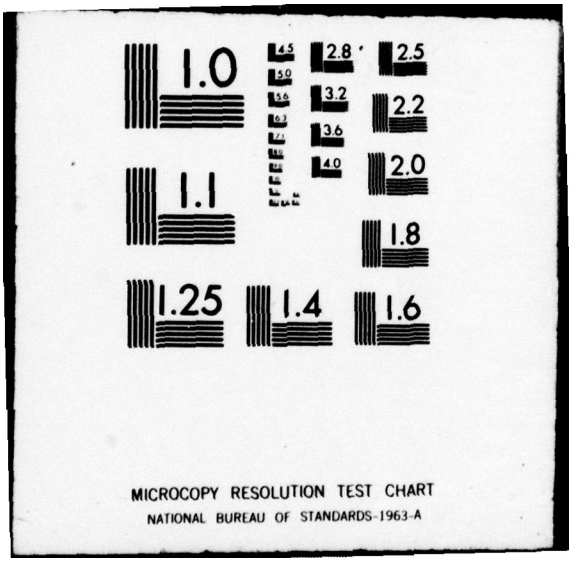
AD-A069 028 HARVARD UNIV CAMBRIDGE MASS AIKEN COMPUTATION LAB F/G 9/2
STUDIES ON THE FEASIBILITY OF SYSTEM SOFTWARE DEBUGGING.(U)
APR 79 R K JAIN, U GAGLIARDI F30602-77-C-0058
UNCLASSIFIED RADC-TR-79-20 NL

1 OF 3
AD
A069 028


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

[redacted]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-79-20	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): STUDIES ON THE FEASIBILITY OF SYSTEM SOFTWARE DEBUGGING	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report June 1977 - October 1978
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Rajendra K. Jain, Ugo Gagliardi	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0058 new
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Harvard University, Aiken Computation Laboratory, Cambridge MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: J.O. 5581 17 02
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB MA 13441	12. REPORT DATE: April 1979 / 13. NUMBER OF PAGES: 202
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 226p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Raymond A. Liuzzi (ISIE)

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

Debugging	Control Theory
System Software	Performance Monitoring
Computers	Virtual Machines
Operating Systems	

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

This report is two separate but parallel efforts on system software debugging studies. The first study describes a theoretical approach on defining new resources management algorithms that can be utilized in an operating system. The second report describes an experimental work station for performing sophisticated operating system software debugging. The actual experimentation is performed by connecting a PDP 11/40 to a PDP-10 through a high speed data bus and installing on the PDP 11/40 a virtual machine monitor.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

405831

EVALUATION

The work described in this report and performed during this effort represents a significant accomplishment in the structuring of resource management algorithms for operating systems. First, a theoretical base was described that is a fresh and general view of the resource management problems in complex operating systems. Second, an experimental facility was built to examine various resource management policies.

A critical component of this facility, a virtual machine monitor, represents the resource management portion of an operating system. This facility allows one to virtualize a complete computer system at a work station without having extensive peripherals at the work station.

This effort demonstrates the types of capabilities that will compose future operating systems.

Raymond A Liuzzi

RAYMOND A. LIUZZI
Project Engineer

Preface

This final report on Contract No. F30602-77-C-0058 is composed of two parts. The first part documents research on novel resource management algorithms used in computing systems. What we are referring to are the CPU, and memory management algorithms that are the heart of the resource allocation function of operating systems.

The contribution made by this research is of a fundamental nature and we expect that it will revolutionize the design of operating systems in the future. In particular a key contribution made by this research is an unification of the page replacement policies in operating system. In fact, we present here a result that shows that many of the commonly accepted page replacement policies can be considered a special case of the more general policy based on an ARIMA (1,1,1) model proposed by this work.

The formulation of a unifying theory, which shows that previous known cases are special cases of a more general case, not only results in conceptual clarity and economy, but it actually sheds further light on the conditions where the special cases are really applicable.

An additional fall out of this research, and a very significant one, is that it becomes possible to design operating system memory management policies which are capable of adaptation. The point being that the general unified theory, which is presented in this report, supplies two key factors. First, as we stated above, an understanding of the conditions under which specific memory management policies are optimal and secondly, a general algorithm which comprises these policies as special cases. Therefore, one is in the position to design an operating system which uses a generalized memory management policy capable of adapting to conditions. In other words, the ARIMA (1,1,1) page replacement algorithm could be embodied in such a computing system and the algorithm would automatically specialize to its extreme special cases when the operating conditions warrant.

Therefore, the key contribution of the first part of the report, is a fresh, much more powerful and general view to the resource management problem in computing systems. This contribution capitalizes on the very extensive and solid body of mathematical theory that has been developed by the control theoreticians.

The second part of the report presents work on new notions on the organization of a debugging environment. The basic principle here is to give the person doing the debugging of complex software an inexpensive work station with a number of tools which enable him to obtain extensive snapshots of the state of a complex software system.

The reason why the work station can be inexpensive and still offer very powerful status capturing and reporting capabilities is that the work station obtains most of its peripherals, through a novel virtualization technique, from the host computer. Since the work station is capable of transferring at a very high data rate large sections of the main computer memory it has the ability to capture extensive status information and reporting it quickly.

These two parallel efforts were undertaken in this program, because the experimental apparatus needed to verify the theories presented in Part I essentially supplies the complete underlining structure for performing the exploration in Part II.

An inexpensive experimental facility was built by connecting a PDP-11/40 to a PDP-10 through a very high speed data bus and by installing on the PDP-11/40 a virtual machine monitor. This kind of facility is of course a very inexpensive and very convenient facility for exploring alternative memory management policies. In fact, the virtual machine monitor is essentially the resource management function of an operating system and the high speed data bus allows one to virtualize a complete computer at the work station without having to have any expensive peripherals at the work station.

Once this facility for experimenting on operating system resource management algorithms had been put in place, it became very natural to use it for exploring approaches to debugging of complex systems running on the host processor and this is exactly the work that is reported in Part II of this report.

Table of Contents

SECTION I - Control Theory for the Dynamic Optimization of Computer Systems Performance

LIST OF FIGURES

LIST OF TABLES

SYNOPSIS

This thesis proposes the application of control theory to the dynamic optimization of computer systems performance. Until now, queueing theory has been extensively used in the evaluation and modeling of computer systems. It is a good design and static analysis tool. However, it provides little run time guidance. For dynamic (run time) optimization we need to exploit modern control theoretic techniques such as state space models, stochastic filtering and estimation, time series analysis, etc. In this thesis, a general control theoretic approach is proposed for the formulation of operating systems resource management policies. The approach is exemplified by formulating policies for CPU and memory management.

The problem of CPU management is that of deciding which task from among a set of ready tasks should be run next. The main problem encountered in the practical implementation of theoretically optimal algorithms is that the service-time requirements of tasks are unknown. The proposed solution is to model the CPU demand as a stochastic process, and to predict the future demands of a job from its past behavior. Several analytical results concerning the effect of prediction errors are derived. An empirical study of program behavior is made to find a suitable predictor. Several different models are compared. Finally, it is shown that a zeroth order autoregressive moving average model is the most appropriate one. Based on this observation an adaptive scheduling algorithm called "SPRPT" (Shortest Predicted Remaining Processing Time) is proposed.

The problem of memory management is also formulated as the problem of predicting future page references from past program behavior. Using a zero-one stochastic process model for page references, it is shown that the process is non-stationary. Empirical analysis is presented to show that the page reference pattern can be satisfactorily modeled by an autoregressive integrated moving average model of order 1,1,1. A two stage exponential predictor is derived for the model. Based on this predictor a new algorithm called "ARIMA Page Replacement Algorithm" is proposed. This algorithm is shown to be easy to implement. It is shown that many conventional page replacement algorithms, including Working Set, are merely boundary cases of the ARIMA algorithm. The conditions under which these conventional algorithms are optimal are described. The limitations of the formulation and possible directions for future extensions are also discussed.

The ARIMA model does not take into account the fact that a binary process takes only two values, 0 or 1. This discrepancy is removed by developing Boolean models for such processes. It is shown that if a binary process is Markov of a finite known order, it can be modeled as the output of a Boolean (switching) system driven by a set of binary white noises. Modeling, estimation, and prediction of the process using the Boolean model is described. A method is developed for optimal non-linear prediction under any given non-linear cost criterion. All the results are then generalized to k-ary processes, i.e., processes which take integer values between 0 and k-1. Finally, the application of the model to the problem of memory management is described.

CONTROL-THEORETIC FORMULATION
OF
OPERATING SYSTEMS RESOURCE MANAGEMENT POLICIES

by

Rajendra Kumar Jain

CHAPTER I

SECTION - I

Research on Resource Management Algorithms Used in Computing Systems

INTRODUCTION

Conventionally an operating system is defined as the set of computer program modules which control the allocation and use of equipment resources such as the central processing unit (CPU), main memory, secondary storage, I/O devices and files [MaD74]. These programs resolve conflicts, attempt to optimize performance, and interface between the user program and computer resources (hardware and system software).

1.1 CONTROL-THEORETIC VIEW OF AN OPERATING SYSTEM

For a control theorist, an operating system is a set of controllers which exercise control over the allocation of some system resource. The goal of each controller is to optimize system performance while operating within the constraints of resource availability.

Figure 1.1 shows some of the components of an operating system. The Controllers are represented by circles. The "load controller" controls the number of jobs allowed to log in. The job controller (job scheduler, or high level dispatcher) controls the transfer of jobs from the "submitted" queue to the "ready" queue. This decision is based upon the availability of resources like memory, magtapes, etc. The CPU controller (task dispatcher, or low level scheduler) controls the allocation of the CPU. It selects a task from the set of ready tasks and allows it to run. The paging controller (page replacement algorithm, or memory management algorithm) controls the transfer of pages from virtual memory (disk or drum) to primary memory, and so on.

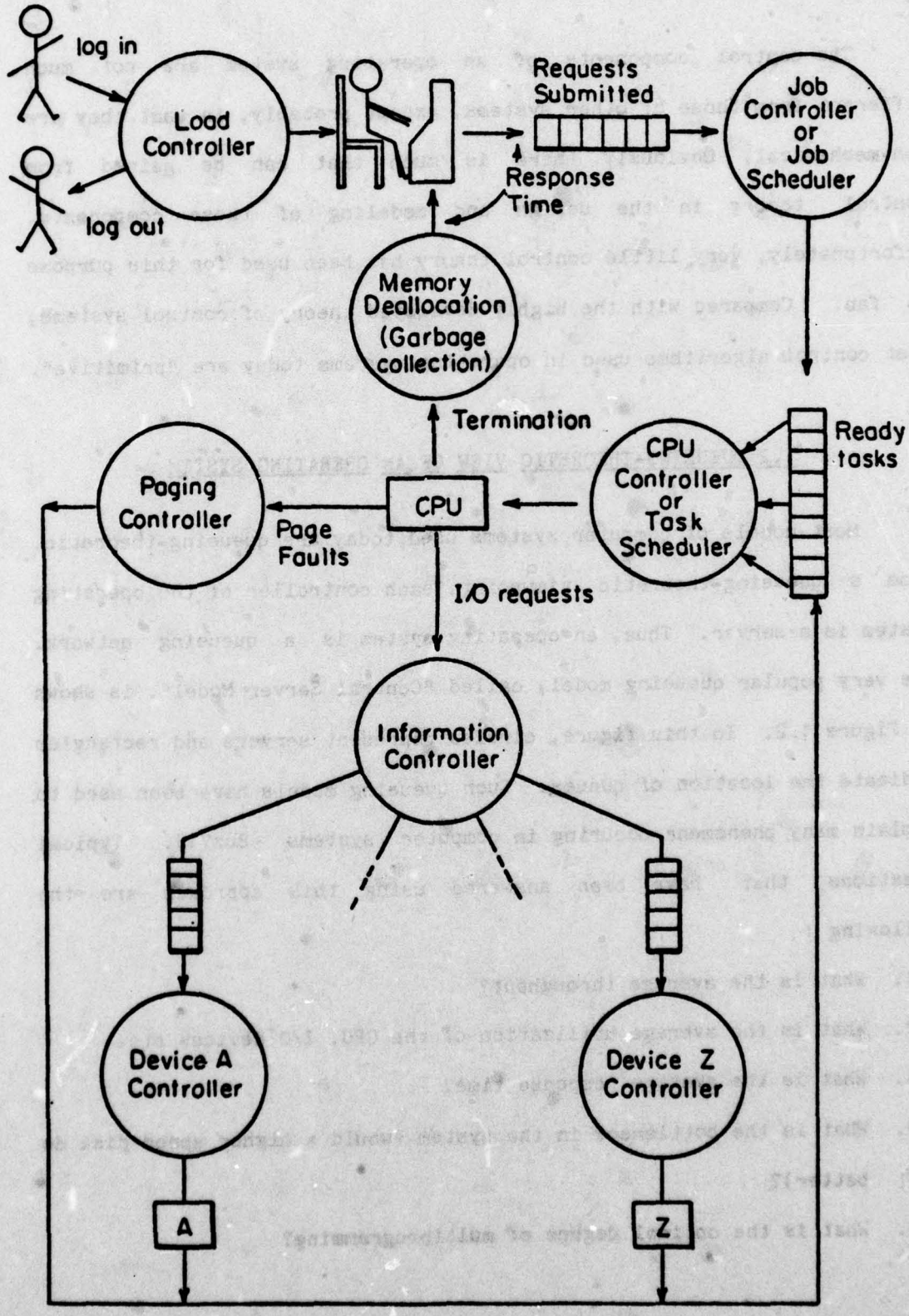


Figure 1.1: Control-theoretic view of an operating system

The control components of an operating system are not much different from those of other systems, except probably, in that they are non-mechanical. Obviously, there is much that can be gained from control theory in the design and modeling of these components. Unfortunately, very little control theory has been used for this purpose so far. Compared with the highly developed theory of control systems, most control algorithms used in operating systems today are "primitive".

1.2 QUEUEING-THEORETIC VIEW OF AN OPERATING SYSTEM

Most models of computer systems used today are queueing-theoretic. From a queueing-theoretic viewpoint, each controller of the operating system is a server. Thus, an operating system is a queueing network. One very popular queueing model, called "Central Server Model", is shown in Figure 1.2. In this figure, circles represent servers and rectangles indicate the location of queues. Such queueing models have been used to explain many phenomena occuring in computer systems [Buz71]. Typical questions that have been answered using this approach are the following :

1. What is the average throughput?
2. What is the average utilization of the CPU, I/O devices etc.
3. What is the average response time?
4. What is the bottleneck in the system (would a higher speed disk do better)?
5. What is the optimal degree of multiprogramming?

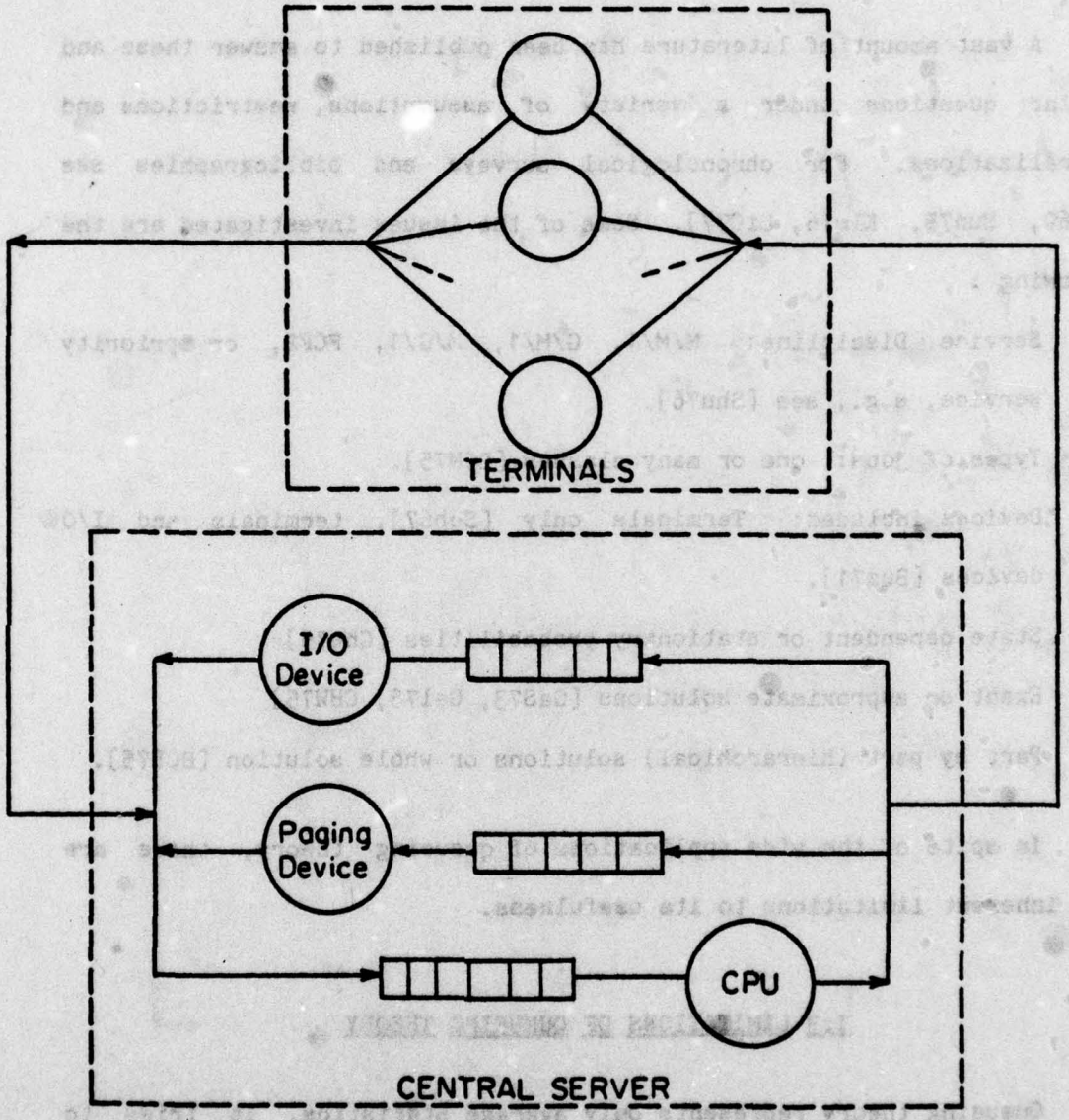


Figure 1.2 : A queueing-theoretic view of an operating system

A vast amount of literature has been published to answer these and similar questions under a variety of assumptions, restrictions and generalizations. For chronological surveys and bibliographies see [McK69, Mun75, Kle76, LiC77]. Some of the issues investigated are the following :

1. Service Discipline: M/M/1, G/M/1, M/G/1, FCFS, or priority service, e.g., see [Shu76].
2. Types of jobs: one or many classes [BCM75].
3. Devices included: Terminals only [Sch67], terminals and I/O devices [Buz71].
4. State dependent or stationary probabilities [Che75]
5. Exact or approximate solutions [GaS73, Gel75, CHW75]
6. Part by part (hierarchical) solutions or whole solution [BCB75].

In spite of the wide applications of queueing theory, there are some inherent limitations to its usefulness.

1.3 LIMITATIONS OF QUEUEING THEORY

Queueing theory represents only average statistics. It tries to represent a number of jobs by the average characteristics of the class. The "individuality" of a job is ignored. In this sense, it is a static analysis. It cannot satisfactorily represent time varying phenomena or dynamics. Therefore, it is good only as a design time tool. It cannot be used at operation time, for which we need adaptive techniques that can adapt to the individual characteristics and time-varying behavior of

jobs. To give a concrete example, a queueing model is ideal for telling whether the disk is the bottleneck in the system or whether a faster CPU will increase efficiency (both design time questions). However, once we have acquired the proper disk and CPU, it does not tell us which job from a given a set of jobs should be given the CPU or the disk next. This is a dynamic decision problem, which can only be solved by the application of techniques from decision and control theory.

Queueing theory is good for modeling a computer system and, to a certain extent, its subsystems. However, when we come down to the level of a program, it cannot model its behavior (because there are no queues to be modeled). Given all the known information about a program, it cannot tell what the program behavior is likely to be in the near future. This is a prediction problem. Again, control theory must be used for this purpose.

Queueing theory cannot model the interaction between the space and time demands of a program. Since the theory cannot model either the space demand behavior of a program or its time demand behavior, it certainly is inadequate for modeling the interaction between the two. Bad memory management may cause frequent page faults and may degrade the performance of an otherwise good scheduling policy. Still, the memory and the CPU allocation policies of most operating systems to date are more or less independent. This is due to a lack of clear understanding of the interaction between them. With the application of control theory we hope to remedy this situation, because, given control-theoretic

models of two systems, their joint model can be obtained by modeling the cross-correlation between the two.

1.4 ADDITIONAL EXPECTATIONS FROM CONTROL THEORY

There are many concepts like stability, controllability, and parameter sensitivity, that are well established in control theory but have not been used in computer systems modeling. We hope that the control-theoretic approach will eventually lead to a better understanding of these concepts as applied to computer systems. For example, take the concept of stability. Instability in computer systems occurs in the form of excessive overhead caused by frequent switching of CPU between jobs, or by frequent oscillation of pages between main and secondary memory. Instability in The control-theoretic approach is especially suitable for stability studies, e.g., for determining the effect of sudden demand variations, or the effect of measurement delays. There are well established techniques for this purpose.

Controllability studies of computer systems could similarly help us to determine whether it is possible to reach the optimum performance state. Parameter sensitivity is already a big issue even in current queueing models. One of the major studies that investigated the applicability of queueing models to a real interactive system was conducted by Moore at the University of Michigan [Moo71]. One conclusion of the study was that queueing models are very sensitive to parameter values which vary considerably with time and load variations.

Again, control theory with its well established techniques for sensitivity analysis provides better hope.

1.5 SURVEY OF APPLICATIONS OF CONTROL THEORY

Wilkes was probably the first to strongly advocate the exploitation of control theory for computer systems modeling. In his paper [Wil73], he stated:

> "We are not yet in a position, and perhaps never will be, to write down equations of motion for computer systems. However this does not exclude the design of a control system. Indeed, it is just in circumstances where the dynamical equation are not fully understood or when the system must operate in an environment that can vary over a wide range that control engineering comes into its own."

The paper presents many arguments for applying control theory. We do not intend to duplicate those arguments here. To illustrate his ideas, Wilkes proposed a general model of paging systems.

Adaptive policies for many components of operating systems have been proposed. Dynamic tuning of allocation policies to improve throughput in multiprogramming systems has been suggested by Wulf [Wul69]. An adaptive implementation of a load controller is described in [Wil71]. Blevins and Ramamoorthy have investigated the feasibility of a dynamically adaptive operating system [BlR76]. Two different techniques for adaptive control of the degree of multiprogramming have been described in [DKL76].

The need for a control-theoretic approach was also stressed by Arnold and Gagliardi [ArG74]. They proposed a state space formulation using resource utilization as the state variables. A dynamic programming approach to memory management and scheduling problems is described in [Lew74, Lew76]. A survey of some early applications of statistical techniques to computer systems analysis can be found in [Ash72].

The work most closely related to this thesis is that of Arnold [Arn75, Arn78]. Using correlation properties of the memory demand behavior of programs, he has investigated the applicability of the Wiener filter theory to the design of a memory management policy.

1.6 PRINCIPAL CONTRIBUTIONS AND ORGANIZATION OF THE THESIS

In this thesis we propose the following general control-theoretic approach to the formulation of resource management policies for operating systems.

1. In order to develop a resource management policy, model the corresponding program behavior as a stochastic process.
2. Using identification techniques and empirical data, identify a suitable model structure for the process* and estimate typical values of model parameters.

---

* The term *process* is used here exclusively in the control-theoretic sense of stochastic process. To avoid confusion, the term *task* is used to denote computer processes e.g., we say "ready tasks" instead of "ready processes".

3. Based on the model, formulate a prediction strategy for the stochastic process, and hence a resource management policy.

The policy so obtained is dynamic in the sense that it varies the allocation of the system resource to a user job depending upon the recent past behavior of the job. It, thus, provides the run time optimization not possible with the queueing theory approach. Also, notice that the individuality of the job is fully exploited. The key step in the approach is the formulation of the stochastic process model in such a way that the allocation problem reduces to a prediction problem. We exemplify this approach by formulating control-theoretic policies for CPU scheduling and page replacement. Policies for allocation of other shared resources (e.g., disks) can be, similarly, formulated.

Formulation of the CPU scheduling policy is described in Chapter II. The time taken by successive compute bursts of a program is modeled as a stochastic process. It is shown that the main problem is that of predicting the future demands of a job from its past behavior. A few analytical results are derived concerning the increase in the mean weighted flow time due to prediction error. Correlation techniques (also called time series analysis techniques) are used to identify a suitable model structure for the stochastic process. Empirical data on the CPU demand behavior of users of an actual time sharing system is used for this purpose. Details of the procedure used for modeling and parameter estimation from the data are included. In particular, it is

shown that the CPU demand process is a stationary stochastic process having very little autocorrelation. The efficiency of several autoregressive moving average (ARMA) models is compared. The final conclusion is that the gains are very small and that a zeroth order non-zero mean white noise (ARMA(0,0)) model is appropriate for the process. Based on this conclusion, several different predicion schemes are proposed. An adaptive scheduling algorithm called "Shortest Predicted Remaining Processing Time" (SPRPT) is proposed.

In Chapter III, the problem of page replacement is formulated as a prediction problem. Using a stochastic process model of memory demand behavior, suggested by Arnold [Arn75], an expression is derived for the cost of prediction error. The identification analysis shows that the process is non-stationary. The non-stationarity is, however, homogeneous in the sense that the first differences of the process are stationary. An autoregressive integrated moving average model of order 1,1,1 (ARIMA(1,1,1)) is shown to be an appropriate model for the process. A two step exponential predictor is derived for the model. Based on this predictor, a new page replacement algorithm called the "ARIMA" algorithm is proposed. Even though the origin of the algorithm lies in complex control-theoretic ideas, its final implementation is very simple. Moreover, it turns out that many conventional page replacement algorithms like the working set algorithm [Den68], Arnold's Wiener filter algorithm [Arn75], and the independent reference model [ADU71] are special cases of the ARIMA algorithm. The control-theoretic derivation of the conditions under which these algorithms are optimal is

presented.

Chapter IV is devoted to developing new techniques for analysis of binary processes like the memory demand process. The ARIMA model does not take into account the fact that a binary process takes only two values, 0 or 1. In this chapter, an attempt is made to remove this discrepancy. It is shown that if a binary process is Markov of a finite known order, it can be modeled as the output of a Boolean (switching) system driven by a set of binary white noises. Modeling, estimation, and prediction of the process using the Boolean model is described. A method is developed for optimal non-linear prediction under any given linear or non-linear cost criterion. All the results are then generalized to k-ary processes, i.e., processes which take integer values between 0 and k-1. The model is shown to be applicable to a class of non-stationary processes also. Finally, the application of the model to the problem of memory management is described.

In this thesis we make extensive use of control-theoretic terms and concepts. However, since a majority of the readers of the thesis are likely to be computer scientists, a tutorial approach is followed in deriving the control-theoretic results. Whenever possible, simple and intuitive explanations of the inferences based on control theory are provided. A brief explanation of ARIMA models, which are used extensively in this thesis, is given in Appendix A. Further details of control-theoretic concepts can be obtained from [Nel73, BoJ70, Ast70, BrH69].

CHAPTER II

A CONTROL THEORETIC APPROACH
TO
CPU MANAGEMENT

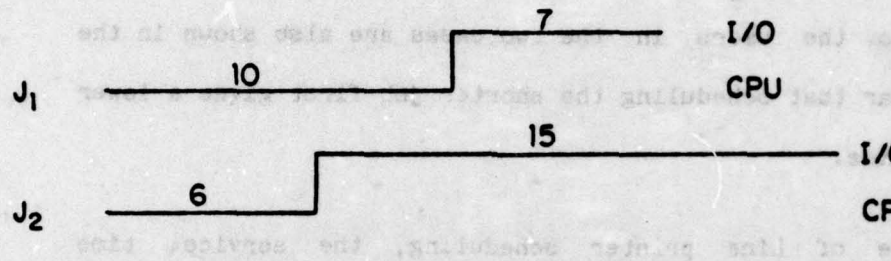
2.1 PROBLEM STATEMENT

The problem of CPU management is that of deciding which task from among a set of ready tasks be given the CPU next. In the literature this problem is also referred to as low level scheduling, short term scheduling, or process dispatching. There has been a considerable amount of work on designing scheduling strategies for optimizing different cost criteria, single or multiprocessor strategies,and for different precedence constraints among the jobs [Cof76]. A common underlying assumption in all these researches is that the CPU time required by each job is known. For example, the simplest scheduling problem is that of scheduling n independent tasks with known CPU time requirements of $t_1, t_2, \ldots, t_n$ respectively on a single processor in such a way as to minimize average finish time for all users. If the jobs were scheduled in lexicographic order (i.e., 1,2,...n), the average finish time would be

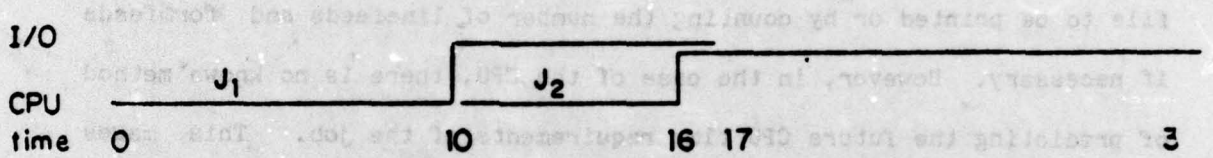
$$R = \frac{1}{n} \sum_{i=1}^{n} (n-i+1)t_i$$

A very well known solution to this problem is due to Smith [Smi56]. This solution is called "SPT" or Shortest Processing Time rule i.e., the jobs are given the CPU in the order of non-decreasing CPU demand. For those not familiar with this fact the following example should prove convincing.

Example : Consider scheduling two jobs J_1 and J_2 with each requiring only one cycle of computation followed by output. The time required for CPU and I/O are shown in Figure 2.1 . The scheduling decision is to



A. Job J_1 is scheduled first. Average Response time = $\frac{17+31}{2} = 24$



B. Job J_2 is scheduled first. Average Response time = $\frac{21+23}{2} = 22$

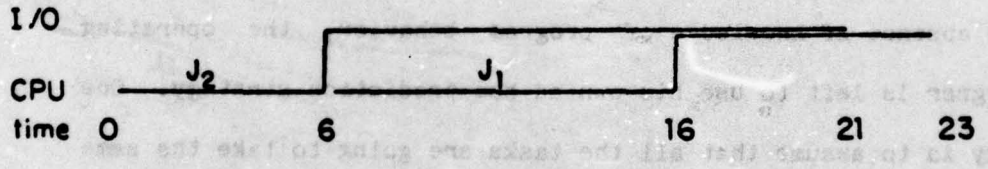


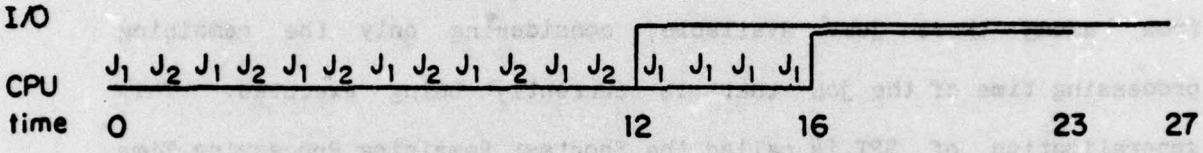
Figure 2.1: Optimal scheduling of two jobs

decide which of the jobs gets the CPU first. Obviously there are only two options: J_1 first, or J_2 first. The calculation of the average response times to the users in the two cases are also shown in the figure. It is clear that scheduling the shorter job first gives a lower average response time.

In the case of Line printer scheduling, the service time requirements can be predicted reasonably accurately from the size of the file to be printed or by counting the number of linefeeds and formfeeds if necessary. However, in the case of the CPU, there is no known method of predicting the future CPU time requirements of the job. This makes SPT and all similar scheduling strategies unimplementable.

In the absence of knowledge of program behavior, the operating system designer is left to use his own ad hoc prediction strategy. One such strategy is to assume that all the tasks are going to take the same (a fixed quantum of) time. The tasks are, therefore, given the CPU in a round robin fashion for the fixed quantum of time, and if a task has not completed by the end of the quantum, it is put back on the run queue. It is obvious that full-information strategies like SPT perform better than no-information strategies like the fixed-quantum round robin. This point is illustrated in Figure 2.2 where it is shown that if job J_1 happens to be the first in the queue the response time is 25; otherwise, it is 24.5. In both cases it is more than the SPT response time.

A. Round robin with J_1 first. Average Response time = $\frac{23+27}{2} = 25$



B. Round robin with J_2 first. Average Response time = $\frac{23+26}{2} = 24\frac{1}{2}$

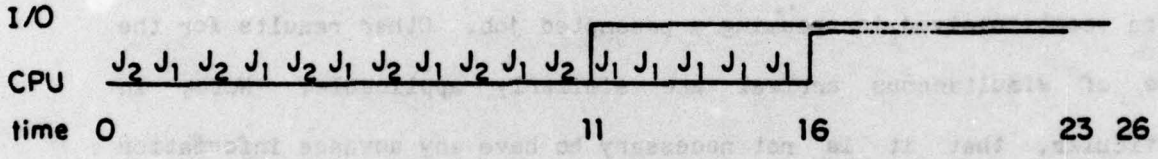


Figure 2.2: Round robin scheduling with unit quantum time

Up till now we have assumed that all the tasks arrive simultaneously and are ready for processing at the same time. Obviously, this is not the case in a real computer system, where, tasks arrive intermittently. The optimal scheduling strategy is still basically the same. At each point in time one makes the best selection from among those jobs available, considering only the remaining processing time of the job that is currently being executed. This generalization of SPT is called the Shortest Remaining Processing Time (SRPT) rule [Sai78]. This minimizes the mean flow time if there is no extra cost involved in resuming a preempted job. Other results for the case of simultaneous arrival are similarly applicable. Note, in particular, that it is not necessary to have any advance information about job arrivals.

2.2 CONTROL THEORETIC FORMULATION

Consider a program in a uniprogramming situation. Figure 2.3 shows the typical time behavior of the program*. The program oscillates between CPU and I/O devices (Disk, Teletype, Card reader, Magnetic tape etc.). Most program have three phases. During the first phase they do very little computation, spending most of the time collecting parameter values from the user. The program then enters a computation phase consisting generally of one or more loops. Finally, the program outputs the results. The computation phase constitutes a major portion of the life of the program. The cyclic nature of this phase (due to loops) makes the program behavior somewhat predictable. While in a loop, the program repeatedly references the same set of pages, and makes similar CPU and I/O demands. Under the name of "Principle of Locality", this behavior has been successfully exploited for memory management. The working set strategy of memory management is partly based on this principle. This strategy states that the set of pages referenced during the last time interval T are more likely to be referenced in the near future than other pages.

The CPU management equivalent of the WS strategy is to say that the length of the last CPU burst is the likely length of the next CPU burst. This strategy has been used in many operating systems, though there are many different forms of its implementations. One

---

* The same is applicable to a program in a multiprogramming situation provided the time scale represents "virtual time".

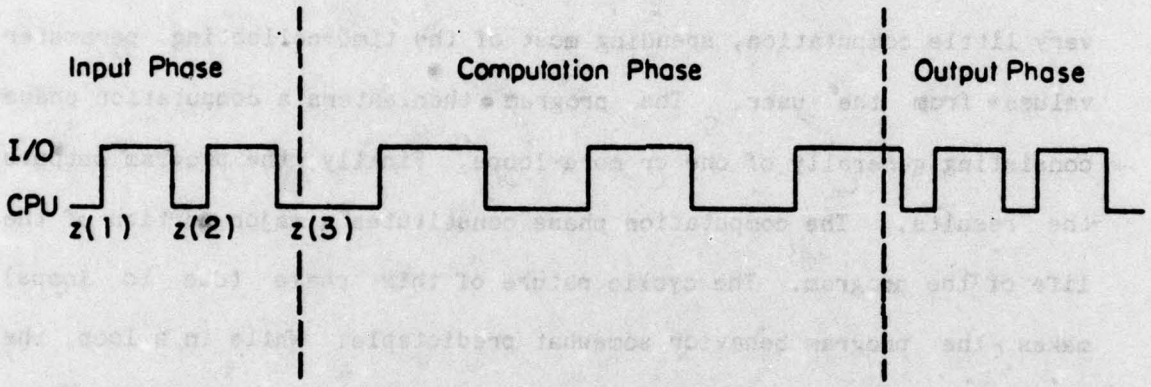


Figure 2.3: CPU and I/O demands of a typical program

implementation method is to put a job taking a lot of CPU time on a low priority queue so that the next time it will get the CPU only afer those jobs which have taken less CPU time this cycle. Unfortunately, this principle, although commonly used, has never been theoretically explained.

One aim of the research reported here was to check the validity of this "Next Equal to Last" (NEL) principle, and, if it was found invalid, to find a strategy for the best prediction of the future CPU demand of a program from its past behavior. We model the CPU demands of a job as a stochastic process. The k^{th} CPU burst is modeled as a random variable z(k). One way of representing a stochastic process is to model it as the output of a control system driven by white noise (see Figure 2.4). Thus, as seen by the CPU scheduler, the program is like a control system which generates successive CPU demands. A general time series model for such a process is given by the following equation:

$$z(t) = f(z(1),z(2),\ldots,z(t-1),e(1),e(2),\ldots,e(t))$$

Where z(t) represents t^{th} CPU burst and e(t) is the t^{th} random shock. A linearized and time invariant form of the above equation is the well known ARMA(p,q) model (see Appendix A for details on ARMA models) :

$$z(t) = w+a_1z(t-1)+\ldots.+a_pz(t-p)+e(t)-b_1e(t-1)-\ldots-b_qe(t-q)$$

We choose this formulation to model the CPU demand behavior of programs, because there are well established techniques to find such models from empirical data. Once a suitable ARMA model is found, it is easy to convert it to other models (e.g., state space model), if necessary.

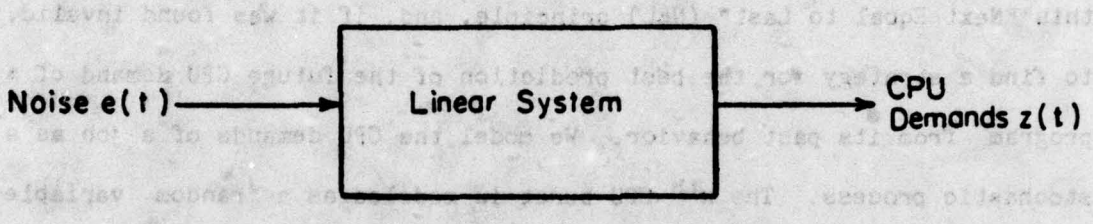


Figure 2.4: CPU demands modeled as a stochastic process

2.3 EFFECT OF PREDICTION ERRORS

In order to study the effect of prediction errors, we need to choose a performance measure. Consider the problem of scheduling n independent tasks with CPU time requirements of $t_1, t_2, \ldots, t_n$ respectively on a single processor. A schedule consists of specifying the sequence in which the tasks should be given to the processor. There are many different performance measures for comparing different schedules. The measure most commonly used for single processor scheduling is "Mean Weighted Finish Time" (MWFT). It is defined as follows:

$$c = \frac{1}{n} \sum_{i=1}^{n} w_i f_i$$

Where f_i is the finishing time of i^{th} task and w_i is the weight or deferral cost of the task. It was shown by Smith [Smi56] that this cost criterion is minimized by arranging the tasks in the order of non-decreasing ratio t_i/w_i. If all the tasks have equal deferral costs, i.e., vi w_i =1, then the cost c is called average finishing time or average response time. It follows from the above that the average response time is minimized by sequencing the tasks in the order of non-decreasing t_i. This rule is commonly known as "Shortest Processing Time" (SPT) rule.

It has been shown that SPT also minimizes the following cost criteria [CMM67]:

1. Mean power of finishing time $\frac{1}{n} \sum_{i=1}^{n} f_i^k$

2. Mean waiting time $\frac{1}{n}\sum_{i=1}^{n}(f_i-t_i)$
3. Mean power of waiting time.
4. Mean lateness (time beyond deadline).
5. Mean tardiness if all jobs are tardy.
6. Mean number of tasks waiting

However, SPT does not optimize the following cost criteria (all of which are functions of due dates):

1. Maximum lateness
2. Maximum tardiness
3. Mean tardiness
4. Number of tardy jobs.

Fortunately, due dates are rarely, if ever, specified for CPU scheduling and hence the above criteria are of no practical interest. For a computer user the most important criterion is a low response time*. Since a job consists of several CPU-I/O cycles (or CPU-I/O tasks), the response time is the sum of the finishing time of these tasks of the job. An increase in the finishing time of a task directly contributes to an increase in the response time.

---

* Some researchers believe that it is the consistency of response time rather than minimality that is of concern to a user [HoP72]. For example, if a program takes 1 minute on one day, it is quite bothersome to the user if it takes 5 minutes on another day. However, the proper control point for this criterion is load control (control of the number of users allowed to log in or the number of batch jobs allowed to run simultaneously). Therefore, we do not consider this criterion.

In the following, we derive a few analytical results concerning the increase in mean weighted finishing time (MWFT) of tasks due to prediction errors. We first consider a very general case where the time requirements of all jobs are to be predicted. Then we consider another case, where only one job is considered for prediction, the compute time requirements of other jobs is assumed to be known. The results are presented as Theorems 2.3.1 and 2.3.2 below. The proofs of theorems are given in Appendix B.

2.3.1 Theorem [Non-deterministic Case] : Consider a set of n tasks T_0, T_1, ..., T_{n-1} with compute time requirements of t_0, t_1, ..., t_{n-1} respectively, where all the times are unknown and are predicted as f_0, f_1, ..., f_{n-1} etc. The predictor is such that the predicted time f_i is a random variable with distribution $F_i(f_i)$. The increase in the mean finishing time (MFT) due to prediction error is given by

$$c = \frac{1}{n} \sum_{i=0}^{n-1} (i-\hat{i})\, t_i$$

where $\hat{i}$ = Predicted position of T_i

$$= \int_0^{\infty} [\sum_{\substack{j=0 \\ j \neq i}}^{n-1} F_j(t)]\, f_i(t)\,dt$$

2.3.2 THEOREM [Deterministic Case] : Given a set of n tasks $T_0, T_1, \ldots, T_{n-1}$ with compute time requirements of $t_0, t_1, \ldots, t_{n-1}$ respectively, where $t_1, \ldots, t_{n-1}$ are known exactly and t_0 is predicted as t_p, then the increase in mean weighted finishing time (MWFT) due to prediction error is given by:

$$c = \frac{1}{n} \sum_{k \in I} |(w_0 t_k - w_k t_0)|$$

Where $I = \{k : \frac{t_k}{w_k} \in J\}$

$$J = \begin{cases} (t_0/w_0,\ t_p/w_0) & \text{if } t_0 < t_p \\ (t_p/w_0,\ t_0/w_0) & \text{if } t_p > t_0 \end{cases}$$

Informally, I is the set of indices of tasks lying between the predicted and the real position of T_0. J is the interval between t_0/w_0 and t_p/w_0.

2.3.2.1 corollary : The increase in mean finishing time ($w_k=1$ w_k) due to t_0 predicted as t_p is given by :

$$c = \frac{1}{n} \sum_{k \in I} |t_k - t_0|$$

where $I = \{k : t_0 < t_k < t_p \text{ or } t_p < t_k < t_0\}$

One implication of this corollary is that only those tasks that lie in between the predicted and actual position of the task contribute to the increase in MFT. Thus if the compute time of various tasks are arranged in increasing order and plotted as shown in the Figure 2.5, then the increase in MFT is represented by the hatched area. In the special case, when these compute times are linearly increasing, the increase in MFT is proportional to square of the prediction error. This fact is stated by the following corollary whose proof is given in Appendix B.

2.3.2.2 corollary : If $t_k = kT$, $k=1,2,\ldots,n-1$ then the increase in MFT due to t_0 predicted as t_p is given approximately by:

$$c \approx \frac{1}{2nT}(t_0 - t_p)^2$$

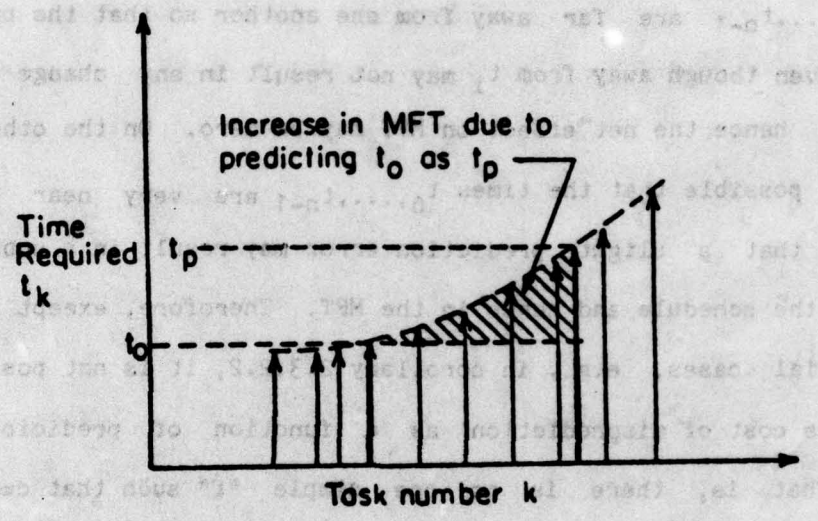


Figure 2.5: Increase in MFT due to prediction error

It is seen from above theorems that the error in prediction of computer time of a job affects the relative placement of all other jobs in a very complicated fashion. For example, it is possible that the time $t_0,\ldots,t_{n-1}$ are far away from one another so that the predicted value t_i even though away from t_i may not result in any change in the order and hence the net effect on MFT may be zero. On the other hand, it is also possible that the times $t_0,\ldots,t_{n-1}$ are very near to each other so that a slight prediction error may result in a substantial change in the schedule and hence in the MFT. Therefore, except in some very special cases, e.g., in corollary 2.3.2.2, it is not possible to express the cost of misprediction as a function of predicion error alone. That is, there is no one simple "f" such that $c=f(t_0,t_p)$ represents the loss function. We, therefore, choose to use the conventional least square criterion to predict the compute time. In other words, we seek to predict in such a way that the average value of square difference between the predicted and the actual value is minimum.

2.4 DATA COLLECTION

This section describes the experiment to collect data on CPU demands of actual programs. The experiment was conducted on a real user environment in our Aiken Computation Laboratory. The laboratory has a DECsystem-10 computer with TOPS-10 operating system. The system is mainly a research facility for use by graduate students.

The TOPS-10 operating system maintains a number of queues among which the jobs are distributed. For example, there is a queue for jobs waiting to be run, a queue for jobs waiting for disc I/O, a queue for jobs waiting for TTY I/O etc. Thus. the easiest way to get the data we require is to watch the queue history of the program i.e., to note the queue the job is in and to repeat the observation at every clock tick*.

Table 2.1 gives major details of the experiment. It consisted of 19 different runs spread over a month. Each run consisted of randomly selecting a user and watching his queue history for a period of about 45 minutes. Along with the queue history which was observed every clock tick, many other parameters like program name, memory used, etc. were also recorded every second.

The data was later translated to produce the CPU demand processes of individual programs. This produced 550 CPU demand processes consisting of the length of successive CPU bursts (total CPU usage

---

* In all subsequent discussions, the unit of time will be a clock tick called "jiffy" in DEC terminology. A jiffy is the cycle period of the line power supply i.e., 1/60th of a second.

TABLE 2.1 : DATA COLLECTION EXPERIMENT

Duration of the experiment	1 month
Number of runs	19
Duration of each run	45 minutes
Number of programs observed	550
Number of programs with 80 or more bursts	33
Number of program histories analyzed	33
Number of user histories obtained	19
Number of user histories analyzed	8

between successive I/O requests). However, most of these program processes were too short i.e., consisted of only a small number of observations (number of CPU bursts). Only 33 processes had 80 or more observations. These were chosen for correlation analysis.

We also obtained 19 user processes - one for each run. These consist of lengths of successive CPU demands of the user regardless of the program being run. Of these user processes, alternate (actually only 8) processes were selected for analysis. The list of the processes selected for analysis is shown in Table 2.2 . The processes are named "XXXXX.YNN" where XXXXX is either "USER" for user processes or the program name. Y is the user identification (letters A, B, C,...) and NN is the serial number of the program in a particular run. Thus MAIN.R55 stands for the 55th program run by user R. "MAIN" is the name of the

program. Table 2.2 also gives the type of the program, number of observations in the process, its mean value, standard deviation s_z, and P-VALUE. The term P-VALUE will be explained later under the Chi-square test.

The developmental nature of the environment is obvious from the table. Notice that 14 (42%) of the programs are editing (SOS and TECO), 7 (21%) are FORTRAN programs, and 4 (12%) are EL1 programs. FORTRAN and EL1 are the main languages used at our laboratory. Most users follow a cycle of editing (TECO), compiling (FORTR), and running the program, and then reediting etc. This is typical of research and development environments. In a production environment in an industry, less amount of editing and more application program execution is expected. However, as we shall see later, the CPU demand behavior of editing programs and application programs are not very different except that the mean value of CPU burst in an editing program tends to be much lower than that in an application program. Therefore, it is plausible that the results obtained here also hold in a production environment.

Table 2.2 : List of CPU Demand Processes Analyzed

S. No.	Process Name	N	$\bar{z}$	s_z	P-VALUE Chi-sq	Program Type
1.	COMP2.G63	158	24.8	85.0	0.000	FORTRAN Program
2.	ECL.B1	81	11.2	18.9	0.241	EL1 Program
3.	ECL.B2	260	74.7	379.9	0.006	EL1 Program
4.	ECL.S1	448	49.0	308.9	0.997	EL1 Program
5.	ECL.S2	270	77.6	528.5	0.999	EL1 Program
6.	FORTR.P21	113	5.4	7.8	0.178	FORTRAN compiler
7.	FORTR.P30	234	6.3	7.6	0.412	FORTRAN compiler
8.	FORTR.P8	349	6.2	6.7	0.000	FORTRAN compiler
9.	FORTR.Q17	253	5.2	6.1	0.001	FORTRAN compiler
10.	FRCDO.C1	141	606.7	1298.0	0.047	FORTRAN Program
11.	FRCDO.C11	158	184.2	578.8	0.000	FORTRAN Program
12.	M786S.U1	504	1.5	5.7	0.000	FORTRAN Program
13.	MAIN.Q10	204	8.4	7.4	0.000	FORTRAN Program
14.	MAIN.Q19	98	8.2	5.9	0.000	FORTRAN Program
15.	MAIN.R55	129	3.4	3.4	0.230	FORTRAN Program
16.	P.A19	222	9.2	23.4	0.000	FORTRAN Program
17.	PIP.G18	140	1.1	0.7	0.088	Peripheral I/O
18.	PIP.G45	84	1.0	0.8	0.000	Peripheral I/O
19.	PIP.G60	225	0.9	0.3	0.000	Peripheral I/O
20.	SOS.A21	422	1.9	2.7	0.000	Text Editor
21.	SOS.A22	85	2.0	2.7	0.646	Text Editor
22.	SOS.A23	103	1.5	1.3	0.374	Text Editor
23.	SOS.A6	110	2.8	3.1	0.606	Text Editor
24.	TECO.B8	90	3.7	6.6	0.916	Text Editor
25.	TECO.F1	92	2.7	4.6	0.540	Text Editor
26.	TECO.F20	199	5.7	6.3	0.018	Text Editor
27.	TECO.G37	116	28.0	122.2	0.272	Text Editor
28.	TECO.G38	221	17.2	64.8	0.140	Text Editor
29.	TECO.G55	114	4.3	4.5	0.000	Text Editor
30.	TECO.H1	168	2.2	2.8	0.001	Text Editor
31.	TECO.J5	90	6.4	6.4	0.174	Text Editor
32.	TECO.P1	138	4.6	12.8	0.979	Text Editor
33.	TECO.P13	84	4.3	5.5	0.568	Text Editor
34.	USER.B	587	37.0	257.7	0.000	
35.	USER.D	259	52.0	329.2	1.000	
36.	USER.F	680	5.5	17.5	1.000	
37.	USER.H	413	6.5	39.9	1.000	
38.	USER.L	372	4.2	7.8	0.000	
39.	USER.N	471	30.2	187.5	0.999	
40.	USER.P	1629	7.1	17.5	0.000	
41.	USER.T	262	25.1	112.0	0.554	

2.5 DATA ANALYSIS

The aim of data analysis is to find one suitable model structure for CPU demand behavior of programs. The two main steps of data analysis are model identification and parameter estimation. The first step consists of studying the first and the second order statistics of the data in order to identify a class of models suitable for the process. In the second step, these models are fitted to each process to find the maximum achievable gain Finally, these different models are compared to give one general model for all CPU demand processes. A large part of the data analysis reported here was done on a time series analysis package TS developed by Professor Vandalae of Harvard Business School.

Statistical techniques are very often misused and results misinterpreted. It is easy to dr}w misleading conclusions unless the statistical procedures are fully understood and used properly. for example, we have noticed that in most of the computer science literature, correlation techniques are used witnout significance tests, parameters estimated without their confidence intervals. and so on. We, therefore, decided to explain the methodology along with the results. In the following we have tried to describe the reasoning behind each inference that we draw. The description is, however, brief due to space limitations and references are provided for further details whenever necessary.

2.5.1 Model Identification:

Model identification consists of studying the characteristic behavior of the first and the second order statistics of the data. The goal of this step is to identify a model structure or a class of models suitable for the data. Notice that this does not include finding an exact model equation; that is part of the next step on parameter estimation. The statistics used for model identification in this analysis are data plots, autocorrelations, partial autocorrelations, inverse autocorrelations, and Chi square test. The inferences drawn from these statistics are now described.

2.5.1.1 Data Plot :

The very first step in any identification procedure must be to plot the data and to study its general time behavior. The plots of CPU demands of some of the programs analyzed are shown in Figure 2.6 . These are typical of all the programs analyzed. Very often a program has just one or two large CPU bursts which if plotted would obscure the details at lower values. Therefore, the Y-axis scales have been so chosen that at least 95% of the data are shown in the graph. Very large value are shown cut off at the largest plottable value. Notice the following characteristic behavior of these graphs:

A. No Trend : A trend (monotonous increase or decrease) in the data is an indication of non-stationarity, though its absence does not confirm stationarity. For a stationary series, the mean of the data does not depend upon time; it is constant. Therefore, such a series takes trips

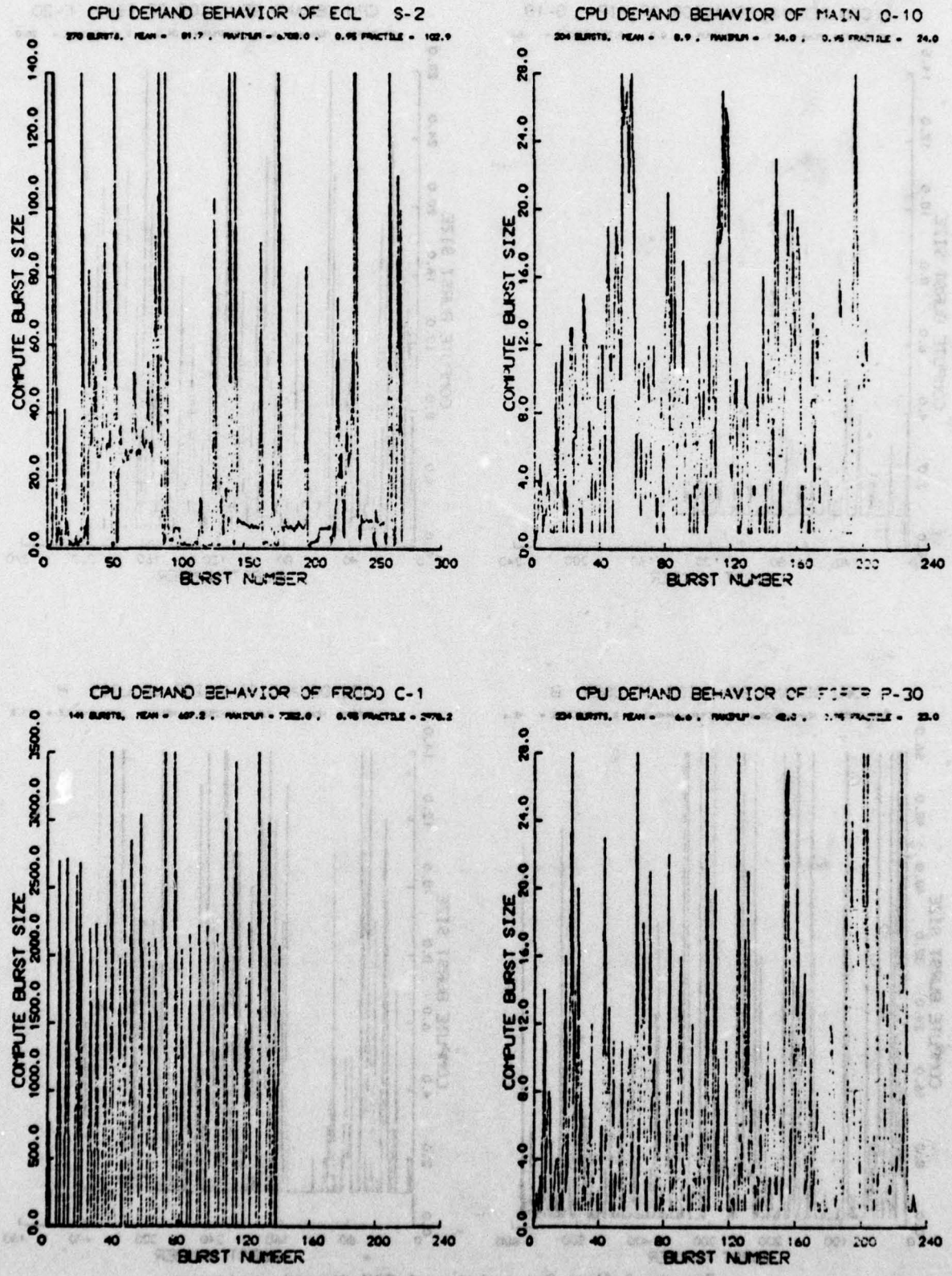


Figure 2.6a : Data plots of CPU demand processes

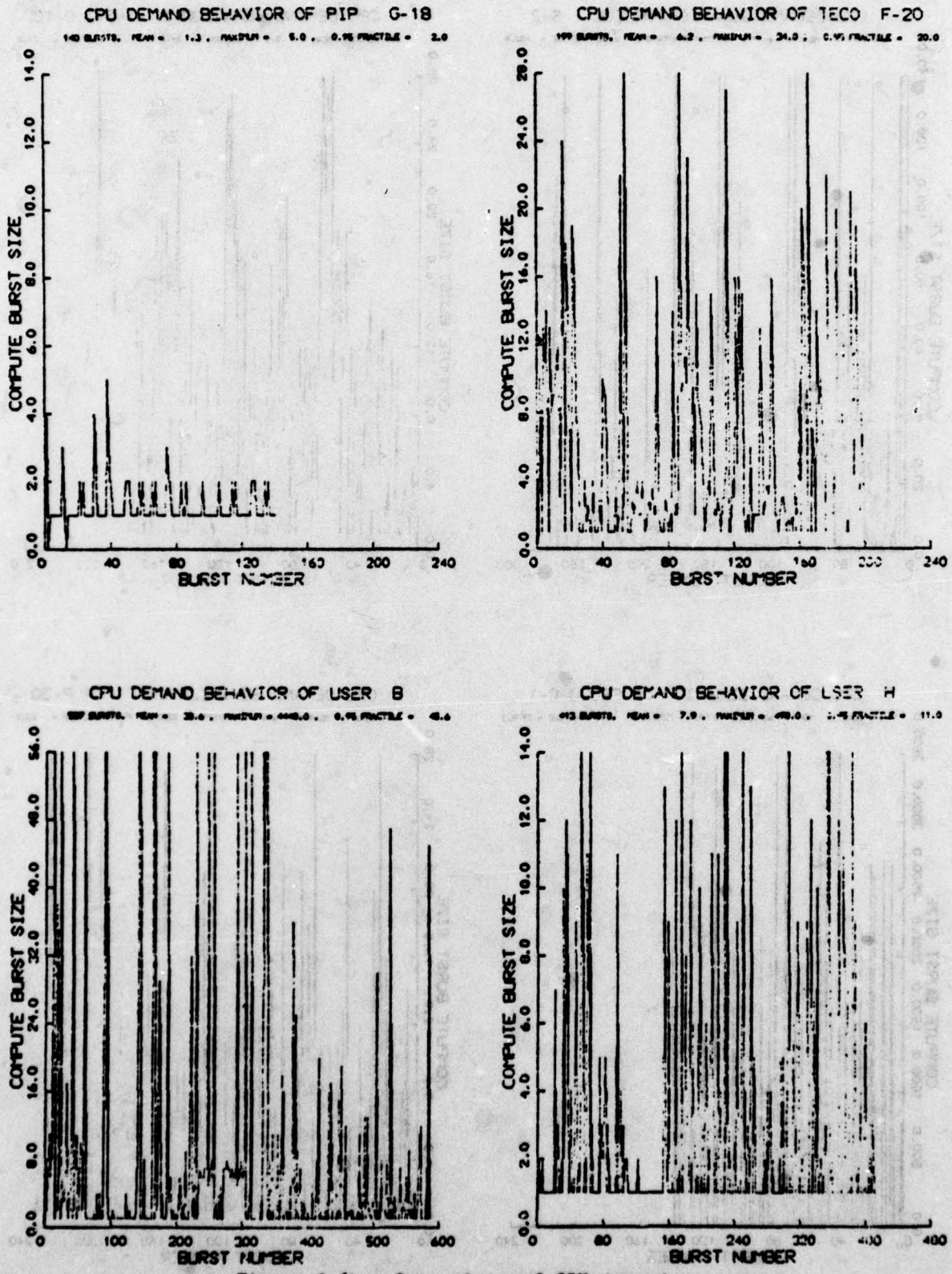


Figure 2.6b : Data plots of CPU demand processes

away from the mean, but it returns repeatedly during its history. Fortunately, none of the CPU demand processes show a trend. Thus we can hope for stationarity. A more conclusive test of stationarity via the autocorrelation function will be described in the next section.

B. Violent Variations : Notice that the series does not stay at any one level even for short intervals. This indicates that a WS-type prediction scheme ($\hat{z}(t+1)=z(t)$, i.e., the current CPU burst size is a good estimate of the next one) is probably not very valid. We may have to use some more sophisticated scheme.

2.5.1.2 Autocorrelation Function :

As the name implies, the autocorrelation function is a measure of the correlation between the present and the past observations. It is therefore, also a measure of the predictability of future from the past. Mathematically, the autocorrelation function is the normalized autocovariance function. The latter is defined as follows:

$$Cov(k) = E[(z(t)-z)(z(t+k)-z)]$$

By dividing the autocovariance function by the variance (Cov(0)) we get the autocorrelation function C(k):

$$C(k) = Cov(k)/Cov(0)$$

Obviously, to be of any value, a stochastic process should have finite memory, i.e., the present observation must be correlated only with those in the finite past. In other words, the autocorrelation function should die down to zero at very large lags. Such processes are called stationary because after a while they achieve "equilibrium" and

their behavior does not depend upon initial conditions (long past).

Autocorrelation functions (ACF) of some of the CPU demand processes are shown in Figure 2.7 . These are typical of all the programs analyzed. The dashed lines indicate the 95% confidence interval of the ACF for the given sample. The expression given above for C(k) is valid only for infinite sample sizes. For finite sample sizes the calculated values are only an approximation to the theoretical ACF. Thus if r(k) denotes the standard deviation of C(k), then a calculated value for theoretically zero autocorrelation (C(k)=0) may lie anywhere between $0 \pm 1.98 r(k)$ with 95% probability. In simple words, any value between the dotted lines can be effectively assumed to be zero with 95% confidence. The variance r(k) can be calculated by Bartlett's formula [Bar46]. In computer science literature, this significance test is almost always omitted, resulting in misleading conclusions.

The characteristic features of the ACF and the inference that we can draw are now described.

A. The ACF dies down to zero very quickly. This indicates that the CPU demand process is stationary. If the ACF had not died down quickly, we would have had to analyze the ACF of the first and higher differences of the process.

B. The ACF is non-zero only for 1 or 2 lags. We can, therefore, restrict our consideration to MA models of order less than 2.

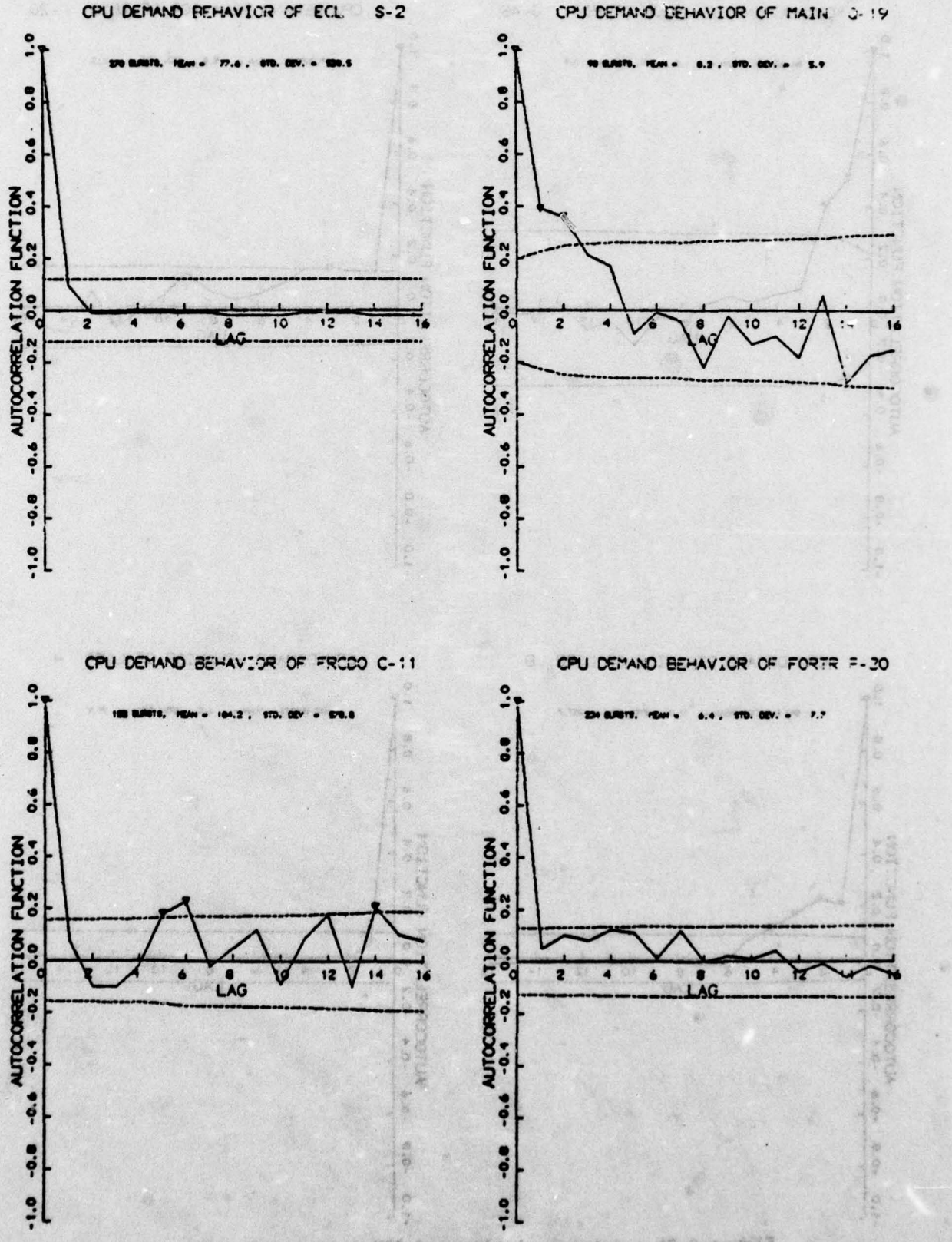


Figure 2.7a: Autocorrelations of CPU demand processses

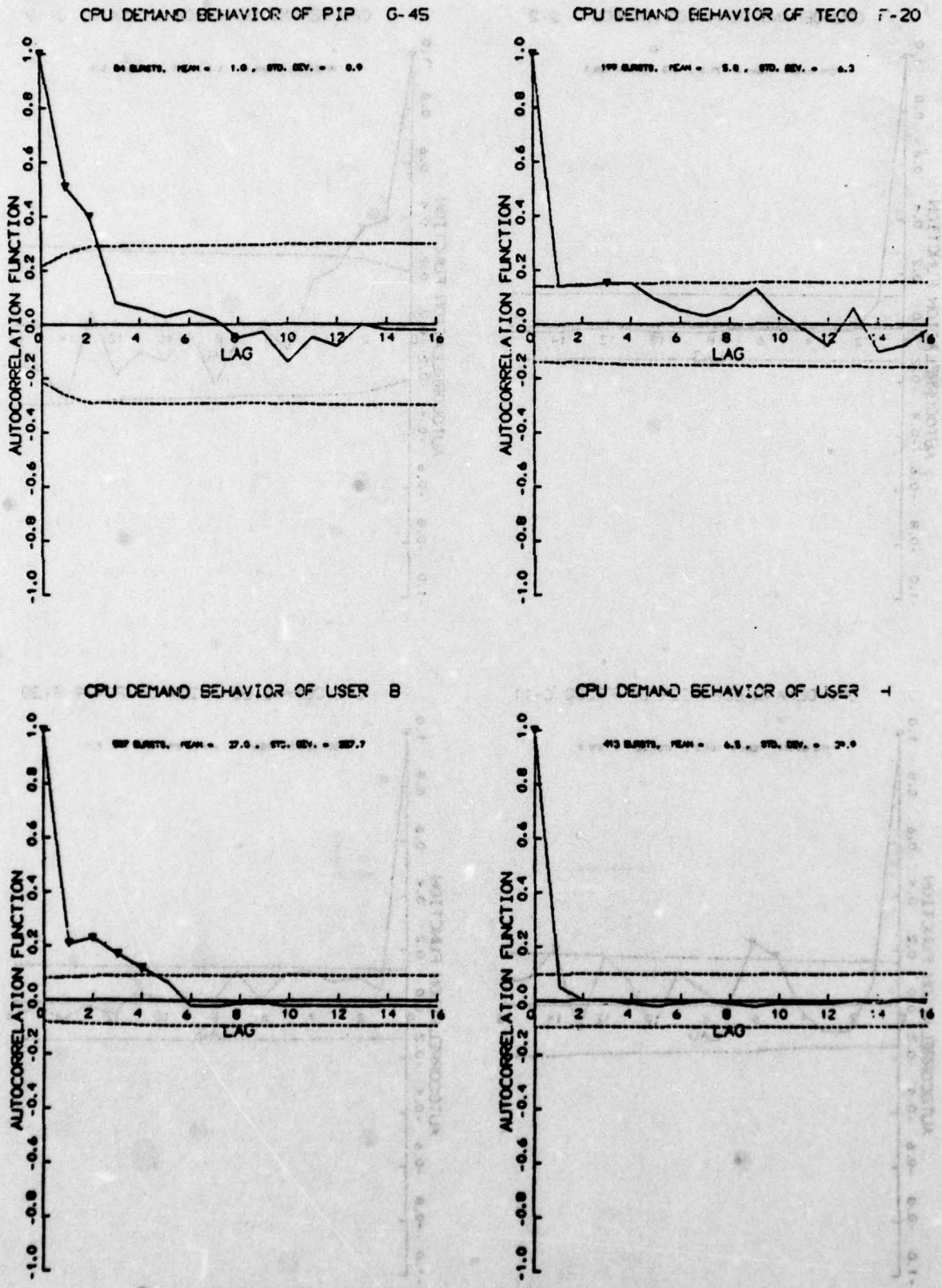


Figure 2.7b : Autocorrelations of CPU demand processes

It is very important to remember that the sample autocorrelations are only estimates of the actual autocorrelations for the process which generated the data at hand. Therefore, the analyst must be on the look out for general characteristics which are recognizable in the sample correlogram and not automatically attach significance to every detail. For example, there is a 5% probability that a theoretically zero correlation will show up as significant (above the dashed lines). Therefore, one or two significant correlations at large lags in some of the cases shown should not alarm us.

C. The ACF is positive. A positive correlation between successive values indicates that a large CPU demand in one cycle implies a large demand in the next cycle. Therefore, a program that took a long CPU time during last cycle can be expected to be CPU bound at least for the next cycle and put on a lower priority queue.

D. The value of ACF is rather small. The ACF at lower lag values even though non-zero and positive is really very small (of the order of 0.1). This partially dulls the hope expressed in the last inference. The correlation being small, the gain in the predictability of the future from the past will be small. In control theoretic terms, we are, perhaps, headed for a zeroth order model.

2.5.1.3 Partial Autocorrelation Function :

The PACF is the dual of the ACF. Like the ACF gives an idea of the order of the MA models, PACF gives an idea of the order of AR models. If the process is modeled by an AR model of order p:

$$z(t) = w + a_1 z(t-1) + a_2 z(t-2) + \ldots + a_p z(t-p) + e(t)$$

Then the coefficient a_p of the last AR term z(t-p) is defined as the value of PACF at lag p. Naturally, if the real process generating the data had an AR model of order n, then we would expect PACF to be zero at all lags greater than n. Thus the cut-off point of the PACF gives the order of AR model.

The PACFs of some of the CPU demand processes are shown in Figure 2.8 . The dashed lines indicate the 95% confidence interval for the PACF for given sample sizes. It was shown by Quenouille [Que49] that the approximate standard error of the PACF is $n^{-0.5}$. The characteristic attributes of these PACFs and their implications are now described.

A. The PACF dies down to zero very quickly. In fact in most cases the PACF is significant (above the dashed lines) only for lags 1 or 2. This means that we do not have to bother about very high order AR models to model these processes. A first or second order model will do.

B. The PACF is positive at low lags. Notice that the PACF for almost all processes is positive at lag 1. Only in 1 or 2 cases is PACF(1) negative. The positive value implies that a CPU bursts gives a positive contribution to the estimate of the next burst. It therefore confirms our previous conclusion that a large CPU burst is more likely to be followed by another large burst.

2.5.1.4 Chi Square Test of Randomness :

One way of viewing the process of modeling a time series is as an attempt to find a transformation that reduces the observed data to

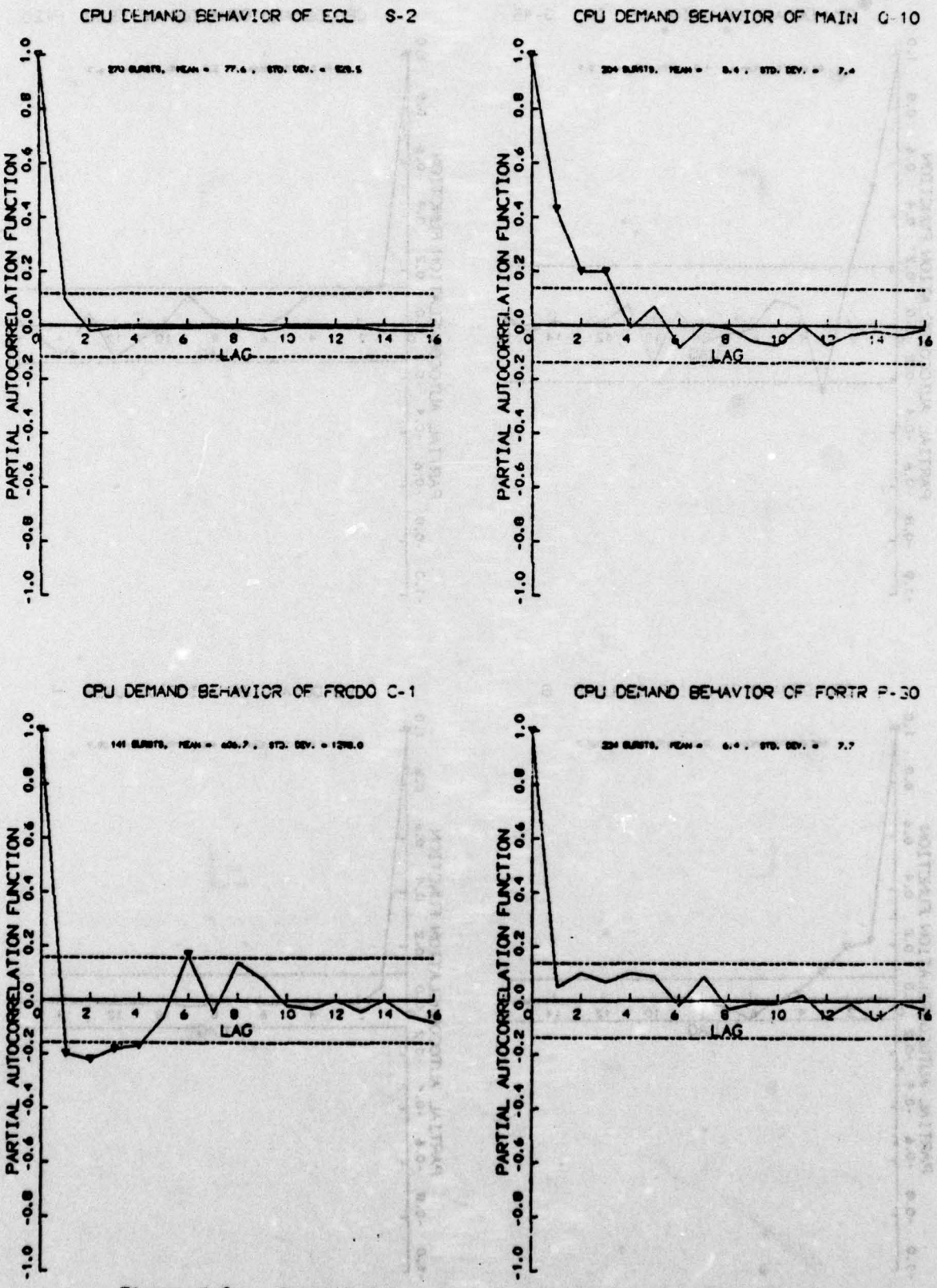


Figure 2.8a : Partial Autocorrelations of CPU demand processes

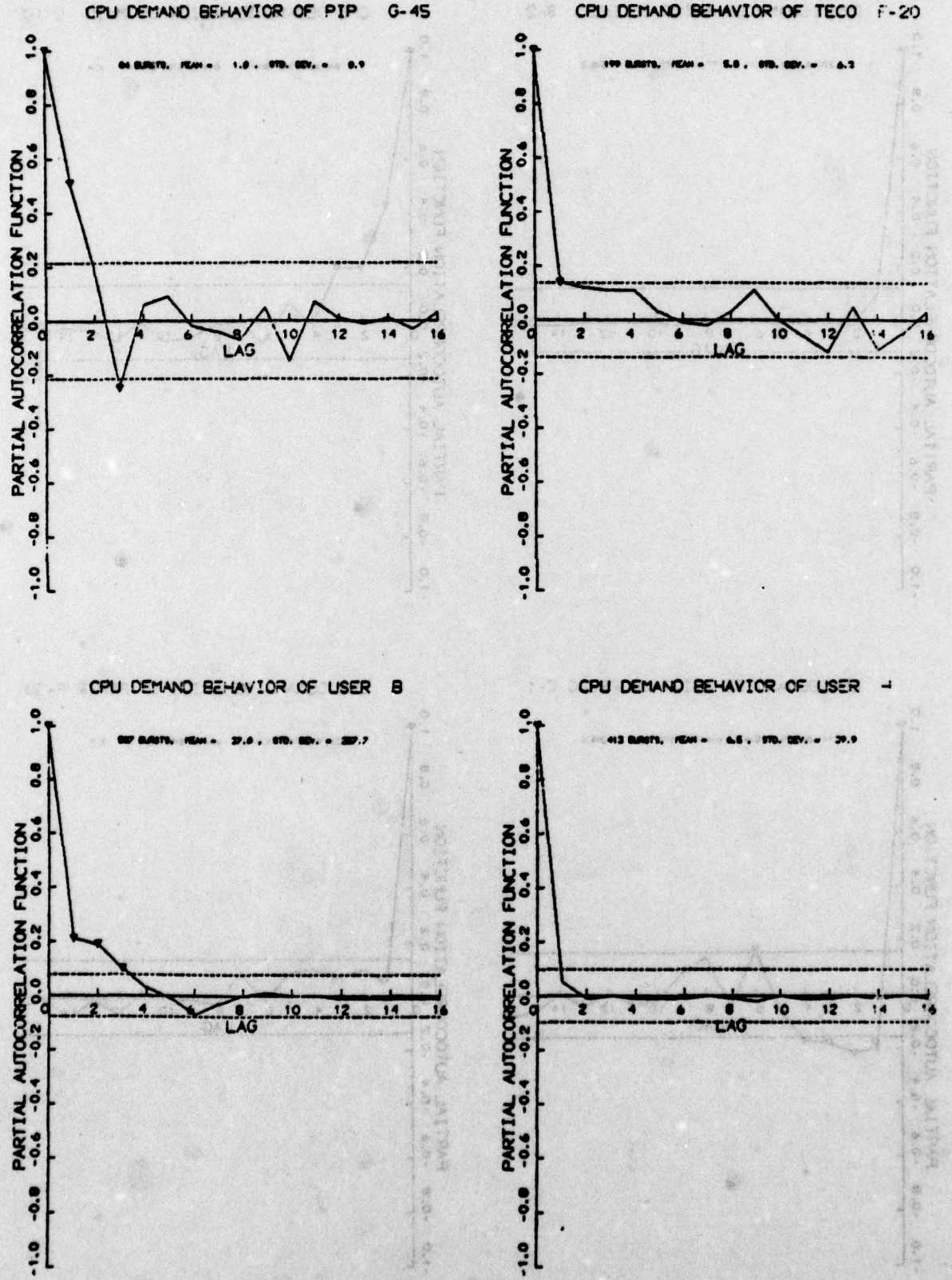


Figure 2.8b : Partial Autocorrelations of CPU demand processes

random noise. The first question, therefore, is whether the data itself is a random noise. Theoretically, the autocorrelation of random noise will be zero at all lags. In practice, it will have small non-zero values. Bartlett's formula for the standard error of the ACF provides some guidance to test the smallness. A better quantitative test of randomness is due to Box and Pierce [BoP70]. They have suggested a statistic that offers a test of the smallness of a whole set of sample autocorrelations for lags 1 through k. This is the Q statistic given by

$$Q = N \sum_{j=1}^{k} C(j)^2$$

Q is approximately Chi-square distributed with k degrees of freedom. Using the Q statistic one can calculate the probability that the given sample came from a white noise process. This probability is listed in the Table 2.2 under P-VALUE. Notice that 22 of the 33 processes analyzed have non-zero P-VALUE, 16 have P-VALUE greater than 10%, and 8 have P-VALUE greater than 50%. Of the 8 user processes analyzed 4 have a P-VALUE of 1, i.e., they are almost surely random noises. The high randomness of the user processes is a result of their being mixtures of several program traces, many of which have no relation to one another.

2.5.1.5 Inverse Autocorrelation Function :

The inverse autocorrelations of a time series are defined to be the autocorrelations associated with the inverse of the spectral density of the series, i.e.,

$$\text{IACF} = \text{Inv. Fourier Transform}\left[\frac{1}{\text{Fourier Transform(ACF)}}\right]$$

The IACFs were first proposed by Cleveland [Cle72]. He claims that they

are useful in identifying non-zero coefficients in an ARMA model. However, their utility in model identification is still a point of debate among statisticians [Par72]. We calculated the inverse autocorrelation functions for all of our CPU demand data processes. However, in most cases these functions did not give much additional information. Only in some (2 or 3) cases, where the processes behaved abnormally (a low order ARMA model was not adequate), did we gain some insight into modeling these particular cases.

In order to illustrate the use of IACF, let us consider one such case : the CPU demand behavior of program FRCDO.C11 . Its ACF and PACF were insignificant everywhere except at lags 5, 6, and 14. Obviously, a low order ARMA model would not work for this process. As we will see in the next section of model fitting, that an AR(2) model resulted in only 1.6% improvement over a zeroth order model. The inverse autocorrelation for this process (assuming orders of 1 through 8 for the AR part of the model) are shown in Table 2.3 . Notice that all columns except 5 and 6 are zero. Cleveland suggests that this indicates an appropriate model would have a 6th order AR part with only 5th and 6th coefficients non-zero and all other coefficient zero, i.e., a model of the following type :

$$z(t) = w + a_5 z(t-5) + a_6 z(t-6) + e(t)$$

Obviously, these high order models are of no interest to us because of their applicability only in rare cases, and also because of the rather small gain even in these cases.

Table 2.3 : Inverse Autocorrelations of FRCDO.C11

m	ri(1)	ri(2)	ri(3)	ri(4)	ri(5)	ri(6)	ri(7)	ri(8)
1	-0.076							
2	-0.091	0.100						
3	-0.074	0.087	0.080					
4	-0.072	0.090	0.077	0.014				
5	-0.078	0.066	0.046	0.038	-0.162			
6	-0.011	0.062	0.026	0.001	-0.135	-0.189		
7	-0.018	0.056	0.027	0.003	-0.132	-0.190	0.016	
8	-0.019	0.095	0.052	-0.003	-0.140	-0.205	0.026	-0.107

ri(n) = n^{th} inverse autocorrelation

m = Order of the AR model used for calculating ri.

2.5.2 Parameter Estimation :

In order to find one general model for all CPU demand processes we fitted several models to each process, and found the best parameter estimates and hence the maximum improvement available. The details of the model fitting procedure and the results obtained are the topic of this section.

The net conclusion of the identification step discussed in the last section are the following :

1. The CPU demand process is a stationary process.
2. The order of the ARMA model required to model the process is rather small - of the order of 1 or 2.

We, therefore, limited our search for the best model to the class of ARMA(p,q) models with $p+q \leq 2$. This class includes the following six models.

S. No.	p,q	Model Type	Model equation
1.	0 0	White Noise	$z(t)=w+e(t)$
2.	0 1	MA(1)	$z(t)=w+e(t)-b_1e(t-1)$
3.	0 2	MA(2)	$z(t)=w+e(t)-b_1e(t-1)-b_2e(t-2)$
4.	1 0	AR(1)	$z(t)=w+a_1z(t-1)+e(t)$
5.	1 1	ARMA(1,1)	$z(t)=w+a_1z(t-1)+e(t)-b_1e(t-1)$
6.	2 0	AR(2)	$z(t)=w+a_1z(t-1)+a_2z(t-2)+e(t)$

Let us consider the general case of fitting an ARMA(p,q) model to a process. The model is

$$z(t)-a_1z(t-1)-\ldots-a_pz(t-p) = w+e(t)-b_1e(t-1)-\ldots-b_qe(t-q)$$

The parameter estimation problem is to find the "best" estimate of the parameters $\theta = \{w, a_1,\ldots,a_p, b_1,\ldots,b_q\}$, and the variance s_e^2 of e(t). Here the best is defined in the sense of maximum likelihood (ML). The likelihood function is the probability $p(z|\theta,s_e^2)$ that a given set of parameter values would have given rise to the observed data. If the noise e(t) is assumed to be normal then it can be shown that the ML estimates are obtained by maximizing the sum-of-square function [Nel72, p94]:

$$SSR(\theta) = \sum_{t=1}^{N} e_t^2(\theta)$$

Once MLE of θ has been obtained, MLE of s_e^2 is just

$$\hat{s}_e^2 = \frac{SSR(\hat{\theta})}{N}$$

The superscript ^ denotes ML estimate. We illustrate the estimation procedure with a sample case.

A Sample Case : Figure 2.9 presents the output from the program ESTIMA for the case of fitting an ARMA(1,1) model to ECL.S2 process. The first portion of the output describes the problem, i.e., number of observations, order of differencing, initial guess values for parameters etc. Then the iterations towards ML estimate begin. The Gauss-Newton method is used to find the optimal. We now describe the importance of each of the results shown in Figure 2.9 .

```
CPU DEMAND BEHAVIOR OF ECL (S-2)

NOBS = 270
INITIAL VALUES

 AR( 1)   0.1000E+00
 MA( 1)  -0.1000E+00
 CONST    0.7500E+02

MODEL WITH D = 0 DS = 0 S = 0

MEAN =   81.67     SD =   528.9    (NOBS = 270)

INIT SSR =   0.7540E+08

ITER     SSR          ESTIMATES
                        1          2          3
   1   0.7473E+08  3.905E-02 -7.193E-02   78.0
   2   0.7472E+08 -1.716E-02 -0.122       82.9
   3   0.7471E+08 -7.497E-02 -0.180       87.7
   4   0.7471E+08 -5.205E-02 -0.157       85.9
   5   0.7471E+08 -6.595E-02 -0.171       87.0

REL. CHANGE IN SSR <=   0.1000E-05
FINAL SSR =  0.7471E+08
5 ITERATIONS
CPU DEMAND BEHAVIOR OF ECL (S-2)

PARAMETER ESTIMATES
==================
          EST        SE      EST/SE     95% CONF LIMITS
 AR( 1)   -0.066     0.569    -0.116    -1.181     1.049
 MA( 1)   -0.171     0.563    -0.304    -1.273     0.932
 CONST    87.001    59.532     1.461   -29.682   203.684

EST.RES.SD =  5.2899E+02
EST.RES.SD(WITH BACK FORECAST) =    5.2899E+02

R SQR          0.011
ADJ R SQR      0.004
D.F. = 267
F =   1.474 (2,267 DF) P-VALUE =  0.231

CORRELATION MATRIX
         AR( 1)  MA( 1)
 MA( 1)  0.994
 CON( 3) -0.780  -0.776
```

(CONTINUED...)

```
CPU DEMAND BEHAVIOR OF ECL (S-2)

AUTOCORRELATIONS OF RESIDUALS
 LAGS   ROW SE
  1 -8    .06   -0.00 -0.01 -0.02  0.04 -0.02 -0.01 -0.01 -0.01
  9-16    .06   -0.02 -0.01 -0.01 -0.01 -0.01 -0.01 -0.02 -0.01

CHI-SQUARE TEST              P-VALUE
Q( 8) =  .679        6 D.F.  0.995
Q(16) =  1.06       14 D.F.  1.000

CROSS CORRELATIONS OF RESIDUALS AND THE SERIES
ZERO LAG =  0.99
 LAGS                          E(T),Z(T+K)
  1 -8     0.10 -0.01 -0.02  0.03 -0.01 -0.01 -0.01 -0.01
  9-16    -0.02 -0.01 -0.01 -0.01 -0.01 -0.01 -0.02 -0.02

CHI-3QUARE TEST              P-VALUE
Q( 8) =  3.59        6 D.F.  0.732
Q(16) =  4.03       14 D.F.  0.995


 LAGS                          E(T+K),Z(T)
  1 -8    -0.00 -0.01 -0.02  0.03 -0.02 -0.01 -0.01 -0.02
  9-16    -0.02 -0.01 -0.01 -0.01 -0.01 -0.01 -0.02 -0.02

CHI-SQUARE TEST              P-VALUE
Q( 8) =  .649        6 D.F.  0.996
Q(16) =  1.10       14 D.F.  1.000
```

Figure 2.9 : Output of the Parameter Estimation Program

A. Stopping Criterion: Regardless of the optimization technique used one has to decide when to stop iteration. Various stopping criteria and justifications for their use have been discussed by Muralidharan and Jain [MuJ75]. The ESTIMA program stops whenever any of the following criteria are satisfied:

1. Relative change in SSR is less than 10^{-6}.
2. Absolute change in SSR is less than 10^{-6}.
3. The step size is less than 10^{-6}.
4. Number of iteration reaches a limit of 30 (a bad likelihood function).

In almost all cases of CPU demand modeling, the optimization program stopped on the first criterion.

B. Confidence Interval : It is important to remember that MLE of the parameters are, after all, random variables since they are functions of the data. It can be shown [BoJ70, p226] that MLE in large samples are joint normally distributed with mean value equal to the true parameter values and variance covariance matrix given by :

$$V(\hat{\theta}) = 2s_e^2 Q^{-1}$$

where the (i,j)th element of the matrix Q is given by

$$Q_{ij} = \frac{d^2S}{d\theta_i d\theta_j} \quad i,j=1,2,\ldots,p+q-1$$

Taking the square root of the diagonal elements of the estimated variance-covariance matrix, we get the estimated standard deviation of the parameter estimates or the standard error denoted $SE(\hat{\theta}_i)$. A 95%

confidence interval for θ_i is given by $\hat{\theta}_i \pm 1.96 * SE(\hat{\theta}_i)$.

C. Utility of the Model : We test the utility of the model in terms of R^2 and F-test. R^2 is the fraction of the variance explained by the model. It is calculated by the following formula :

$$R^2 = 1 - \frac{var[e(t)]}{var[z(t)]}$$

$$= 1 - \frac{1/N \sum_{t=1}^{N} e^2(t)}{1/n \sum_{t=1}^{N} (z(t)-\bar{z})^2}$$

This criterion for measuring the utility of a model does not penalize the model for its use of parameters. Generally the addition of any parameter to a model may be expected to reduce SSR and s_e^2 since the additional parameter offers one additional degree of freedom along which to reduce them. Consequently, to penalize a model for its use of parameter or degrees of freedom, one may compute estimates of s_e^2 by dividing SSR by N-k i.e., the net remaining degrees of freedom. This corrected measure of improvement is called Adjusted R^2.

$$R^2_{adj} = 1 - \frac{1/(N-k) \sum e^2(t)}{1/(N-1) \sum (z(t)-\bar{z})^2}$$

$$= \frac{N-1}{N-k} R^2 - \frac{k-1}{N-k}$$

A negative or very low value of R^2_{adj} indicates that the model is really not worth the trouble.

The utility of a model can also be assessed by the F-test. This test compares two hypotheses :

H1 : $\theta \neq 0$

H0 : $\theta = 0$

It can be shown that the likelihood ratio F given by

$$F = \frac{P(z/H1)}{P(z/H0)} = \frac{R^2/k}{(1-R^2)/(N-k)}$$

is F-distributed with k and N-k degrees of Freedom [BoJ70 p266]. The probability that a F-distributed variable has the calculated F value is given in the output as P-VALUE. A large P-VALUE implies that the parameter values are significantly different from zero. For example, the ARMA(1,1) model for ECL.S2 has a P-VALUE of 0.231. This low value indicates futility of the ARMA model for this process.

D. Back Forecasting : The values of e(t) in the expression for SSR are calculated as follows

$$e(t) = z(t) - \sum_{i=1}^{p} a_i z(t-i) + \sum_{i=1}^{q} b_i e(t-i) \quad - w$$

It is immediately apparent that there is a problem here because we have no value for $e_0,\ldots,e_{-q+1}$ and $z(0),\ldots,z(-p)$. One solution is to assume e(1) through e(q) as zero and start using the above equation from t=q+1. An alternative solution to this starting value problem is a back forecasting procedure suggested by Box and Jenkins [BoJ70, p212]. The ESTIMA output gives the standard deviation of the residuals with and without the back forecasting.

E. Correlation Matrix : Also given in the ESTIMA output is the estimated correlation matrix of the parameter estimates. The correlation between two parameters is obtained by taking their estimated covariance form the variance-covariance matrix described before and dividing it by the product of their standard errors. In this example, the correlation between a_1 and b_1 is estimated to be 0.994. This high correlation indicates that one of the two parameters is highly dependent on the other and therefore one of them can be omitted from the model and the model order reduced without seriously affecting the performance.

F. Diagnostic Checks on the Residuals : There are two kinds of tests that can be applied to residuals to test the adequacy of a model. These are the whiteness test and the cross-correlation test. If the model is adequate, we would expect to find that the residuals $\hat{e}(t)$ have the property of random noise - in particular, that they are not serially correlated. Autocorrelation, if evident in the residuals, may help to suggest the direction in which the model should be modified. To test whiteness, we use the Q-statistic discussed before. In the example shown, the Q-statistic is 1.06 which corresponds to a probability (P-value) of 1.000. This high p-value confirms the uncorrelatedness of residuals.

The cross-correlation test is based on the correlation between the residuals and the process. An important property of the theoretical disturbances is that they are correlated with the present and future values of z, but not with the past values, i.e.,

$$\text{Cov}[z(t-k),e(t)] = 0 \quad k>0$$

$$\text{Cov}[z(t+k),e(t)] \neq 0 \quad k\geq 0$$

As an additional check on the model, corresponding sample cross-correlations between residuals and the process are displayed in ESTIMA output.

2.5.3 Choosing a General Model:

The results of parameter estimation for 5 different models of 41 different CPU demand processes analyzed are listed in Tables 2.4 through 2.8 . In these tables, w, a_1, a_2, b_1, and b_2, if present, are model parameters; s_e is the standard deviation of the residuals. R^2, R^2_{adj}, P-value for F-test, and P-value for Chi-square test are as explained previously in sections 2.5.2.C and 2.5.1.4 .

Our next task is to choose the model which best represents CPU demand behavior of programs. There are many ways of defining the "best". For example, one criterion that we first explored was the following :

> For each process find the best model (the one giving the highest R^2, or R^2_{adj}), and choose the model that is best for a majority of the processes.

We rejected this criterion on the grounds that it does not reflect the fact that for programs with large variances even a small R^2 is good, whereas for programs with low variances even a large R^2 is not much use. Thus the net reduction in SSR rather than R^2 should be the criterion for selection. We, therefore, decided to use the following criterion :

> For each type of model, find the total (sum of) reduction in SSR achieved by the model for all programs, and choose the model that gives the highest reduction.

Table 2.4 : Parameter Estimation for ARMA(1,1) Model

$$z(t) = w + a_1 z(t-1) + e(t) - b_1 e(t-1)$$

Process Name	w	a_1	b_1	s_e	R^2	R^2_{adj}	P-VAL F-test	P-VAL Chi-sq
COMP2.G63	7.00	0.71	0.38	77.3	0.185	0.175	0.000	0.190
ECL.B1	8.98	0.19	-0.17	18.1	0.118	0.095	0.007	0.884
ECL.B2	20.89	0.72	0.53	368.7	0.066	0.058	0.000	0.969
ECL.S1	27.55	0.44	0.44	309.6	0.000	0.004	0.000	0.990
ECL.S2	83.68	-0.08	-0.19	527.4	0.011	0.004	0.214	0.000
FORTR.P21	0.74	0.86	0.75	7.7	0.042	0.025	0.094	0.377
FORTR.P30	0.76	0.88	0.80	7.6	0.025	0.017	0.053	0.925
FORTR.P8	0.10	0.98	0.93	6.7	0.044	0.038	0.000	0.049
FORTR.Q17	0.82	0.84	0.69	6.0	0.068	0.061	0.000	0.364
FRCDO.C1	394.10	0.36	0.70	1232.1	0.112	0.100	0.000	0.364
FRCDO.C11	243.75	-0.32	-0.42	579.7	0.010	0.003	0.470	0.001
M786S.U1	1.60	-0.01	-0.39	5.4	0.127	0.123	0.000	0.000
MAIN.Q10	1.18	0.85	0.55	6.5	0.244	0.237	0.000	0.955
MAIN.Q19	1.81	0.77	0.42	5.3	0.207	0.190	0.000	0.000
MAIN.R55	1.41	0.59	0.41	3.4	0.043	0.028	0.064	0.591
P.A19	2.70	0.70	0.47	22.5	0.093	0.085	0.000	0.000
PIP.G18	0.53	0.56	0.35	0.7	0.049	0.035	0.031	0.409
PIP.G45	-0.04	1.01	0.33	0.6	0.500	0.488	0.000	0.552
PIP.G60	-0.02	0.10	0.82	0.3	0.292	0.285	0.000	0.910
SOS.A21	0.03	0.98	0.92	2.6	0.077	0.072	0.000	0.000
SOS.A22	0.26	0.87	0.73	2.7	0.064	0.041	0.067	0.997
SOS.A23	0.41	0.74	0.83	1.4	0.015	0.005	0.476	0.370
SOS.A6	-0.04	1.01	0.96	3.1	0.029	0.011	0.210	0.557
TECO.B8	6.16	-0.66	-0.53	6.6	0.031	0.008	0.258	0.966
TECO.F1	0.24	0.10	0.98	4.6	0.023	0.001	0.348	0.594
TECO.F20	0.82	0.86	0.73	6.2	0.056	0.047	0.003	0.731
TECO.G37	1.74	0.94	0.91	123.0	0.005	0.013	0.763	0.268
TECO.G38	2.47	0.85	0.78	64.6	0.019	0.010	0.124	0.221
TECO.G55	0.69	0.84	0.59	4.2	0.160	0.145	0.000	0.960
TECO.H1	0.56	0.74	0.52	2.7	0.100	0.089	0.000	0.706
TECO.J5	0.82	0.87	0.72	6.3	0.069	0.048	0.044	0.649
TECO.P1	4.45	0.05	-0.10	12.8	0.021	0.006	0.240	0.000
TECO.P13	2.57	0.41	0.25	5.6	0.027	0.003	0.325	0.967
USER.B	9.12	0.75	0.55	247.2	0.082	0.079	0.000	0.779
USER.D	45.22	0.13	0.10	330.3	0.001	0.007	0.864	0.000
USER.F	0.82	0.85	0.84	17.6	0.001	0.002	0.743	0.000
USER.H	7.01	-0.07	-0.13	40.0	0.003	0.002	0.548	0.000
USER.L	1.44	0.66	0.59	7.8	0.009	0.003	0.195	0.001
USER.N	23.87	0.21	0.13	187.3	0.006	0.002	0.222	0.000
USER.P	1.64	0.77	0.63	17.1	0.048	0.047	0.000	0.096
USER.T	12.11	0.52	0.39	111.3	0.022	0.014	0.059	0.852

Table 2.5 : Parameter Estimation for AR(1) Model

$$z(t) = w + a_1 z(t-1) + e(t)$$

Process Name	w	a_1	s_e	R^2	R^2_{adj}	P-VALUE F-test	P-VALUE Chi-sq
COMP2.G63	14.39	0.41	77.6	0.172	0.167	0.000	0.014
ECL.B1	7.33	0.34	18.0	0.113	0.102	0.002	0.867
ECL.B2	60.56	0.19	373.8	0.036	0.032	0.002	0.531
ECL.S1	49.31	-0.01	309.3	0.000	0.002	0.893	0.995
ECL.S2	69.39	0.11	526.6	0.011	0.007	0.085	0.000
FORTR.P21	5.19	0.05	7.9	0.003	0.006	0.572	0.297
FORTR.P30	6.07	0.05	7.7	0.002	0.002	0.485	0.521
FORTR.P8	5.57	0.10	6.8	0.010	0.007	0.058	0.003
FORTR.Q17	4.44	0.10	6.1	0.024	0.020	0.014	0.029
FRCDO.C1	727.58	0.10	1277.3	0.039	0.032	0.019	0.058
FRCDO.C11	169.95	0.08	579.0	0.006	0.001	0.339	0.001
M786S.U1	1.06	0.33	5.5	0.110	0.108	0.000	0.872
MAIN.Q10	4.75	0.10	6.7	0.184	0.180	0.000	0.091
MAIN.Q19	4.79	0.41	5.5	0.161	0.152	0.000	0.000
MAIN.R55	2.81	0.10	3.4	0.031	0.023	0.046	0.593
P.A19	6.99	0.24	22.9	0.057	0.052	0.000	0.000
PIP.G18	0.94	0.22	0.7	0.041	0.034	0.017	0.457
PIP.G45	0.17	0.87	0.7	0.441	0.434	0.000	0.000
PIP.G60	0.49	0.49	0.4	0.164	0.160	0.000	0.000
SOS.A21	1.92	0.02	2.7	0.001	0.002	0.622	0.000
SOS.A22	1.68	0.18	2.7	0.031	0.019	0.107	0.970
SOS.A23	1.57	-0.03	1.4	0.001	0.009	0.766	0.379
SOS.A6	2.66	0.06	3.1	0.004	0.006	0.538	0.666
TECO.B8	4.17	0.10	6.6	0.015	0.004	0.246	0.941
TECO.F1	2.93	-0.08	4.7	0.007	0.004	0.440	0.638
TECO.F20	4.92	0.14	6.3	0.021	0.016	0.041	0.266
TECO.G37	27.70	0.01	122.7	0.000	0.009	0.896	0.303
TECO.G38	16.89	0.02	65.0	0.000	0.004	0.753	0.148
TECO.G55	2.87	0.34	4.3	0.113	0.105	0.000	0.708
TECO.H1	1.60	0.28	2.7	0.079	0.074	0.000	0.660
TECO.J5	4.84	0.10	6.3	0.059	0.048	0.021	0.797
TECO.P1	4.01	0.14	12.8	0.021	0.013	0.092	0.000
TECO.P13	3.67	0.15	5.6	0.023	0.011	0.170	0.954
USER.B	29.04	0.21	251.9	0.046	0.044	0.000	0.010
USER.D	50.41	0.04	329.7	0.001	0.003	0.586	0.000
USER.F	5.54	0.01	17.5	0.000	0.001	0.782	0.000
USER.H	6.18	0.05	39.9	0.003	0.000	0.276	0.000
USER.L	4.11	0.04	7.8	0.002	0.001	0.433	0.001
USER.N	27.82	0.08	187.1	0.006	0.004	0.085	0.000
USER.P	5.82	0.18	17.2	0.033	0.032	0.000	0.000
USER.T	21.97	0.12	111.4	0.015	0.012	0.044	0.791

Table 2.6 : Parameter Estimation for MA(1) Model

$$z(t) = w + e(t) - b_1 e(t-1)$$

Process Name	w	b_1	s_e	R^2	R^2_{adj}	P-VALUE F-test	P-VALUE Chi-sq
COMP2.G63	24.76	-0.31	79.7	0.127	0.121	0.000	0.000
ECL.B1	11.13	-0.32	18.0	0.112	0.101	0.002	0.921
ECL.B2	74.70	-0.14	375.6	0.026	0.023	0.009	0.270
ECL.S1	49.00	0.01	309.3	0.000	0.002	0.891	0.995
ECL.S2	77.55	-0.11	526.5	0.011	0.008	0.080	0.000
FORTR.P21	5.49	-0.05	7.9	0.002	0.007	0.605	0.283
FORTR.P30	6.36	-0.04	7.7	0.002	0.003	0.522	0.498
FORTR.P8	6.21	-0.09	6.8	0.009	0.006	0.079	0.002
FORTR.Q17	5.26	-0.11	6.1	0.017	0.013	0.039	0.012
FRCDO.C1	610.29	0.40	1253.3	0.075	0.068	0.001	0.070
FRCDO.C11	184.04	-0.09	578.6	0.007	0.001	0.291	0.001
M786S.U1	1.58	-0.38	5.4	0.127	0.125	0.000	0.000
MAIN.Q10	8.39	-0.34	6.9	0.134	0.130	0.000	0.000
MAIN.Q19	8.17	-0.26	5.7	0.102	0.093	0.001	0.000
MAIN.R55	3.42	-0.14	3.4	0.024	0.017	0.078	0.532
P.A19	9.19	-0.17	23.1	0.039	0.035	0.003	0.000
PIP.G18	1.19	-0.17	0.7	0.031	0.024	0.036	0.392
PIP.G45	1.03	-0.43	0.8	0.213	0.204	0.000	0.054
PIP.G60	0.95	-0.30	0.4	0.098	0.094	0.000	0.000
SOS.A21	1.96	-0.02	2.7	0.000	0.002	0.665	0.000
SOS.A22	2.04	-0.17	2.7	0.029	0.017	0.122	0.959
SOS.A23	1.53	0.04	1.4	0.001	0.009	0.741	0.372
SOS.A6	2.83	-0.06	3.1	0.003	0.006	0.542	0.665
TECO.B8	3.71	0.09	6.6	0.011	0.000	0.316	0.940
TECO.F1	2.71	0.10	4.7	0.007	0.004	0.443	0.633
TECO.F20	5.76	-0.12	6.3	0.017	0.012	0.067	0.178
TECO.G37	28.04	-0.01	122.7	0.000	0.009	0.897	0.303
TECO.G38	17.26	-0.02	65.0	0.000	0.004	0.761	0.148
TECO.G55	4.35	-0.24	4.3	0.077	0.069	0.003	0.210
TECO.H1	2.24	-0.21	2.8	0.059	0.053	0.002	0.262
TECO.J5	6.41	-0.24	6.3	0.059	0.048	0.021	0.805
TECO.P1	4.69	-0.14	12.8	0.021	0.014	0.091	0.000
TECO.P13	4.34	-0.12	5.6	0.018	0.006	0.221	0.911
USER.B	36.98	-0.16	253.5	0.034	0.032	0.000	0.000
USER.D	52.25	-0.04	329.7	0.001	0.003	0.587	0.000
USER.F	5.59	-0.01	17.5	0.000	0.001	0.783	0.000
USER.H	6.53	-0.05	39.9	0.003	0.000	0.274	0.000
USER.L	4.28	-0.03	7.8	0.001	0.001	0.473	0.001
USER.N	30.22	-0.08	187.2	0.006	0.004	0.089	0.000
USER.P	7.10	-0.15	17.3	0.027	0.027	0.000	0.000
USER.T	25.11	-0.10	111.6	0.013	0.009	0.069	0.754

Table 2.7 : Parameter Estimation for AR(2) Model

$$z(t) = w + a_1 z(t-1) + a_2 z(t-2) + e(t)$$

Process Name	w	a_1	a_2	s_e	R^2	R^2_{adj}	P-VAL F-test	P-VAL Chi-sq
COMP2.G63	12.71	0.37	0.11	77.4	0.183	0.172	0.000	0.137
ECL.B1	8.00	0.37	-0.09	18.0	0.120	0.097	0.007	0.906
ECL.B2	49.42	0.15	0.18	368.4	0.067	0.060	0.000	0.980
ECL.S1	49.94	-0.01	-0.01	309.6	0.000	0.004	0.956	0.991
ECL.S2	71.08	0.11	-0.02	527.4	0.012	0.004	0.211	0.000
FORTR.P21	4.62	0.05	0.11	7.9	0.014	0.004	0.456	0.214
FORTR.P30	5.46	0.04	0.10	7.7	0.012	0.003	0.260	0.766
FORTR.P8	5.09	0.09	0.08	6.7	0.017	0.012	0.048	0.010
FORTR.Q17	3.51	0.10	0.21	6.0	0.065	0.058	0.000	0.325
FRCDO.C1	888.04	0.10	-0.22	1251.6	0.084	0.071	0.002	0.105
FRCDO.C11	187.83	0.10	-0.10	577.8	0.016	0.004	0.281	0.000
M786S.U1	1.19	0.37	-0.13	5.4	0.124	0.121	0.000	0.000
MAIN.Q10	3.69	0.34	0.21	6.6	0.220	0.213	0.000	0.582
MAIN.Q19	3.32	0.31	0.27	5.3	0.220	0.203	0.000	0.002
MAIN.R55	2.43	0.10	0.13	3.4	0.048	0.033	0.045	0.665
P.A19	5.35	0.18	0.23	22.3	0.106	0.098	0.000	0.000
PIP.G18	0.81	0.18	0.14	0.7	0.056	0.042	0.020	0.428
PIP.G45	-0.03	0.10	0.38	0.6	0.52-	0.514	0.000	0.911
PIP.G60	0.30	0.33	0.36	0.3	0.232	0.225	0.000	0.024
SOS.A21	1.62	0.02	0.15	2.7	0.023	0.019	0.007	0.000
SOS.A22	1.58	0.17	0.06	2.7	0.034	0.011	0.237	0.964
SOS.A23	1.72	-0.03	-0.09	1.4	0.009	0.011	0.628	0.449
SOS.A6	2.62	0.06	0.01	3.1	0.004	0.015	0.819	0.595
TECO.B8	3.47	0.10	0.16	6.6	0.041	0.019	0.159	0.994
TECO.F1	2.92	-0.08	0.00	4.7	0.007	0.016	0.744	0.565
TECO.F20	4.30	0.13	0.13	6.3	0.036	0.026	0.027	0.397
TECO.G37	27.51	0.01	0.01	123.2	0.000	0.018	0.989	0.243
TECO.G38	16.21	0.02	0.04	65.1	0.002	0.007	0.802	0.117
TECO.G55	2.20	0.26	0.23	4.2	0.157	0.142	0.000	0.965
TECO.H1	1.36	0.24	0.15	2.7	0.100	0.089	0.000	0.751
TECO.J5	4.98	0.25	-0.03	6.3	0.059	0.038	0.069	0.752
TECO.P1	4.08	0.15	-0.02	12.8	0.021	0.006	0.239	0.000
TECO.P13	3.25	0.14	0.11	5.6	0.035	0.011	0.241	0.987
USER.B	23.51	0.17	0.19	247.6	0.080	0.077	0.000	0.684
USER.D	50.35	0.04	0.00	330.3	0.001	0.007	0.862	0.000
USER.F	5.49	0.01	0.01	17.6	0.000	0.003	0.944	0.000
USER.H	6.22	0.05	-0.01	40.0	0.003	0.002	0.549	0.000
USER.L	3.69	0.04	0.10	7.8	0.012	0.006	0.116	0.001
USER.N	27.53	0.08	0.01	187.3	0.006	0.002	0.222	0.000
USER.P	5.28	0.16	0.09	17.1	0.041	0.040	0.000	0.003
USER.T	19.67	0.11	0.10	111.0	0.026	0.018	0.033	0.937

Table 2.8 : Parameter Estimation for MA(2) Model

$$z(t) = w + e(t) - b_1 e(t-1) - b_2 e(t-2)$$

Process Name	w	b_1	b_2	s_e	R^2	R^2_{adj}	P-VAL F-test	P-VAL Chi-sq
COMP2.G63	24.75	-0.39	-0.21	77.8	0.175	0.165	0.000	0.051
ECL.B1	11.09	-0.38	-0.10	18.0	0.121	0.099	0.006	0.891
ECL.B2	74.57	-0.12	-0.17	370.8	0.055	0.047	0.001	0.834
ECL.S1	49.00	0.01	0.01	309.6	0.000	0.004	0.961	0.991
ECL.S2	77.56	-0.11	0.01	527.4	0.012	0.004	0.213	0.000
FORTR.P21	5.48	-0.01	-0.07	7.9	0.008	0.010	0.640	0.198
FORTR.P30	6.36	-0.03	-0.08	7.7	0.009	0.000	0.350	0.682
FORTR.P8	6.20	-0.08	-0.06	6.8	0.014	0.008	0.090	0.005
FORTR.Q17	5.25	-0.12	-0.15	6.0	0.049	0.041	0.002	0.171
FRCDO.C1	615.91	0.31	0.21	1226.4	0.121	0.108	0.000	0.429
FRCDO.C11	184.48	-0.07	0.09	578.4	0.014	0.002	0.330	0.001
M786S.U1	1.58	-0.38	0.00	5.4	0.127	0.123	0.000	0.000
MAIN.Q10	8.38	-0.31	-0.19	6.8	0.163	0.155	0.000	0.003
MAIN.Q19	8.13	-0.30	-0.22	5.5	0.170	0.153	0.000	0.000
MAIN.R55	3.41	-0.14	-0.16	3.4	0.047	0.032	0.047	0.702
P.A19	9.16	-0.13	-0.23	22.5	0.087	0.079	0.000	0.000
PIP.G18	1.20	-0.18	-0.18	0.7	0.057	0.043	0.018	0.398
PIP.G45	1.03	-0.44	-0.56	0.7	0.416	0.401	0.000	0.964
PIP.G60	0.95	-0.35	-0.27	0.4	0.162	0.155	0.000	0.000
SOS.A21	1.96	0.01	-0.13	2.7	0.019	0.014	0.018	0.000
SOS.A22	2.04	-0.16	-0.02	2.7	0.029	0.005	0.299	0.941
SOS.A23	1.53	0.05	0.10	1.4	0.011	0.009	0.581	0.434
SOS.A6	2.83	-0.06	-0.01	3.1	0.003	0.015	0.830	0.593
TECO.B8	3.70	0.12	-0.16	6.6	0.040	0.018	0.172	0.992
TECO.F1	2.71	0.08	-0.01	4.7	0.007	0.016	0.744	0.565
TECO.F20	5.76	-0.11	-0.09	6.3	0.028	0.018	0.064	0.276
TECO.G37	28.03	-0.01	-0.01	123.2	0.000	0.018	0.990	0.243
TECO.G38	17.25	0.00	-0.04	65.1	0.002	0.008	0.847	0.110
TECO.G55	4.33	-0.23	-0.27	4.2	0.137	0.122	0.000	0.819
TECO.H1	2.24	-0.24	-0.20	2.7	0.095	0.084	0.000	0.743
TECO.J5	6.41	-0.25	-0.03	6.3	0.060	0.038	0.068	0.757
TECO.P1	4.69	-0.15	-0.01	12.8	0.021	0.006	0.240	0.000
TECO.P13	4.32	-0.15	-0.16	5.5	0.041	0.017	0.185	0.991
USER.B	36.95	-0.15	-0.16	249.9	0.063	0.060	0.000	0.096
USER.D	52.27	-0.04	-0.00	330.3	0.001	0.007	0.862	0.000
USER.F	5.59	-0.01	-0.01	17.6	0.000	0.003	0.947	0.000
USER.H	6.53	-0.05	0.00	40.0	0.003	0.002	0.548	0.000
USER.L	4.28	-0.01	-0.11	7.8	0.012	0.007	0.110	0.001
USER.N	30.21	-0.08	-0.02	187.3	0.006	0.002	0.224	0.000
USER.P	7.10	-0.16	-0.09	17.2	0.036	0.035	0.000	0.000
USER.T	25.07	-0.12	-0.11	111.0	0.026	0.019	0.032	0.941

Table 2.9 : Comparison of Different Models

Model	Net Reduction in Total SSR	
	Program Processes	User Processes
ARMA(1,1)	6.3%	3.9%
AR(1)	2.6%	2.3%
MA(1)	4.0%	1.7%
AR(2)	5.2%	3.8%
MA(2)	6.6%	3.0%

The total reduction in SSR for various model types are listed in Table 2.9 . The reductions have been expressed as a fraction of total SSR for 0th order model ($z(t) = z_{+}e(t)$). We see that for program processes the maximum reduction achievable is only 6.6% if we choose the MA(2) model. The gain is only 3.9% in case of USER processes. The next question is whether with this little reduction it is worth while having a two parameter model. In our judgment*, it is too much work for too

* The analysis presented in this section is more of a qualitative nature than quantitative. Hence, personal preferences and biases of the analysts may well affect the final conclusion. However, it is the approach rather than the result that we deem more important. It is quite possible for some analyst to disagree with our conclusions. However, they can still follow our approach and come up with a scheduling algorithm based on control theoretic arguments.

little gain and the zeroth order model is good enough. Our conclusion is also backed up by many of the observations made during the identification step, viz., violent variations in the process, small values of ACF and PACF, non-zero probabilities in chi-square tests etc.

2.6 SCHEDULING ALGORITHM BASED ON THE ZEROTH ORDER MODEL

The net conclusion of the analysis so far is that the CPU demand behavior of programs is best represented by the following 0th order model:

$$z(t) = \bar{z} + e(t)$$

Since, e(t) is uncorrelated zero mean noise, it cannot be predicted, and the best estimate of the future CPU demand is its mean value, i.e.,

$$\hat{z}(t) = \bar{z}$$

where $\bar{z} = \frac{1}{N}\sum_{k=1}^{N} z(k)$

The problem in using the above formula is that $\bar{z}$ can be calculated only after all values of z(t), t=1,2,...,N are known. What we need now is an adaptive technique to calculate $\bar{z}$ and update it each time a new observation is obtained. Some of the possible adaptive methods are discussed below.

1. Current Average : Average of all values observed up to t-1.

$$\bar{z}_t = \frac{1}{t-1}\sum_{k=1}^{t-1} z(k) \qquad t>1$$

$$= \frac{t-2}{t-1}\bar{z}_{t-1} + \frac{1}{t-1}z(t-1)$$

$$= (1-a_t)\bar{z}_{t-1} + a_t z(t-1) \qquad \text{where } a_t = \frac{1}{t-1}$$

Here, $\bar{z}_t$ denotes the current estimate of the mean.

2. Exponentially Weighted Average* :

$$z_t = (1-a)z_{t-1} + az(t-1)$$

This is a specialization of case 1 above with a_t taken to be a constant rather than a variable.

3. Average of the last n values : n=constant

$$z_t = \frac{1}{n}\sum_{k=1}^{n} z(t-k)$$

Notice that the special case n=1 corresponds to the NEL (Next Equal To Last) strategy.

Regardless of which formula is used for prediction, the scheduling algorithm basically remains the same. We call it SPRPT (Shortest Predicted Remaining Processing Time) algorithm. It can be stated as follows:

> Each time a job leaves CPU for I/O, the CPU time taken by the job is noted and the current estimate of mean value is updated. This gives the estimate of the next CPU demand of this job. Scheduling is done at suitable intervals (e.g., clock interrupts, or whenever a job changes state etc.), and the job with the shortest predicted remaining time is selected

---

* A recent survey of current operating systems revealed that Dijkstra's T.H.E. operating system uses a scheduling algorithm based on exponentially weighted average of previous CPU demands [McW76]. However, the algorithm was based on the simple argument that I/O bound progress should be given preferential CPU allocation, and that a program should not be classified as CPU bound simply because it took large CPU time during the last burst. The exponential weighted average was thought to be a better indicator of CPU boundedness.

for CPU allocation.

Notice that the SPRPT algorithm does not require any extra book keeping other than what is already done by the operaing system. Most operating systems record CPU time used by programs for accounting and billing purposes.

2.7 CONCLUSION

A control theoretic formulation of the CPU management problem has been presented. The problem has been formulated as one of predicting the future CPU demand of a job based on its previous demands. several analytical expressions for the effect of prediction errors on the mean finishing time of tasks have been derived. The results of an experiment to study the behavior of actual program have been reported. The empirical study shows that the CPU demands of program follow a white noise model. The best least-squares predictor for the next CPU burst is, therefore, the current mean. Three different schemes for adaptive prediction have been proposed. An adaptive scheduling algorithm called SPRPT has been proposed.

CHAPTER III

A CONTROL THEORETIC APPROACH TO MEMORY MANAGEMENT

3.1 PROBLEM STATEMENT

Memory management is the technique whereby an operating system creates an illusion of virtually unlimited memory even though the actual physical memory is limited. Thus, a user program having memory requirement larger than the available physical main memory can be run on the system. This is accomplished by dividing the user program into several equal size (say 1K Words) pieces called pages. The whole program is stored on a secondary memory (drum or disk) and only a few pages are loaded in the primary (core) memory. The program is then allowed to run. Obviously, the program will be interrupted when it tries to reference a page that is not in the primary memory. This situation is called "page fault".

On a page fault, the demanded page is brought in to the core. Space for the incoming page is obtained by removing either a page of this same program or a page of some other program residing in the core. In the first case, total core memory available to each program remains fixed, and in the second case, it varies with time. The former scheme is known as fixed partitioning and the latter as variable or dynamic partitioning. In either case, when a new page is brought in, an old page must be removed from the core. The page to be removed is determined by using a page replacement algorithm. Thus, the chief problem in memory management is that of page replacement.

Intuitively, the best page to remove is the one that will never be needed again or, at least, not for a long time. In fact, it has been proved that for fixed memory partitioning, the best page to remove is the one that will not be referenced for the longest interval of time. This policy called 'MIN' is optimal in the sense that it minimizes the total number of page faults [Bel66]. However, this requires advance knowledge of the future page references (a prediction problem!). A realizable approximation to MIN is the Least Recently Used (LRU) policy which assumes that the page that has not been referenced for the longest interval in the past is the one that will not be referenced for the longest interval in future, and is the candidate for replacement.

In case of variable memory partitioning, it has been shown that MIN and its LRU approximations are not optimal. The optimal page replacement policy in this case (called VMIN algorithm) is to remove all those pages that will not be referenced during the next T time interval (t, t+T), where T = R/U is the ratio of the cost of bringing a new page in the main memory from secondary memory to the cost of keeping a page in the main memory for unit time [PrF76]. Again, this is only of theoretical interest, because it requires knowledge of the future page reference string.

A realizable approximation to VMIN policy is the Working Set (WS) Policy [Den68]. According to this policy, the pages most likely to be referenced in the next T interval (t, t+T) are those which have been referenced during last T interval (t-T, t). All other pages can

therefore be removed. The interval T is called the window size.

Both LRU and WS try to predict the future reference pattern from the past behavior of the program. Efficient operation of these algorithms is dependent upon the degree of locality of reference in programs. In statistical terms, the principle of locality states that there is a high correlation between the immediate future and the recent past behavior of a program.

3.2 CONTROL THEORETIC FORMULATION

It is obvious from the previous discussion that the problem of page replacement is a prediction problem. If we can somehow model the page reference string as a stochastic process, we can use modern control theoretic prediction algorithms such as Wiener filter, or Kalman filter etc. to predict future page reference string.

There are many ways to model the reference string as a stochastic process. Ideally the model should be such that it incorporates all the information contained in the page reference string. However, such a model becomes very complex and difficult to analyze. We, therefore, choose to begin with a rather simple stochastic process model suggested by Arnold [Arn75]. More complexity may be introduced in the future work. The implications of this simplification, and limitations of the conclusion drawn from this model are discussed in the last section of this chapter. It turns out that even this simplified model gives us much useful insight in to the problem. The stochastic process is, therefore, described next.

The page reference pattern of a given (say i^{th}) page of a program can be modeled as a zero-one process as follows :

$$z(k) = \begin{cases} 1 \text{ if the page is referenced in the kth interval } ((k-1)T < t < kT) \\ 0 \text{ otherwise} \end{cases}$$

A sample trajectory of the process is shown in Figure 3.1. The problem of page replacement is that of predicting z(k) given trajectory up to time (k-1)T, i.e., finding the best estimate $\hat{z}(k)$ of z(k) from measurements up to time (k-1)T. This problem is well known in control theory. There, much work has been done on the prediction of stochastic processes.

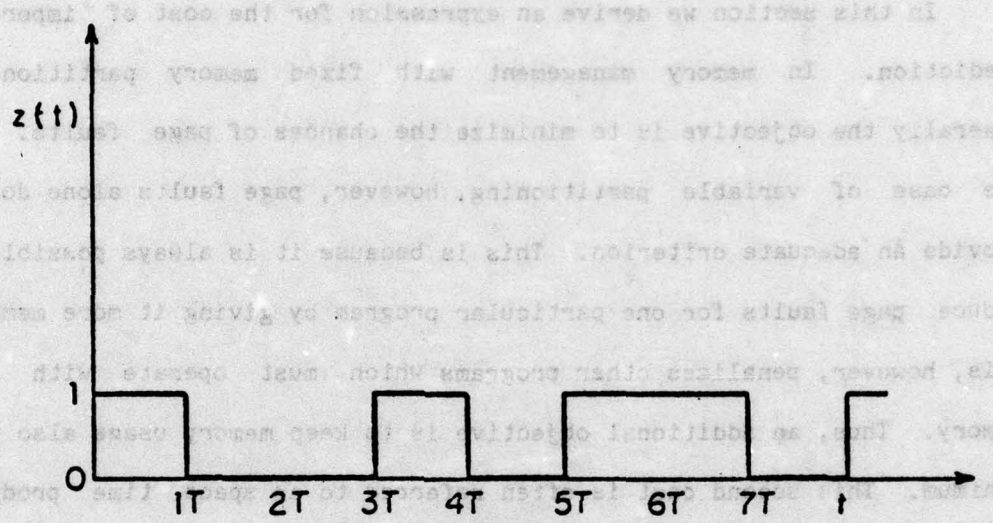


Figure 3.1 References to a page modelled as a binary stochastic process

3.3 COST EXPRESSION

In this section we derive an expression for the cost of imperfect prediction. In memory management with fixed memory partitioning, generally the objective is to minimize the chances of page faults. In the case of variable partitioning, however, page faults alone do not provide an adequate criterion. This is because it is always possible to reduce page faults for one particular program by giving it more memory. This, however, penalizes other programs which must operate with less memory. Thus, an additional objective is to keep memory usage also to a minimum. This second cost is often referred to as space time product. The total cost is, therefore, calculated as follows.

Let R = Cost of a page fault

= Cost of bringing a new page in to memory

and U = Cost of memory usage

= Cost of withholding one page of memory from other users for unit time

Let $\hat{z}(k)$ denote the predicted value of z(k) from information available at time (k-1)T. Due to imperfect knowledge of the future, z(k) and $\hat{z}(k)$ are not the same. A price has to be paid for errors in prediction. If both z and $\hat{z}$ can take only 0, 1 values, then there are only 4 cases to be considered as shown in Table 3.1.

Thus the additional cost due to imperfect prediction of z is given by :

$$C = Rz\bar{\hat{z}} + UT\bar{z}\hat{z}$$

TABLE 3.1 : Costs of memory management

z	$\hat{z}$	Decision Based on $\hat{z}$	Additional Cost	Remark
0	0	Remove	0	
0	1	Keep	UT	The page is not referenced but still kept.
1	0	Remove	R	A page fault occurs.
1	1	Keep	0	The page is referenced and it is in the memory.

Our aim should be choose $\hat{z}$ such that the expected cost E[C] is minimum.

$$E[C] = E[Rz\bar{z} + UTz\hat{z}]$$

If we choose our decision interval T such that T=R/U or R=UT, we have

$$E[C] = E[\, R(z\bar{z} + z\hat{z})]$$

$$= E[\, R(z-\hat{z})^2]$$

$$= R\, E[(z-\hat{z})^2]$$

= R times the mean square prediction error

Note that the second equality above holds only if both z and $\hat{z}$ are zero-one valued variables, not otherwise.

A classic solution to the least square prediction problem is due to Wiener[Pap65, p. 408]. It consists of designing a linear system (Wiener filter) with impulse response h(u) such that the output of the system is the estimate $\hat{z}(t)$ when input is z(k), $0 < k < t-1$ (see Figure 3.2).

$$\hat{z}(t) = \sum_{u=1}^{\infty} h(u)z(t-u)$$

The impulse response h(u) can be obtained by solving the Wiener-Hopf

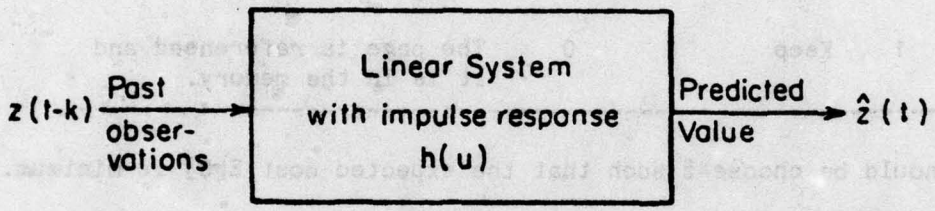


Figure 3.2: Wiener filter predictor. h(u) is given by solution to Wiener-Hopf equation.

equation :

$$C(k) = \sum_{u=1}^{\infty} c(k-u)h(u) \qquad k=0, 1, 2, \ldots$$

where C(k) = autocorrelation function of z(t). The memory management problem can, therefore, be solved by measuring C(k) and solving the Wiener-Hopf equation.

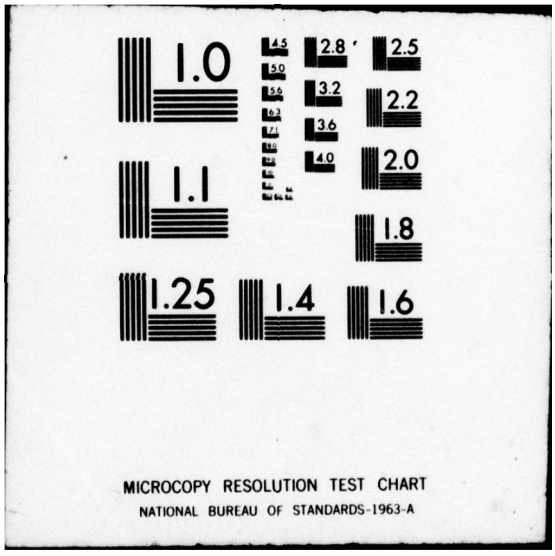
Strictly speaking the Wiener filter technique is not applicable to binary processes. For example, the output of the predictor will not be necessarily 0 or 1. It may take any value. The analysis is, therefore, approximate. The reason for the choice of this method for initial analysis is that there is no as convenient a way of modeling binary processes* as for continuous processes. In fact, the techniques for modeling, estimation, and prediction of continuous processes are so well developed that it is no longer necessary to solve the Wiener-Hopf equation in order to find the optimal predictor. Simply, by looking at the shape of the autocorrelation function, it is possible to guess the model of the system that could have generated the process [BoJ70 & Nel73]. For example, an exponentially decaying autocorrelation function implies an AR(1) model, i.e., the impulse response h(u) (the solution of the Wiener-Hopf equation) is zero every where except at u=1. These "Time Series Analysis Techniques" provide very convenient means for modeling empirical data.

---

* We have developed some techniques for modeling binary processes. These techniques and their applications to page reference process are described in the next chapter. In this chapter we report the results using conventional techniques.

AD-A069 028 HARVARD UNIV CAMBRIDGE MASS AIKEN COMPUTATION LAB F/G 9/2
STUDIES ON THE FEASIBILITY OF SYSTEM SOFTWARE DEBUGGING.(U)
APR 79 R K JAIN, U GAGLIARDI F30602-77-C-0058
UNCLASSIFIED RADC-TR-79-20 NL

2 OF 3
AD
A069 028


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3.4 NON-STATIONARITY OF THE PAGE REFERENCE PROCESS

Arnold [Arn75] has reported the results of autocorrelation measurements on a number of programs. His conclusion is that in most cases the autocorrelation function has the following form :

$$C(k) = p + (1-p)q^k \quad \text{with } p > 0 \text{ and } q=\text{constant}$$

Arnold reports in his paper that one of the main findings of his measurements is the fact that the autocorrelation function does not go to zero, i.e., the constant $p \neq 0$.

An important implication of this observation is that the page reference process is a non-stationary stochastic process. In fact, a commonly used test for stationarity is to verify that the autocorrelation function C(k) dies down to zero at large lags [BoJ70]. A simple explanation is that if the correlation between z(k) and z(0) is zero for large k, the effect of the initial conditions will not be felt after large enough k, and the process will eventually reach a state of "statistical equilibrium" called stationarity.

There are unlimited number of ways in which a process can be non-stationary. However, most of the real world non-stationary processes exhibit a "homogeneous" non-stationary behavior such that some suitable difference of the process is stationary. For example, if the process exhibits homogeneity in the sense that apart from local level (i.e., local mean), one part of the series behaves much like any other part, then the first difference of the process may be found to be

stationary.

To model such non-stationary processes, therefore, one studies the autocorrelation function of 1st, 2nd, 3rd, ... differences until a stationary process is obtained. Thus, to model z(t) we should study the autocorrelation functions of

$$Dz(t) = z(t) - z(t-1)$$

$$D^2z(t) = Dz(t) - Dz(t-1)$$

...

...

$$D^dz(t) = D^{d-1}z(t) - D^{d-1}z(t-1)$$

till a stationary process $D^dz(t)$ is found.

The non-stationarity of page reference process z(t) can be explained as follows. Even though the program behavior may be stationary in one locality, the frequency of reference to a particular page varies as the program progresses from one locality to the next. Thus, the process z(t) may behave like a set of locally stationary processes, i.e., like a homogeneous non-stationary process whose mean value varies. If this is so, the first difference of z(t) must be stationary. At this point this is just a hypothesis. The identification results presented in the next section confirm this hypothesis.

3.5 ARIMA MODEL OF PAGE REFERENCE BEHAVIOR

This section describes the results of modeling the 1st and higher differences of the page reference process. The data for analysis was supplied by Arnold. It consisted of a reference string trace of the MUDDLE compiler. About 5 different pages were chosen for analysis.

The autocorrelation functions of some of the pages studied is shown in Figure 3.3. The broken lines indicate the 95% confidence interval of the ACF for the given sample. It is obvious from this figure that the process is non-stationary and further differencing is necessary. The first difference process y(t) is defined as

$$y(t) = z(t)-z(t-1)$$

In almost all cases studied the first differences turned out to be stationary. Sample autocorrelation (ACF) and partial autocorrelation (PACF) functions are shown in Figure 3.4. The common characteristics of these functions and the inferences that we can draw from these are now described.

A. The ACF cuts off at large lags. This implies that the 1st differences are stationary and no further differencing is necessary. Thus the appropriate model for the page reference process z(t) would be an ARIMA(p,1,q) model (Auto-Regressive Integrated Moving Average model of order p,1,q) for some suitable p, and q.

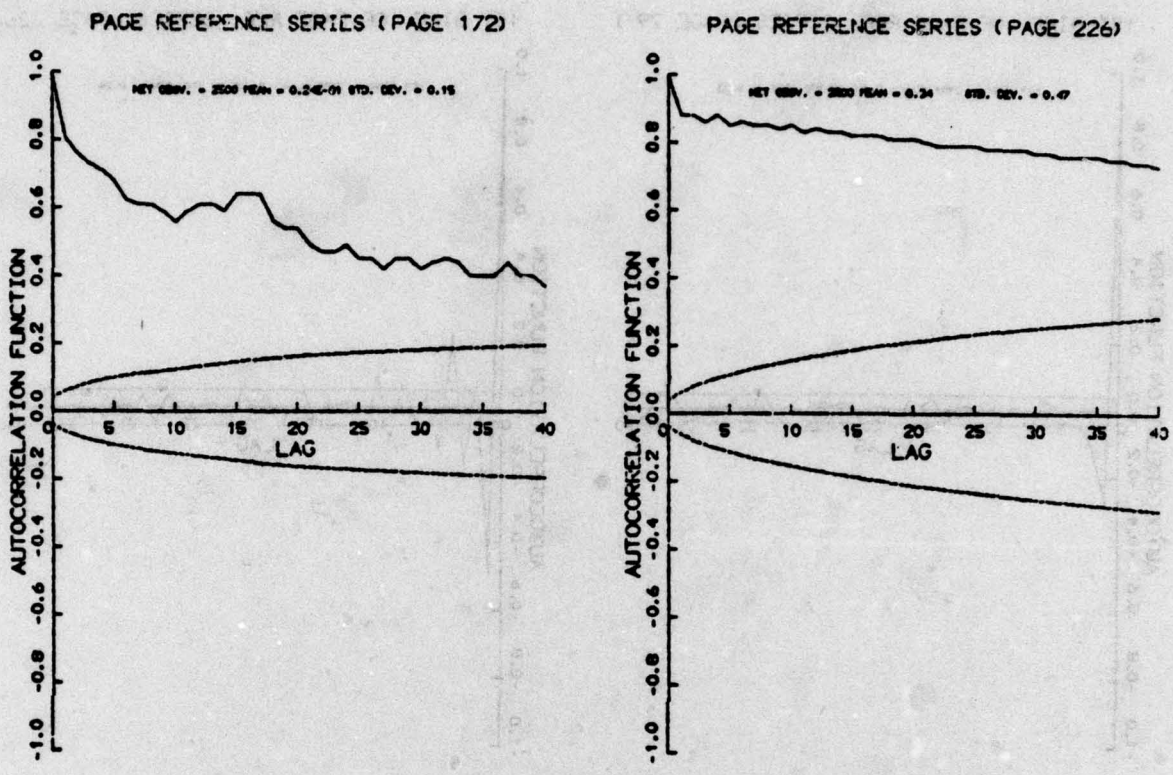


Figure 3.3 : The Autocorrelation function of page reference processes

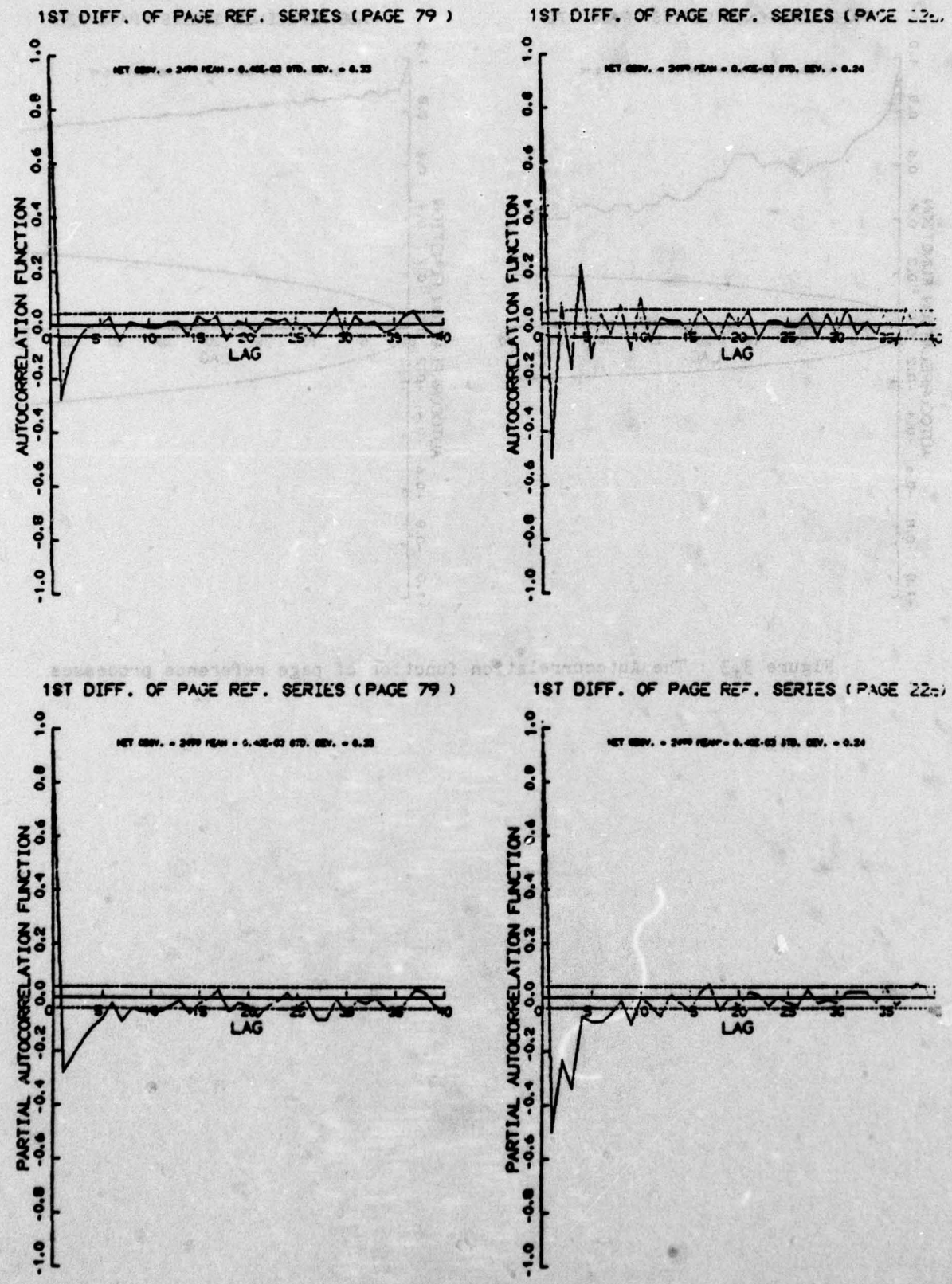


Figure 3.4 : The ACF and PACF of 1st difference of the page reference processes

B. The mean of the difference process is almost zero. This means that the constant term in the ARMA model for y(t) could be taken as zero. This property of y(t) is least surprising because a little arithmetic shows that this must be so.

$$y(t) = z(t)-z(t-1)$$

Hence, mean of $y(t) = \frac{1}{N-1}\sum_{t=2}^{N} y(t)$

$$= \frac{1}{N-1}\sum_{t=2}^{N} [z(t)-z(t-1)]$$

$$= \frac{z(N)-z(1)}{N-1}$$

$$=0 \text{ or } \pm\frac{1}{N-1}$$

$$\approx 0$$

C. Both ACF and PACF are largest at lag 1, after which they die out slowly. Of course, there are a few jumps at other lags* and the fall is not smooth. The main point is that C(1) and Ø(1) are not insignificant as was the case for CPU demand processes. Therefore, suitable low order model for y(t) is ARMA(1,1) model. To see this clearly, consider the

---

* Some pages show periodic peaks in the ACF even at large lags. This happens for the pages that are in a big (with respect to interval T) program loop. If the total time of the loop is kT (say), the page is referenced every k intervals. Thus z(t) and z(t+k) are highly correlated, and so are z(t) and z(t+jk), j=2,3,... This will cause peaks in ACF at lags jk, j=1,2,3,... In conventional time series analysis this behavior is called "seasonal". Though it is not very difficult to model this behavior, we will not consider this here in order to keep the analysis simple.

ARMA model :

$$y(t) - ay(t-1) = e(t) - be(t-1)$$

The ACF and PACF of this process for different relative values of parameters a and b are shown in Figure 3.5. The exact expressions are as follows :

$C(0)$	$= 1$	$\emptyset(0)$	$= 1$
$C(1)$	$= \frac{a-b}{1-2ab+b^2}$	$\emptyset(1)$	$= a-b$
$C(k+1)$	$= aC(k)\ k>0$	$\emptyset(k+1)$	$= b\emptyset(k)\ k>0$

The comparison of Figure 3.4 with Figure 3.5 shows that a ARMA(1,1) model with b>a>0 could be used for y(t).

D. The ACF as well PACF are negative at lag 1. This observation along with the expressions for C(1) and $\emptyset(1)$ given above further confirm the constraint guessed above, i.e., b>a. The fact that the successive values of y(t) will always be negatively correlated can easily be seen as follows:

$y(t)=1 \Rightarrow z(t)=1 \Rightarrow y(t+1)=0$ or -1

$y(t)=-1 \Rightarrow z(t)=0 \Rightarrow y(t+1)=0$ or 1

Thus a positive value of y(t) implies that that the next value will be zero or negative and vice versa.

The parameter values obtained for the cases analyzed are listed in Table 3.2. Notice that a and b do satisfy the constraints (b>a>0) conjectured above. In addition we notice that b is almost always nearer to 1 and a is nearer to 0. Also listed in the table is the relative

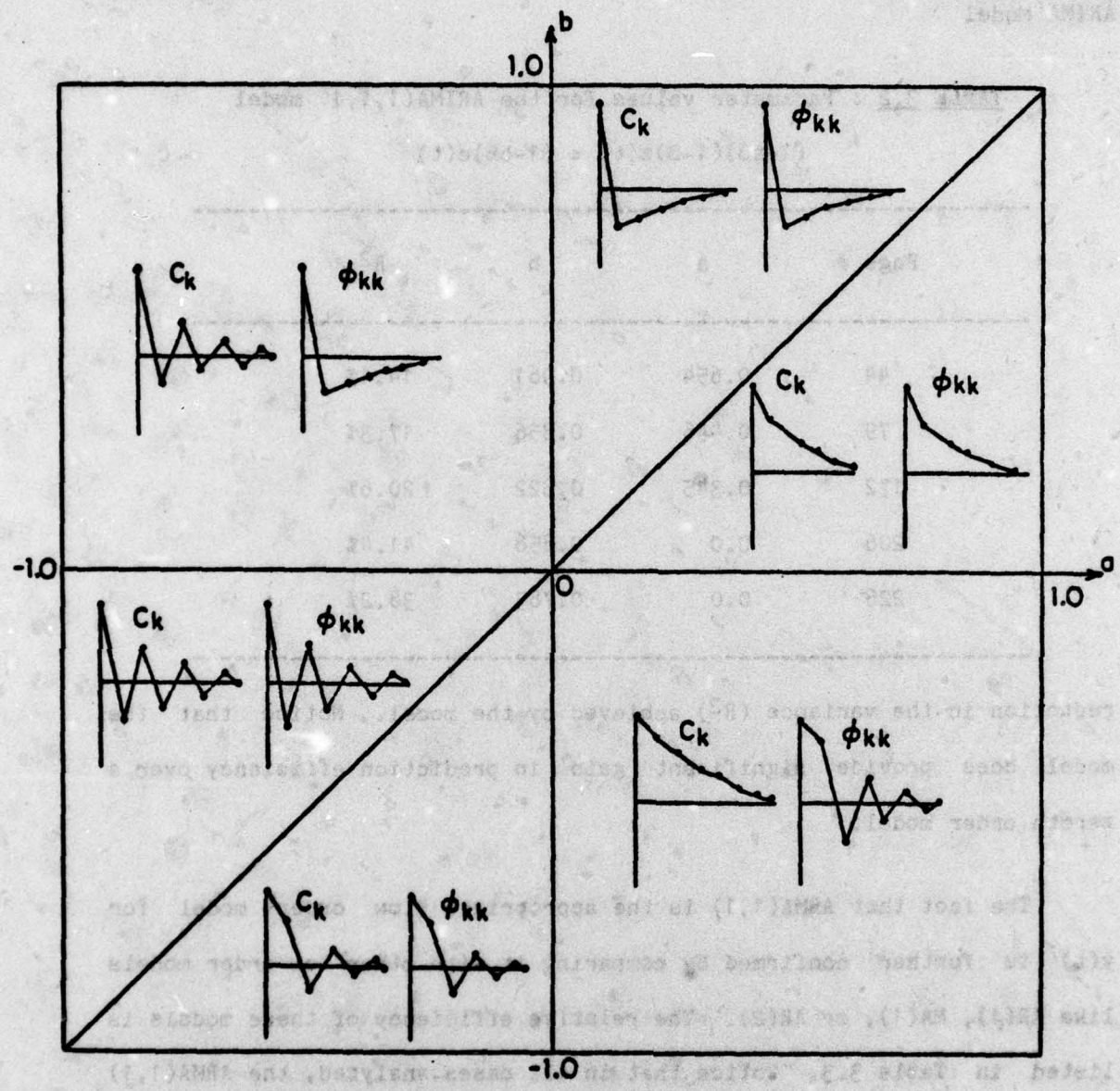


Figure 3.5: Autocorrelation and partial autocorrelation functions C_k and ϕ_{kk} for ARMA (1,1) Model

$$y(t) - ay(t-1) = e(t) - be(t-1)$$

TABLE 3.2 : Parameter values for the ARIMA(1,1,1) model

$$(1-aB)(1-B)z(t) = (1-bB)e(t)$$

Page #	a	b	R^2
44	0.654	0.961	14.1%
79	0.446	0.856	17.3%
172	0.345	0.822	20.6%
206	0.0	0.858	41.4%
226	0.0	0.783	38.2%

reduction in the variance (R^2) achieved by the model. Notice that the model does provide significant gain in prediction efficiency over a zeroth order model.

The fact that ARMA(1,1) is the appropriate low order model for y(t) is further confirmed by comparing it with other low order models like AR(1), MA(1), or AR(2). The relative efficiency of these models is listed in Table 3.3. Notice that in all cases analyzed, the ARMA(1,1) model turns out to be the most efficient.

Since y(t) is the first difference process of z(t), z(t) is said to be the first "integrated" process of y(t). Thus an Auto Regressive Moving Average (ARMA) model of order 1,1 for y(t) implies an Auto Regressive Integrated Moving Average (ARIMA) model of order 1,1,1 for z(t). The model equation for y(t) is

$$(1-aB)y(t) = (1-bB)e(t)$$

where B is the backward shift operator ($By(t) = y(t-1)$). Further since,

$$y(t) = z(t)-z(t-1) = (1-B)z(t)$$

the model equation for z(t) is given by the following equation :

$$(1-aB)(1-B)z(t) = (1-bB)e(t)$$

TABLE 3.3 : Comparison of ARMA models of 1st difference process

$$y(t) = z(t) - z(t-1)$$

List of R^2 (% reduction in variance) for different models of y(t)

Page #	AR(1)	AR(2)	MA(1)	ARMA(1,1)
44	4.6%	6.0%	6.4%	14.1%
79	7.6%		13.2%	17.3%
172	13.2%	16.9%	19.1%	20.6%
206	25.0%	29.0%	41.4%	41.4%
226	25.4%	29.5%	38.2%	38.2%

3.6 PAGE REFERENCE PREDICTION USING THE ARIMA(1,1,1) MODEL

The net conclusion of the last section is that the page reference behavior of programs can be appropriately modeled by an ARIMA(1,1,1) model :

$$(1-aB)(1-B)z(t)=(1-bB)e(t)$$

where z(t) is the binary page reference process, e(t) is white noise, B is the backward shift operator, and a, b are model parameters with b>a>0. Using this model, we can derive equations for prediction of z(t) based on process observations up to time t-1. In the following, we derive two such sets of equations called "open loop" and "closed loop" predictors. An implementation of open loop predictor using two exponential weighted averages is also discussed.

3.6.1 Open Loop Predictor : The usual way to derive a predictor for any ARIMA model is to transform it into an equivalent AR model. For our ARIMA(1,1,1) model, this is done as follows:

$$e(t) = \frac{(1-aB)(1-B)}{1-bB} z(t)$$

$$= \left[1 - (1+a-b)B - (b-a)(1-b) \sum_{i=2}^{\infty} b^{i-2} B^i \right] z(t)$$

$$= z(t) - (1+a-b)z(t-1) - (b-a)(1-b) \sum_{i=2}^{\infty} b^{i-2} z(t-i)$$

$$\text{or, } z(t) = e(t) + (1+a-b)z(t-1) + (b-a)(1-b) \sum_{i=2}^{\infty} b^{i-2} z(t-i)$$

Since e(t) is white noise and can not be estimated or predicted from the previous observations, the best estimate of z(t) from observations up to time t-1 is given by:

$$\hat{z}(t) = (1+a-b)z(t-1)+(b-a)(1-b)\sum_{i=2}^{t-1} b^{i-2}z(t-i) \qquad [3.1]$$

This equation is very inconvenient to use because it requires knowledge of all previous observations. One way to simplify it is to ignore the higher order terms (terms with small coefficients). In practice, terms after the 5th lag can be easily ignored without much loss in accuracy.

A more ingenious procedure is to rewrite the equation 3.1 as follows:

$$\hat{z}(t) = (1-c)z(t-1) + c\ z(t-2) \qquad [3.2]$$

where c=b-a>0 and z(t-2) is defined as (1-b) times the summation term in the equation 3.1. It can be recursively calculated as follows:

$$z(t-2) = (1-b)z(t-2) + bz(t-3) \quad \text{with } z(0)=0 \qquad [3.3]$$

Notice that both the equations 3.2 and 3.3 represent exponential weighted averages. However, the weighting coefficients in the two equations are quite different, because b-c = a ≠ 0.

3.6.2 Closed Loop Predictor : A somewhat different equation for the predictor can be derived as follows:

$$z(t) = \frac{1-bB}{(1-aB)(1-B)}\,e(t)$$

$$= e(t) + \frac{(1+a-b) - aB}{(1-aB)(1-B)}Be(t)$$

$$= e(t) + \frac{(1+a-z) - aB}{(1-aB)(1-B)}\,e(t-1)$$

$$= e(t) + \frac{(1+a-b) - aB}{(1-aB)(1-B)} \frac{(1-aB)(1-B)}{1-bB} z(t-1)$$

$$= e(t) + \frac{(1+a-b) - aB}{1-bB} z(t-1)$$

Hence $\hat{z}(t) = \frac{(1+a-b) - aB}{1-bB} z(t-1)$

or $(1-bB)\hat{z}(t) = [(1+a-b) - aB]z(t-1)$

or $\hat{z}(t) - b\hat{z}(t-1) = (1+a-b)z(t-1) - az(t-2)$

or $\hat{z}(t) = (1+a)z(t-1) - az(t-2) - b[z(t-1) - \hat{z}(t-1)]$

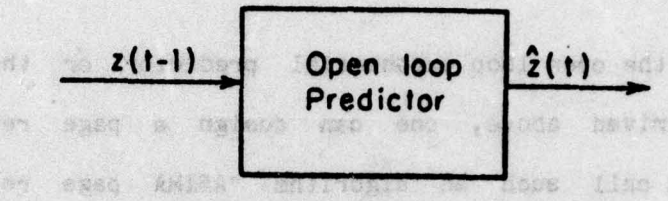
$$= (1+a)z(t-1) - az(t-2) - be(t-1) \qquad [3.4]$$

where $e(t) = z(t) - \hat{z}(t)$

= Error in prediction at t

= Innovation sequence

The block diagram representations of predictors given by equation 3.1 and 3.4 are shown in Figure 3.6. It is obvious from the diagrams why we call these predictors open loop and closed loop respectively.



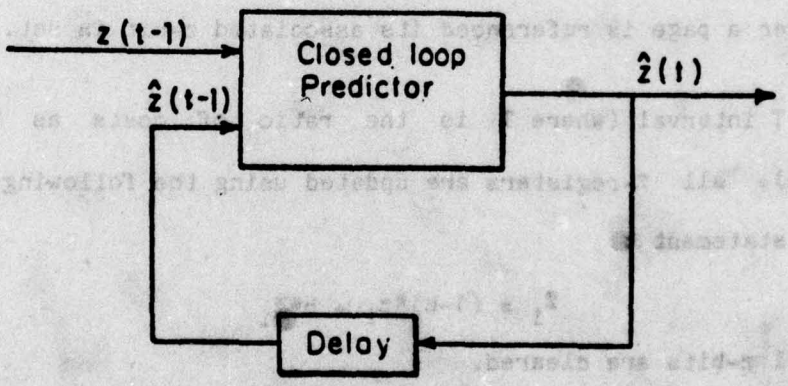


Figure 3.6: Block Diagram representation of Predictors

3.7 ARIMA PAGE REPLACEMENT ALGORITHM

Using either the open loop exponential predictor or the closed loop equation derived above, one can design a page replacement algorithm. We will call such an algorithm "ARIMA page replacement algorithm". In the following, we describe the algorithm based on the exponential predictor. The closed loop version can, similarly, be designed. The algorithm is as follows.

1. Associated with each page i is a hardware register Z_i (called Z-register). Also associated is a bit z_i (called z-bit).

2. Whenever a page is referenced its associated z-bit is set.

3. Every T interval (where T is the ratio of costs as described before), all Z-registers are updated using the following (FORTRAN like) statement :

$$Z_i = (1-b)*z_i + b*Z_i$$

and all z-bits are cleared.

4. When a new page is loaded in the memory the z_i bit is cleared and Z_i is initialized to 0.

5. At the time of page replacement, which could be every T interval, or more appropriately at page fault, a quantity called $\hat{Z}_i$ is calculated as follows :

$$\hat{Z}_i = (1-c)*z_i + c*Z_i$$

Based on $\hat{Z}_i$, a decision is made regarding the page to be replaced.

There are many possible ways of making this decision. Some examples of such decision rules are described below :

a. The page with least $\hat{z}_i$ is replaced.

b. All pages with $\hat{z}_i < k$, where k is some suitable cut off point (say 0.5) are replaced.

c. The first page encountered with $\hat{z}_i < k$ is replaced.

d. Among the pages with $\hat{z}_i < k$, the page to be replaced is selected using LIFO, FIFO, or LRU algorithms.

Of course, one may also use a combination of two or more rules, for example, in the case b above, if there is no page with $\hat{z}_i < k$, then use the rule a, or try varying k depending upon the page fault frequency and so on.

The main overhead involved in this algorithm is the update of z registers every T interval. This overhead is not excessive considering that it involves only one multiplication, one addition, and a complementation. A simple hardware circuitry could be used to do this task as shown in Figure 3.7. At the time of replacement the same circuitry could be used for prediction by replacing b by c.

The question that we have ignored so far is what value of parameters b and c should be used in the ARIMA algorithm. Ideally, one would like to estimate these parameters separately for each page. The estimation technique should be an adaptive one so that b_i and c_i are updated along with z_i every T-interval. Alternately one could use some suitable fixed value. This latter procedure has much less overhead and

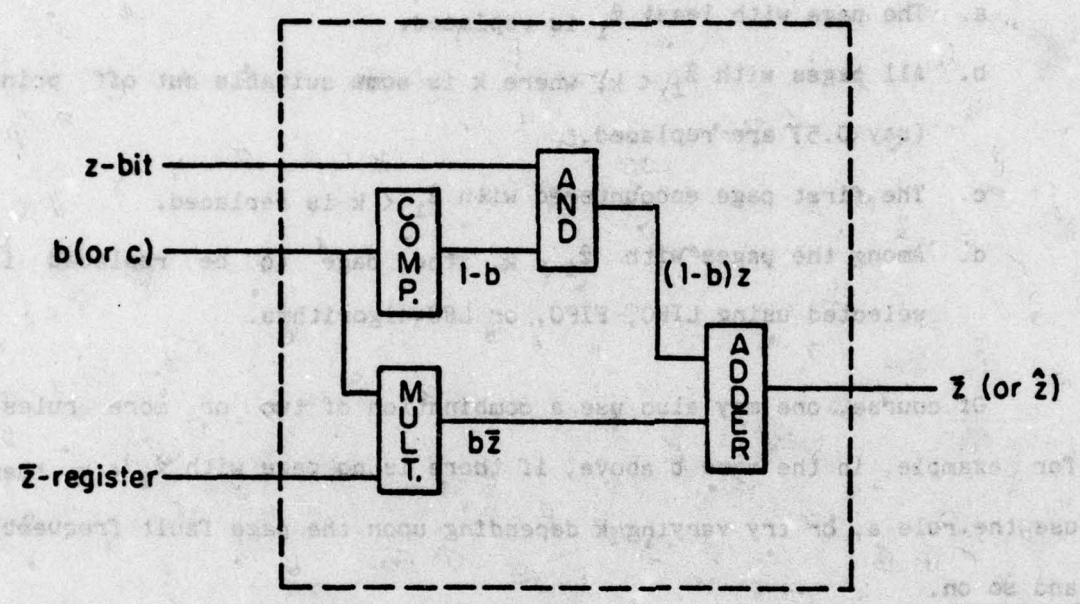


Figure 3.7: Hardware Implementation of ARIMA Algorithm

is more practical. However, in this case, program behavior monitoring should be done from time to time to detect drastic variations in user program behavior. The main consideration in choosing these values is that they should be representative of user program behavior, and also they must be easy to represent. For example, for the pages we analyzed, we found that the average values of b and c were 0.856, and 0.567 respectively. Therefore, b=7/8, and c=1/2 seem appropriate, considering that we are going to use binary arithmetic.

3.8 SPECIAL CASES OF THE ARIMA ALGORITHM

In this section we show that working set, reference frequency, and few other algorithms are special cases of the ARIMA algorithm. Another special case is an extended working set, wherein the window size is dynamically adjusted to match the program locality size. Recall from the last section that the exponential predictor for the ARIMA(1,1,1) model is given by :

$$\hat{z}(t) = (1-c)z(t-1) + cz(t-2)$$

$$z(t-2) = (1-b)z(t-2) + bz(t-3)$$

We shall refer to these two equations as prediction equation, and update equation respectively. The four special cases of the ARIMA algorithm occur when the parameters b and c take their extreme values 0 and 1. These cases are now described.

Case I [c=0] : In this case, the prediction equation becomes

$$\hat{z}(t) = z(t-1)$$

i.e., the pages that were referenced in the last T interval are the ones that are likely to be referenced in the next T interval. All other pages can be replaced. This is exactly what Working Set policy also says. For this special case of the ARIMA page replacement algorithm described above, the Z-registers are no longer necessary, and the pages with z-bit on constitute the working set. Also notice that c=b-a=0 implies that the model equation for this case is

$$(1-B)z(t) = e(t)$$

i.e., an ARIMA(0,1,0) model. In this case the process z(t) is an

"Integrated white noise" or Wiener process. Thus working set is optimal for programs whose page reference processes constitute a Wiener process.

Since using WS is equivalent to using a white noise model for the first difference process y(t), the percentage improvements listed in Table 3.3 are also percentage paging cost improvements achievable by various ARIMA models.

Case II [b=1] : For this case the update equation takes the following form :

$$\bar{z}(t-2) = \bar{z}(t-3) = \bar{z} \text{ (say)}$$

i.e., the mean is time invariant, the prediction equation therefore reduces to an AR(1) predictor :

$$\hat{z}(t) = (1-c)z(t-1)+c\bar{z}$$

This is Arnold's Wiener Filter model [Arn75]. Notice that this is applicable only if the mean is time invariant, i.e., if the z(t) process is stationary.

Case III [c=1, b=1] : For this special case, like case II above, the update equation implies a time invariant mean and the predictor equation becomes

$$\hat{z}(t) = \bar{z}$$

i.e., the pages are expected to be referenced with fixed frequency (mean value) and the page with least $\bar{z}$ is the candidate for replacement. This is the Reference Frequency policy of page replacement. This model is also known as Independent Reference Model (IRM). For this case the model equation becomes

$$(1-B)z(t) = (1-B)e(t) \text{ since } a=b-c=0$$

or, $$z(t) = e(t)+z$$

This is an ARIMA(0,0,0) or white noise model. Thus the reference frequency model is appropriate only if the page references constitute white noise*.

Case IV [c=b] : In this case, the predictor and the update equations become similar.

$$\hat{z}(t) = (1-b)z(t-1) + bz(t-2)$$

$$z(t-2) = (1-b)z(t-2) + bz(t-3)$$

These equations can be rewritten as follows:

$$\hat{z}(t) = z(t-1)$$

$$z(t-1) = (1-b)z(t-1) + b\hat{z}(t-1)$$

This is an extension of the independent reference model. Here the reference frequency is assumed to be time varying and is computed adaptively using an exponential weighted average. This policy could, therefore, be called "Adaptive Independent Reference Model" (AIRM). This is optimal when a=c-b=0 and the process model is ARIMA(0,1,1):

$$(1-B)z(t) = (1-bB)e(t)$$

i.e., z(t) is the integration of a first order (colored or correlated) noise. It could, therefore, be called "Colored Wiener Process".

---

* This same conclusion was reached by Aho, Denning, nd Ullman [ADU71]. They call it Ao policy and show that the policy is optimal when the probability of reference of a page depends neither on time (stationarity) nor on previous program behavior (no-autocorrelation).

We conclude this section on special cases of the ARIMA(1,1,1) model by depicting all the four cases discussed above on a single diagram as shown in Figure 3.8. The ARIMA model operates in the triangular region 0<c<b<1. It is obvious from this diagram that the ARIMA(1,1,1) is a general model and that Working set, Arnold's Wiener Filter, Independent Reference Model, and Adaptive Independent Reference Model are all its boundary cases.

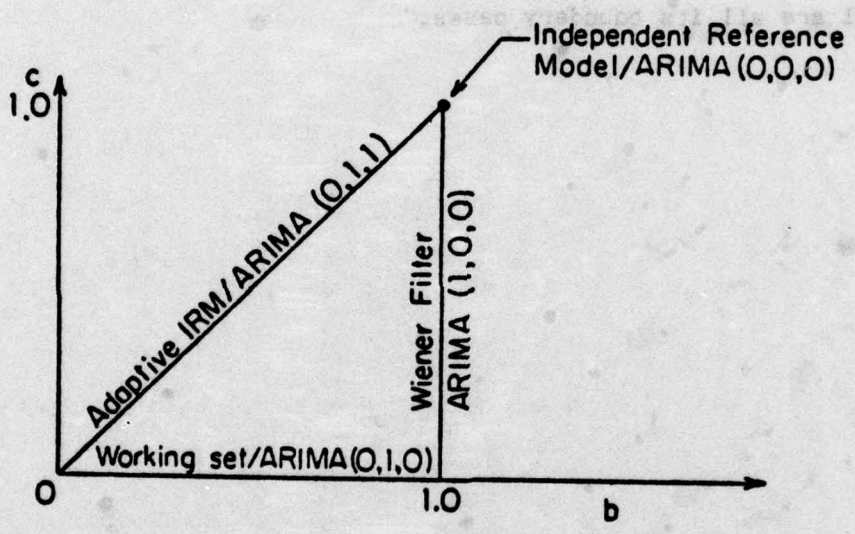


Figure 3.8: Special cases of ARIMA (1,1,1) Algorithm

3.9 LIMITATIONS OF THE FORMULATION

In the last section it was shown that the working set policy is a special case of the ARIMA policy. Therefore, a question that naturally arises is whether there is any relation between the ARIMA and another popular page replacement algorithm LRU, and whether the LRU is also a special case of the ARIMA. The answer is probably "no". This is because of the limitations of the zero-one stochastic formulation that we started with. It uses only a subset of the available past information.

A deeper insight into the information structure can be obtained by considering the information used by various algorithms. VMIN - the optimal variable space algorithm uses the complete page reference string. WS requires knowledge of set of pages referenced in the last T interval. It does not require the order in which the pages are referenced or the number of times they are referenced. A general ARIMA model would use all the sets of pages referenced in successive T interval. Finally, LRU uses the set of last referenced m pages along with their order of reference. The Venn diagram of information used by these algorithms is shown in Figure 3.9. The broken line in this figure separates the past from the future. There are two inferences to be drawn from this figure :

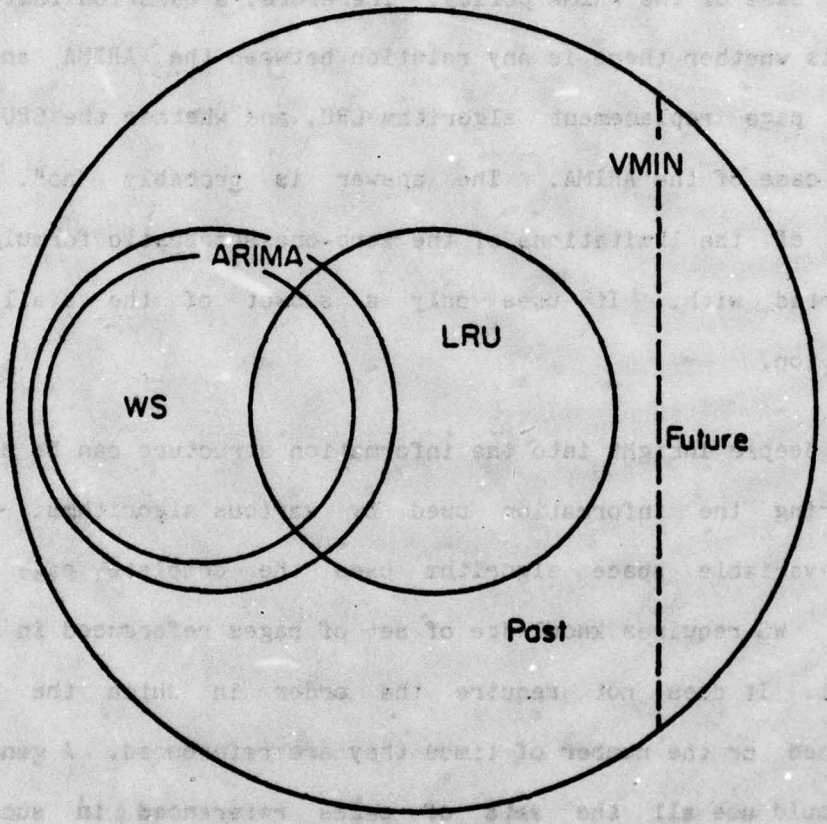


Figure 3.9: Information used by various page replacement algorithms.

1. The information used by the WS is a subset of that used by the ARIMA. This explains why ARIMA policies can always be specialized to WS set policies with proper window sizes.

2. There is much information, (like the frequency of reference of a page in an interval, the order of reference of various pages, and cross-correlation between different page processes), that is not used in the zero-one stochastic process formulation used to derive the ARIMA policy. If we could somehow develop a formulation which uses the complete past information, then both the WS and the LRU will be special cases of the generalized model. The conclusions drawn would then be universal in scope.

3.10 CONCLUSION

The page reference behavior modeled as a zero-one binary stochastic process exhibits a non-stationary behavior. An ARIMA(1,1,1) model was shown to be appropriate for the process. This model is then used to design a memory management policy.

The main results achieved in this chapter can be stated as follows:

1. We have shown that the cost of imperfect prediction is proportional to the square of the difference between the predicted and the actual value.

2. Using empirical results, we have shown that the ARIMA(1,1,1) model is an appropriate model for page reference processes.

3. We have designed a new page replacement algorithm called ARIMA page replacement algorithm. The algorithm is shown easy to implement.

4. We have shown that many conventional algorithms like Working Set, Reference frequency, and Arnold's Wiener Filter algorithm are merely boundary cases of the ARIMA algorithm. Also we have described conditions under which these boundary cases are optimal. In particular we, thus, have a control theoretic derivation of the WS policy.

In the analysis presented in this chapter, approximations were introduced due to Gaussian assumption. We, therefore, expect that the development of identification methods for discrete binary processes will lead to better understanding and management of program memory behavior.

CHAPTER IV

BOOLEAN MODELS FOR BINARY PROCESSES WITH APPLICATION TO MEMORY MANAGEMENT

4.1 INTRODUCTION

In this chapter we present a new approach for analysis of binary processes. A process z(t) is called binary if the variable z(.) can take only two values 0 and 1*. A classic example of a binary stochastic process is the so called "Semi-random Telegraph Signal", which consists of a sequence of independent identically distributed binary random variables.

In computer science, binary process are a common occurrence. For example, as was shown in the last chapter, the reference pattern of a particular page constitutes a binary stochastic process which can be used to design new memory management policies. Similarly, in database management, record reference patterns constitute binary processes which can be used to detect changes in reference patterns and to determine optimum points for database reorganization. In computer networks, packet arrivals at a node can be modeled by a zero-one process. Several similar examples can be constructed in the areas of weather prediction, signal detection, medical diagnosis, and information theory.

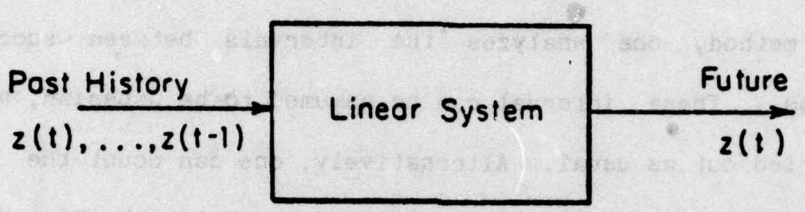
---

* In fact, z(.) can take any two values say a and b. The analysis presented here can still be used by transforming it to another process $y(t)=\frac{Z(t)-a}{b-a}$. Notice that the process y(t) is a zero-one process.

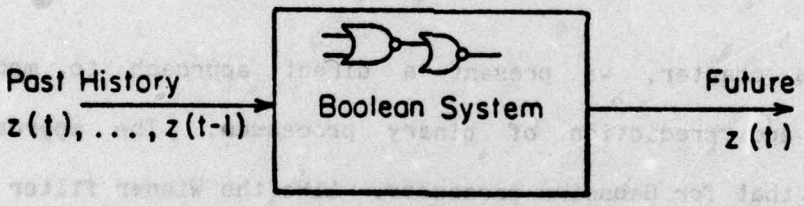
In spite of the fact that binary processes are so common, it is surprising that no direct technique for identification and prediction of such processes has been described in the published literature. The two known methods for analyzing such processes are both indirect [CoL66]. In the first method, one analyzes the intervals between successive $z(t)=1$ pulses. These interval can be assumed to be Gaussian, and the analysis carried out as usual. Alternatively, one can count the number of $z(t)=1$ pulses over suitable intervals of equal length and model the resulting "count" process as a Gaussian process. It is obvious that both these approaches for "modeling" of the process are not suitable for the prediction of $z(t)$ given its history upto time t.

In this chapter, we present a direct approach to modeling, estimation, and prediction of binary processes. The approach is analogous to that for Gaussian processes. Like the Wiener filter for a Gaussian process (see Figure 4.1), we design a system (a Boolean system) whose output is the predicted value $\hat{z}(t)$, and the input is the past history of the process. Our model is more general than the Wiener filter in the following respects:

1. The measure of goodness of the predictor is not limited to a fixed criterion, e.g., least-squares in the case of Wiener filter. Our method applies to any given criterion: linear or non-linear.
2. We do not impose the linearity condition on the system. Our method gives the optimal non-linear predictor for the process. Further, if the optimal predictor is not unique, our method gives all the predictors.



a. Wiener Predictor for Gaussian Processes



b. Boolean Predictor for Binary and k-ary Processes

Figure 4.1: Analogy between Wiener predictor and Boolean Predictor

3. Our model is not restricted to stationary processes alone; it is applicable to some non-stationary processes also.

An additional feature of our model is that it gives zero-one estimates of a zero-one process. Since z(t) is binary it is not meaningful to have fractional estimates of z(t). For example, it is meaningless to say $\hat{z}(t) = 0.73$ (though it is meaningful to say that the probability of z(t) = 1 is 0.73).

The only restriction in our model, which does not appear in the Wiener filter, is that the process is assumed to be Markov of a given order n. A process is called Markov if the probability distribution of z(t) given all the past history of the process depends only on a finite past. In particular, z(t) is Markov of order n if

$$P[z(t)|z(1),z(2),\ldots,z(t-1)] = P[z(t)|z(t-n),z(t-n+1),\ldots,z(t-1)]$$

Here P[.] denotes the probability of an event.

In this chapter we develop a general probabilistic model relating z(t) to its past values. Based on this model, an expression is derived for the likelihood function, and hence, for the maximum likelihood estimates (MLE) of the model parameters. We show how the model is used for optimum prediction and derive a formula for the total cost due to prediction errors. Then we extend all results to the more general case of k-ary processes. In this case, the process takes integer values from 0 through k-1. Finally, we show how the model can be used for page replacement.

Introduction

In the analysis presented in this paper we make frequent use of the properties of pseudo-Boolean functions. The essential elements of the theory of such functions are, therefore, briefly reviewed in the next section (adopted from [HaR68]). The material in the other sections of this paper is original and, as far as is known to the author, has not appeared anywhere in the published literature.

4.2 BOOLEAN FUNCTIONS - FUNDAMENTALS

The definition of Boolean functions varies widely among authors. In an attempt to generalize the concepts, even the pioneers of this theory, Rudeanu and Hammer, have changed the definition over time (e.g., in [HaR68], and [Rud74]). In this thesis, we adopt the following definition from [HaR68].

4.2.1 Definition : By a "Boolean function" $f(x_1,x_2,\ldots,x_n)$ of n variables we mean a mapping

$$f:\{0,1\}^n \rightarrow \{0,1\}$$

i.e., a zero-one valued function of zero-one valued variables.

An example of a Boolean function is $f(x_1,x_2) = x_1 + x_2 - 2x_1x_2$. The usual way to express a Boolean function is by using the Boolean operations (e.g., conjunction, disjunction, and negation). For example, the above function is usually written as

$$f(x_1,x_2) = \bar{x}_1x_2 \vee x_1\bar{x}_2$$

where "v" is the disjunction (inclusive OR) operator, bar indicates negation, and conjunction is denoted by juxtaposition. The transformation between the two representations is a result of the following equivalences:

$$\bar{x} = 1-x \qquad \forall x \in \{0,1\}$$

$$x_1 \vee x_2 = x_1 + x_2 - x_1x_2 \qquad \forall x_1,x_2 \in \{0,1\}$$

A notation which is commonly used in the literature on Boolean functions is the following

$$x^0 = \bar{x} \quad x^1 = x$$

Where x^0 is "x sup zero" (not x raised to the power zero). To avoid confusion, we will use $(x)^i$ to denote the i^{th} power of a binary variable x. Continuing with the notation, if $X = \{x_1, x_2, \ldots, x_n\}$ is a set of n binary variables and $i_1 i_2 \ldots i_n$ is the n digit binary representation of i, $0 \leq i \leq 2^n - 1$, then

$$q_i(X) = X^i = x_1^{i_1} x_2^{i_2} \ldots x_n^{i_n}$$

is called the i^{th} fundamental product. For example, for n=3

$$q_5 = x_1^1 x_2^0 x_3^1 = x_1 \bar{x}_2 x_3 \text{ and } \quad q_0 = x_1^0 x_2^0 x_3^0 = \bar{x}_1 \bar{x}_2 \bar{x}_3$$

An important property of fundamental products is that $q_i(X)=1$ if and only if X=i. Thus, the fundamental products are "mutually exclusive", i.e.,

$$q_i q_j = \begin{cases} 0 & i \neq j \\ q_i & i = j \end{cases}$$

There are many ways of representing a Boolean function. A few examples are given below:

$1 - x_1 - x_2 + 2x_1x_2$	(Polynomial form)
$\bar{x}_1\bar{x}_2 \vee x_1x_2$	(Disjunctive Form)
$(x_1 \vee x_2)(\bar{x}_1 \vee \bar{x}_2)$	(Conjunctive Form)
$1 \oplus x_1 \oplus x_2$	(Reed-Muller form)
$\bar{x}_1\bar{x}_2 + 0\bar{x}_1x_2 + 0x_1\bar{x}_2 + x_1x_2$	(Sum of Products form)

In our analysis, we use the sum of products form. Using Shannon's decomposition theorem, any Boolean function can be expressed in this form as follows:

$$f(X) = \sum_{i=0}^{2^n-1} f_i q_i(X)$$

where, $f_i = f(X|X=i)$, i.e., $f(X)$ when $x_1=i_1$, $x_2=i_2$, ..., $x_n=i_n$.

The concept of Boolean functions can be generalized to other functions - not necessarily zero-one valued. Such functions are called "pseudo-Boolean functions".

4.2.2 Definition : Let R be the field of real numbers; by a pseudo-Boolean function f we mean a mapping

$$f:\{0,1\}^n \rightarrow R$$

i.e., a real valued function of binary variables.

An example of pseudo-Boolean functions is the following function :

$$f(x_1,x_2) = 0.5(x_1)^3 + 3x_1 - 2(x_2)^2$$

In fact, all functions (including Boolean functions) of binary variables are pseudo-Boolean functions. Therefore, the adjective "pseudo-Boolean" may be dropped whenever it is clear from the context.

Again using Shannon's decomposition theorem, any function of binary variables can be reduced to a "sum of products form":

$$f(X) = \sum_{i=0}^{2^n-1} f_i q_i(X)$$

For example,

$$f(x_1,x_2) = 1 - 0.5(x_1)^3 + 3x_1x_2 - 2(x_2)^2$$

For this function $f_0 = 1$, $f_1 = -1$, $f_2 = 0.5$, and $f_3 = 1.5$, hence,

$f(x_1,x_2) = 1\bar{x}_1\bar{x}_2 + (-1)\bar{x}_1x_2 + 0.5x_1\bar{x}_2 + 1.5x_1x_2$

Similarly, $x_2 e^{x_1} = 0\bar{x}_1\bar{x}_2 + 1\bar{x}_1x_2 + 0x_1\bar{x}_2 + ex_1x_2$

Notice that when expressed in the sum of products form, every function of binary variables becomes linear in each variable (i.e., each variable appears only as its first power), although the function itself is non-linear (due to the presence of product terms).

4.3 DEVELOPMENT OF THE BOOLEAN MODEL

Let Z_{ji} denote the set of i observations immediately preceding z(j), i.e., {z(j-i), z(j-i+1), ..., z(j-1)}. Thus Z_{tt-1} = {z(1),z(2),...z(t-1)} denotes the complete past history of the process. Let $p_t = P[z(t)=1|z(1),z(2),...,z(t-1)]$ denote the probability of z(t)=1 given the past history of the process. The simplest binary process is the so-called "Binary White Noise" (BWN) or Bernoulli Process. It is defined as the sequence of independent identically distributed binary random variables. The semi-random telegraph signal described previously is a BWN. Also if we associate a time index to successive Bernoulli trials, they will constitute a BWN. A BWN can also be obtained by filtering and clipping a Gaussian white noise. A BWN with parameter p will be denoted by BWN(p)

For a Markov process of order n, p_t depends only on the past n values Z_{tn}={z(t-n),z(t-n+1),...,z(t-1)}. We can represent the most general non-linear dependence of p_t on Z_{tn} by saying that $p_t = h(Z_{tn})$, where h is some non-linear function of Z_{tn}, such that $0 \leq h \leq 1$. In the sum of products form, we have

$$p_t = P[z(t)=1|Z_{tn}] = \sum_{i=0}^{2^n-1} l_i q_i(Z_{tn}) \qquad [4.1]$$

where $l_i = h(Z_{tn}|Z_{tn}=i)$

= Value of $h(Z_{tn})$ when z(t-n),...,z(t-1) take values corresponding to the binary expansion of i.

and $q_i(Z_{tn})$ = i^{th} fundamental product of z(t-n),...,z(t-1)

The equation for z(t) corresponding to this equation is

$$z(t) = \sum_{i=0}^{2^n-1} e_i(t) q_i(Z_{tn}) \qquad [4.2]$$

where, $e_i(t) \sim BWN(h_i)$.

By taking expectations of both sides of equation (4.2) , it can be shown to be equivalent to equation (4.1). Notice that Z_{tn} denotes the "state" of the process. The process can be in any one of 2^n states corresponding to $Z_{tn}=i$, $i=0,1,\ldots 2^n-1$. The distribution of the future value z(t) in state i is Bernoulli with parameter h_i.

For example, the Boolean model of a second order Markov process is

$$z(t) = e_0(t)\bar{z}(t-2)\bar{z}(t-1) + e_1(t)\bar{z}(t-2)z(t-1) + e_2(t)z(t-2)\bar{z}(t-1) + e_3(t)z(t-2)z(t-1)$$

4.4 LIKELIHOOD FUNCTION AND PARAMETER ESTIMATION

The proposed model (equation 4.1 or 4.2) has 2^n parameters $h_0, h_1, \ldots, h_{2^n-1}$. In this section we develop a likelihood function for the observations, and find the expression for maximum likelihood estimates of these parameters. To develop the result in the form of Theorem 4.4.2, we need the following lemma.

4.4.1 Lemma [Function Lemma] : Let f be a mapping f :R->R, i.e., a real valued function of a real variable, and let

$$p = \sum_{i=0}^{2^n-1} h_i q_i(X)$$

Then, $f(p) = \sum_{i=0}^{2^n-1} f(h_i) q_i(X)$

Proof : Let $p = h(X)$

so that $f(p) = f(h(X))$

Since the right hand side of the above equation is a pseudo-Boolean function, it can be written in a sum of products form:

$$f(h(X)) = \sum_{i=0}^{2^n-1} f(h(X|X=i)) q_i(X)$$

$$= \sum_{i=0}^{2^n-1} f(h_i) q_i(X)$$

[Q.E.D.]

Some examples of the use of the Function Lemma are given below.

$$p^2 = \sum_{i=0}^{2^n-1} h_i^2 q_i(X)$$

$$p^{-1} = \sum_{i=0}^{2^n-1} h_i^{-1} q_i(X)$$

$$\log(1-p) = \sum_{i=0}^{2^n-1} \log(1-h_i)\, q_i(X)$$

4.4.2 Theorem [Estimation Theorem] : The maximum likelihood estimate of h_i based on N observations {z(1), z(2), ..., z(N)} is given by

$$h_i = \frac{m_{i1}}{m_{i0} + m_{i1}} \quad i=0,1,\ldots 2^n-1$$

where m_{i0} = # of times the sequence $Z_{tn}=i$ is followed by $z(t)=0$

and m_{i1} = # of times the sequence $Z_{tn}=i$ is followed by $z(t)=1$

Proof : Let $H = \{h_0, h_1, \ldots, h_{2^n-1}\}$ be the set of parameters.

Let, $$p_t = P[z(t)=1 | Z_{tn}, H] = \sum_{i=0}^{2^n-1} h_i q_i(Z_{tn})$$

$$\bar{p}_t = 1-p_t = P[z(t)=0 | Z_{tn}, H]$$

The above two equations for p_t and $\bar{p}_t$ can be combined as follows:

$$P[z(t) | Z_{tn}, H] = P[z(t)=1 | Z_{tn}, H]^{z(t)} P[z(t)=0 | Z_{tn}, H]^{\bar{z}(t)}$$

$$= p_t^{z(t)} \bar{p}_t^{\bar{z}(t)}$$

Therefore,

$$P[z(N), z(N-1), \ldots, z(1) | z(-n+1), z(-n+2), \ldots, z(0), H]$$

$$= P[z(N) | z(N-1), \ldots, z(1), Z_{1n}, H] P[z(N-1) | z(n-2), \ldots, z(1), Z_{1n}, H] \ldots$$

$$\ldots P[z(1) | Z_{1n}, H]$$

$$= \prod_{t=1}^{N} P[z(t)|Z_{tn},H]$$

The above equation gives the likelihood that the N observations came from a model with parameter $H = \{h_0,h_1,\ldots h_{2^n-1}\}$. Notice that we assume the initial conditions $z(-n+1),\ldots,z(0)$ to be given (or to be assumed equal to zero). Only the parameters are to be estimated. The likelihood function is

$$L(H) = \prod_{t=1}^{N} p_t^{z(t)} \bar{p}_t^{\bar{z}(t)}$$

taking the log of the above equation we get the log likelihood function

$$l(H) = \log\{L(H)\}$$

$$= \sum_{t=1}^{N} (z(t)\log p_t + \bar{z}(t)\log \bar{p}_t)$$

Now using the Function Lemma,

$$\sum_{t=1}^{N} z(t)\log p_t = \sum_{t=1}^{N} z(t) \sum_{i=0}^{2^n-1} (\log h_i) q_i(Z_{tn})$$

$$= \sum_{i=0}^{2^n-1} \log(h_i) \sum_{t=1}^{N} z(t) q_i(Z_{tn})$$

$$= \sum_{i=0}^{2^n-1} m_{i1} \log(h_i)$$

where $m_{i1} = \sum_{t=1}^{N} z(t) q_i(Z_{tn})$

$$= \#\text{ of times } Z_{tn}=i \text{ is followed by } z(t)=1$$

The last equality is a result of the observation that $z(t)q_i(Z_{tn})$ is 1 if and only if $z(t)=1$, and $Z_{tn}=i$. Similarly,

$$\sum_{t=1}^{N} \bar{z}(t) \log \bar{p}_t = \sum_{i=0}^{2^n-1} m_{i0} \log(\bar{h}_i)$$

Therefore, the log likelihood function is given by

$$l(H) = \sum_{i=0}^{2^n-1} \{m_{i1} \log h_i + m_{i0} \log(1-h_i)\}$$

The maximum likelihood estimate of h_i is obtained by setting the first derivative of the log likelihood function equal to zero, i.e.,

$$\frac{dl}{dh_i} = 0 = \frac{m_{i1}}{h_i} - \frac{m_{i0}}{1-h_i}$$

$$\text{or } h_i = \frac{m_{i1}}{m_{i0} + m_{i1}}$$

[Q.E.D.]

4.5 MEASURES OF GOODNESS

In the case of Gaussian random variables it is common to define the "best estimate" in the least-squares sense (LSE), i.e., $\hat{z}$ is the best estimate of z if $E[(z-\hat{z})^2]$ is minimum. In the case of binary variables, the role is played by what we propose to call the "Least XOR Estimate" (LXE), and the "Least Cost Estimate" (LCE).

3.1 Least XOR Estimate : Since both z and $\hat{z}$ can take only two values, there are only 4 cases to be considered as shown below. Here e is used to denote the error variable.

z	$\hat{z}$	Error	e
0	0	No	0
0	1	Yes	1
1	0	Yes	1
1	1	No	0

It is easy to see from the above table that $e = z\oplus\hat{z}$ (exclusive-or of z and $\hat{z}$). The minimum number of error cases will be obtained if $E[z\oplus\hat{z}]$ is minimum. The estimate $\hat{z}$ which minimizes $E[z\oplus\hat{z}]$ is the least XOR estimate (also the least error estimate). It is easy to verify that LXE is equivalent to LSE for binary variables, i.e., $(z-\hat{z})^2 = z\oplus\hat{z}$.

3.2 Least Cost Estimate : In formulating LXE, it was assumed that both kinds of error $z=1,\hat{z}=0$ and $z=0,\hat{z}=1$ are equally costly. In the signal processing area, these two errors are called "miss signal" and "false

alarm" respectively. The cost of these two types of errors is generally different. For example, in the case of weather prediction, the cost of predicting a storm and not actually getting one is quite different from that of getting an unpredicted storm. Similarly, in the case of memory management, the cost of a page fault (miss signal) is not always the same as the cost of keeping an unused page for some time (false alarm).

In such cases, therefore, we propose a generalized concept to be called the "least cost estimate" or LCE. In this case, the cost function $C(z,\hat{z})$ is a given, not necesarily linear, function of z and $\hat{z}$. Now by Shannon's decomposition theorem, we can express C as follows :

$$C(z,\hat{z}) = c_0\bar{z}\bar{\hat{z}} + c_1\bar{z}\hat{z} + c_2 z\bar{\hat{z}} + c_3 z\hat{z}$$

Where $c_0 = C(0,0)$, $c_1 = C(0,1)$, $c_2 = C(1,0)$, $c_3 = C(1,1)$.

Here c_2 and c_1 are the costs of a miss signal and a false alarm respectively. Without loss of generality, we can assume that $c_0 = c_3 = 0$. This is because

$$C(z,\hat{z}) = \{c_0\bar{z} + c_3 z\} + \{(c_1-c_0)\bar{z}\hat{z} + (c_2-c_3)z\bar{\hat{z}}\}$$

The part within the first set of braces is independent of $\hat{z}$, and hence the problem is equivalent to one with cost of miss signal c_2-c_3, cost of false alarm c_1-c_0, and zero cost for correct prediction.

4.6 PREDICTION

We now return to our original problem of finding the Boolean function g, such that the estimate $\hat{z}(t) = g(Z_{tn})$ minimizes a given cost function. The prediction method that we are going to describe is based on the two theorems below.

4.6.1 Theorem [Prediction Theorem] : Given the model relating z(t) to Z_{tn}

$$z(t) = \sum_{i=0}^{2^n-1} e_i(t) q_i(Z_{tn})$$

the estimate $\hat{z}(t)$ which minimizes the expected value of cost function $C(z(t),\hat{z}(t))$ for N observations is given by

$$\hat{z}(t) = \sum_{i=0}^{2^n-1} \hat{e}_i q_i(Z_{tn})$$

where $\hat{e}_i$, $i=0,1,\ldots,2^n-1$ are zero-one valued variables chosen as follows :

$$\hat{e}_i = \begin{cases} 1, & \text{if } h_i > r \\ 0, & \text{if } h_i < r \\ d, & \text{if } h_i = r \end{cases}$$

where $r = \dfrac{c_1}{c_1 + c_2}$

and d represents a "don't care" condition, i.e., either 0 or 1 would do equally well.

Proof : Let the desired estimate be

$$\hat{z}(t) = g(Z_{tn})$$

where $g(Z_{tn})$ is a Boolean function of Z_{tn}. Again, using the Decomposition Theorem, we have

$$\hat{z}(t) = \sum_{i=0}^{2^n-1} \hat{e}_i q_i(Z_{tn})$$

where $\hat{e}_i$ is a zero-one valued variable given by $\hat{e}_i = g(Z_{tn}|Z_{tn}=i)$.

Since $$z(t) = \sum_{i=0}^{2^n-1} e_i(t) q_i(Z_{tn})$$

the exclusion property of the fundamental products enables us to write the cost function as follows:

$$C(z(t),\hat{z}(t)) = c_1 z(t)\hat{z}(t) + c_2 z(t)\bar{\hat{z}}(t)$$

$$= c_1 \sum_{i=0}^{2^n-1} \bar{e}_i(t)\hat{e}_i q_i(Z_{tn}) + c_2 \sum_{i=0}^{2^n-1} e_i(t)\bar{\hat{e}}_i q_i(Z_{tn})$$

$$= \sum_{i=0}^{2^n-1} \{c_1 \bar{e}_i(t)\hat{e}_i + c_2 e_i(t)\bar{\hat{e}}_i\} q_m(Z_{tn})$$

Taking expectation, we have

$$E[C(z(t),\hat{z}(t))] = \sum_{i=0}^{2^n-1} (c_1 \bar{h}_i \hat{e}_i + c_2 h_i \bar{\hat{e}}_i) E[q_i(Z_{tn})]$$

$$= \sum_{i=0}^{2^n-1} C(h_i,\hat{e}_i) E[q_i(Z_{tn})]$$

Thus we have decomposed the expected cost into 2^n small components each of which can be independently optimized. Consider the i^{th} component $C(h_i,\hat{e}_i)$:

$$C(h_i,\hat{e}_i) = c_1 \bar{h}_i \hat{e}_i + c_2 h_i \bar{\hat{e}}_i = c_2 h_i + \hat{e}_i\{c_1 - (c_1 + c_2)h_i\}$$

The last expression is linear in $\hat{e}_i$. It is minimum if each $\hat{e}_i$ is chosen as stated in the theorem.

[Q.E.D.]

Notice the similarity in expressions for $z(t)$ and $\hat{z}(t)$. The expression for $\hat{z}(t)$ can be obtained from that for $z(t)$ by replacing $e_i(t)$ by $\hat{e}_i$. In fact, $\hat{e}_i$ is the best estimate of the binary white noise $e_i(t)$ if the cost function is $C(e_i(t),\hat{e}_i)$.

4.6.2 Theorem [Total Cost Theorem] : The total cost of imperfect prediction for N observations by using the Prediction Theorem is

$$TC = \sum_{i=0}^{2^n-1} \min(c_2 m_{i1}, c_1 m_{i0})$$

Proof : $$TC = \sum_{j=1}^{N} C(z(j),\hat{z}(j))$$

$$= \sum_{j=1}^{N} (c_1 \bar{z}(j)\hat{z}(j) + c_2 z(j)\bar{\hat{z}}(j))$$

Now $$\sum_{j=1}^{N} \bar{z}(j)\hat{z}(j) = \sum_{j=1}^{N} \bar{z}(j) \sum_{i=0}^{2^n-1} \hat{e}_i q_i(Z_{jn})$$

$$= \sum_{i=0}^{2^n-1} \hat{e}_i \sum_{j=1}^{N} \bar{z}(j) q_i(Z_{jn})$$

$$= \sum_{i=0}^{2^n-1} \hat{e}_i m_{i0}$$

Similarly, $$\sum_{j=1}^{N} z(j)\bar{\hat{z}}(j) = \sum_{i=0}^{2^n-1} \bar{\hat{e}}_i m_{i1}$$

Hence, $$TC = \sum_{i=0}^{2^n-1} (c_1 \hat{e}_i m_{i0} + c_2 \bar{\hat{e}}_i m_{i1})$$

$$= \sum_{i=0}^{2^n-1} \min(c_1 m_{i0},\ c_2 m_{i1})$$

The last equality follows from the observation that $\hat{e}_i$ is a binary variable, hence the sum $c_1\hat{e}_i m_{i0} + c_2\bar{\hat{e}}_i m_{i1}$ is either $c_1 m_{i0}$ or $c_2 m_{i1}$. The prediction theorem chooses $\hat{e}_i$ in such a way that the coefficient of the larger term is zero.

[Q.E.D.]

4.7 TABULAR METHOD OF BINARY PROCESS PREDICTION

This method is a result of combining the three theorems described before, viz., the Estimation Theorem, the Prediction Theorem, and the Total Cost Theorem. The method consists of the following steps :

1. Summarize the observed data in terms of frequency of occurrence of various fundamental product terms. The summary is arranged in a tabular form as shown in Table 4.1 . The table has 2^n-1 rows and 5 columns. The columns are named Z, M, H, $\hat{e}$, and TC respectively.
2. The Z column consists of n subcolumns corresponding to n variables $z(t-n), z(t-n+1), \ldots, z(t-1)$. The i^{th} row in this column is simply the n digit binary expansion of i-1.
3. The M column consists of 2 subcolumns corresponding to $z(t)=0$, and $z(t)=1$ respectively. The entry in the i^{th} row of the first subcolumn is m_{i0}, i.e., the number of observations with $z(t)=0$ and $z_{tn}=i$. Similarly, the entry in the second subcolumn is the number of observations with $z(t)=1$ and $Z_{tn}=i$.
4. The entries in the h_i column are obtained from those in the M column as follows :

$$h_i = \frac{m_{i1}}{m_{i0} + m_{i1}}$$

$$= \frac{\text{entry in the } z(t)=1 \text{ subcolumn}}{\text{sum of entries in the } z(t)=1 \text{ and } z(t)=0 \text{ subcolumns}}$$

5. The entries in the $\hat{e}$ column are either 0, 1, or d according as h_i is less than, greater than, or equal to the ratio $r = c_1/(c_1 + c_2)$.

If in a particular row, both m_{i0}, and m_{i1} are zero, the $\hat{e}$ entry in that row is d.

6. The entries in the TC column are calculated according to the Total Cost Theorem, i.e., the i^{th} entry is $\min(c_2 m_{i1}, c_1 m_{i0})$.
7. Synthesize the Boolean function represented by the $\hat{e}$ column. This is the optimum predictor. In sum of product form the function is simply $\sum_{i=0}^{2^n-1} \hat{e}_i q_i(Z_{tn})$.
8. The goodness of fit is given by the total cost calculated by summing up the TC column.

We now illustrate the method with an example.

4.7.1 Example : The data consists of 144 observations on a 4th order binary process. The actual observations have not been included here, instead, the frequency of occurrence of the various combinations is presented in Table 4.2. The cost of a false alarm is twice that of a miss signal, i.e., $c_1=2$ and $c_2=1$. The ratio $r = \frac{2}{2+1} = \frac{2}{3}$. The h_i column is constructed as usual. The entries in the $\hat{e}$ column are 1 or 0 according as the entries in the h_i column are greater or less than 2/3. Two of the h_i's are exactly equal to 2/3. Hence, the $\hat{e}$ entries in these rows are "don't care" entries marked as d_1 and d_2 respectively. The predictor corresponding to $d_1 d_2 = 00$ is

$$\begin{aligned}\hat{z}(t) = {} & z(t-4)z(t-3)z(t-2)z(t-1) + z(t-4)z(t-3)z(t-2)z(t-1) \\ & + z(t-4)z(t-3)z(t-2)z(t-1) + z(t-4)z(t-3)z(t-2)z(t-1) \\ & + z(t-4)z(t-3)z(t-2)z(t-1) + z(t-4)z(t-3)z(t-2)z(t-1) \\ & + z(t-4)z(t-3)z(t-2)z(t-1)\end{aligned}$$

$$= 1 - z(t-1) - z(t-3) - z(t-4) + z(t-1)z(t-3) + 2z(t-1)z(t-4)$$
$$+ z(t-2)z(t-4) + 2z(t-3)z(t-4) - z(t-1)z(t-2)z(t-4)$$
$$- 3z(t-1)z(t-3)z(t-4) - 2z(t-2)z(t-3)z(t-4)$$
$$+ 3z(t-1)z(t-2)z(t-3)z(t-4)$$

Similar equations can be written for 3 other equally good predictors corresponding to d_1d_2=01, 10, 11. All these predictors give the same total cost of 50. □

TABLE 4.1 : Tabular Arrangement for Boolean Model

z(t-n)	...	z(t-2)	z(t-1)	# of obsv. with z(t)=0	# of obsv. with z(t)=1	h_i	$\hat{e}_i$	TC_i
0	...	0	0	m_{00}	m_{01}	$\frac{m_{01}}{m_{00}+m_{01}}$	...	...
0	...	0	1	m_{10}	m_{11}	...	...	...
0	...	1	0	m_{20}	m_{21}	...	...	...
0	...	1	1	m_{30}	m_{31}	...	...	...
	...							
	...							
	...							
1	...	1	1	$m_{2^n-1,0}$	$m_{2^n-1,1}$	...	...	...

TABLE 4.2 : Frequency Distribution for Data of Example 4.7.1

z(t-4)	z(t-3)	z(t-2)	z(t-1)	m_{i0}	m_{i1}	h_i	$\hat{e}$	$\min(2m_{i0}, m_{i1})$
0	0	0	0	1	9	0.90	1	2
0	0	0	1	8	2	0.20	0	2
0	0	1	0	3	8	0.73	1	6
0	0	1	1	7	1	0.13	0	1
0	1	0	0	3	2	0.40	0	2
0	1	0	1	9	7	0.44	0	7
0	1	1	0	2	4	0.67	d_1	4
0	1	1	1	6	0	0.00	0	0
1	0	0	0	5	3	0.38	0	3
1	0	0	1	1	8	0.89	1	2
1	0	1	0	2	9	0.82	1	4
1	0	1	1	0	7	1.00	1	0
1	1	0	0	2	8	0.80	1	4
1	1	0	1	7	5	0.42	0	5
1	1	1	0	1	2	0.67	d_2	2
1	1	1	1	3	9	0.75	1	6

Total Cost = 50

4.8 GENERALIZATION TO K-ARY VARIABLES

In this section we generalize the analysis done so far to the case where the process may take k values 0,1,...,k-1. To do this generalization, we use the concept of Boolean functions extended for k-ary variables. This concept is due to Rosenberg [HaR68, p. 301].

Let $B_k = \{0,1,\ldots,k-1\}$. A Boolean function is now defined as a mapping $f : (B_k)^n \to B_k$ and a pseudo-Boolean function as $f : (B_k)^n \to R$. For any xGBc¬bc¬ , we define the so called "Lagrangean Development" x^i (x sup i) as :

$$x^i = \frac{x(x-1)\ldots(x-i+1)(x-i-1)\ldots(x-k+1)}{i(i-1)\ldots1(-1)\ldots(i-k+1)}$$

k

mapping B_k into B_2. For example, when k=3:

$$x^0 = \frac{1}{2}(x-1)(x-2) \qquad x^1 = -x(x-2) \qquad x^2 = \frac{1}{2}x(x-1)$$

Notice that

$$x^i = \begin{cases} 1 & \text{if } x=i \\ 0 & \text{otherwise} \end{cases}$$

Let $i_1,i_2,\ldots,i_n$ be the k-ary expansion of i, and $X = \{x_1,x_2,\ldots x_n\}$

then $X^i = q_i(X) = x_1^{i_1}x_2^{i_2} \ldots x_n^{i_n}$

$= i^{th}$ Fundamental product

Any pseudo-Boolean function has a Lagrangean development (sum of products form) :

$$f(x_1,\ldots,x_n) = \sum_{i=0}^{k^n-1} f(i)X^i = \sum_{i=0}^{k^n-1} f_i q_i(X)$$

Again,

$$q_i(X) = \begin{cases} 1 & \text{if } X=i \\ 0 & \text{otherwise} \end{cases}$$

Therefore, the fundamental products are mutually disjoint, i.e.,

$$q_i(X)q_j(X) = \begin{cases} 0 & i \neq j \\ q_i(X) & i=j \end{cases}$$

So, the Function Lemma holds, i.e., if f is a real valued function of a real variable, and

$$\text{if} \quad p = \sum_{i=0}^{k^n-1} h_i q_i(X)$$

$$\text{then } f(p) = \sum_{i=0}^{k^n-1} f(h_i) q_i(X)$$

4.8.1 Model : The relationship between z(t) and Z_{tn} in its most general form is given by

$$z(t) = \sum_{i=0}^{k^n-1} e_i(t) q_i(Z_{tn})$$

where $e_i(t)$ is a k-ary white noise (sequence of independent identically distributed random variables) with $P[e_i(t)=u]=h_{iu}$. Hence,

$$P_{ut} = P[z(t)=u|Z_{tn}] = \sum_{i=0}^{k^n-1} h_{iu} q_i(Z_{tn}), \qquad u=0,1,\ldots,k-1$$

There is an additional constraint, however, that

$$\sum_{u=0}^{k-1} P_{ut} = 1$$

This constraint implies that

$$\sum_{u=0}^{k-1} h_{iu} = 1 \qquad i=0,1,\ldots,k^n-1$$

With this model, all the results of the binary case, viz., Estimation theorem, Measures of goodness, Prediction theorem, and Total Cost theorem, can be generalized to the k-ary case. These generalizations are stated below. The proofs of the theorems, being similar to the binary case, are given in Appendix C.

4.8.2 Estimation Theorem : The maximum likelihood estimates of h_{iu} based on N observations $\{z(1),z(2),\ldots,z(N)\}$ are given by

$$h_{iu} = \frac{m_{iu}}{\sum_{u=0}^{k-1} m_{iu}} \qquad u=0,1,\ldots,k-1$$

where m_{iu} = # of times $Z_{tn}=i$ is followed by $z(t)=u$.

4.8.3 Measures of Goodness : In the case of k-ary variables the least cost estimate $\hat{z}(t)$ is obtained by minimizing a general cost function $C(z,\hat{z})$. The function can be expressed in the sum of products form as follows :

$$C(z,\hat{z}) = \sum_{u=0}^{k-1}\sum_{v=0}^{k-1} c_{uv} z^u \hat{z}^v$$

where $c_{uv} = c(u,v)$ = cost of misprediction when $z=u$ and $\hat{z}=v$. It is often easier to specify C as a k by k matrix whose $(u+1,v+1)^{th}$ element is c_{uv}. A special case of the least cost estimate occurs when

$$C(z,\hat{z}) = \frac{1}{2}\sum_{j=0}^{k-1} z^j \hat{z}^j$$

It is easy to verify that in this case,

$$c_{uv} = \begin{cases} 0 \text{ if } u=v \\ 1 \text{ otherwise} \end{cases}$$

i.e., all errors cost the same. Thus, by minimizing this cost function, one obtains the least number of errors. The estimate obtained could be called "Least XOR Estimate", because of the form of the cost function. In general, LXE is not the same as LSE except in the binary case.

4.8.4 Prediction Theorem : Given the model equation

$$z(t) = \sum_{i=0}^{k^n-1} e_i(t) q_i(Z_{tn})$$

The estimate $\hat{z}(t)$ which minimizes the expected value of the cost function $C(z(t),\hat{z}(t))$ is given by

$$\hat{z}(t) = \sum_{i=0}^{k^n-1} \hat{e}_i q_i(Z_{tn})$$

where $\hat{e}_i$, $i=0,1,\ldots,k^n-1$ are k-ary variables chosen as follows :

$$\hat{e}_i = \arg \min_v \sum_{u=0}^{k-1} c_{uv} h_{iu}$$

$$= \arg \min_v \sum_{u=0}^{k-1} c_{uv} m_{iu}$$

4.8.4.1 Corollary : The least XOR estimate ($c_{uv}=1$, $u \neq v$) is given by

$$\hat{e}_i = \arg \max_v m_{iv}$$

4.8.5 Total Cost Theorem : Given a set of N observations on z(t), the total cost of imperfect prediction by using the Prediction Theorem is

$$TC = \sum_{i=0}^{k^n-1} \min_v \sum_{u=0}^{k-1} m_{iu} c_{uv}$$

4.8.5.1 Corollary : The total cost in the case of LXE is given by

$$TC = N - \sum_{i=0}^{k^n-1} \max_v m_{iv}$$

4.8.6 Tabular Method : The method is very similar to that for the binary case. The only addition is an MC column, which is obtained by post-multiplying the M column by the C Matrix. We illustrate the procedure with an example.

4.8.6.1 Example : Consider the problem of predicting a ternary process $z(t)$ of 2^{nd} order. A total of 137 observed values of the process are available. The data, summarized in tabular form, ar shown in Table 4.3. The cost function is

$$C(z(t),\hat{z}(t)) = |z(t) - \hat{z}(t)|$$

Therefore, the cost matrix is

$$C = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

The calculations are shown in the table. The MC column is obtained by post-multiplying the M column by the C matrix. Notice from the table that in the last row, two MC entries are equal.. Therefore, the corresponding $\hat{e}$ entry is d_{01}, which stands for "don't care as long as it is 0 or 1". The optimum regression function corresponding to $d_{01}=0$ is

$$\begin{aligned}\hat{z}(t) &= z^0(t-2)z^0(t-1) + z^0(t-2)z^1(t-1) + z^0(t-2)z^2(t-1) \\ &\quad + 2z^1(t-2)z^0(t-1) + 2z^1(t-2)z^1(t-1) + 2z^1(t-2)z^2(t-1) \\ &= z^0(t-2) + z^1(t-2) \\ &= -\frac{3}{2}(z(t-2))^2 + \frac{5}{2}z(t-2) + 1\end{aligned}$$

An equivalent, rather simple, expression for the above $\hat{z}(t)$ is

$$\hat{z}(t) = 1 +_3 z(t-2) \text{ where } +_3 \text{ denotes "addition modulo 3".}$$

A second predictor, corresponding to $d_{01}=1$, can, similarly, be written. The total cost in either case is 101.

TABLE 4.3 : Boolean Predictor for Data of Example 4.8.6.1

z(t-2)	z(t-1)	m_{i0}	m_{i1}	m_{i2}	h_{i0}	h_{i1}	h_{i2}	MC_0	MC_1	MC_2	$\hat{e}_i$	TC_i
0	0	8	3	6	0.47	0.18	0.35	15	14	19	1	14
0	1	5	6	7	0.28	0.33	0.39	20	12	16	1	12
0	2	2	5	6	0.15	0.38	0.46	17	8	9	1	8
1	0	5	1	7	0.38	0.08	0.54	15	12	11	2	11
1	1	9	1	11	0.43	0.05	0.52	23	20	19	2	19
1	2	3	4	8	0.20	0.27	0.53	20	11	10	2	10
2	0	7	2	2	0.64	0.18	0.18	6	9	16	0	6
2	1	9	4	5	0.50	0.22	0.28	14	14	22	d_{01}	14
2	2	6	3	2	0.55	0.27	0.18	7	8	15	0	7

												101

4.9 ON MODEL ORDER DETERMINATION

In the theory that we have developed so far we have assumed that the model order n is known. In practice, this may not always be true. In the case of Gaussian processes there are many criteria and tests (e.g., see [Aka74]), that allow us to determine an optimal model order from empirical data. The corresponding results for Boolean models are yet to be developed. Some rudimentary ideas on this problem are presented in this section.

It should be obvious that the prediction error (or the total cost of prediction) monotonically decreases as the model order is increased. A quantitative formula for the increase in error is given by the following theorem.

4.9.1 Theorem : The increase in cost in going from a $(n+1)^{st}$ order model to n^{th} order model is given by:

$$TC(n)-TC(n+1) = \sum_{i=0}^{2^n-1} [\, \min\{ c_2(m_{i1}+m_{i',1}), c_1(m_{i0}+m_{i',0})\} - \min(c_2 m_{i1}, c_1 m_{i0}) - \min(c_2 m_{i',1}, c_1 m_{i',0}) \,]$$

where the m-values are for the $(n+1)^{st}$ order model, and $i'=2^n+i$.

Proof : Let m' denote the m-values for the n^{th} order model. For example,

m'_{i1} = # of times $Z_{tn} = i$ is followed by $z(t)=1$
= # of times $\underline{z(t-n-1)=0}$, $Z_{tn}=i$ is followed by $z(t)=1$
+ # of times $\underline{z(t-n-1)=1}$, $Z_{tn}=i$ is followed by $z(t)=1$

$$= \#\text{ of times } Z_{tn+1} = i \text{ is followed by } z(t)=1$$

$$+ \#\text{ of times } Z_{tn+1} = 2^n+i \text{ is followed by } z(t)=1$$

$$= m_{i1} + m_{i'1}$$

Similarly, $m'_{i0} = m_{i0}+m_{i'0}$

$$\text{Now } TC(n) = \sum_{i=0}^{2^n-1} \min(c_2 m'_{i1}, c_1 m'_{i0})$$

$$\text{and } TC(n+1) = \sum_{i=0}^{2^{n+1}-1} \min(c_2 m_{i1}, c_1 m_{i0})$$

$$= \sum_{i=0}^{2^n-1} [\min(c_2 m_{i1}, c_1 m_{i0}) + \min(c_2 m_{i'1}, c_1 m_{i'0})]$$

Notice that in the last equation the upper limit of the summation is 2^n-1 instead of $2^{n+1}-1$. The difference of the above two equations gives the theorem as stated.

[Q.E.D.]

There are two implications of this theorem. Firstly, each summation term is of the form "the minimum of sums minus the sum of minima". Hence, each term is non-negative. This proves the statement that the cost monotonically goes down. The second implication, which becomes obvious from the proof, is that the m-values for the n^{th} order model can be obtained from those of the $(n+1)^{st}$ order model by summing up values that are 2^n apart. Thus, once the data has been summarized in a tabular form for high enough n, all lower order models can be easily calculated. An example is shown in Table 4.4. Here m_{21} for the 2^{nd} order model is obtained by adding m_{21} and m_{61} ($6 = 2+2^n$, $n=2$) of the 3^{rd} order model and so on. Thus, starting from a large n, one can calculate

the total cost TC(n) for that and all lower order models. A plot of TC(n) vs n will look similar to that shown in Figure 4.2a.

In choosing the model order, a compromise must be made between the amount of computation required for the model and the improvement obtainable by the model. The complexity of the model is exponential, i.e., $O(2^n)$. Hence, the net utility of an n^{th} order model is $TC(n)-a2^n$, where a is some normalizing constant. The optimal order is obviously the one that maximizes this utility (see Figure 4.2). Another fact that should be pointed out in this regard is that as the model order increases, the number of parameters to be estimated increases and, hence, the precision (or confidence) of parameter values may go down. As at present, we do not have formulae for parameter confidence.

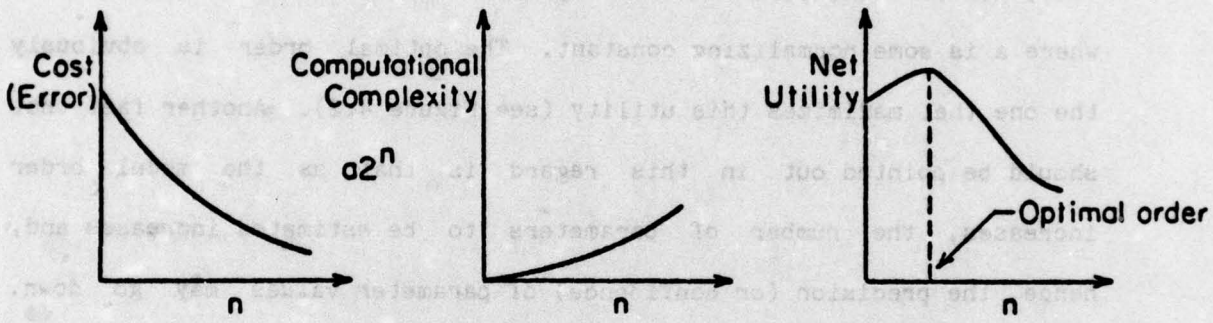


Figure 4.2: Determination of model order n.

Table 4.4 : Calculation of the Total Cost TC(n) for Lower Order Models
$c_1 = 2,\ c_2 = 1,\ TC_i = \min(c_1 m_{i0},\ c_2 m_{i1})$

n=3				n=2				n=1				n=0			
i	m_{i0}	m_{i1}	TC_i	i	m_{i0}	m_{i1}	TC_i	i	m_{i0}	m_{i1}	TC_i	i	m_{i0}	m_{i1}	TC_i
0	8	12	12	0	12	22	22	0	19	45	38	0	60	84	84
1	9	10	10	1	25	22	22	1	41	39	39			TC(0) =	84
2	5	6	6	2	7	23	14			TC(1) =	77				
3	7	8	8	3	16	17	17								
4	4	10	8			TC(2) =	75								
5	16	12	12												
6	2	17	4												
7	9	9	9												
		TC(3) =	69												

4.10 ON STATIONARITY

A stochastic process is called stationary if its probabilistic behavior is independent of the time origin. In our Boolean model we assumed $P[z(t)|Z_{tn}] = h(Z_{tn})$ to be independent of time. For a stationary process this is obviously a valid assumption. For a general non-stationary process the model should be

$$P[z(t)|Z_{tn}] = h(t,Z_{tn}) = \sum_{i=0}^{2^n-1} h_i(t)q_i(Z_{tn})$$

i.e., the model parameters are functions of time. We do not know how to estimate these time varying parameters. Nor do we have any tests for stationarity (similar to the ACF going to zero for continuous processes). However, what we do know is that the time-independent Boolean model applies also to the so called "Homogeneous non-stationary" processes. A non-stationary process is called homogeneous if its d^{th} difference is stationary for some d. Recall that in the case of continuous processes ARIMA rather than ARMA models are used to model such homogeneous processes. The following theorem proves the above statement.

Theorem : If the d^{th} difference of a k-ary process z(t) follows an n^{th} order time-independent Boolean equation then the process itself follows a $(d+n)^{th}$ order time-independent equation.

Proof : Consider the 1^{st} difference process y(t)

$$y(t) = z(t) - z(t-1)$$

So that $$\sum_{j=1}^{t} y(j) = z(t)-z(0)$$

Hence, $P[z(t)|Z_{tn+1}] = P[z(t)|z(t-n-1), z(t-n),\ldots,z(t-1)]$

$$= P[z(0)+\sum_{j=1}^{t} y(j)|z(0)+\sum_{j=1}^{t-n-1} y(j),\ z(0)+\sum_{j=1}^{t-n-2} y(j),\ \ldots,$$

$$z(0)+\sum_{j=1}^{t-1} y(j)]$$

$$= P[y(t)|z(0)+\sum_{j=1}^{t-n-1} y(j),\ y(t-n),\ y(t-n+1),\ \ldots,\ y(t-1)]$$

$$= P[y(t)|y(t-n),\ y(t-n+1),\ \ldots,\ y(t-1)]$$

= Independent of time if y(t) is n^{th} order

Thus we see that if the first difference y(t) has n^{th} order time-independent Boolean model, then z(t) has $(n+1)^{st}$ order time-independent model. By taking successive difference, the theorem for d^{th} difference follows.

[Q.E.D.]

The implication of this theorem is that we can use the theory developed so far for our page refergnce process whose 1^{st} difference was shown to be stationary in the last chapter.

4.11 PAGE REPLACEMENT USING THE BOOLEAN MODEL

There are two ways of using the Boolean model for page replacement. The first is simply to use the model to predict z(t) from the knowledge of z(t-1), ..., z(t-n). In this case, one must choose as the modeling interval T = R/U, the ratio of replacement and usage costs. As was shown in the last chapter, the cost criterion in this case is least-squares. This method is straightforward and we do not develop it any further here.

An alternative method arises from the fact that with the Boolean model we are not restricted to the least-squares cost criterion. Hence, we can design a page replacement policy without any restriction on T. In this section we develop such a policy. The policy is a realizable version of a theoretically optimal, but unrealizable, policy called VMIN [PrF76]. In order to see the optimality of VMIN, consider a particular page. Without loss of generality, we can assume that the modeling interval T is unity. Supposing we know the complete page reference process (past as well as future), let s be the length of time for which the page is not referenced following t, i.e., z(t), z(t+1),..., z(t+s-1) are all zero and z(t+s) is 1.

Let d(t) = Decision to remove the page at time t.

$$= \begin{cases} 1 \Rightarrow \text{page is removed} \\ 0 \Rightarrow \text{page is kept in the main memory} \end{cases}$$

The cost of the decision d(t) over the interval (t, t+s) is

$$C = Rd(t) + sUd(t) = sU + (R-sU)d(t)$$

Since the cost is linear in d(t), the optimal decision is d(t)=1 iff R-sU < 0, i.e., d(t)=1 iff $s > \frac{R}{U}$.

Thus the optimal policy is to keep the page if it is going to be referenced in the next R/U interval. This is the VMIN policy. However, this is unrealizable because it uses both past and future information. A realizable version that uses only past information can be derived as follows.

Since the future is unknown, the "forward recurrence time" s is a random variable and the expected cost E[C] = Rd(t) + U$\bar{d}$(t)E[s] is to be minimized. The optimal decision based on the past information is, therefore, to choose d(t)=1 iff E[s] is greater than R/U. The distribution of s, and hence its expected value can be derived from the Boolean model of the process as described by the following two theorems.

4.11.1 Theorem* : The "reverse cumulative distribution" (1-cumulative distribution) of s is given by

$$r_{iu} = P[s>u|Z_{tn}=i]$$

$$= \prod_{j=0}^{u} \bar{h}_{2^j i}$$

---

* In this chapter we use the convention that whenever i appears in a subscript, the expression upto i is evaluated modulo 2^n. Anything following i is simply another subscript. Thus r_{iu} is "r sub i comma u", whereas, h_{3i} is "h sub (3 times i modulo 2^n)". For example, for n=2, i=3, $h_{3i}=h_9=h_1$.

Proof : $r_{i0} = P[s>0|Z_{tn}=i]$

$= P[z(t)=0|Z_{tn}=i]$

$=h_i$

$r_{iu} = P[s>u|Z_{tn}=i]$

$= P[z(t+u)=0, z(t+u-1)=0, \ldots, z(t)=0|Z_{tn}=i]$

$= P[z(t+u)=0|Z_{tn}=i, z(t)=0,\ldots,z(t-u-1)=0]$

$P[z(t-u-1)=0,\ldots,z(t)=0|Z_{tn}=i]$

$= P[z(t+u)=0|z_{u+t,n}=2^{u}i]r_{i,u-1}$

$=h_{2^{u}i}r_{i,u-1}$

The above equation gives a recursive expression for r_{iu}. By applying the recursion for successively decreasing value of u, and using the initial condition r_{i0} we get the theorem as stated.

[Q.E.D.]

4.11.1.1 Corollary : $p_{iu} = P[s=u|Z_{tn}=i] = h_{2^{u}i}\prod_{j=0}^{u-1}\bar{h}_{2^{j}i}$

Proof : $p_{iu} = r_{i,u-1} - r_{i,u} = \prod_{j=0}^{u-1}\bar{h}_{2^{j}i} - \prod_{j=0}^{u}\bar{h}_{2^{j}i} = h_{2^{u}i}\prod_{j=0}^{u-1}\bar{h}_{2^{j}i}$

[Q.E.D.]

4.11.2 Theorem : Let $s_i = E[s|Z_{tn}=i]$. Then, s_i is given by the following recursive equation:

$s_i = \bar{h}_i(1+s_{2i}) \qquad i=0,1,\ldots,2^{n}-1$

and $s_0 = \dfrac{\bar{h}_0}{h_0}$

Proof : $s_i = E[s|Z_{tn}=i]$

$$= \sum_{u=0}^{\infty} u p_{iu}$$

$$= \sum_{u=0}^{\infty} u(r_{i,u-1} - r_{iu})$$

$$= 1(r_{i0}-r_{i1}) + 2(r_{i1}-r_{i2}) + 3(r_{i2}-r_{i3}) + \ldots$$

$$= r_{i0} + r_{i1} + r_{i2} + \ldots$$

$$= \bar{h}_i + \bar{h}_i\bar{h}_{2i} + \bar{h}_i\bar{h}_{2i}\bar{h}_{4i} + \ldots$$

$$= \bar{h}_i (1 + \bar{h}_{2i} + \bar{h}_{2i}\bar{h}_{4i} + \ldots)$$

$$= \bar{h}_i (1 + s_{2i})$$

By substituting i=0, in the above equation we get

$$s_0 = \bar{h}_0 (1+s_0)$$

or $$s_0 = \frac{\bar{h}_0}{\bar{h}_0}$$

Alternatively, one can derive an expression for s_0 from $\sum_{u=0}^{\infty} u P[s=u|Z_{tn}=0]$. The result is the same as above.

[Q.E.D.]

Using theorem 4.11.2 one can get an expression for all s_i, $i=0,1,2,\ldots,2^n-1$ in terms of h_i. However, one must follow a particular order. After calculating s_i for i=v, calculate s_i for i=v/2 and $2^{n-1}+v/2$. For example, for n=2 the following expressions are obtained.

$$s_0 = \frac{\bar{h}_0}{\bar{h}_0} = \frac{m_{00}}{m_{01}}$$

$$s_2 = \bar{h}_2(1+s_4) = \bar{h}_2(1+s_0) = \frac{\bar{h}_2}{\bar{h}_0} = \frac{m_{20}}{m_{20}+m_{21}} \quad \frac{m_{00}+m_{01}}{m_{01}}$$

$$s_1 = \bar{n}_1(1+s_2) = \bar{n}_1(1+\frac{\bar{h}_2}{\bar{h}_0}) = \frac{m_{10}}{m_{10}+m_{11}}(1+\frac{m_{20}(m_{00}+m_{01})}{m_{01}(m_{20}+m_{21})})$$

$$s_3 = \bar{n}_3(1+s_6) = \bar{n}_3(1+s_2) = \bar{n}_3(1+\frac{\bar{h}_2}{\bar{h}_0}) = \frac{m_{30}}{m_{30}+m_{31}}(1+\frac{m_{20}(m_{00}+m_{01})}{m_{01}(m_{20}+m_{21})})$$

Thus, using the estimated value of the forward recurrence time for the current state, one can decide whether to replace a page or not. This version of the "VMIN" algorithm, although realizable, is too expensive to implement in practice. This is because for an n^{th} order model 2^{n+1} registers are required to hold m-values. This number is prohibitively large. Even for n=2, eight registers are required for each page. To manage a page of 1048 words, using eight registers is not economical. Therefore, at the present time we do not discuss implementation aspects of the algorithms developed in this chapter. However, with rapidly advancing memory technology it is quite possible that pages of future will be much bigger and eight registers per page would then not be an expensive proposition. If that happens, it might be interesting to do empirical analysis of page reference process and develop a Boolean model of it.

4.12 CONCLUSION

In this chapter we have proposed a direct approach to modeling, estimation, and prediction of a k-ary process. The process is modeled as the output of a Boolean system driven by a set of k-ary white noises. The model makes use of the special properties of pseudo-Boolean functions in sum of products form. An expression for the likelihood function has been developed. Using this expression a formula for calculating the maximum likelihood estimate of the model parameters has been derived. A method of finding the optimal non-linear predictor has been developed. The method makes use of the sample frequency distributions of the fundamental products.

Two different ways of designing a memory management policies based on the Boolean model have been presented. The algorithms, although physicaly realizable, are not economical enough for practical implementation at the current state of technology. However, the research reported here is valuable from the control-theoretic point of view for application to other systems.

In the case of Gaussian variables, the joint probability distribution of n variables is completely specified by specifying the mean and covariance matrix. Therefore, while analyzing Gaussian processes, we summarize the data in terms of the autocorrelation function. In the case of binary variables, we find that autocorrelation has no importance. Instead, the role is played by nth order moments -

the expected values of fundamental product terms. Thus, whereas the sufficient statistic* of a sample of n Gaussian variables has $n(n + 1)/2$ terms (n means, $n(n-1)/2$ variances), that of n binary variables has 2^n-1 (k^n-1 in the k-ary case) terms.

We have partly resolved the question of "representation". In the case of Gaussian processes, the representation theorem [Ast70] states that every stationary stochastic process with rational spectral density can be represented as the output of a linear system driven by white noise. In this chapter, we have shown that any stationary finite order Markov process can be represented as the output of a Boolean system driven by a set of k-ary noises.

The Boolean approach to the analysis of k-ary process parallels to that conventionally used for Gaussian processes. However, as this is the first time that this approach has been taken, many issues remain to be resolved. In particular, the problem of order determination needs further research. Nevertheless, some of the results obtained are more general than those known for Gaussian processes. For example, our model gives the optimal non-linear predictor for any given linear or non-linear cost function, whereas most of the literature on Gaussian processes deals with the optimal linear predictor for the least-squares cost function.

---

* The sufficient statistic is the minimal set of statistical summaries that contains all the useful information in the sample data.

CHAPTER V

CONCLUSIONS

5.1 SUMMARY OF RESULTS

Most resource management problems are basically prediction problems. Therefore, we advocate the use of modern stochastic control theory to formulate operating systems resource management policies. In this thesis, we have proposed a general approach to the prediction of resource demands of a program based on its past behavior.

We exemplified the approach by applying it to the problems of CPU management and memory management. One interesting outcome of the research reported here is that our control-theoretic approach also provides an explanation for many previously described policies that are based on completely non-control-theoretic principles.

In the case of CPU management, it was shown that the successive CPU demands of a program constitute a stationary white noise process. Therefore, the best predictor for the future demand is the current mean value. Several different schemes for adaptively predicting the demand were proposed. An adaptive scheduling algorithm called SPRPT was described. It turns out that Dijkstra's "T.H.E." operating system uses a scheme similar to one of the proposed ones. Thus, we also have a control-theoretic derivation and explanation of T.H.E.'s CPU management policy.

In the case of memory management, we started with a very simple stochastic process model and still obtained significant results. We showed that the cost of memory management is proportional to the square

of the prediction error. Empirical analysis showed that the process is non-stationary and that an ARIMA(1,1,1) model is appropriate. A new page replacement algorithm called "ARIMA" was, therefore, proposed. The algorithm is not only easy to implement, it also unifies many other algorithms previously cited in the literature. In particular, Working Set and the Independent Reference Models were shown to be boundary cases of the algorithm proposed in this thesis.

The memory management process is a binary process. The absence of suitable techniques for prediction of such processes led us to develop new techniques for modeling, estimation and prediction of binary processes. We later extended these techniques to k-ary processes also. Our approach was to model these processes as the output of a Boolean system. This "Boolean approach" allowed us to find the optimal non-linear predictor for the process under any given non-linear cost function. The model was shown to be applicable to a subclass of non-stationary processes also. However, when applied to the memory management problem, the resulting algorithm, though optimal, is rather expensive to implement for currently used page sizes. Nevertheless, the research reported here is important from the control-theoretic viewpoint for application to other systems.

5.2 DIRECTIONS FOR FUTURE RESEARCH

There are many avenues along which the research reported in this thesis can be extended. The first possibility is to investigate the problem of joint management of CPU and memory. In this thesis, CPU and memory demands have been modeled as independent processes. Strictly speaking this is not true; the CPU demand is affected by the memory policy. For example, a bad memory policy may result in frequent page faults causing tasks to be descheduled prematurely.

As far as the analysis of binary or k-ary processes is concerned, there are many issues that need to be resolved. In particular, the problem of order determination needs further research. Tests for stationarity and models for non-stationary k-ary processes should be developed. The possibility of using less expensive, though suboptimal, predictors should be investigated. This is particularly desirable for application to memory management.

The control-theoretic approach can be extended to the management of other resources, e.g., disks. The disk scheduling policy can be optimized if the disk demand behavior of programs is predicted in advance.

The approach can also be used for the modeling of other systems. For example, in a database, the record access patterns can be modeled as a stochastic process and its prediction used to determine the optimal organization and, hence, the reorganization points of the database. In

the case of computer networks, the arrival patterns of packets at a node can be modeled as a binary stochastic process. The forecast of future packet arrivals can then be used for flow control or to avoid congestion in the network.

The essence of our philosophy in this thesis is that control-theorists have made good use of computers to develop better and faster modeling, estimation and prediction techniques. It is now time for computer scientists to use these techniques to develop better and faster computer systems.

APPENDIX A: A PRIMER ON ARIMA MODELS

A stochastic process is a sequence of random variables, say, z(1), z(2),...,z(t),... . The simplest stochastic process is white noise e(t). It has the property that any two elements e(t) and e(t+k) of the process are not correlated. The process in which only consecutive elements, i.e., z(t) and z(t+1), t=1,2,... are correlated is represented by a moving average model of order 1 (MA(1)):

$$z(t) = w + e(t) - b_1 e(t-1)$$

Here the expression "$e(t) - b_1 e(t-1)$" represents a moving average of the white noise process e(t), and w is a constant used to balance the mean on the two sides of the equation. A moving average process of order q (MA(q)) is similarly represented by

$$z(t) = w + e(t) - b_1 e(t-1) - b_2 e(t-2) - \ldots - b_q e(t-q)$$

On the other hand the process represented by

$$z(t) - a_1 z(t-1) - a_2 z(t-1) - \ldots - a_p z(t-p) = w + e(t)$$

is called an autoregressive process of order p (AR(p)). The name clearly indicates that the process z(t) depends (regresses) on its p past values. A process which has both AR and MA parts is called an "ARMA" process. The ARMA(p,q) model is given by the following equation:

$$z(t) - a_1 z(t-1) - \ldots - a_p z(t-p) = w + e(t) - b_1 e(t-1) - \ldots - b_q e(t-q)$$

For a process z(t) its d^{th} difference is defined as follows:

$$Dz(t) = z(t) - z(t-1)$$

$$D^2 z(t) = Dz(t) - Dz(t-1)$$

$$\vdots$$

$$D^d z(t) = D^{d-1} z(t) - D^{d-1} z(t-1)$$

Appendix A
ARIMA models

Now if $y(t) = D^d z(t)$ is the d^{th} difference process of $z(t)$, then $z(t)$ is called the d^{th} integrated process of $y(t)$. Thus if $y(t)$ is shown to be an ARMA(p,q) process, $z(t)$ is said to be an autoregressive integrated moving average process of order p,d,q, i.e., ARIMA(p,d,q). Using the backward shift operator B, $Bz(t) = z(t-1)$, the ARIMA(p,d,q) model can be written as

$$(1-a_1B-\ldots-a_pB^p)(1-B)^d z(t) = w + (1-b_1B-\ldots-b_qB^q)e(t)$$

Further details on ARIMA models are given in [BoJ70].

APPENDIX B

PROOFS OF CPU SCHEDULING THEOREMS

B.1 Proof of Theorem 2.3.1: Let us assume that the tasks $T_0, \ldots, T_{n-1}$ have been so numbered that

$$t_0 < t_1 < \ldots < t_{n-1}$$

This assumption of numbering does not cause any loss of generality. If the tasks are not in the required order, we sort them in the required order. Let k' denote index of the k task in the sorted sequence, then, $t_{0'}, t_{1'}, \ldots, t_{n-1'}$ form the required ordered sequence. The rest of the proof can now be carried our with non-primed subscripts.

Also notice that we assume an increasing sequence rather than a non-decreasing one. Thus we are excluding the possibility of two t_k's being equal. This is only to keep the proof simple. If equality is allowed, the optimal sequence is no longer unique. However, the MFT is same for all optimal sequences, and hence the final cost expression remains the same.

The minimum MFT with known t_i's is

$$MFT_0 = \frac{1}{n} \sum_{i=0}^{n-1} (n-i)t_i$$

Now, if due to predicion error, i^{th} task T_i is placed in $k_i{}^{th}$ position

then

$$MFT_p = \frac{1}{n}\sum_{i=0}^{n-1}(i-k_i)t_i$$

So that the expected increase in MFT is

$$c = E[MFT_p]-MFT_0$$
$$= \sum_{i=0}^{n-1}(i-\bar{i})t_i \qquad \text{where } \bar{i} = E[k_i]$$

It only remains to find an expression for $E[k_i]$.

Since, $E[k_i] = \int_0^\infty E[k_i|t_i=u]f_i(u)du$

We need only show that

$$E[k_i|t_i=u] = \sum_{\substack{j=0\\ j\neq i}}^{n-1} F_j(u)$$

The easiest way to show this for any n and i is by the method of induction. This is obviously true for n=1. Assuming that it is true for n, we now show that it is true for n+1. For a set of n tasks, let $P[k_i=v|t_i=u]=p_{vn}$. Now if a $(n+1)^{st}$ task T_n is added, the task T_i will change position, at most, by 1. Therefore,

$$p_{v,n+1} = p_{vn}\,P[t_n>u] + p_{v-1,n}\,P[t_n\leq u]$$
$$= p_{vn}\{1-F_n(u)\} + p_{v-1,n}\,F_n(u) \qquad v=0,1,\ldots,n-1$$

The boundary conditions are

$$p_{0,n+1} = p_{0n}[1-F_n(u)] \text{ and } p_{n,n+1} = p_{n-1,n}\,F_n(u)$$

Therefore, the expected position of T_i among n+1 tasks is

$$E[k_i|f_i=u,n+1 \text{ tasks}] = \sum_{v=0}^{n} v p_{vn+1}$$

$$= np_{n,n+1} + \sum_{v=1}^{n-1} v p_{vn}[1-F(u)] + v p_{v-1,n} F_n(u)$$

$$= np_{n-1,n} F_n(u) + [1-F_n(u)] \sum_{v=1}^{n-1} v p_{vn} + F_n(u) \sum_{v=1}^{n-1} v p_{v-1,n}$$

$$= [1-F_n(u)]E[k_i|f_i=u,n \text{ tasks}] + F_n(u)\{1+E[k_i|f_i=u,n \text{ tasks}]\}$$

$$= F_n(u) + E[k_i|f_i=u,n \text{ tasks}]$$

$$= \sum_{\substack{j=0 \\ j\neq i}}^{n} F_j(u)$$

[Q.E.D]

B.2 Proof of Theorem 2.3.2: Again, as in theorem 2.3.1 we assume that the tasks $T_1,\ldots,T_{n-1}$ have been so numbered that $\frac{t_k}{w_k}$ form an increasing sequence, i.e.,

$$\frac{t_1}{w_1} < \frac{t_2}{w_2} < \ldots < \frac{t_{n-1}}{w_{n-1}}$$

Let the predicted value t_p be such that the optimal position of task T_o according to SPT is after task i, i.e.,

$$\frac{t_i}{w_i} < \frac{t_p}{w_o} < \frac{t_{i+1}}{w_{i+1}}$$

whereas the real value t_o is such that the optimal position of the task T_o would be j, i.e.,

$$\frac{t_j}{w_j} < \frac{t_o}{w_o} < \frac{t_{j+1}}{w_{j+1}}$$

Hence MWFT with T_o after task T_i is given by:

$$MWFT_p = \frac{1}{n}\left(w_o f_p + \sum_{k=1}^{n-1} w_k f_k \right)$$

where f_k = finishing time of task k with T_o after T_i

$$= \begin{cases} \sum_{m=1}^{k} t_m, & 1 \le k \le i \\ t_o + \sum_{m=1}^{k} t_m, & i+1 \le k \le n-1 \end{cases}$$

and $f_p = t_o + \sum_{m=1}^{i} t_m$

Notice that we use t_o (and not t_p) in the above expression for f_k. This is because the prediction error results only in misplacement of the task. When executed it still takes only t_o.

Similarly, MWFT with T_o after task T_j is given by:

$$MWFT_o = \frac{1}{n}\left(w_o f_o + \sum_{k=1}^{n-1} w_k f'_k \right)$$

where f'_k = finishing time of task T_k with T_o after T_j.

$$= \begin{cases} \sum_{m=1}^{k} t_m, & 1 \le k \le j \\ t_o + \sum_{m=1}^{k} t_m, & j+1 \le k \le n-1 \end{cases}$$

and $f_o = t_o + \sum_{m=1}^{i} t_m$

The increase in MWFT due to prediction error is given by :

$$c = MWFT_p - MWFT_o$$

$$= \frac{1}{n}[w_o(f_p - f_o) + \sum_{k=1}^{n-1} w_k(f_k - f'_k)]$$

Now there are two possible cases: i>j or j>i.

Case I : i>j The predicted position is higher than the real position.

In this case,

$$f_k - f'_k = \begin{cases} 0 & 1 \le k \le j \\ -t_o & j+1 \le k \le i \\ 0 & i+1 \le k \le n-1 \end{cases}$$

and $f_p - f_o = \sum_{k=j+1}^{i} t_k$

Therefore,

$$c_I = \frac{1}{n}[w_o \sum_{k=j+1}^{i} t_k + 0 + \sum_{k=j+1}^{i} -t_o w_k + 0]$$

$$= \frac{1}{n}[w_o \sum_{k=j+1}^{i} t_k + \sum_{k=j+1}^{i} -t_o w_k]$$

$$= \frac{1}{n} \sum_{k=j+1}^{i} (w_o t_k - t_o w_k)$$

$$= \frac{1}{n} \sum_{k \in I} |w_o t_k - t_o w_k|$$

where I=[j+1,i]

Case II : j>i The predicted position is less

In this case,

$$f_k - f'_k = \begin{cases} 0 & 1 \le k \le i \\ t_o & i+1 \le k \le j \\ 0 & j+1 \le k \le n-1 \end{cases}$$

and $f_p - f_o = - \sum_{k=i+1}^{j} t_k$

Therefore,

$$c_{II} = \frac{1}{n}\left[-w_o \sum_{k=i+1}^{j} t_k + t_o \sum_{k=i+1}^{j} w_k \right]$$

$$= \frac{1}{n} \sum_{k=i+1}^{j} (-w_o t_k + t_o w_k)$$

$$= \frac{1}{n} \sum_{k \in I} |w_o t_k - t_o w_k|$$

where I=[i+1,j]

The two cases can now be combined together by redefining I as follows:

$$c = \frac{1}{n} \sum_{k \in I} |w_o t_k - t_o w_k|$$

where $I = \{k : \frac{t_k}{w_k} \in J\}$

$$\text{and } J = \begin{cases} \left(\frac{t_o}{w_o}, \frac{t_p}{w_o}\right) & \text{if } t_p > t_o \\ \left(\frac{t_p}{w_o}, \frac{t_o}{w_o}\right) & \text{if } t_o > t_p \end{cases}$$

[Q.E.D.]

B.3 Proof of Corollary 2.3.2.1: The corollary follows straightforwardly from theorem 2.3.2 by substituting $w_0=w_p=1$. (It can also be obtained from theorem 2.3.1 by substituting impulse functions for probability density functions of f_i).

[Q.E.D]

B.4 Proof of Corollary 2.3.2.2 : Substituting $w_o=w_k=1$ in the expression for c_I in the above proof we get

$$c_I = \frac{1}{n}\sum_{k=j+1}^{i}(t_k-t_o)$$

$$= \frac{1}{n}\left[\sum_{k=j+1}^{i} t_k - t_o(i-j)\right]$$

$$= \frac{1}{n}\left[\sum_{k=j+1}^{i} kT - t_o(i-j)\right]$$

$$= \frac{1}{n}\left[T\left(\frac{i(i+1)}{2} - \frac{j(j+1)}{2}\right) - t_o(i-j)\right]$$

$$= \frac{1}{2nT}[(i^2+i-j^2-j)T^2 - 2t_o(i-j)T]$$

Appendix B
Proofs of Scheduling Theorems

Since $iT < t_p < (i+1)T$, $t_p \equiv (i+\frac{1}{2})T$

Similarly, $t_o \equiv (j+\frac{1}{2})T$

Therefore,

$$(t_p-t_o)^2 \equiv [(i-j)T]^2$$

$$= (i^2+j^2-2ij)T^2$$

$$= (i^2+i-j^2-j+2j^2-i+j-2ij)T^2$$

$$= (i^2+i-j^2-j)T^2 - 2(j+\frac{1}{2})T(i-j)T$$

$$= 2nTc_I$$

i.e.,

$$c_I \equiv \frac{1}{2nT}(t_p-t_o)^2$$

Similarly for c_{II}.

[Q.E.D.]

APPENDIX C

PROOFS OF THEOREMS ON K-ARY PROCESSES

Proof of Estimation Theorem : The proof is essentially similar to the binary case except that now we have to maximize the likelihood function under k^n constraints:

$$\sum_{u=0}^{k-1} h_{iu} = 1 \qquad i=0,1,\ldots,k^{n-1}$$

Let $p_{tu} = P[z(t)=u|Z_{tn},H] = \sum_{i=0}^{k^n-1} h_{iu} q_i(Z_{tn}) \qquad u=0,1,\ldots,k-1$

The above k equations can be combined into one as follows :

$$P[z(t)|Z_{tn},H] = P[z(t)=0|Z_{tn},H]^{z^0(t)}\ P[z(t)=1|Z_{tn},H]^{z^1(t)} \ldots$$

$$\ldots\ P[z(t)=k-1|Z_{tn},H]^{z^{k-1}(t)}$$

$$= \prod_{u=0}^{k-1} p_{tu}^{\,z^u(t)}$$

where $z^u(t)$ is the u^{th} Lagrangean function of $z(t)$.

The likelihood function for N observations is given by

$$L(H) = P[z(N),z(N-1),\ldots,z(1)|z(-n+1),\ldots,z(0),H]$$

$$= P[z(N)|z(N-1),\ldots,z(1),Z_{1n},H]P[z(N-1)|z(N-2),\ldots,z(1),Z_{1n},H]\ldots$$

$$\ldots P[z(1)|Z_{1n},H]$$

$$= \prod_{t=1}^{N} P[z(t)|Z_{tn},H]$$

$$= \prod_{t=1}^{N} \prod_{u=0}^{k-1} p_{tu}^{z^u(t)}$$

Again as in the case of binary processes, we assume that the initial conditions $z(-n+1),\ldots,z(0)$ are given (or are assumed equal to zero), and only the parameters are to be estimated. The log likelihood function is

$$l(H) = \log\{L(H)\}$$

$$= \sum_{t=1}^{N} \sum_{u=0}^{k-1} z^u(t) \log p_{tu}$$

Now, using the Function Lemma we have,

$$\log p_{tu} = \sum_{i=0}^{k^n-1} \log(h_{iu})\, q_i(Z_{tn})$$

Hence,

$$l(H) = \sum_{t=1}^{N} \sum_{u=0}^{k-1} z^u(t) \sum_{i=0}^{k^n-1} \log(h_{iu})\, q_i(Z_{tn})$$

$$= \sum_{i=0}^{k^n-1} \sum_{u=0}^{k-1} \log(h_{iu}) \sum_{t=1}^{N} z^u(t) q_i(Z_{tn})$$

$$= \sum_{i=0}^{k^n-1} \sum_{u=0}^{k-1} m_{iu} \log(h_{iu})$$

where, $m_{iu} = \sum_{t=1}^{N} z^u(t) q_i(Z_{tn})$

$= \#$ of times $Z_{tn}=i$ is followed by $z(t)=u$

The last equality is a result of the fact that the quantity $z^u(t)q_i(Z_{tn})$

is 1 if and only if $z(t)=u$, and $Z_{t_n}=i$. The maximum likelihood estimates of the parameters are obtained by maximizing $l(H)$ under the constraints

$$\sum_{u=0}^{k-1} h_{iu} = 1 \qquad i=0,1,\ldots,k^n-1$$

We use the method of Lagrange multipliers for constrained maximization. The modified objective function is

$$l'(H) = \sum_{i=0}^{k^n-1} \sum_{u=0}^{k-1} m_{iu}\log(h_{iu}) + \sum_{i=0}^{k^n-1} w_i\{1 - \sum_{u=0}^{k-1} h_{iu}\}$$

$$= \sum_{i=0}^{k^n-1} w_i + \sum_{i=0}^{k^n-1} \sum_{u=0}^{k-1} m_{iu}\log(h_{iu}) - w_i h_{iu}$$

Here w_i are Lagarange multipliers. The necessary conditions for maximization are

$$\frac{dl'}{dh_{iu}} = \frac{m_{iu}}{h_{iu}} - w_i = 0, \quad i=0,1,\ldots,k^n-1, \quad u=0,1,\ldots,k-1$$

and $\sum_{u=0}^{k-1} h_{iu} = 1 \qquad i=0,1,\ldots,k^n-1$

The first equation above implies that $h_{iu} = \frac{m_{iu}}{w_i}$. The second equation implies that

$$\frac{1}{w_i} \sum_{u=0}^{k-1} m_{iu} = 1 \quad \text{or} \quad w_i = \sum_{u=0}^{k-1} m_{iu}$$

Therefore, the desired MLE of the parameters is

$$h_{iu} = \frac{m_{iu}}{\sum_{u=0}^{k-1} m_{iu}} \qquad i=0,1,\ldots,k^n-1, \quad u=0,1,\ldots,k-1$$

[Q.E.D.]

Appendix C

Proofs of k-ary Process Theorems

Proof of Prediction Theorem : Let the desired estimate be

$$\hat{z}(t) = g(Z_{tn})$$

where g is a Boolean function of Z_{tn}. Again, using the Lagrangean development of g we have,

$$\hat{z}(t) = \sum_{i=0}^{k^n-1} \hat{e}_i q_i(Z_{tn})$$

where $\hat{e}_i$ is a k-ary variable given by $\hat{e}_i = g(Z_{tn} | Z_{tn}=i)$.

Applying the Function Lemma to the model and the predictor equation we have

$$z^u(t) = \sum_{i=0}^{k^n-1} e_i^u(t) q_i(Z_{tn})$$

$$\hat{z}^v(t) = \sum_{i=0}^{k^n-1} \hat{e}_i^v q_i(Z_{tn})$$

and, therefore,

$$C(z(t),\hat{z}(t)) = \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} \hat{z}^v(t) z^u(t)$$

$$= \sum_{i=0}^{k^n-1} \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} \hat{e}_i^v e_i^u(t) q_i(Z_{tn})$$

$$\text{or } E[C(z(t),\hat{z}(t))] = \sum_{i=0}^{k^n-1} \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} \hat{e}_i^v h_{iu} E[q_i(Z_{tn})]$$

$$= \sum_{i=0}^{k^n-1} E[q_i(Z_{tn})] \sum_{v=0}^{k-1} \hat{e}_i^v \sum_{u=0}^{k-1} c_{uv} h_{iu}$$

The cost function is linear in $\hat{e}_i^v$. Obviously, $\hat{e}_i$ should be so chosen

that the $\hat{e}_i^v$ that has the smallest coefficient is one (all other $\hat{e}_i^v$'s will then automatically become zero), i.e.,

$$\hat{e}_i = \arg \min_v \sum_{u=0}^{k-1} c_{uv} h_{iu}$$

Now, if h_{iu} is determined according to the Estimation Theorem, then h_{iu} is proportional to m_{iu}. Hence, the above formula for $\hat{e}_i$ is equivalent to that stated in the theorem.

[Q.E.D.]

Proof of Corollary 4.8.4.1 : In this case, $c_{uv} = 1$ iff $u \neq v$. Hence,

$$\sum_{u=0}^{k-1} c_{uv} m_{iu} = -m_{iv} + \sum_{u=0}^{k-1} m_{iu}$$

Hence, the v that maximizes m_{iv} also minimizes the left hand side of the above equation and hence the cost function.

[Q.E.D.]

Proof of Total Cost Theorem :

$$TC = \sum_{t=1}^{N} C(z(t), \hat{z}(t))$$

$$= \sum_{t=1}^{N} \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} z^u(t) \hat{z}^v(t)$$

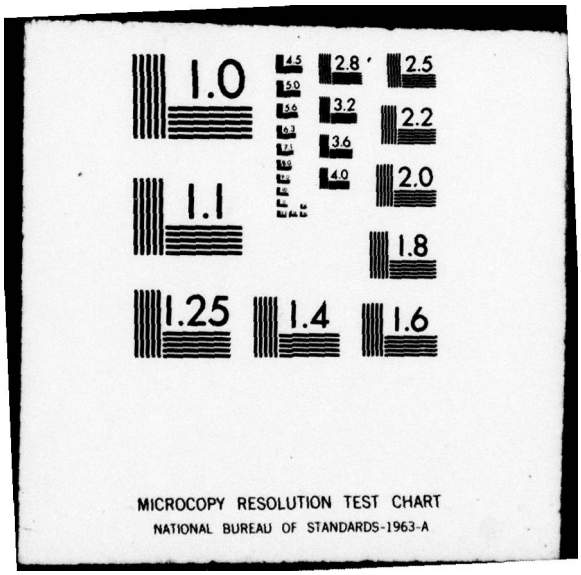
$$= \sum_{t=1}^{N} \sum_{i=0}^{n-1} \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} z^{u(t)} \hat{e}_i^v q_i(Z_{tn})$$

$$= \sum_{i=0}^{n-1} \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} \hat{e}_i^v m_{iu}$$

AD-A069 028 HARVARD UNIV CAMBRIDGE MASS AIKEN COMPUTATION LAB F/G 9/2
STUDIES ON THE FEASIBILITY OF SYSTEM SOFTWARE DEBUGGING.(U)
APR 79 R K JAIN, U GAGLIARDI F30602-77-C-0058
UNCLASSIFIED RADC-TR-79-20 NL

3 OF 3
AD
A069 028

END
DATE
FILMED
6-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

$$= \sum_{i=0}^{k^n-1} TC_i$$

where $TC_i = \sum_{v=0}^{k-1} \sum_{u=0}^{k-1} c_{uv} \hat{e}_i^v m_{iu}$

$$= \sum_{v=0}^{k-1} \hat{e}_i^v \sum_{u=0}^{k-1} c_{uv} m_{iu}$$

$$= \min_v \sum_{u=0}^{k-1} c_{uv} m_{iu}$$

The last equality is valid because $\hat{e}_i$ is chosen according to the Prediction Theorem.

Hence, $TC = \sum_{i=0}^{k^n-1} \min_v \sum_{u=0}^{k-1} c_{uv} m_{iu}$

[Q.E.D.]

Proof of Corollary 4.8.5.1 : This corollary follows straightforwardly from the total cost theorem by substituting $c_{uv}=1$ iff $u \neq v$.

[Q.E.D.]

REFERENCES

[Aka74] H. Akaike, "A new look at the statistical model identification," IEEE Trans. on Automatic Control, vol. AC-19, no. 6, Dec 1974, pp. 716-723.

[Arn75] C. R. Arnold, "A control theoretic approach to memory management," Proceedings Ninth Asilmor Conference on Circuits, Systems, and Computer, Pacific Grove, Calif., November 1975.

[Arn78] C. R. Arnold, "Optimization of computer operating systems," Ph. D. thesis, Harvard University, Cambridge, Mass., In preparation.

[ArG74] C. R. Arnold and U. O. Gagliardi, "A state-space formulation of the resource allocation problem in computer operating systems," in Proc. 8th Asilmar Conf. on Circuits, Systems, and Computers, Pacific Grove, Calif., December 1974, pp. 713-722.

[Ash72] R. Ashany, "Application of control theory techniques to performance analysis of computer systems," in Proc. 6th Asilmar Conf. on Circuits, Systems, and Computers, Pacific Grove, Calif., November 1972, pp. 90-101.

[Ast70] K. J. Astrom, Introduction to Stochastic Control Theory. New York: Academic Press, 1970.

References

[ADU71] A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement," JACM, vol. 18, no. 1, January 1971, pp. 80-93.

[Bar46] M. S. Bartlett, "On the theoretical specification of the sampling properties of autocorrelated time series," Journal of the Royal Statistical Society, B8:27 1946.

[Bel66] L. A. Belady, "A study of replacement algorithms for a virtual storage computer," IBM Systems Journal, vol. 5, no. 2, 1966, pp. 78-101.

[BlR76] P. R. Blevins anf C. V. Ramamoorthy, "Aspects of a dynamically adaptive operating system," IEEE Trans. Comput., vol. C-25, no. 7, July 1976, pp. 713-725.

[BoJ70] G. E. P. Box and G. M. Jenkins, Time Series Analysis Forcasting and Control. San Fransisco: Holden-Day Inc., 1970.

[BoP70] G. E. P. Box and D. A. Pierce, "Distribution of residual autocorrelation in autoregressive moving average time series models," Journal of the American Statistical Association 64, 1970.

[BrH69] A. E. Bryson, Jr. and Y. C. Ho, Applied Optimal Control. Waltham, Mass: Ginn and Co., 1969.

[Buz71] J. P. Buzen, "Queueing network models of multiprogramming," Ph. D. Thesis, Harvard University, Cambridge, Mass. 1971.

[BCB75] J. C. Browne, K. M. Chandy, et al, "Hierarchical techniques for the development of realistic models of complex computer systems," Proc. IEEE, vol. 63, no. 6, June 1975, pp. 966-975.

[BCM75] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open closed and mixed networks of queues with different classes of customers," JACM, vol. 22, no. 2, April 1975, pp. 248-260.

[Cle72] W. S. Cleveland, "The inverse autocorrelations of a time series and their applications," Technometrics vol. 14 no. 2 May 1972, pp. 277-293.

[Cof76] E. G. Coffman Jr., Computer and Job Shop Scheduling Theory. John Wiley & Sons, 1976.

[Cou75] P. J. Courtois, "Decomposability, instabilities, and saturation in multiprogramming systems," CACM, vol. 18, no. 7, July 1975, pp. 371-377.

[CoL66] D. R. Cox and P. A. W. Lewis, The Statistical Analysis of a Series of Events. London: Methuen, 1966.

[CHW75] K. M. Chandy, U. Herzog, and L. Woo, "Approximate analysis of general queueing networks," IBM J. Res. Develop., vol. 14, no. 1, January 1975, pp. 43-49.

References

[CMM67] R. W. Conway, W. L. Maxwell, and L. W. Miller, Theory of Scheduling. Reading, MA: Addision Wesley, 1967, pp. 27-30.

[Den68] P. J. Denning, "The working set model for program behavior," CACM, vol. 11, no. 5, May 1968, pp. 323-333.

[DKL76] P. J. Denning, K. C. Kahn, J. Leroudier, D. Potier, and R. Suri, "Optimal Multiprogramming," Acta Informatica, vol. 7, fasc. 2, 1976, pp. 197-216.

[GaS73] D. P. Gaver and G. S. Shedler, "Approximate models for processor utilization in multiprogrammed computer systems," SIAM J. Comput., vol. 2, no. 3, September 1973, pp. 183-192.

[Gel75] E. Gelenbe, "On approximate computer systems models," JACM, vol. 22, no. 2, April 1975, pp. 261-269.

[HaR68] P. L. Hammer (Ivanescu) and S. Rudeanu, Boolean Methods in Operations Research and Related Areas. New York: Springer-Verlag, 1968.

[HoP72] C. A. R. Hoare and R. H. Perrot, Operating Systems Techniques. Proc. of International Seminar held at Queen's University of Belfast 1971, Academic Press 1972.

[Kas77] R. L. Kashyap, "A bayesian comparison of different classes of dynamic models using empirical data," IEEE Trans. on Automatic Control, vol. AC-22, no. 5, October 1977, pp. 715-727.

[Kle69] L. Kleinrock, "Certain analytic results for time-shared processors," in Information Processing 68. A. J. H. Morrel, Ed., Amsterdam: North-Holland Publishing Co., 1969, pp. 838-845.

[Kle76] L. Kleinrock, Queueing systems. vol. 2: Computer Applications, ch. 4, New York: Wiley-Interscience, 1976.

[Lew74] A. Lew, "Optimal resource allocation and scheduling among parallel processes," in Parallel Processing. Tse-Yun Fung, Ed., Springer-Verlag, Berlin, 1974.

[Lew76] A. Lew, "Optimal control of demand-paging systems," Information Sciences, vol. 10, no. 4, 1976, pp. 319-330.

[LiC77] L. Lipsky and J. D. Church, "Applications of a queueing network model for a computer system," Computing Surveys, vol. 9, no. 3, September 1977, pp. 205-221.

[MaD74] S. E. Madnick and J. J. Donovan, Operating Systems. New York: McGraw-Hill Book Co., 1974.

[McK69] J. M. McKinney, "A survey of analytical time-sharing models," Computing Surveys, vol. 1, no. 2, June 1969, pp. 105-116.

[McW76] R. M. Mckeag and R. Wilson, Studies in Operating Systems. Academic Press 1976, Chapter 4.

[Moo71] C. G. Moore, III, "Network models for large-scale time-sharing systems," Ph. D. Thesis, Univ. of Michigan, Ann Arbor, 1971.

[Mun75] R. R. Muntz, "Analytic modeling of interactive systems," Proc. IEEE, vol. 63, no. 6, June 1975, pp. 946-953.

[MuJ75] R. Muralidharan and R. K. Jain, "MIN : An interactive educational program for function minimization," Technical Report no. 658, DEAP, Harvard University, December 1975.

[Nel73] C. R. Nelson, Applied Time Series Analysis for Managerial Forecasting. San-Fransisco: Holden-Day Inc., 1973.

[Pap65] A. Papoulis, Probability, Random Variables, and Stochastic Processes. New York: McGraw Hill Book Co., 1965.

[Par72] E. Parzen, Discussion no. 2, Technometrics vol. 14 no. 2, May 1972, pp. 295-298.

[PrF76] B. G. Prieve and R. S. Fabry, "VMIN - An optimal variable space page replacement algorithm," CACM, vol. 19, no. 5, May 1976, pp. 295-297.

[Que49] M. H. Quenouille, "Approximate tests of correlation in time series," Journal of the Royal Statistical Society, B11:68, 1949.

[Rud74] S. Rudeanu, Boolean Functions and Equations. Amsterdam: North-Holland Publishing Company, 1974.

[Sch67] A. Scherr, An Analysis of Time-Shared Computer Systems. Research monograph 36, Cambridge, Mass.: M.I.T. Press, 1967.

[Shu76] A. W. C. Shum, "Queueing models for computer systems with general service time distribution," Ph. D. Thesis, Harvard University, Cambridge, Mass. Dec 1976.

[Smi56] W. E. Smith, "Various optimizations for single-stage production," Naval Res. Logist. Quart., 3(1956) pp. 59-66.

[Smi78] D. R. Smith, "A new proof of the optimality of the shortest remaining processing time discipline," Operations Research, vol. 26, no. 1, Jan-Feb 1978, pp. 197-199.

[Wil71] M. V. Wilkes, "Automatic load adjustment in time-sharing systems," in Proc. ACM-SIGOPS workshop on System Performance Evaluation, Harvard University, Cambridge, Mass. April 1971, pp. 308-320.

[Wil73] M. V. Wilkes, "The dynamics of paging," Computer Journal, vol. 16, no. 1, February 1973, pp. 4-9.

[Wul69] W. A. Wulf, "Performance monitors for multiprogramming systems," in Proc. 2nd Symp. on Operating Systems Principles, Princeton Univ, October 1969, pp. 175-181.

CHAPTER VI

SECTION II

System Software Debugging

Abstract

This report discusses the use of satellite computers in system software debugging. In this approach a satellite computer keeps track of the system states and provides useful information if the observed system crashes due to a software bug. As a byproduct of this approach, the data gathered can also be used to measure and monitor the system performance. This approach has therefore been named "SODEM" (Satellite Observer DEbugger and Monitor) approach.

The design considerations and structural modules of SODEMs have been discussed in detail. The ideas have been illustrated with examples from an actual implementation of a SODEM to observe a DECsystem-10 using a PDP-11 processor.

I. Introduction

The high cost of software development and maintenance has raised a demand for new tools for software debugging. In the case of application software, this demand has resulted in at least two kinds of new developments. Firstly, in the development of many interpretive languages e.g., BASIC and EL1 which can be easily debugged, and secondly in the development of special debugging software for non-interpretive languages e.g., FORDDT for FORTRAN.

In the case of system software such as an operating system the problem of debugging is complicated by the necessity for it to be done on-line, i.e. the system must be debugged while several other users are using it. Of course, this does not apply during the initial development phase at the vendor's plant where it is easy to have a machine for each software development. The scenery that we have in mind is a more usual one: the one at a customer's site where he has only one copy of the hardware and a relatively bug free system software which he has to frequently modify to satisfy the changing user demands. The users are allowed to run on the modified software. Normally the system runs fine, except that once in a while a bug hidden somewhere brings the system down. To locate this bug what we need is an independent observer which keeps track of

the system states during its use and provides helpful information for post crash analysis.

One approach to provide this independent observer is to use a virtual machine monitor to provide several copies of the hardware (GaG72). The system software is run on one copy and the observer on another so that the latter remains alive when the former crashes. This idea, though theoretically sound, may not work so well in practice. This is because hardware malfunctions, which are more often then not the causes of system failure, will bring all the virtual machines down simultaneously. Strictly speaking this should not be taken as the demerit of this approach because the proposed observer is not supposed to detect the hardware malfunctioning. However, this approach does make it difficult to pinpoint the cause of failure.

Another approach to providing the independent observer could be to put it on a separate small satellite computer connected to the main processor. The observer can then run independntly of the system software as well as hardware, and regardless of hardware or software nature of failure the observer provides the expected service.

The idea of satellite computers is nothing new. In the past they have been extensively used as I/O interfaces better known as peripheral processors. For example, CDC-6600 uses 10 peripheral

processors around its main processor (McW76). Satellite Processors have also been used as measurement tools to monitor the performance of computer systems. An example is a HEMI monitor implemented on one of the peripheral processor of the CYBER system (Svo76).

The scheme under consideration here is to use a satellite processor to keep track of the states of the system software on the main processor. The primary purpose of these observations is to help debug the system software. Under normal running periods the record of system states can also be sued to measure and monitor the system performance. Therefore, we suggest the use of the acronym SODEM (Satellite Observer DEbugger and Monitor) for this new role of satellite computers. Although the term SODEM refers to the union of satellite computer hardware, its software and its interface with the main processor, we will frequently use it to denote the software alone.

The SODEM computer does not necessarily have to be a computer smaller than the main computer, not is it necessary for it to be dedicated for observing purpose. In fact, it can be any other computer connected to the observed system by a hardware link. The existence of such links is beocming very common these days. In the light of recent rapid growth of computer

networks, and tremendous success of ARPANET*, the importance of SODEM approach is obvious.

At Harvard University, we have developed an experimental SODEM. Our Aiken Computation Laboratory has a TOPS-10 monitor on a DECsystem-10 connected to ARPANET. Also, it has hardware links connecting it to a PDP-1, a PDP-11/10, a PDP-11/40, a PDP-11/70, and a GT-10 computer. We chose the PDP-11/10 as the satellite processor to implement the experimental SODEM called SODEM-1011 (SODEM-ten-eleven) and to study various aspects including the feasibility of this appraoch. The details of this implementation and the knowledge that we have gained from this are the topic of this report.

---

* In the ARPANET the IMPs keep observing the neighboring IMP's. When an IMP comes down, the neighbors not only inform other IMPs in the network but also reload the IMP if possible. In this sense IMPs provide observation and monitoring function of SODEMs.

II. Design Considerations

The three main components involved in a SODEM are the system to be observed, the link, and the satellite processor. In the following, we discuss how the choice of each of these components affects the design and capability of a SODEM. The ideas have been illustrated with our own experience from the design of SODEM-1011.

2.1 Data Selection and Data Rate

Data selection involves determining exactly what information should be observed i.e., what constitutes the system state and how it can be extracted and recorded. Stirctly speaking, the system state is the state of all system's memories (main memory, secondary memory, etc.). Continuous collection of all this information is neither feasible nor desirable. Therefore, one has to identify a small subset of important memory location (such as control registers, operating system tables, etc.) for observation and recording. Also there is the problem of data availability. Some information, although important, may not be obtained.

Closely coupled with the question of data selection is the question of data rate i.e., how often should the information be collected. This rate could be different for different subsets of collected data. For example, CPU status bits may have to be observed every instruction cycle whereas, teletype status could be observed every few milliseconds or so.

In case of SODEM-1011 we have chosen to observe key tables that the TOPS-10 monitor keeps permanently in the core. This does not include many of the disk management tables which are kept on the disk or the job data areas which are swapped in and out with the job. This choice is quite arbitrary and is based on the belief that in the case of a crash these other tables could be very reliably obtained form the secondary storage. Depending upon its variability, each monitor table has been assigned a cycle time at which it is to be recorded.

2.2 Communication Link

Ideally the link connecting the SP (Satellite Processor) to MP (Main Processor) should be such that the SP can access the complete main memory of MP. The second best choice is to let the MP have privilege to read/write SP memory. In the latter case, SP sends an interrupt signal to MP whenever it wants some portion of its memory to be read.

The hardware link connecting our PDP-10 and PDP-11, called X-BUS, was designed much before the SODEM project began. It provides a master-slave relationship between the PDP-10 and the PDP-11. To be more precise, the PDP-10 can use the X-BUS to read/write the PDP-11 memory, to start/stop/interrupt the PDP-11 or to sense states of PDP-11. The PDP-11 can not use it in the same way to control the PDP-10. The usual way for the PDP-11 to communicate with PDP-10 is to write it in a

buffer and wait till PDP-10 reads it. This is a rather undesirable situation because it not only makes communication slow but also the observer has to depend upon the observed to get the data.

2.3 Satellite Processor

Generally the satellite processor is much smaller and slower than the main processor (for obvious economical reasons). However, it is desireable that the two processor have the same word or byte size. This saves a lot of overhead in data transformation and manipulation.

In the design of SODEM-1011 we were faced with completely uncompatible word sizes - 36 bits in PDP-10 and 16 bits in PDP-11. It means that 4.5 bytes (8 bits each) are required in PDP-11 to store one PDP-10 word. Thus, we had to write a complete set of data manipulation routines to add, subtract, multiply, divide these odd size numbers. We did save some time by sacrificing some PDP-11 memory and using 5 bytes for each word instead of 4.5 bytes.

III. Structure of a SODEM Software

The basic functions of a SODEM software are the following:

1. To collect data at MP.
2. To transfer the data to SP.
3. To transform it in to a form suitable for SP.
4. To display it to the human user as SP debugging the software.
5. To record the data in a permanent storage for future use.

Thus, it is obvious that a SODEM software should have at least 5 modules, one for each of the above 5 functions. In addition it should have a module where it keeps information on when and what is to be observed. The interaction between these various modules is shown in Figure 3.1. The details of these modules with examples from SODEM-1011 are now explained.

3.1 Data Definition Module

As said above, this module contains information on what is to be observed, and when it is to be observed. In case of SODEM-1011, this consists of a list of monitor tables to be observed. Each table has a name assigned to it, and consists of several words called entries. Each entry has several bytes - collection of variable number of bits.

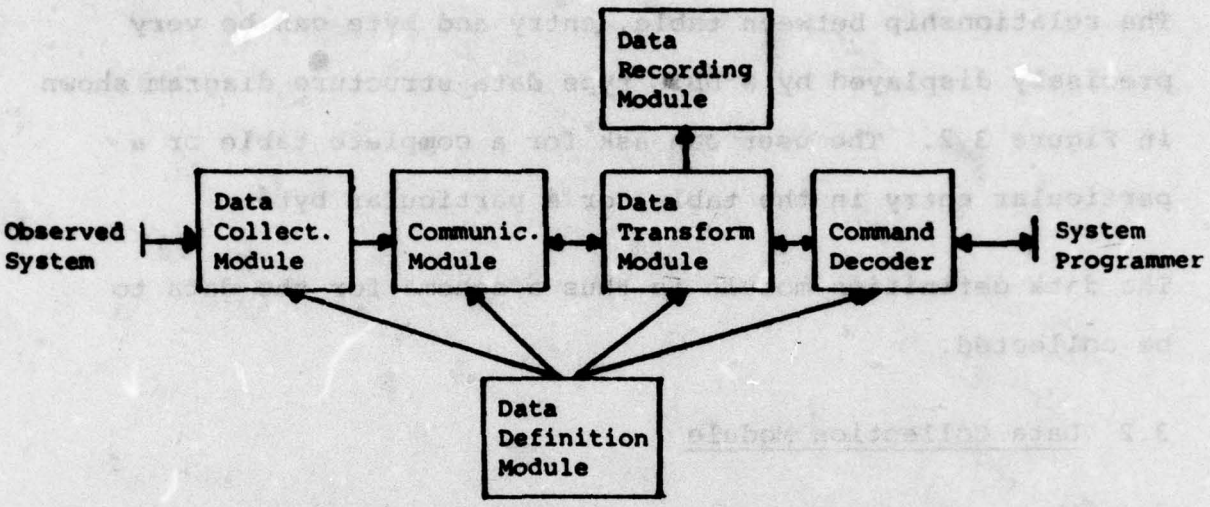


Figure 3.1: Structure of a SODEM Software

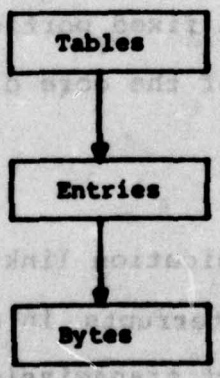


Figure 3.2: Data Structure Diagram of SODEM-1011 Data Base

The relationship between table, entry and byte can be very precisely displayed by a DBTG type data structure diagram shown in Figure 3.2. The user can ask for a complete table or a particular entry in the table, or a particular byte. The data definition module is thus a schema for the data to be collected.

3.2 Data Collection Module

The function of this module is to collect the data from MP memory. If the communication link between MP and SP is such that SP can directly read any location in MP memory then this module would not be required.

In SODEM-1011, the data collection module resides on PDP-10, it reads in the data from PDP-10 memory. Currently, it runs as a user program, therefore it has to use monitor service calls known as GETTAB UUO to get monitor tables. However, eventually it will reside in a fixed portion of core and have a privilege to read all of the core directly.

3.3 Communication Module

This module handles the communication link between MP and SP. This means that it handles interrupts in either direction, and checks the data for correct transmission. It makes use of the system clock to do periodic observations.

In SODEM-1011 this module resides partly in PDP-10 and partly in PDP-11. This is because of the peculiarity of our X-BUX which does not allow PDP-11 to read/write PDP-10 memory or interrupt it. Therefore, the PDP-11 portion of the communication module writes the data request messages in fixed locations to be read and serviced later by PDP-10 portion.

3.4 Data Transformation and Manipulation Module

This module transforms data from the format used on MP to that on SP. It would not be required if both the processors had identical word sizes. Unfortunately, in SODEM-1011 the word sizes are not only different they are highly incompatible. As said before, we use 5 bytes in PDP-11 to store one PDP-10 word. Thus the data transformation module consists of routines to interpret these 5-byte words in Ascii, Sixbit, Octal, Floating point formats. Also it contains routines for arithmetic operations like addition, subtraction, multiplication, and division on these odd size numbers.

3.5 Command Decoder

This module provides the debugging facility to the human programmer. It is recommended that its command structure be similar to that of other debugging programs used at the installation. For example, in SODEM-1011 the commands are very similar to DDT used for debugging assembly language programs.

In SODEM-1011 this module resides partly in PDP-10 and partly in PDP-11. This is because of the peculiarity of our X-BUX which does not allow PDP-11 to read/write PDP-10 memory or interrupt it. Therefore, the PDP-11 portion of the communication module writes the data request messages in fixed locations to be read and serviced later by PDP-10 portion.

3.4 Data Transformation and Manipulation Module

This module transforms data from the format used on MP to that on SP. It would not be required if both the processors had identical word sizes. Unfortunately, in SODEM-1011 the word sizes are not only different they are highly incompatible. As said before, we use 5 bytes in PDP-11 to store one PDP-10 word. Thus the data transformation module consists of routines to interpret these 5-byte words in Ascii, Sixbit, Octal, Floating point formats. Also it contains routines for arithmetic operations like addition, subtraction, multiplication, and division on these odd size numbers.

3.5 Command Decoder

This module provides the debugging facility to the human programmer. It is recommended that its command structure be similar to that of other debugging programs used at the installation. For example, in SODEM-1011 the commands are very similar to DDT used for debugging assembly language programs.

In SODEM-1011 this module resides partly in PDP-10 and partly in PDP-11. This is because of the peculiarity of our X-BUX which does not allow PDP-11 to read/write PDP-10 memory or interrupt it. Therefore, the PDP-11 portion of the communication module writes the data request messages in fixed locations to be read and serviced later by PDP-10 portion.

3.4 Data Transformation and Manipulation Module

This module transforms data from the format used on MP to that on SP. It would not be required if both the processors had identical word sizes. Unfortunately, in SODEM-1011 the word sizes are not only different they are highly incompatible. As said before, we use 5 bytes in PDP-11 to store one PDP-10 word. Thus the data transformation module consists of routines to interpret these 5-byte words in Ascii, Sixbit, Octal, Floating point formats. Also it contains routines for arithmetic operations like addition, subtraction, multiplication, and division on these odd size numbers.

3.5 Command Decoder

This module provides the debugging facility to the human programmer. It is recommended that its command structure be similar to that of other debugging programs used at the installation. For example, in SODEM-1011 the commands are very similar to DDT used for debugging assembly language programs.

Thus "JBTSTS/" types the table JBTSTS, "JBTSWP_2/ types the second entry in the table JBTSWP. A linefeed types the next entry and an up-arrow "^" types the previous entry. Also there are commands to display the last typed data in octal, sixbit, decimal, half-word formats. A short description of SODEM-1011 commands is given in the Appendix-A.

3.6 Data Recording Module

This module writes the data on to a secondary storage device for later use. In SODEM-1011, unfortunately, there is no secondary storage device on the satellite computer PDP-11. Hence no recording is currently done. However, there exists a virtual machine monitor designed previously for this PDP-11 and PDP-10 combination. The VMM allows virtual I/O facility to PDP-11 users in the sense that with its help they can read or write on PDP-10 devices. Eventually, we plan to use this facility to record the SODEM-1011 data on PDP-10 disk.

IV. Conclusion

A satellite computer can be a very useful tool for system software debugging. It also helps measure the system performance as a byproduct. The chief consideration in the design of a SODEM (Satellite Observer DEbugger and Monitor) are deciding what data should be observed, the design of the connecting link and the choice of appropriate satellite processor. The main structural modules of SODEM software are data collection module, communication module, data transformation module, data display maodule, data recording module, and data definition module.

A SODEM has been implemented successfully at Harvard University to observe a PDP-10 using a PDP-11 as a satellite processor. The different word sizes of the two computers have increased the size of the software. Also the uni-directional nature of the X-BUS connecting the two computers has made the SODEM a little slow and dependent on the observed software. Nonetheless, it has demonstrated the feasibility of this approach.

The main advantage of SODEM approach is that the observed system remains intact even when there are hardware or software failures of the observed system. However, like most other system software, it is generally not portable. It can be used only with a specific type of main processor, operating system, and satellite processor combination.

Appendix A - Using SODEM-1011

This appendix describes the working of the experimental SODEM developed at Harvard to observe a DECsystem-10 using a PDP-11 satellite processor. As explained before, the state of the TOPS-10 monitor system is given largely by the contents of its various monitor tablees. SODEM-1011 lets a user sitting at PDP-11 teletype to look at almost all the monitor tables during normal operation or after a crash. The user has freedom to observe a particular table, any one entry in a table, or any bit or set of bits in an entry. Of course, the user must be familiar with the internal structure and working of the TOPS-10 monitor to be able to find the bugs or to make any sense out of these observations.

The command structure of SODEM-1011 has been deliberately kept very similar to that of DDT used here for debugging other programs. For example, a entry name followed by "/" displays that entry, a linefeed displays the enxt entry, and so on. The key element in learning to use SODEM-1011 is to know how to specify the name of the data to be observed.

A.1 Name Specification

Each table in the TOPS-10 monitor has a name associated with it. For example, the table containing the status of various jobs is called 'JBTSTS'.

These tables, in general, are several words long. The term ENTRY denotes a word or a group of consecutive words having similar structure. For example, the first five words of configuration table give the name of the system. This entry is named '.CNFGO'. The names of various entries are described in the DECsystem-10 documentation (DEC74). The entries that do not have any name have to be specified by their relative location in the table.

The different bits of set of bits in each word of the table have different significance. For example, the 3rd bit of job status word indicates whether that job number has been assigned to a job, similarly bits 10 thru 14 of this word specify the wait status code of the job. The term BYTE is used to denote such bits or set of bits. Many bytes have names assigned, and many do not. For example, the 3rd bit of job status word is called 'JNA'. Again, these names can be found either from monitor listings of from system documentations. Also, the help command (see below under commands) of SODEM-1011 can be used to get names of tables, entries, or bytes.

A name specification consists of a table name followed by an entry name or number, followed by byte name or number. The character "_" (back arrow, or underline) is used as a separator between table and entry names, and between entry and byte names. A few examples of name specifications are shown in Table A.1.

Table A.1: Examples of Name Specifications

S. No.	Specification	Meaning
1.	CNFTBL/	Display the whole table.
2.	CNFTBL_.CNDBG/	Display the debuggin status word entry only.
3.	CNFTBL_4_3/	Display the 3rd bit of the 4th word in the table.
4.	.CNDBG/	Same as in 2.
5.	RUN/	Display RUN bits of all entries in the job status table.
6.	JBTSTS__3/	Display 3rd bit of all entries in the table.
7.	CNFTBL/	Same as 1. use the same format for all entries.
8.	CNFTBL_/	Display each entry in its associated format.
9.	CNFTBL__/	Display all bytes of all entries in their (byte's) format.

Since the names are unique, one can omit table (entry) name if entry (byte) name is specified. For example, see specifications 4 and 5 in the Table A.1. The entry or byte names can also be nulls, in which case all the entries or bytes satisfying the specification are displayed (see example 6 in the Table A.1).

A.2 SODEM Commands

The information in PDP-10 can be in several different formats, e.g., ASCII, SIXBIT, octal, decimal, etc. Therefore, the data transformation module of SODEM-1011 lets the user view the information in any formt he deisres. The format is specified by the command character following the name specification. For example, the "=" command types the data in octal format.

The data definition module, which keeps a dictionary of all data names, also has a format for each piece of data. Thus each table, entry, or byte has a particular type out format assigned to it. For example, the entry".CNOPR" (name of the OPR TTY) is to be typed in SIXBIT, whereas, ".CNNSM" (number of nano seconds per memory cycle) is typed in decimal. The"/" (Slash) command displays the data in the format specified in the data definition module. The type out routines chosen are those associated with the lowest level specified (even though that level specification may be null) in the name specification. For example, see specifications 7 thru 9 in the Table A.1.

Table A.2: List of SODEM-1011 Commands

S. No.	Command	Function
1.	/	Normal type out.
2.	#	Non-zero entries only.
3.	&	SIXBIT format type out.
4.	'	ASCII Format.
5.	$	Decimal format.
6.	=	Octal format.
7.	%	Half word format.
8.	!	Binary format.
9.	<LF>	Type the next entry in the table.
10.	^	Type the entry before the current one.
11.	<CR>	Do nothing. Just prompt again.
12.	?	Help

Very often one wants to skip over the zero data values. For example, if only 3 jobs are currently logged in, there is not point in typing the whole job status table. The "#" (number) command of SODEM-1011 facilitates this. It is identical to the slash command except that only non-zero values are displayed.

There are many more commands accepted by SODEM-1011. Further, the modular structure of the SODEM allows for easy extensions, of the command table, if needed. The current list of commands is given in Table A.2. One command which is very helpful is the "?" (help) command. It helps the user by listing the various options that he has at a particular level. Thus, if a question mark is typed after specifying say a table and an entry name, SODEM-1011 will display the byte names associated with that entry. The list of available commands, entry names, and table names can be similarly obtained.

A.3 Sample Protocols

The effectiveness of SODEM-1011 in helping the user to debug the system depends upon his familiarity with the internal structure of the system. With enough familiarity and imagination, one can easily extract some very useful information about the system (and the bugs). We illustrate it with a sample protocol. This is just a very small subset of the possible questions one might want to ask after a system crash. The user type-in is underlined and our comments follow semicolons.

Question: List all the users that were logged in.

Protocol:

```
>.PDNM1#                ;TYPE NON-ZERO NAMES (FIRST
                        ;HALF).

.PDNM1 1/OPERA
.PDNM1 3/JAIN
.PDNM1 4/OPERA
```

Answer: Operator was logged in as job #1 and 4, and Jain as #3.

Question: Which jobs were privileged to POKE the monitor?

Protocol:

```
>JP.POK#                ;TYPE NON-ZERO POKE PRIV BITS.
JBTPRV_1_JP.POK/1
JBTPRV_4_JP.POK/1
```

Answer: Jobs 1 and 4 had the privilege.

Question: Which job poked last?

Protocol:

```
>.CNPUC/                ;TYPE THE JOB # AND # OF POKES
CNFTBL_.CNPUC/  4,16

>^                      ;TYPE THE ENTRY BEFORE .CNPUC
CNFTBL_.CNPOK/ 0,2300
```

Answer: Job #4 poked location 2300. Total number of pokes was 16 (octal).

Question: Who, if any, meddled with his sharable segment?

Protocol:

```
MEDDLE#                    ;TYPE NON-ZERO MEDDLE BITS
JBTSGN_3_MEDDLE/1
```

Answer: Job #3 meddled with his high segment.

A Sample Protocol with SODEM-1011

REFERENCES

(GaG72) Gagliardi, U.O., and Goldberg, R.P., "Virtualizable Architectures", Proceeding of 1972 ACM AICA International Comp. Symp., Venice, Italy, April 1972, pp. 527-538.

(McW76) Mckeag, R.M., and Wilson, R., "Studies in Operating Systems", Academic Press 1976.

(Svo76) Svobodova, L., "Computer Performance Measurement and Evaluation Methods: Analysis and Applications", Elsevier 1976.

(DEC74) "DECsystem-10 Monitor Calls", Digitial Equipment Corporation, Pub. #DEC-10-OMCMA-A-D, May 1974.

☆U.S. GOVERNMENT PRINTING OFFICE: 1979-614-023/116