions we can reduce the number to something less than 10^5 states.

In any event it is desirable to perform the optimization as efficiently as possible. Unfortunately, our cost function is rather ill-behaved and does not lend itself to minimization by any of the well developed, classical methods (dynamic programming, branch and bound, etc.). Therefore, we have developed a procedure which is tailored to our particular problem.

Basically, the procedure is as follows:

- (1) Estimate a least upper bound for the cost function and set the best cost so far T^* , to this value.
- (2) Assign values to some (fixed) subset, S , of the 39 decision variables and estimate a lower bound, L , for the cost function.
- (3) If L is less than or equal to T^* , systematically assign values to the remaining unfixed variables and compute the actual cost, T_i , at each of the states determined in this way. If the least cost $T_m = \min_i T_i$ is less than T^* , set T^* to T_m .
- (4) If not all possible assignments to the variables in S have been considered, go to step (2). Otherwise, T^* is the optimal solution.

Of course, the techniques for determining the least upper bound for the cost function and for determining L for each assignment of values to S are very much dependent upon our particular problem.

This technique compares very favorably with a less sophisticated technique attempted earlier. For comparison, the earlier technique required approximately 1400 seconds of CPU time on an IBM 360/67 for solution of a problem which, using the current technique, requires only 90 seconds or less.

5.2 Computer Graphics Systems

The objective of the research reported in this section is to describe the topological structure of an interactive computer display picture and to compare the time which is required to operate on various implementation of this structure. The purpose of an interactive graphical display system is to make efficient use of both man and machine, allowing the user to interact with the computer while his program is running in order to influence the course of his computations. The designer may communicate with the computer by means of line drawings and in some cases it may suffice to use a simple list of x, y-coordinates of the endpoints of the straight lines which make up the drawing. Depending upon the application requirements, however, varying degrees of sophistication and complexity are often used in the data structures representing the display pictures. For example, an interactive graphics system which is to be used for computer-aided design has to satisfy certain special requirements. A user may reference, at any

moment, any part of the drawing he has created. He may indicate, with a light pen, a particular line or a segment of the drawing. When the pen sees the light from the display, an interruption of the normal drawing occurs, and the program can determine which line or the segment is being referred. The structure should be flexible enough to find out how the line or part fits into the rest of the structure before any actions (erase) can be performed. Therefore, the display structure should be implemented to allow the necessary searching and accessing of picture parts to be performed efficiently. Also, during a design process actions like adding or deleting, and moving or rescaling of picture parts may be required. So the structure should be implemented to allow quick updating. Clearly these requirements are application and equipment configuration dependent. The purpose of this work is to determine the optimum implementation of the topological structure for a given equipment configuration and application. Since the details of this work have already been published in the report entitled, "Optimal Implementation of Topological Structures for Interactive Computer Displays", [2], we shall briefly describe the general approach taken in this work and the major decisions which affect the choice of an implementation.

5.2.1 The General Approach

The scope of this work may be clarified with the aid of Figure 5.2.1, which represents a simplified form of the general interactive computer display program as described by W. R. Sutherland [3]. Each input which affects the picture is an element of a one-dimensional control language,

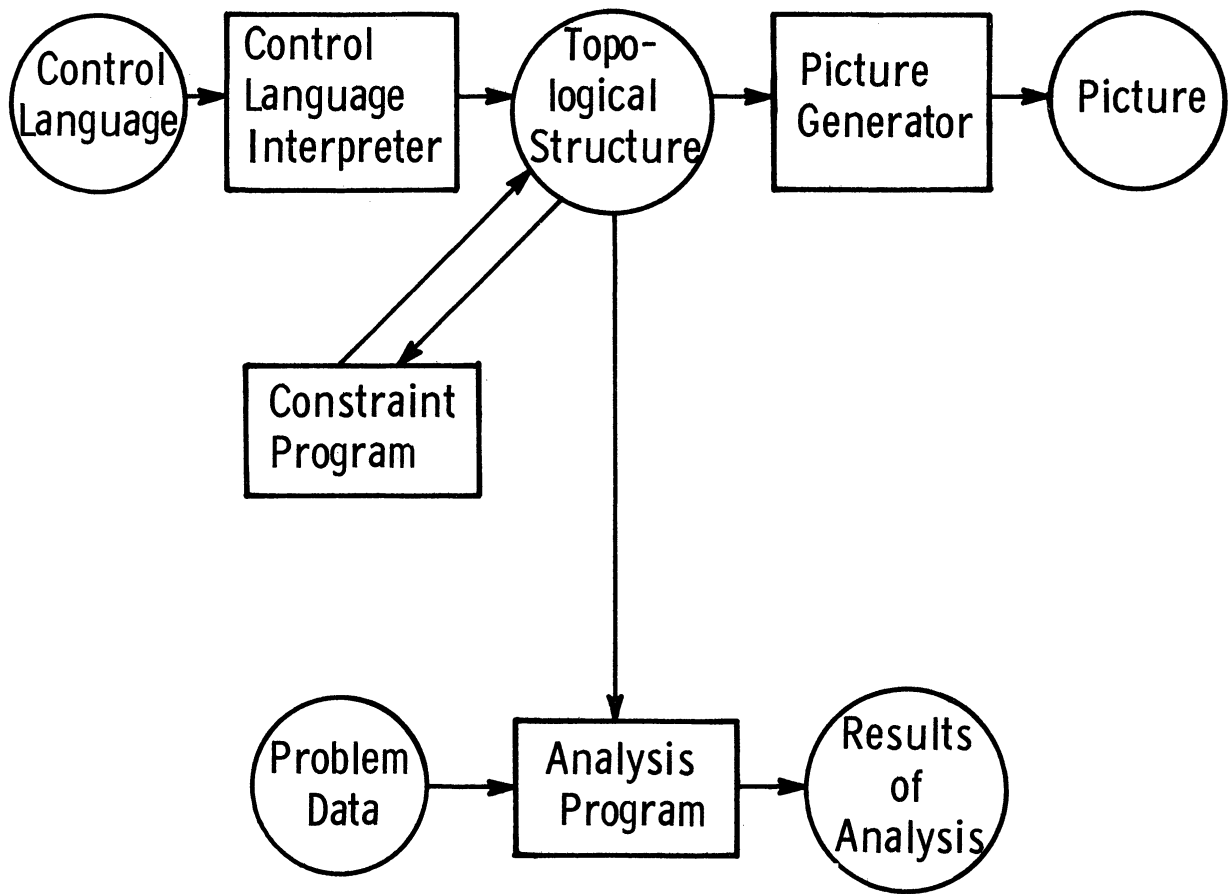


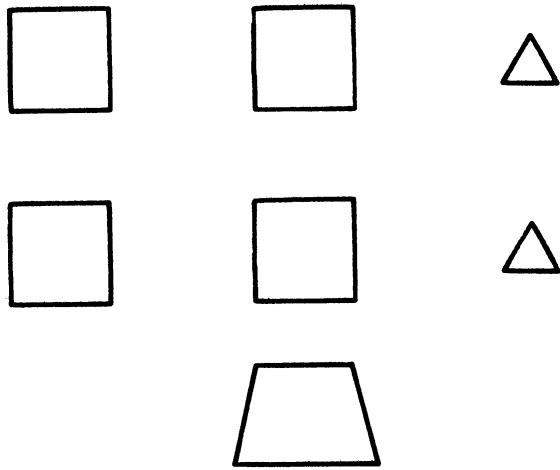
Figure 5.2.1

Interactive Computer Display Program
as described by W. R. Sutherland

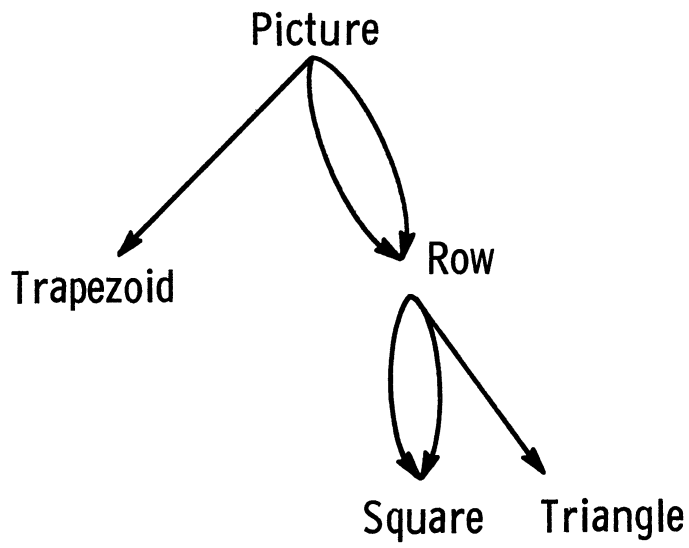
which consists of sequences of light pen motions, push button hits, etc. In response to these inputs, the control language interpreter modifies the structure of the picture which is being displayed. The resultant structure is then further modified by a constraint program which compensates for the coarseness of the control language inputs, and it is interpreted by the picture generator to produce a new picture. We are concerned with the topological structure which is interpreted to generate the picture, the picture generator, and the picture which is produced.

5.2.1.1 Topological Structure

The need for representing the structure of a picture, in addition to its appearance, was originally demonstrated by SKETCHPAD [4]. A software system which provided for the representation of the topology of a picture on a more sophisticated hardware configuration was developed at Bell Telephone Laboratories for use with GRAPHIC-2 [5]. Generally, a topological structure describes both the topology of picture parts and the coordinate transformations which are necessary to display these picture parts. One use of this information is to permit the display of multiple copies of a subpicture. The ability to display multiple copies of a subpicture eliminates the necessity to store individual copies of identical subpictures. Consequently, since the storage required for topology information is small relative to that required to store copies of subpictures, the representation of the topology of the picture in memory results in a reduced total storage requirement.



a. Picture



b. Representation of Topology

Figure 5.2.2
A Picture Representation of Its Topology

Furthermore, all picture parts which are represented as copies of a common subpicture can be modified simultaneously.

The topology of a picture identifies which entities are parts of other entities, where the entities are defined to be the picture parts (or the entire displayed picture) which are to be interpreted as a unit by the graphics program. This procedure may be illustrated with the aid of Figure 5.2.2. Figure 5.2.2a represents a picture to be displayed and Figure 5.2.2b shows a representation of the topology of this picture. In Figure 5.2.2b, each directed line indicates that a copy of the subpicture to which it points is used as part of the subpicture at its tail. For this example, the picture consists of a trapezoid and two rows of objects, each of which consists of two squares and a triangle. The subpictures which are represented are the trapezoid, the square, the triangle, the row, and the entire picture. Only three of these subpictures are used to generate polygons in the picture: the trapezoid, the square, and the triangle. However, seven polygons appear in the picture because four copies of the square are displayed and two copies of the triangle are displayed.

In order to use a copy of a subpicture as part of a larger subpicture, some information about how the smaller subpicture is to appear in the larger subpicture is needed. In SKETCHPAD, this information was represented by four variables: x and y coordinates, a rotation angle, and a scale factor. (The values actually stored were the x and y coordinates and the products of the scale factor by the sign and cosine of the angle of

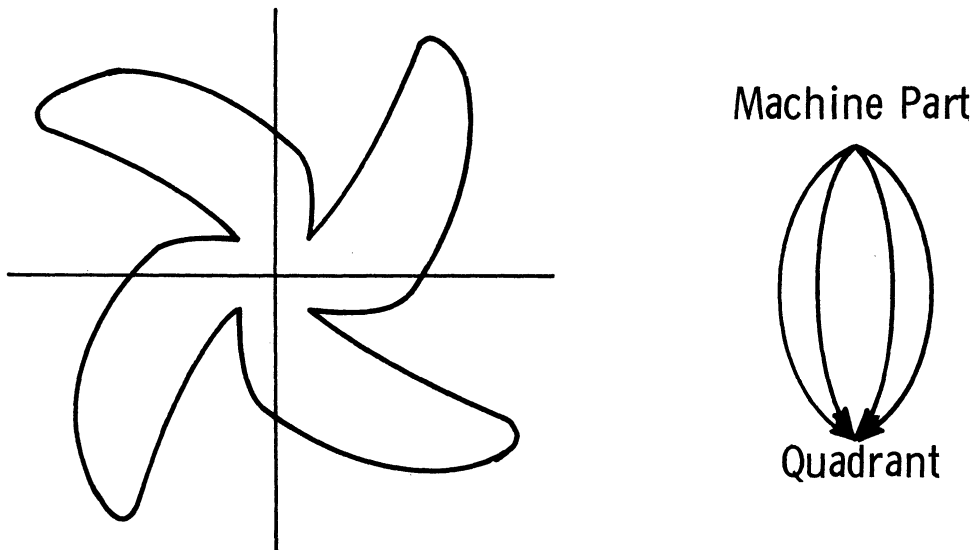
rotation.) These four variables represented a transformation of coordinates to be applied to each point in the smaller picture when it was displayed. A set of values for these four variables, together with a subpicture, was called an instance. A subpicture could then be defined in terms of smaller subpictures as a collection of instances.

For the example in Figure 5.2.2, there are two instances of the square, one instance of the trapezoid, one instance of the triangle, and two instances of the row. Each of these instances is represented as a directed line in Figure 5.2.2b. The number of instances of a subpicture is not necessarily the number of times that that subpicture appears in the overall picture. In this example, there are four squares in the picture, yet there are only two instances of the square. Likewise, the variables which help define an instance are not measured relative to the overall picture. In this example, the coordinates of the two instances of the square and the instance of the triangle are measured relative to the coordinates of each row.

As mentioned above, the topology of the picture is also used to identify those other subpictures which are modified whenever a subpicture is referenced by a demonstrative control language input (i.e., an input which consists of identifying a picture part by pointing to it with a light pen, stylus, etc.). Some of the applications of the topology of the picture to this problem may be illustrated with the aid of Figure 5.2.3. Figure 5.2.3a shows a resistor symbol as it might appear in a circuit analysis program. The symbol is



a. A Resistor from an Electrical Network Program



b. A Machine Part From a Drafting Program

Figure 5. 2. 3

Examples of Pictures for which the Topology of Each Picture Facilitates Response to Control Language Inputs

considered to be composed of two ports, i. e., attachment points for connections to other elements, a parameter value, and a body. In the picture, each port appears as a small circle, the parameter value appears as the number 1.0, and the body appears as the rest of the symbol. For purposes of illustration, assume that a program is running which allows the user to point at the symbol with a light pen and either (1) change the value of the parameter to a value which has just been inputted on some other device, or (2) move the symbol with the light pen if no parameter value has been inputted. In either case, the light pen may reference any one of several primitive subpictures: the port, the parameter, or the body of the symbol. The resistor subpicture is identified as the subpicture which consists of a set of instances of primitive subpictures and which was being displayed at the time of the light pen hit. In the former case, the parameter subpicture to be replaced is identified as the primitive subpicture which is part of the resistor subpicture and which represents a parameter. (Whether or not a subpicture represents a parameter is determined from information external to the topology, e. g., from an ordering imposed on the parts of the resistor subpicture.) In the latter case, the coordinates associated with the instance of the resistor subpicture which was referenced with the light pen are modified in order to move the symbol.

Figure 5. 2.3b represents a picture which maintains symmetry about the origin of the two axes shown. The program is assumed to be capable

of modifying only the quadrant subpicture in response to control language inputs. The machine part subpicture is displayed as four instances of this quadrant subpicture at angles 0 , $\pi/2$, π , and $3\pi/2$. Not only does the topological structure provide the information necessary to identify what other changes should be made to the picture when a quadrant is modified, but it guarantees that the other quadrants will be correspondingly modified without intervention by the program.

In this work two topological structures are considered. One structure provides for the independent modification of picture parts which are not topologically related. However, as shown in Figure 5.2.1, a constraint program is usually included as part of the interactive graphics program to refine the structure after it has been modified in response to a control language input. For example, when a character is removed from a line of text which is displayed, the other characters in the line may be adjusted so that no gap is left in the line. In a picture of a mechanical system, many parts may move when a specified part is moved in order to simulate mechanical constraints. The second structure provides for the enforcement of concatenation constraints among picture parts. In this structure coordinate transformations are represented so that they depend on portions of the structure. The subpictures which are used to generate entities may then be concatenated in a manner similar to that described by Shaw [6] . The applicability of one structure over the other depends the application and is normally determined by the frequency with which the entities are to be concatenated.

5. 2. 1. 2 The Picture Generator

The structures used by SKETCHPAD and GRAPHIC-2 were similar in that they both described picture topologies. However, pictures were generated from these two structures in quite different fashions. SKETCHPAD interpreted a list structure to produce a table of the coordinates of points to be plotted on the display screen. Then the program generated the picture to be displayed by transferring the coordinates in each point represented in the table to the display processor. Consequently, the SKETCHPAD picture generator was essentially a program. However, in GRAPHIC-2, the general-purpose computer program interpreted the list structure only to compute a pair of coordinates at which a primitive subpicture was to be displayed and to locate a display processor program (leaf) for that subpicture. The display processor itself then executed the leaf and produced the subpicture at the computed coordinates. Hence, the GRAPHIC-2 picture generator was partly a program and partly display processor hardware.

A third form of a picture generator has also been implemented [7]. This structure was implemented entirely as a display processor program. Recent effort in the design of display processors has yielded devices which interpret 3-dimensional data and plot perspective projections of this data on 2-dimensional screens. Examples of such devices are the Adage Graphics Terminal [8] and the Evans and Sutherlands LDS-1 [9]. The latter of these devices is sufficiently versatile to generate a picture from a topological structure without the aid of a general-purpose computer program.

As evidenced by the above examples, the picture generator may be either a program, a combination of a program and display processor hardware, or entirely display processor hardware. No one of these schemes has a clear advantage over the others for all applications. A software point-plotting picture generator, such as the one which was employed by SKETCHPAD, can be implemented with even the most primitive display processor hardware. However, this scheme requires storage both for the structure of the picture and for a display buffer. Furthermore, unless the picture is produced on a storage tube or some similar device, much computation time is consumed merely maintaining a static picture. Less computation time is required for this process if a scheme such as that used in GRAPHIC-2 is employed. Furthermore, no storage is required for a display buffer. However, this scheme requires that a data channel be available to send information to the display processor and that the display processor include a display generator which is capable of producing some basic geometric forms, such as vectors, which may be displayed relative to given coordinates. Unlike the software point-plotting technique, this technique often limits the flexibility with which copies of a subpicture may be displayed. For example, a single leaf could not be executed so that copies of its corresponding subpicture would be displayed at various angles of rotation in GRAPHIC-2. Even more limitations are imposed if the display processor hardware itself is to be used as the picture generator. In particular, the words which appear in the topological structure must be

executable display processor commands. Furthermore, if this procedure is used and demonstrative control language inputs are to be identified, the display processor must contain a subroutining facility which saves return addresses on a push-down stack which may be examined by the general-purpose computer. However, this procedure has the advantage over the others that no computation time is required to maintain a static picture.

5. 2. 1. 3 Representation of the Topological Structure

The data structure used most generally in large graphics systems is the hierarchical tree and ring structure. A hierarchical structure is a structure with levels of hierarchy and is constructed with rings. From one ring, there may be branches off from any element to its logically related elements. Branches from these related elements to other related elements may occur. This structure allows access from any data item to any other data item via the rings. This structure is very flexible and those operations which interrogate the structure can be handled very easily. However, the operations which modify the structure are quite involved because additional pointers have to be updated. Extra overhead in storage is also caused.

5. 2. 1. 4 The Optimum Implementation

All of the above considerations are applied to determine the optimum implementation of the topological structure for a given equipment configuration and application. As a preliminary step, a cost function which is a

measure of the performance of an implementation of the topological structure must be defined. The cost of an implementation of the structure is considered to be the average time which is required to modify and interpret the structure in response to a control language input. For simplicity, the computer which controls the display processor is assumed to be dedicated to the use of that display processor. The discussion is also restricted to display processors whose picture must be continually refreshed. Furthermore, in order to avoid momentary losses of the picture, the process of refreshing the picture is assumed to have priority over the process of responding to control language inputs. Display processors whose picture need not be continually refreshed are not considered because the response time for these devices is independent of the picture generation process and is generally insignificant when compared to the time which is required for this process.

Certain constraints are also imposed by the hardware configuration. One of these is a limitation on the amount of storage which is available to represent the structure. A second constraint is a limitation on the amount of storage which is available to represent the structure. A second constraint is a limitation on the amount of time which is available to produce a picture from the structure. This time is the smallest time interval which produces objectionable flicker (typically about $1/30$ sec.).

Equations which express the above cost function in terms of major design decisions (e.g., whether to use the display processor or a computer

program as the picture generator, whether to use the topological structure which enforces concatenation constraint or not, presence or absence of redundant pointers in the representation of the topological structure) are derived. Each design decision is considered to be a binary-valued variable and it represents a choice of one of the two possible methods of implementing some feature of the topological structure. The optimal implementation is then described by the set of these design decisions which minimizes the cost.

The storage which is required by the topological structure, the total time which is arrested from the computer in order to generate the picture, the time which elapses while the picture is generated, and the cost are expressed as functions of these design decisions. Eleven design parameters, each of which may assume two values, are considered. The number of implementations of the topological structure which can be specified for a given hardware configuration is $2^{11} = 2048$. However, only 480 of these implementations are possible because the design parameters are not independent. Furthermore, for certain applications, not all of these 480 implementations are possible because of the constraints imposed by the equipment configuration. The number of implementations is small enough so that these implementations may be easily enumerated. Consequently, the optimal design of the topological structure for a specified hardware configuration and application may be determined.

A program which enumerates the possible implementations for a specified hardware configuration and application, and the results obtained by applying this program to several applications for the DEC 339 are described in [2].

References for Section 5

1. L. S. Randall, "A Relational Model and its Optimization for the Representation of Structured Data within a Random-Access Computer Memory", Systems Engineering Laboratory Technical Report no. 54, The University of Michigan, Ann Arbor, to be published.
2. J. H. Jackson, "Optimum Implementation of Topological Structures for Interactive Computer Displays", Systems Engineering Laboratory Technical Report no. 51, The University of Michigan, Ann Arbor, January 1971.
3. W. R. Sutherland, "The On-Line Graphical Specification of Computer Procedures", Ph. D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Lexington, Massachusetts, January 1966.
4. I. E. Sutherland, "SKETCHPAD - A Man-Machine Graphical Communication System", Proceedings of the Spring Joint Computer Conference, 1963, pp. 329-346.
5. C. Christensen and E. N. Pinson", "Multi-Function Graphics for a Large Computer System", Proceedings of the Fall Joint Computer Conference, 1967, pp. 697-711.
6. A. C. Shaw, "Parsing of Graph-Representable Pictures", Journal of the ACM, vol. 17, no. 3, July 1970, pp. 453-481.
7. J. H. Jackson, "An Executive System for a DEC 339 Computer Display System", Concomp Project Technical Report 15, University of Michigan, Ann Arbor, Michigan, December 1968.
8. System Reference Manual -- Adage Graphics Terminal, Adage, Inc., Boston, Massachusetts, March 1968.
9. Evans and Sutherland Line Drawing System Model 1 System Reference Manual, Evans and Sutherland Computer Corporation, Salt Lake City, Utah, January 1970.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D		
<i>(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)</i>		
1. ORIGINATING ACTIVITY (Corporate author) The University of Michigan Systems Engineering Laboratory Ann Arbor, Michigan 48104		2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED
		2b. GROUP N/A
3. REPORT TITLE A STUDY OF INFORMATION IN MULTIPLE-COMPUTER AND MULTIPLE-CONSOLE DATA PROCESSING SYSTEMS		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) Technical Report April 1970 - March 1971		
5. AUTHOR(S) (First name, middle initial, last name) K.B. Irani I.S. Uppal J.W. Boyse, et al		
6. REPORT DATE August 1971	7a. TOTAL NO. OF PAGES 177	7b. NO. OF REFS 57
8a. CONTRACT OR GRANT NO. F30602-69-C-0214 Job Order No. 55810000	9a. ORIGINATOR'S REPORT NUMBER(S) Annual Report No. 4	
	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) RADC-TR-71-160	
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.		
11. SUPPLEMENTARY NOTES RADC PROJECT ENGINEER: Rocco F. Iuorno (ISIS) AC 315 330-7011	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (ISIS) Griffiss Air Force Base, New York 13440	
13. ABSTRACT This report documents the achievements from April 1970 to March 1971 of continuing research into the development and application of mathematical techniques for the analysis and optimization of multiple-computer, multiple-user systems. A summary of the theoretical investigations conducted, the major conclusions reached, and some typical applications are included. The material covers the following areas: message processing and communication system, multiprogrammed and multiprocessor computer systems, central processor design, and data structures and their representation.		

DD FORM 1473
1 NOV 65

UNCLASSIFIED

Security Classification



UNCLASSIFIED

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Mathematical Models Multiprogramming Paging Algorithm Data Base Structures Microprogramming File Systems Interactive graphics						

UNCLASSIFIED

Security Classification