1996

# Dynamic Storage Allocation Using Simon's Model of Information Usage.

Rachelle Folse Cope
*Louisiana State University and Agricultural & Mechanical College*

# INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**UMI**

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

# DYNAMIC STORAGE ALLOCATION USING SIMON'S MODEL OF INFORMATION USAGE

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Interdepartmental Program in Business Administration

by
Rachelle Folse Cope
B.S., Nicholls State University, 1987
August 1996

## ACKNOWLEDGMENTS

My dissertation committee was composed of the following members: Dr. Ye-Sho Chen, Dr. Peter Kelle, Dr. Ishwar Murthy, Dr. Kwei Tang, and Dr. Dennis Webster. I thank all of these members for the guidance that they gave me during the doctoral program and in preparation of this dissertation. I owe a great deal to Dr. Chen, my dissertation chairman, for his patience, enthusiasm, and compassion.

My appreciation is extended to all of my friends and colleagues who have supported me through this process. I give special thanks to Dean Joseph Miller and Dr. Sam Cappel for their encouragement and belief in me when the road ahead seemed very long.

It is very hard to express the depth of the gratitude and respect that I have for my parents, Dr. and Mrs. Raymond Folse. They have guided and supported me through every minute of my education, and I know that I could not have made it through this process without them. I especially thank my father who has always been, in my eyes, the epitome of that which an educator should be.

To my husband, Bobby, I give my love and heartfelt appreciation for all that he has sacrificed in order to share this dream with me. This journey brought many gifts to me, but I am most grateful that it brought Bobby into my life.

## TABLE OF CONTENTS

**ABSTRACT**

In today's rapidly changing field of Management Information Systems (MIS) one problem faced by organizations is the consumption of storage capacity due to the growing base of software assets. Research has shown that very few firms effectively monitor program usage. Storage management issues arise due to the fact that many of the programs occupying valuable storage space are used infrequently. In this dissertation, we apply Simon's model of information usage in order to model the dynamic behavior of program usage. This methodology enables organizations to identify the changing usage frequencies of software assets. We propose a classification scheme which MIS personnel can use in order to effectively monitor their program usage tendencies. This classification scheme may then serve as a basis for storage allocation decision-making. Through a study of the dynamic behavior of programs, we have formulated a minimum cost model for hierarchical storage allocation. We will show the value of incorporating dynamic usage frequencies into algorithms which have traditionally considered only a static view.

# CHAPTER 1 INTRODUCTION

## 1.1 Problems of Storage Management

From its inception, Management Information Systems (MIS) technology has searched for solutions to the many problems which have ensued with the growth of the field. Literature is replete with ideas, methodologies and models established in order to simplify the tasks performed by information system department (ISD) personnel. The diversity of the number and kinds of ISD problems which exist are too numerous to totally identify. Consequently, ISD personnel have the major tasks of identifying the problems of greatest concern and employing methodologies to remedy them. One well-known author in the field of information technology, Edward Yourdon (1992), speaks of the deficiencies of programming languages, people, automated tools, structured techniques, and software reengineering, only to name a few.

This dissertation, in part, addresses one of the inherent problems that cannot be escaped by any MIS --- the problem of limited storage capacity with growing software and data arsenals. It is intended in this research to invoke a methodology which will permit organizations to better allocate their programs to secondary storage. Ultimately, it is desirable to show that the problem of storage

allocation is greatly affected by the dynamic nature of program usage.

### 1.1.1 Storage Limitations

It is no secret to users of computers, from the student with a floppy disk to the computer operator who backs up 10 gigabytes of data each night, that at some point storage of software and data assets becomes a problem. Large organizations have traditionally used ad hoc methodologies in order to free hard disk storage space. Unfortunately, the problem of limited storage capacity continues in its complexity due to the evolving environments from which we obtain information. At one time, storage was only a problem of mainframe systems. With the growing number of applications being performed on personal computers and local area networks (LANs), problems inherent to mainframe systems have expanded to all types of computer systems (Busse, 1994).

The high cost and low productivity associated with storage management is causing many organizations to clamor for products capable of freeing valuable hard disk space. These types of tools are still in their infancy, and for the moment the utopian notion of a totally open storage management platform is nowhere to be found (Korgeniowski, 1994).

Although computer storage capacity has changed dramatically over the past five decades, the evolution from magnetic tape to optical disk storage has not eliminated the need for increased storage. Viewing changing storage needs, many organizations have expressed the desire for hierarchical storage systems segregating high cost disk storage from lower cost magnetic tape, optical disk and mass storage devices. Thus, in addressing the problem of storing software assets in organizations, we propose that allocation decisions for high, midrange, and low cost storage options can be made in order to increase efficiency while minimizing cost (Kay, 1994).

### 1.1.2 The Changing Priorities of Software

Intuitively, one may not think that the issue of how to store programs and data is very complex in most organizations. The perception may be that organizations possess unlimited storage capacity. Another view is that additional storage can be purchased when necessary. Even if economic resources are not critical in storage expansion and acquisition, the manner in which the additional storage is managed is critical.

There are also the added complications introduced by network environments. Managing data storage and back-up in today's client/server environment is no small feat. The swelling ranks of workstations being added to networks plus

the growing use of sophisticated graphical applications are causing network capacity requirements to grow by leaps and bounds (Abele, 1994).

The premise of this research is that organizations have the need to identify those items which have critical usage. Naturally, those programs and data with greater usage should be more accessible and should not be stored in a manner that is costly and untimely in retrieval. For some organizations, the question of program usage may not seem complex. MIS personnel may have some idea of which programs are most heavily used. But, more often than not, large organizations store a tremendous number of programs on hard disk which are seldomly used or becoming obsolete. It is especially important to note that most organizations perform little to no regular assessment of which software is being used and which is not (Yourdon, 1992; Ferrill, 1994).

With these considerations, it is necessary for organizations to understand the changing priorities of their software. We concentrate here on software rather than data since program usage and data usage are evaluated in different fashions. In many cases, organizations can make an accurate assessment of the behavior of their data, but the growth of program applications becomes less predictable. Knowing that most organizations have the ability to monitor their software

usage, they should use this information for strategic advantage. Organizations need to realize that their programs of present vital importance, or even marginal importance, may become obsolete and unused in the future. Likewise, programs which are seldomly used today may eventually increase in usage and become of great importance in the future. This dissertation proposes a methodology for organizations to more easily assess the usage behavior of their programs and move away from methodologies which are based on a static view of usage. In other words, the goal is to be able to identify the programs which are most heavily used while tracking the dynamic usage rate to indicate tendencies for usage in the future.

### 1.1.3 Considerations in Storage Decisions

As mentioned earlier, there are very definite costs associated with the type of medium selected for storing programs and data. For example, suppose that one decides to store a program on magnetic tape. The purpose of making this choice would be perception of seldom usage. Often, files stored on magnetic tape are kept at a location off-site (Korgeniowski, 1994). This imposes a charge for delivery and pick-up of tapes being stored. If this file has the need to be accessed more frequently than it was intended, there is certainly an added cost to storing it on that particular

medium. Therefore, cost is a key consideration in making the decision of where to store program files (Merenbloom, 1993).

Using the same scenario stated above, one must also consider the value of the time needed to retrieve a file which is stored on a less expensive medium. While some tapes are stored in-house, many are stored off-site. In either case, there is a substantial time lag associated with locating and loading these files. In this research, we will use the premise that organizations possess at least two different options for storage of program files. The notion is that the decision for a storage option should be made based on the dynamic usage frequency of the program.

## 1.2 The Need for Software Usage Modeling

At this point it is necessary to examine the current processes being used to handle storage decision-making. Some organizations and individuals treat the problem of limited storage with more seriousness than others. To some extent, the issue has attempted to be addressed with the development of various software packages to aid users in freeing valuable hard disk space. In this section, the approach used by many storage analyzing software packages will be discussed. This will show the need for a software usage modeling process to better indicate the behavior of software usage. The need for software usage modeling will also be justified by addressing

the nature of stored information and revealing the current organizational practices used in order to track usage.

We will begin the discussion here by reporting on the technology used in several off-the-shelf software packages designed to monitor hard disk storage. In order to investigate available software packages, several sources were consulted for information on the most widely used products. These sources included software developers, software salesmen, and publicized advertisements.

One of the current packages available is called WindoWasher sold by Micah Development. They advertise that the package builds a picture of file use over time, tracking which files are used most and least often. It boasts the use of a process called intelligent data compression, allowing one to free disk storage space by exiling little-used files. The software provides several views of file usage. One method displays files in a pie chart, labeling them as frequently used, occasionally used, and never used.

The description of the software in the advertisement leads one to believe that the software will solve all hard disk overcrowding problems. Further investigation of the design of the software package revealed many shortcomings in its capabilities. The development company revealed that the software has no capability of tracking overall software

usage. The views of software usage only assess information on a directory by directory basis. There is no ability to compare usages from different directories. Within a directory usage can only be computed based on number of accesses to executable programs. This implies that a document created by an executable file is not included if it is later accessed. Many document files are critical components of hard disk consumption. Thus, it is very misleading to only include executable files in a usage assessment. Also, the software does not provide for a break-in period for execution of files. Naturally, not all files will be executed immediately and the usage display will not be accurate for this period of time. In addition, no items will appear in the "never used" category once every file in a directory has been executed. The analysis of this particular software proved that no estimation for future use was given. Only a static view of executed files in a single directory was given. This provides no potential for planning and does not accommodate an expanding base of software. Here we do not include the various types of software which are being developed to manage hierarchical storage in LAN environments. These will be discussed in the literature review of storage management in Chapter 2.

The purpose in investigating the aforementioned software was to justify the need for a modeling process which captures the constantly changing usage patterns of organizational software. This tendency was observed through examination of the daily logs which the operating system of a national financial corporation produced. On a daily basis, this corporation has reports printed showing for each program used the average I/O time, average memory used, the size of the code plus the data, and a count of the number of times that the program was used. These programs are categorized by most heavily used, worst processor usage, worst I/O usage, and worst memory usage. Other analyses performed are listings of those programs having unusual terminations and listings of the most heavily used programs by usercode.

In questioning the head of this corporation's MIS department on the benefits of the daily reports produced, the answers received were somewhat startling. In actuality, the voluminous collection of daily reports were seldomly if ever used. In fact, the determination of their very existence was also in question.

The observations found at this corporation gave further justification for the need of a modeling process for software usage. Here was an organization spending time-consuming effort to produce reports which added no insight into the

status of program usage. The insight gained through these initial observations justified not only the need for usage evaluation but also proved that the data needed to test the proposed model is available.

The value of a mechanism for software usage modeling is only worth investigating if it can add a predictive component to the behavior of software usage. That is, an organization can access system data and determine for a given point in time which programs have greater and lesser usage. For instance, it is conceivable that a system's usage report would indicate that a certain transaction processing program was highly used on a given day. This might even be true for many days. The problem is that the examination of a few snapshots in time does not tell the entire story of a program's usage profile. Certainly, it would be very difficult for a large corporation with thousands of programs to track the day by day usage of every program. That is why this research serves to provide a methodology whereby a predictive model could give a breakdown of which programs' usages warrant careful attention and which do not. Also, model results can provide a heuristic approach for determining necessary intervals for program re-evaluation.

## 1.3 Phenomena in Software Usage

Usages in information systems usually follow a skew distribution. As mentioned earlier, some programs are used more frequently than others. It is proposed in this dissertation that static views of program usage frequency are insufficient when they are used for storage allocation decision making. Therefore, it is necessary to study the implications of the use of dynamic frequencies in storage allocation algorithms.

The study of usage phenomena has led us to the valuable usage modeling tool that we will use in this dissertation. Examples have been made in issues of personal wealth, income, size of business firms or cities, publication frequencies, and word frequencies (Ijiri and Simon, 1977). In Ijiri and Simon's words, these "social phenomena" exhibit distributions which closely fit a Pareto distribution. Therefore, this phenomenon is known as the Pareto Principle or 80/20 Rule.

The simplest way to describe the pattern of usage concentration is to assign some kind of quantitative measurement to it. Vilfredo Pareto (1909) first reported that in his native country about 80% of the wealth was concentrated in about 20% of its population. Since then, many other sociological, economic, and natural phenomena have been observed to follow the similar pattern. J. M. Juran, a

well-known figure in quality management, claims credit for coining the term Pareto Principle, which in effect is the 80/20 Rule (Sanders, 1987). According to Zunde (1984):

> It has been observed that many other empirical phenomena, both in the domain of information science and in other fields, obey this [Pareto] probability distribution law or exhibit dependencies derivable from it.

The 80/20 measure is not a ratio. It has been used mostly as a heuristic to differentiate the "significant few" from the "trivial many," and it was not originally intended as a rule for action (Sanders, 1987). For example, approximately 80% of the information usages might involve only about 20% of the resources. Similarly, in libraries (Lancaster and Lee, 1985) roughly 80% of transactions involve 20% of holdings. The measure may be 85/35, 88/40, or 95/25. The concentration of "significant few" to "trivial many" strictly depends on the example chosen.

The application of this rule often emphasizes the "significant few." For example, Boehm (1987) suggested that 80% of rework costs in software development typically result from 20% of the problems. The implication is that software verification and validation should focus on identifying and eliminating the high-risk problems in software projects.

In this dissertation the empirical phenomena of the 80/20 rule will be used in order to show the concentration of

those few programs that are used the majority of the time in organizations. The distribution produced by the 80/20 Rule is only a snapshot in time. If a distribution is developed in order to assess the wealth in a given country, then the distribution is accurate at the time that it is evaluated. The same is true with the Fortune 500, which gives the ranking of the top 500 companies each year which possess the greatest amount of wealth. It is known that a company in the top 10 in one year has no guarantee of holding that position in the future. This important observation about the static nature of the 80/20 Rule serves as a foundation for much of the work of this dissertation. Any model of usage concentration will not be valuable unless it is able to capture the dynamic nature of items moving in and out of the "significant few" and "trivial many." For this reason, the traditional use of Simon's Model will be revised in order to predict the changes in usage tendency for programs or software.

### 1.4 Problem Statement

This introduction has emphasized the current lack of storage allocation practices in organizations. The trend is that valuable resources are being spent in order to acquire increased hard disk storage capacity to accommodate software that is often outdated and unused. With this having been

stated the problem that this dissertation addresses can be stated in three parts:

1) The need for a methodology that can be adapted by organizations which would enable them to easily assess the overall behavior of their software usage. In knowing the overall tendencies in usage decline and usage increase, they can adapt the proposed re-evaluation scheme which best suits their software usage profile.

2) The need for a usage model that has the ability to capture the dynamic progressions of software ranking status in organizations. This requires an adjustment in the previously explored versions of Simon's Model.

3) The development of an allocation model for multi-level storage systems which accommodates the dynamic usage frequencies of programs rather than only a static view. This model must show the value of dynamic usage frequency from a cost perspective.

## 1.5 Dissertation Purpose and Organization

### 1.5.1 Dissertation Purpose

Having stated the above problems, the purpose of this dissertation is threefold. First, it addresses a very

current problem found in all large organizations. This problem is the consumption of resources in expanding storage capacity. It is proposed that this problem can be remedied. The ultimate goal is to provide organizations with a methodology that will enable them to categorize their overall software usage behavior. With this information, they may chose the appropriate scheme for software storage decision-making.

Second, this dissertation adds a dynamic component to the use of Simon's Model. That is, it will be adapted in order to model a real world scenario which is prevalent in the storage of programs. This will show the flexibility that Simon's Model has when it is used as a tool for studying dynamic usages.

Lastly, this dissertation will incorporate a unique storage allocation model which considers dynamic usage frequencies while minimizing storage cost.

### 1.5.2 Dissertation Organization

The dissertation is organized as below. Chapter 2 gives a comprehensive view of the methodologies which have been proposed for multi-level storage over the past three decades. Chapter 3 gives a synopsis of the development and use of Simon's Model. It includes the origin of the model along with its different versions and appropriateness in this

research. Chapter 4 provides a description of the design of the experiment to be performed. This includes the simulation program set-up and methodologies used in order to correctly define the scope of the study being performed. Chapter 5 provides simulation results and model validation. It was necessary to use real data in a comparison with simulated results in order to show that Simon's Model is appropriate for capturing program usage behavior. Chapter 6 uses the validated model in order to produce and compare results over a range of model parameters. With a complete spectrum of results, it can be shown that program usage behavior fits into one of five general schemes. This chapter also shows the simulated results when the parameters for Simon's Model are dynamically changed in order to capture more specific instances of program usage. Chapter 7 uses the insight gained from the simulation study in order to recommend a dynamic allocation model for multi-level storage while minimizing cost. Chapter 8 is the final chapter giving conclusions and plans for future research.

# CHAPTER 2 LITERATURE REVIEW OF STORAGE MANAGEMENT

In this chapter, a review of the literature in the field of storage management is given. With the realization that storage management is a very broad area of research, this review concentrates on the methodologies used in hierarchical storage and allocation of information. Previous research in this area spans the last three decades. We explore some of the various approaches for allocation of information resources used in mainframe and PC LAN environments while enumerating the strengths and weaknesses of the various methodologies. The mainstream research explored in this chapter will serve as a foundation for the dynamic allocation methodology developed in this dissertation.

## 2.1 Definition of Storage Management

Storage management can mean anything from data backup and monitoring the health and performance of disk drives to the use of new distributed file systems. A better definition looks past the means to the ends of storage management. The definition might state that the goal of storage management is to provide users with optimal system performance and minimum downtime while protecting electronic assets (Reinhardt, 1994). As organizations become more dependant on information technology, they must also become more aggressive in their strategies for protecting information assets.

Storage is an important and expensive component of a computer system. Many types of storage such as magnetic disks, magnetic tapes, and optical disks are available, and each has a different cost and set of physical attributes. Implementing rigorous media monitoring, backup, and file grooming procedures may seem like a burden to users, but it is the only way to ensure the integrity of data and information assets.

A major storage management problem encountered by database and network managers is the increasing difficulty in keeping step with users' insatiable appetite for storage. In many organizations, adding more disk drives is a short term solution since the cost of disk storage is moderate and declining. However, management costs and losses in productivity increase with the amount of storage and dominate the total cost of ownership. In the long run, simply adding disks to expand storage is neither economical nor even feasible (Ghandeharizadeh et al., 1994).

Another reason for the opposition to the continual expansion of storage is stated by Robert Wright, president of Avail Systems Corporation (Mangold, 1995). He states, "About 80 percent of corporate information is rarely accessed. About 20 percent is very active." Therefore, arises the situation of adding storage for files which are seldomly used. Organizations attempt to remedy this situation in part

by using storage media with slower access times. Organizations can create three or four tiers of migration from the quick-access hard disk. For example, files may first be moved to an optical jukebox, where access takes a few seconds. Subsequently, files may be moved to a tape library where it can take several minutes to retrieve a file (Willet, 1994). Thus, modern computer systems generally consist of several different types of storage devices. Such an assemblage is called a multi-level storage system of a storage hierarchy system (Chen, 1973). Improper management of storage at one or multiple levels may lead to an added organizational burden. Management may not always be aware of the various hidden costs and difficulties. In this section we examine the literature on allocating files in multi-level storage systems.

We begin with some recurrent issues in storage management. Then, we trace the evolvement of algorithms and tools in multi-level storage systems in order to form a basis for an extension to existing methodologies.

## 2.2 Recurrent Issues in Storage Management

The true cost of storage is much greater than the cost of the different forms of storage media themselves. Strategic Research, Inc. determined that, while the cost of disk drives was less than one dollar per megabyte to acquire, management costs were over seven dollars per megabyte

(Mangold, 1995). Management costs include user disk and file management, productivity losses form storage failures, backup and restore costs, storage services and repair, archive costs and drive installation labor. Therefore, simply adding disk drives is much more expensive than most people realize. In order to lower the cost of ownership would require lowering management costs by simplifying administration and leveraging storage capacity (Iida, 1995).

Backup and disaster recovery become increasingly difficult or even impossible as storage arsenals grow since the time required to perform the operations increases with the amount of stored data and eventually exceeds the time allotted. Consider the task of restoring 18.4 gigabytes of data. This task would take fifteen hours at a standard 1.2 gigabyte/hour backup rate. If the amount of data grows to 63 gigabytes over a two year period, restoration will require over 53 hours. Most companies cannot afford that much downtime, especially in mission-critical situations (Imagery Software, 1995).

The scenarios described above have forced information technology to search for scalable, application sensitive storage solutions. In the following sections of this chapter, we will trace the evolvement of algorithms and tools for storage systems in order to form a basis for an extension to the existing hierarchical storage methodologies.

## 2.3 Optimization of Information Storage Systems

### 2.3.1 Minimization of Average Access Time

There is a methodology for optimization of hierarchical storage systems that involves the selection of types and sizes of memory devices such that the average access time to an information block is a minimum for a particular cost constraint. This approach was forwarded by Ramamoorthy and Chandy (Ramamoorthy and Chandy, 1970). The analysis used is based on the following program and device characteristics.

1. Given a program and its associated data, its memory requirements are known as well as the usage frequencies of the program components (independent modules). The usage frequency of the data is also known by analysis or measurement during previous runs.
2. The information is stored in a memory hierarchy which is an assemblage of storage devices. The memory devices are characterized by their cost which is a function of their storage capacity and their average access time.

The basic problem solved using this approach can be described as follows. Given the maximum permissible costs of memory and given "N" different memory types of different costs and access times, determine the optimum allocation of resources to different memory types so as to minimize program average access time described by an activity profile and stored in memory.

The theory developed by Ramamoorthy and Chandy can be applied in evaluating competing memory organizations, storage units and peripheral devices. It can also be used to compute the change in throughput caused by using multi-precision arithmetic rather than single precision. The technique can be extended to determine the optimum allocation of memory to various programs running in parallel so as to maximize their combined throughput.

#### 2.3.1.1 Notation

For the sake of clarity, it is necessary to define the following notation used in this methodology. The first set of terminology is based upon the fact that when a given computer program is run, certain blocks of information are accessed more frequently than others. For example, in a sorting program, the instructions in the basic comparison loop are accessed more frequently than in other parts of the program. A table could be drawn giving the number of information blocks accessed at a given frequency. Thus, we can describe a program consisting of equal-sized blocks such that $W_i$ is the number of blocks accessed at a frequency $F_i$, let $F_1$, $F_2$, $F_3$, . . . , $F_M$ be the M different frequencies and let $W_1$, $W_2$, $W_3$, . . . , $W_M$ be the corresponding number of blocks accessed at those frequencies. We can then define activity $P_i$ as being directly proportional to the access frequency $F_i$. Quantitatively, it is given by

$$P_i = F_i / \sum_{i=1}^{M} F_i W_i .$$

Thus, the activities are positive quantities such that

$$\sum_{i=1}^{M} P_i W_i = 1 .$$

Let the activities of a program be expressed by a vector **P** of dimension M, i.e., **P** = ($P_1$, $P_2$, ..., $P_M$) and such that $P_i > P_{i+1}$. Associated with such a vector **P**, there is an M dimensional vector **W** such that its ith component $W_i$ represents the number of blocks of information accessed at activity $P_i$.

The ordered pair of vectors (**P**,**W**) is defined as the activity profile of the given program. Activity profiles can be determined either analytically or by simulation and experimentation (Martin, 1965, 1967; Ramamoorthy, 1965). Also, let there be N types of memory devices. Let $T_i$ be the cycle time or accessing time of the ith type and let $C_i$ be the cost of one block of that memory type.

Generally, the memory hierarchy includes a memory type known as the mass memory. It has the least cost per memory block and the largest access time. The capacity of the mass storage is large enough so that it can accommodate the entire set of programs that is scheduled for future computer runs.

Generally, the type, size and cost of mass storage units are fixed. The practical problem, then, is to determine the remaining members of the hierarchy to minimize the average access time.

**2.3.1.2 Formulation of the Problem**

Based on the aforementioned notation, a description of the problem formulation and its solution for determining the cost/time characteristic in the single program case is given. It is worth noting that the model was developed first for the single program case and then extended to the multiple program case.

Let the following be given:

1. The maximum permissible cost G of the entire storage system,
2. N different types of memory where the cost per block and the average time to access one block are $C_i$ and $T_i$ respectively,
3. The activity profile of the information to be stored in the hierarchy is given by the 2 X M matrix

$$P_1 P_2 \ \ldots \ P_M$$
$$W_1 W_2 \ \ldots \ W_M$$

where $P_i > P_{i+1}$ and $\sum_{i=1}^{M} P_i W_i = 1$. The P's are activities and the W's are the corresponding number of blocks.

It is required to determine the sizes of the different storage types and the location of information blocks in the storage such that

1. the total cost does not exceed G and
2. the average access time to any information block stored in the hierarchy is minimized.

Without losing any generality we can assume that $G_{Mass}$ is the cost of the mass storage or Type 0 memory, and it is large enough to accommodate all the blocks in the program. We shall assume that one block of information occupies one unit of memory space.

Let $V_{ik}$ be the number of information blocks of activity $P_i$ stored in memory type k ($1 \leq k \leq N$). Let

$$V_{i0}=W_i-\sum_{k=1}^{N} V_{ik}.$$

Thus, the problem is given that

$$G_{Mass}+\sum_{k=1}^{N} C_k\sum_{i=1}^{M} V_{ik}\leq G.$$

$$\sum_{i=1}^{N} V_{ik}=W_i \qquad \text{for } i = 1, 2, \ldots, M.$$

$$V_{ik}\geq 0 \qquad \text{for } 1 \leq i \leq M; \quad 0 \leq k \leq N$$

Minimize

$$\overline{T}=\sum_{i=1}^{M}\sum_{k=0}^{N} P_i T_k V_{ik}.$$

The size of memory type k, $U_k$ is then

$$(k\neq 0):\sum_{i=1}^{M} V_{ik}=U_k.$$

A set $V_{ik}$ for $(1 \leq i \leq M)$ and $(1 \leq k \leq N)$ which satisfies these conditions is called an optimal solution.

**2.3.1.3 Appropriateness of the Model**

It is necessary to state the set of assumptions which was made in this model.

1. The memory size is divisible. This is only an initial assumption which is modified later.
2. The probability of accessing any block does not depend upon the past history of the computation.
3. The cost of associated hardware such as disks or memory control units is not included here but can be taken into account.
4. It is assumed that the activity profile of a program is constant in time or static.
5. The model fits well if we consider the computing system as consisting of two entities: the central processor and its associated memory, and the memory hierarchy.

The activity profile takes into account only those events which require accessing to information during the course of normal computation and not those events which require accessing to information due to the computation environment. Thus, in this model the interest is only in the accesses made to the files which are independent of environment.

**2.3.1.4 The Algorithm**

After determination of the derived hierarchy, it is necessary to determine the variation of the minimum access times over total memory costs for a given program with a specific activity profile. Next, for a fixed memory cost and a specific activity profile, it is necessary to determine the minimum average access time, the sizes of memory types in the hierarchy and the location of information blocks within the hierarchy.

After the derived hierarchy is determined, the procedure consists of three parts.

1. Determination of transfer priorities

   Given the derived hierarchy, we determine the transfer values $X_j$ where

$$X_j = \frac{T_{(j-1)} - T_j}{C_j - C_{(j-1)}}, \quad j = 1, 2, \ldots, n.$$

The transfer priority $Z_{i,j}$ is determined as

$$Z_{i,j} = P_i \cdot X_j \ , \ i = 1, \ \ldots, \ M; \quad j = 1, \ \ldots, \ n$$

The $Z_{i,j}$ are ordered by decreasing magnitude

2. Determination of minimum average access time-total cost characteristic by finding optimal solutions by means of simple transfers.

The transfer implies by $Z_{i,j}$ is the shift of all information of activity $P_i$ from memory type j-1 to memory type j. The incremental cost required to implement this transfer is $\Delta C = W_i(C_j - C_{j-1})$. The incremental change in average access time due to this transfer is $\Delta T = -W_i \cdot P_i \cdot (T_{j-1} - T_j)$. Thus, if e was the cost of the hierarchy before the transfer, $e + \Delta C$ is the cost of the hierarchy after the transfer. Similarly, if r was the average time taken to access an information block stored in the hierarchy before the transfer then $r + \Delta T$ is the average access time after the transfer.

All of the information is stored in the mass memory. The size of a memory type k, where $k \neq 0$, is zero. The cost of this initial hierarchy is $G_{Mass}$. The average access time for this hierarchy is $\overline{T} = T_o$. Let e(k) and r(k) be the cost and average access time of the hierarchy after the kth transfer. Let $\Delta C(k)$ and $\Delta T(k)$ be the change in cost and average access time due to the kth transfer. Then,

$$e(k) = e(k-1) + \Delta C(k),\ k = 1, 2, 3, \ldots$$

and

$$r(k) = r(k-1) + \Delta T(k),\ k = 1, 2, 3, \ldots$$

$$e(0) = G_{Mass}$$

$$r(0) = T_o.$$

3. Determination of the exact composition of the optimal memory hierarchy for a specified total memory cost and its minimum average access time.

In this stage, we need to determine the number k' such that $e(k') \leq G \leq e(k'+1)$. If $e(k) < G$ for all k, then store all the information in memory type n, the quickest access memory type. Then in the optimal memory hierarchy, the size of memory type n, $Un = \Sigma Wi$ and the size of memory type k, $k \neq 0$ and $k \neq n$ is zero. If there exists a k' such that $e(k') \leq G \leq e(k'+1)$ holds, let the ordered transfer priorities $y_1, y_2, \ldots, y_{(k'+1)}$ be $Z_{i(1),j(1)}, Z_{i(2),j(2)}, \ldots, Z_{i(k'+1),j(k'+1)}$.

To determine the location of information of activity $P_u$, inspect the ordered transfer priorities from right to left until we find Zi(l),j(l) such that $i(l) = u$ and $i(q) \neq u$ for $k'+1 \geq q > l$.

(i) If no such $Z_{i(1),j(1)}$ exists all the blocks of information with activity $P_u$ are stored in the mass memory.

$V_{u0} = W_u$.

$V_{uk} = 0$ for $k \neq 0$.

(ii) If such a $Z_{i(l),j(l)}$ exists and $l \neq k'+1$ then all the blocks of information with activity $P_u$ are stored in memory type g, where $g = j(l)$

$V_{u0} = W_u$

$V_{uk} = 0$ for $k \neq g$

(iii) If such a $Z_{i(l),j(l)}$ exists and $l = k'+1$ then the number of blocks of information with activity $P_u$ stored in memory type g where $g = j(l)$ is

$$V_{ug} = \frac{G - e(k')}{C_g - C_{g-1}}$$

The remaining blocks of activity $P_u$ are stored in memory type g-1

$V_{u(g-1)} = W_u - V_{ug}$

Vuk = 0 for $k \neq g$ and $k \neq g-1$.

The size of memory type k in an optimal memory hierarchy is $U_k = \Sigma V_{ik}$ for k = 1, ..., n.

### 2.3.1.5 Application of the Algorithm

The algorithm described above has several applications. Here we will mention two of these. First, the optimum sizes of a number of different memory types in a hierarchy of memories is determined by the algorithm. The process of determining the optimum sizes is carried out in two steps.

The two steps involve selecting a subset of memory types (a derived hierarchy) and determining the optimum sizes of the individual memory types in the derived hierarchy. After the two step process is performed, the preference between the mass memory types is determined by:

1. the typical activity profile
2. the cost of the total memory organization and
3. the other memory types.

Another important application of this technique is in automatic language translation. In the process of translating a language, the number of worlds is large and the activities vary. Therefore, the words are stored over a memory hierarchy. Knowing the activity profile of the words in a language, the algorithm is used to design an optimal memory hierarchy where the translation rate is maximized.

### 2.3.2 Minimization of Storage Cost

Another approach for optimization of file allocation is based on the minimization of storage cost. The methodology was produced by P. P. Chen (1973). In this section, we analyze the procedure for minimizing total storage cost while satisfying one mean overall system response time requirement. This approach can also be extended to include the case for individual response time requirements.

#### 2.3.2.1 Notation

We define the following notation which is used in this model.

$L$ = the total number of files.

$c_j$ = the cost per block of storage using device j.

$n_{ij}$ = the number of blocks of file i allocation to device j.

$\mu_j$ = the service rate.

$\lambda_j$ = the arrival rate.

$N_i$ = the number of blocks in file i.

$V$ = a response time bound.

$R(\{n_{ij}\})$ = the weighted sum of the mean response time.

#### 2.3.2.2 General Statement of Problem

When storage cost is a factor in allocation of information, the following problem arises: allocate the files such that the storage cost is minimized and the mean response time is bounded. That is,

Minimize

$$\sum_{j=1}^{M} C_j \sum_{i=1}^{L} n_{ij} \qquad (1)$$

Subject to

$$R(\{n_{ij}\}) \leq V \qquad (2)$$

$$\sum_{i=1}^{L} n_{ij} f_i < \mu_j, \qquad j = 1, \ldots, M \qquad (3)$$

$$n_{ij} \geq 0, \qquad i = 1, \ldots, L, \; j = 1, \ldots, M \qquad (4)$$

$$\sum_{j=1}^{M} n_{ij} = N_i \qquad i = 1, \ldots, L \qquad (5)$$

The expression (1) above denotes the total storage cost. (2) is the mean system response time constraint where V is the given bound.

**2.3.2.3 The Optimal Solution**

The optimal solution for the problem will follow an allocation strategy. This allocation strategy (strategy A) orders files according to their relative frequency of access, and allocates files to devices in order. Start by allocating the file with the highest reference frequency to the faster device.

Algorithm 2.1 is proposed for the case where M = 2. For the case where M ≥ 3, Algorithm 2.2 (a conventional algorithm) is chosen, and a methodology is proposed for obtaining the initial feasible solution. These two algorithms are stated below.

Algorithm 2.1 (for M = 2)

Step 1. Calculate the total input rate λ using

$$\lambda = \sum_{j=1}^{M} \lambda_j = \sum_{i=1}^{L} \sum_{i=1}^{L} n_{ij} f_i, \qquad (6)$$

Step 2. Find $\lambda_1'$.

Step 3. Allocate files according to allocation strategy A, ensuring that the mean input rate to the faster device equals to $\lambda_1'$.

Algorithm 2.2 (for M ≥ 3)

Step 1. Calculate the total input rate λ using (6), and the input rates.

Step 2. If $R(\lambda_1^*, \lambda_2^*, \ldots, \lambda_M^*) > V$, then terminate.

Step 3. Use $\lambda_1^*, \lambda_2^*, \ldots, \lambda_M^*$ and allocation strategy A to calculate {nij}, the feasible solution.

Step 4. Use this initial feasible solution and the Sequential Unconstrained Minimization Technique to find the optimal solution.

**2.3.2.4 Consideration of Individual Response Time**

Files may have individual response time requirements. This means that the inequality in (2) is replaced by L individual response time requirements (one for each file). In addition, there are many situations in which the probability of the response time for an individual request exceeding some given bound must be limited. Thus, the cumulative response time probability distribution must enter the analysis.

With the inequality in (2) replaced by L individual inequalities, another important type of file allocation problem can be formulated. That is, allocate files minimizing the storage cost and limiting the probability that the response time for file i exceeds some given bound.

This type of problem is very complicated, exhibiting one or more nonconvex feasible regions. Discussion of this type of problem was found in the work of Chen and Mealy (1973).

## 2.4 Long Term File Migration

The steady increase in the power and complexity of modern computer systems has encouraged the implementation of automatic file migration systems which move files dynamically between mass storage devices and disk in response to user reference patterns. The limited storage capacity of the disk system generally results in some form of file migration, whereby active files are kept on or moved to the disk and inactive files are moved to or kept on tape or mass storage. This migration may be managed or done automatically by the system.

As with any decision-based procedure there are a number of problems which may arise. The problems associated with file migration are as follows: when to migrate a file from mass storage to disk (fetch algorithm), where to place it (placement algorithm), and when to remove that file to mass storage when the disk space is again needed (replacement

algorithm). The file migration algorithms explored in this section are all replacement algorithms.

There has been considerable study of the replacement algorithm problem in the context of main memory paging (Smith, 1976 and 1978; Chu and Opderbeck, 1976; Morgan and Levin, 1977; and Reich, 1978). A useful study of replacement algorithms with regard to files was performed by Stritter (1977). Other research has considered the file migration problem in general or descriptive terms (Boyd, 1978; Lum et al., 1975).

### 2.4.1 File Reference Patterns

We will begin this review of long term file migration with analysis which was performed on various file reference patterns (Smith, 1981). It was observed that most files were used on two or fewer days during the measurement period. Of those files studied, there was a portion whose reference patterns permitted useful statistical analysis. In the portion statistically analyzed, about one third were found to show a declining rate of reference with age. The only files included in the analysis were those with no declining trend. Of those files with no declining trend, about 5 percent showed statistically significant serial correlation of the interreference intervals, and about 40 percent were found to have interreference time distributions that were more skewed than the geometric distribution. Significant differences in

various parameters were found between files of different sizes and classes.

### 2.4.2 Comparison of File Replacement Algorithms

#### 2.4.2.1 Methodology

In Smith's (1981) study, the goal was to compare the relative performance of file replacement algorithms. A mechanism was designed in order to provide a reasonable criterion for evaluation. The criterion were effectively compared by graphical means. The graphical results show a comparison of operating points of the form [S(A,P), M(A,P)]. The methodology for deriving operating points is as follows: Define A to be the replacement algorithm and P to be the parameter associated with that algorithm. Let M(A,P) be the fraction of all file "references" that require a fetch from mass store. S(A,P) is the mean number of disk tracks occupied by on-line files. Therefore, the pair [S(A,P), M(A,P)] constitutes an operating point and may be plotted on an x-y plot. The better algorithm has an (S,M) curve which is to the left and below the worse algorithm. It is straightforward for each algorithm to compute the values for S(A,P) and M(A,P).

#### 2.4.2.2 Definition of Algorithms

The purpose of exploring various file replacement algorithms here is not aimed at finding the best algorithm. Instead, the purpose is to investigate the various file

replacement algorithms in order to lay a foundation for expansion to the existing methodologies.

Smith's work first discussed a type of realizable algorithm (Smith, 1981). A realizable algorithm must deal only with known information, i.e., past history. Therefore, it can at best estimate which file will not be used. A realizable algorithm which provides accurate estimates of the time to next use can be expected to yield good performance. In order to derive effective realizable algorithms a Q(A, H, t, Sz, C, P) policy is used for removing files. The parameters are defined as follows: A is the algorithm; H is the entire previous history of reference to the file; t is the time (date); Sz is the size of the file; C is the class of the file; and P is the "cost" of fetching the file when it is next referenced (should it be removed). P is the parameter that is varied in order to produce the (S,M) curve. The algorithm describes various procedures used in order to reduce the number of parameters for Q and find the stochastically optimal policy. The intricate details of this algorithm will not be stated here, but it is worth noting that distribution of time to next reference is not used in this algorithm.

The expected time to next reference is a simpler approach than the approach above. It involves the selection of files whose (time to next reference) x (size) is maximal.

This method is not optimal since the mean expected time to next reference contains far less information than the entire interreference time distribution. The reason for this nonoptimality can be shown by the following example: Let the distribution of time to the next reference be bivalued with the probability 0.5 that the reference is in day 1 and 0.5 that the reference is in 99 days. Then, the expected time to next reference is 50 days. The optimal policy would most likely keep the file for one more day and then if it is not used, discard it. A policy based on only the expected time to next reference would probably discard the file immediately since the expected time is so large. Without explicitly defining the steps of the algorithm, we will instead focus on observations of the methodology. First, the expected time to next reference is generally increasing with the time since the last reference. Thus, the desirability of retaining a file will decline with the time since the last reference. Second, the algorithm cannot effectively fit a curve to the expected time to next reference.

The two algorithms mentioned thus far have relied on interreference time distributions for the file reference process. It is also possible to define some algorithms which do not use measured data but which implicitly assume some model of file reference behavior. Comparing methods developed from the data with either ad hoc algorithms or

those used elsewhere would be necessary to indicate performance improvement. We will now define Working Set (WS), Space-Time Working Set (STWS), and Space-Time Product (y is the parameter and ** stands for exponentiation). Working Set is that algorithm which removes a file when it has been unreferenced for P or more days. This algorithm was originally designed for main memory paging, and in that circumstance it has been shown to work very well. It has the defect that it takes no account of file size, and thus small files are as likely to be removed as large files. The Space-Time Working Set is the straightforward and obvious extension of WS to variable size objects. It removes any file for which the product (time since last reference) x (file size) is greater than the parameter P. The implicit assumption here is that the file that is likely to incur the largest cost of retention to the next reference is that which has already accumulated the largest retention cost since last reference. It was observed in the study that the expected time to next reference climbs quite steeply with time since last reference. It was also shown that larger files are used more frequently than smaller files. These two facts suggest that a modification of STWS which weighted time since last reference more heavily than file size should perform better than STWS. This modification gives the STP**y class of algorithms. In STP**y the following is computed: [Size x

(time since last reference)**y. If the value of the expression given is greater than the parameter P, the file is removed.

A contention of Stritter was that most files observed displayed reference patterns that could be characterized as Poisson. Stritter treated a discrete-time series as a continuous-time series. It was found in Smith's study that of those files testable, the majority could not be characterized as Bernoulli and therefore not Poisson. Nevertheless, experimentation was done which assumed that the actual file reference process was Bernoulli for each file. The Bernoulli process is such that the expected time until the next reference is constant regardless of the time since the last reference. In this way the replacement decision can be made immediately after reference to a file. The only problem is to estimate the rate at which the file is being reference. The time to next reference was predicted with an exponential estimator. All algorithms of this class that were tested were found to perform very poorly (Stritter, 1977).

## 2.5 Current Issues in Hierarchical Storage Management

The models and algorithms that have been discussed thus far have shown the various principles behind the decision for migrating files. These various tools for storage decision-making have been incorporated into an approach called

Hierarchical Storage Management (HSM). HSM is the general term for the combination of hardware and software which migrates less active files to less expensive storage devices. HSM was once used only on mainframes, but is now being adopted by network and system administrators. In this section, a summary of the key issues surrounding HSM will be discussed.

### 2.5.1 Definition

HSM is an automated management system that can be configured to intelligently place data in storage repositories, moving it through a series of storage devices according to immediacy of need (Reinhardt, 1994). HSM places frequently accessed data and files on high performance, high quality disk storage and then moves less frequently accessed data and files to slower but less expensive rewritable optical, CD-ROM, or tape. Simply put, HSM moves less critical data into less expensive secondary or tertiary storage where it no longer clutters local disk drive capacity.

Although files have been physically relocated, HSM maintains all the end-user references to the data. The HSM system then automatically recalls the files as they are accessed. The end-user therefore can view HSM as a never-ending storage pool, where the physical storage medium no loner matters. To database and network administrators, HSM

automates the maintenance of the file system, ensuring that storage is always available and that resources are being used most efficiently.

### 2.5.2 Benefits of HSM

With the use of HSM, the cost of increasing capacity is lower as units of optical storage units, not units of hard disk space, are added. Compared to disk storage, optical storage costs half as much per megabyte. Migrating files to even less expensive, tertiary CD-ROM storage leads to further cost reductions.

HSM improves data availability by maintaining high-speed access to required data and by ensuring that storage is always available. This, in turn, decreases the costs associated with reduced productivity due to poor data availability. HSM also boasts a reduction in the backup and restore burden, and can reduce data restoration time from fifteen hours to six hours (Mangold, 1995).

### 2.5.3 HSM Pricing

There are three ways in which HSM is generally priced. It can be based on the amount of storage, the number of users, or according to a fixed-pricing scheme. If the price is based on the amount of storage, the price is measured in gigabytes. This implies that a set amount of storage will be supported with unlimited users. This is the approach currently used by Intergraph Corporation to price its HSM

products. Number of user's price is used for a specified number not to exceed a fixed license price. Fixed-price scheme price is a flat rate which is independent of the number of users and the amount of storage (Brisse, 1996).

### 2.5.4 Levels of HSM

There are different levels of HSM. Although HSM is not a new technology, there are still many misconceptions about the claims of HSM. Peripheral Strategies, Inc. has provided these various level definitions.

Level 1

involves automated file migration with transparent retrieval

Level 2

Real-time, dynamic load balancing of disk space based on multiple predefined thresholds, manages two or more levels of the storage hierarchy

Level 3

It provides for transparent management of three or more levels of the storage hierarchy. Storage thresholds between different levels in the hierarchy are dynamically balanced and managed. Volume management, job queuing, and device performance optimization are performed.

Level 4

Policy management and administration at all levels of the hierarchy are a concern. It provides for storage management of diverse platforms. These services include maintaining the ownership and location of data.

Level 5

This is object level management. It preserves the relationships of objects at all levels in the hierarchy.

### 2.5.5 Concerns of HSM

As with any technology, there are certain pitfalls to HSM. One problem is the inability to retain file's original dates. When files are recalled, the date on the file becomes the date of the recall, not the original file's date. This is problem for users who need to see when a file was created.

Flexibility is another problem area. HSM does not always accommodate a dynamic environment. The HSM tracks files according to the volume on which they currently exist or where they formerly existed for those that have been migrated. If the system administrator moves files that have already been migrated to a new volume that becomes an issue (Ryan, 1994).

## 2.6 Summary

In this chapter we have made an in depth exploration into the various methodologies which have been proposed for solutions to storage problems. In all of these

methodologies, we have found a common thread. This common thread is the assumption of static usage frequencies by program and data files. It is the goal of this dissertation to show that this is a weak assumption. In developing an algorithm for optimal storage allocation of programs, we can still use much of this literature as a foundation. But, we will show the improvement which can be made if dynamic usage frequencies are considered in the model. Simon's model of information usage will be an important tool in capturing the dynamic nature of program usage. In the following chapter, a foundation for the use of this model will be made.

# CHAPTER 3 THE USE OF SIMON'S MODEL

## 3.1 The Need for Software Usage Assessment

In order to address the issue of software storage, it is necessary to determine the behavior of the usage patterns of those software items in organizations. In the introduction to this dissertation, it was stated that software (programs) in organizations possess the characteristic that few are used frequently and many are used infrequently. In addition, most organizations employing their own information systems department have the added burden of keeping track of new software development and purging their systems of programs no longer in use. Since one of the goals of this research is to provide organizations with a methodology for more efficient software storage, it is necessary to study the usage behavior of programs. Naturally, no two organizations possess the same usage pattern when using software developed to suit individual organizational needs. But, it is important to realize that software in all organizations have some common characteristics. That is, all organizations acquire new programs while knowing that the usage rate of older programs declines with changing needs. Thus, it would be of great value to organizations to have a predictive ability for their changing software storage needs. This chapter will explain

the use of Simon's Model as an effective tool in capturing the dynamic nature of software usage.

## 3.2 The Nature and Use of Simon's Model

It is natural when any methodology is used to ask the question, "Why is this the best approach to solving the problem?" The choice of applying Simon's Model of information usage is no exception to the rule. Therefore, the nature and origin of the model will be described followed by a justification of its use.

### 3.2.1 The Study of Empirical Phenomena

In our everyday lives it is natural that we observe human behavior and speculate as to why matters occur as they do. Many of these observations can be explained by empirical laws. According to Zunde (1984), we call a proposition of science an "empirical law" if "it contains only constructs that refer to observables or are operationally definable ... and laws have been extensively verified and found to hold under a variety of conditions."

#### 3.2.1.1 The Pareto Principle

An example of an empirical phenomenon is the Pareto Principle or 80/20 Rule which is the most often cited concentration measurement (Sanders, 1987). The basis of the 80/20 Rule is that items can be classified into two categories, "the significant few" and "the trivial many." The Pareto Principle is based on the research of the Italian

economist Vilfredo Pareto. In studying the concentration of wealth in his native country, he found that 80% of the wealth was concentrated in 20% of the population. This observation gave birth to the 80/20 rule or Pareto Principle (Flores et al., 1985). The phenomenon of the 80/20 rule can be observed in very numerous aspects of human activities. For example, research shows that 80% of library circulations involve only 20% of holdings. It has also been shown that elaborate studies for justification of computer equipment allocate 80% of budgeted funds for equipment accounting for less than 20% of total cost (Chen et al., 1993).

Application areas of the Pareto Principle have grown in number. For example, in order to obtain the major benefits of global data planning without having to invest in a full-scale strategic data planning process, Goodhue et al. (1988) proposed an 80/20 planning method that 80% of the benefits can be achieved with 20% of the total work. Boehm (1987) also saw use for the Pareto Principle in suggesting that software maintenance methodologies should aim at eliminating 20% of the high-risk problems which compose 80% of rework costs.

In viewing the diverse set of applications of the 80/20 Rule, it will be shown that this empirical phenomenon can be seen in the usage patterns of organizational software. We must also realize that the concept of the Pareto Principle

implies that the "significant few" and "trivial many" are observed in a variety of different concentrations. That is, the principle may reveal a distribution of 70/30, 85/16, 77/22, etc.

#### 3.2.1.2 Problems of Applying Empirical Phenomena in IS

A noted characteristic of the 80/20 Rule and other empirical phenomena is that they hold in general, and the assessment should only be used with careful inspection. Here we will discuss two inherent problems of the Pareto Principle.

The first problem of the Pareto Principle arises based on the fact that the distribution produced by the 80/20 Rule is only a photograph in time of a usually dynamic system. In the examples of the 80/20 Rule earlier stated, usage frequency was the criterion which divided the significant few from the trivial many. The distribution developed was correct at the time the count was taken and the analysis made, but it was not necessarily true over time. That is, the 80/20 Rule is static. It tells nothing about either the past or the future.

Based on this fact, the "trivial many" should not be eliminated because with some likelihood they could be tomorrow's "significant few." It takes a series of photographs over time to show how objects move in and out of critical status.

This important observation about the static nature of the 80/20 Rule serves as a foundation for much of the work of this dissertation. It naturally follows that the same problem continues to the applications where more than two categories of concentration are to be identified. That is, one may desire to observe three categories of usage concentration such as A, B and C categories. Regardless of the number of categories that are to be observed, the assessment is not insightful unless it captures the dynamic nature of the phenomena.

The second problem comes from the natural tendency to use the Pareto Principle to its extreme. It would seem intuitive to disregard the "trivial 80%" of problems and give total attention to the "significant 20%." This would certainly be abuse of the information gathered, and the effects of this abuse will now be discussed.

Let us first look at a scenario given by Sanders (1987). Suppose that there is a situation where the volume of sales from a sales force is examined. It is found that 20% of the salespeople are generating 73% of the sales, and 16% of products account for 85% of sales. In addition, 22% of customers are producing 77% of sales. This is not an unusual situation.

When the salesforce is examined, it is discovered that salesperson Black has 100 active accounts. Twenty of these

accounts produce 80% of Black's sales. Salesperson Green has 100 counties where 80% of the customers are concentrated in 24 counties. Salesperson White sells 30 different products. Six of these products account for 81% of White's sales. Given the above information described in the scenario, how could sales be improved? It would be intuitive to advise Black to concentrate efforts on the 20 accounts that are providing 80% of sales. White should concentrate on the six products that are returning over 80% of sales. Thus, the advice is to concentrate effort in productive areas.

Once the interrelationships between the number of accounts, their geographic location, profitability and so on are integrated, and once management has assessed the cost of serving the accounts, their profitability and further factors, there will emerge a new distribution. This new distribution will indicate that Black and his associates should sell a limited number of products in a smaller geographic area in order to maximize profitability for the company. Even after the most sophisticated analysis of sales territories, the 80/20 Rule will still apply, and management will still opt for heavy concentration on the few accounts rather that the many.

The point is that the information gathered by dividing items into segregated categories should not be carried to excess. A scenario of carrying the principle to excess would

be as follows. Black has 100 active accounts where only 20% produce 77% of sales. It could be decided to terminate Brown and have Black handle Brown's accounts. By the same strategy, sales people Green and White may also be eliminated. This gives Black about 100 accounts all contributing to 80% of sales in their territories.

Black is now working much harder in trying to handle these productive accounts. The problem is that Black cannot concentrate on these 100 accounts as well as he did before. The discovery is that Black's 100 new accounts still follow the 80/20 Rule.

This example is based on the nature of the 80/20 Rule. It has been shown that the 80/20 Rule gives us a type of single criterion categorization by usage. The point is that the Pareto Principle should not be used strictly as a rule for action. It is a rule based on empirical observation. This example has shown that a less than judicious application of the principle can result in disaster.

Another example showing the problem of applying empirical phenomena was cited by Nash (1992). Among the 10 most important systems development issues in 1990, only 5 remained in the top 10 in 1992. Of the five remaining in the top ten, their rankings were reordered. On the other hand, issues that ranked 19 and 17 in 1990 now ranked 10 and 9,

respectively, in 1992. Also, some of the top 10 system development issues in 1992 did not even exist in 1990.

Hence, to give validity to the proposition of this dissertation -- that the 80/20 Rule be used to classify software and address storage problems -- conditions must be found that would allow the classifications to be declared stable. Simon's approach to empirical modeling will serve as the stabilization of the methodology used in this dissertation. If the 80/20 Rule is applied in such a way that the dynamic nature of changing usage concentration can be captured, then scenarios like the ones stated above do not need to be a concern.

### 3.3 Simon's Approach to Empirical Modeling

According to Simon (1968), extreme hypotheses are "assertions that a particular specific functional relationship holds between the independent and the dependent variable." Building and testing a model of empirical phenomena is an example of extreme hypothesis. A standard practice for testing extreme hypotheses is the use of goodness-of-fit tests, and Simon argued that those testing procedures are fundamentally insufficient since "an extreme hypothesis cannot be sensibly identified with the null hypothesis without shifting completely the burden of proof that is supposed to be assured by a new theory, and what is worse, without making the tacit assumption that the

correctness of a theory is an all-or-none matter and not simply a matter of goodness of approximation."

In accordance with the beliefs stated above, Simon emphasized that the central preoccupation of a science should be the discovery of theories that are worth entertaining rather than simply testing them. That is, theories should be produced from data which is collected instead of data being collected to fit pre-existing theories.

Simon's alternative approach to hypothesis testing is stated here. According to Ijiri and Simon (1977), "We are interested in knowing what part of the variance of the data is explained by the theory, and how the remaining variance can be accounted for by successive approximations, rather than in testing whether the variance can be proved to be statistically significant."

Simon's approach, which follows, is a basis for his views of estimation. It is based on the belief that one cannot find phenomena that fit empirical data until one has approximate data that look as though a smooth mathematical function could generate them (Simon, 1991). Simon (1955) proposed a five-step modeling process which is listed below:

(1) Begin with empirical data, not hypothesis;

(2) Draw a simple generalization that approximately summarizes striking features of the data;

(3) Find limiting conditions under which deviations from a generalization are small;

(4) Construct simple mechanisms to explain the simple generalizations; and

(5) Propose the explanatory theories that go beyond simple generalizations and create experiments for empirical observations.

## 3.4 Simon's Basic Models

Based on his modeling process, Simon (1963) proposed a generating mechanism to simulate the information usage patterns. For the modeling of the changes of usage frequency in software applications, the following three versions of Simon's model are used to study the Pareto Principle and then to study concentrations involving more than two classes.

### 3.4.1 Version I: Constant Entry Rate of New Items

According to Simon (1955), the model of constant entry rate is based on two assumptions.

Assumption I: The probability that the (k+1)st information item accessed will be an information item that has not previously been accessed, is $\alpha(k)$; and

Assumption II: The probability that the (k+1)st information item accessed will be an information item that has already been accessed i times ($i \geq 1$) is proportional

to $i \cdot f(i,k)$, where $f(i,k)$ is the number of distinct information items that have been accessed exactly i times in the first k accesses.

The first assumption differentiates two classes of software items: (1) the ones that have not yet been classified in terms of classification because they are "new" software items and (2) those software items which are classified as "old." The parameter $\alpha$, therefore, determines whether software needs to be moved from the class of "non production" to the class of "current production." If the software is currently being used in production, the history of use in production is determined through the second assumption. The second assumption describes the fact that software which is already in use has a greater tendency to remain in use than software which has never been put into production.

### 3.4.2 Version II: Decreasing Entry of New Items

Simon and Van Wormer (1963) began by assuming that the parameter $\alpha$ in assumption I is a constant, thus independent of the number of accesses, k, that have taken place. They noted that the Version I model is only a simple generalization that approximately summarizes the striking features of the observed data. Therefore, this version I model of first approximation was further refined later on

into version II by modifying $\alpha$ to be a decreasing function $\alpha(k)$ i.e., there is a probability function $\alpha(k)$, where $0 \leq \alpha \leq 1$, $\alpha(k + 1) \leq \alpha(k)$, $k = 1, 2, 3, \ldots$, that the $(k + 1)$st accessing is to an information item used for the first time. The parameter $\alpha$ and functional form of $\alpha(k)$ in version I and II models are determined by the environment in which the selection process takes place; some environments encourage new selection while others may discourage them. For example, using a computer program as an example of an information item, a program may be developed but never put into production. Familiarity increases the use of programs. As a result of this, an unused program will likely remain unused.

### 3.4.3 Version III: The Autoregressive Growth of Old Items

The probability of usage for already-chosen information is assumed to be proportional to its previous usage in Simon's basic models. However, the information not used has a tendency of being "forgotten." In other words, the probability of the usage will decrease with time if an item is not used. From analyzing firm sizes which have the same skew-distribution as the information accessing, Simon refined his first two versions of models to better reflect reality (Ijiri and Simon, 1977). This model takes into consideration the recent usage when assigning probability of selection for

"used" information. Simon's autoregressive model is described below.

In terms of the use of information items or programs, the identity of each information item is maintained from one time period to the next. The selection of the next item is governed by a stochastic process, which depends on how much the item has been used before, and also upon when it was last selected. For simplicity in his computations, Simon assumed that only one item is selected at each time period. The probability that an item is chosen next is assumed proportional to a weighted sum of its usage history, where the weight of a usage decreases geometrically, at a rate $\gamma$, from the time it was last used. Since the time interval is functionally equivalent to the number of selections, we only indicate the number of selections in lieu of time interval in this dissertation.

Simon formalized these notations as follows: Let $y_j(k)$ denote our selection of jth information item during the kth selection, where $y_j(k)$ is either unity or zero. Then total usage of the jth item at the end of the kth selection is simply $\sum_{\tau=1}^{k} y_j(\tau)$. The expected usage of the jth item for the $(k+1)$st selection is $p[y_j(k+1)=1] = 1/W_k \sum_{\tau=1}^{k} y_j(\tau)\gamma^{k-\tau}$ where $W_k$ is a function of the total number of usages by the end of selection and is the same for all items, and $\gamma$ is the

fraction that determines how rapidly the influence of past usage on new selection dies out.

Simon selected a unit time interval in his version III model so that there is exactly one information item accessed per any one given time period. He further assumed that there is a constant probability, $\alpha$, that the selected one is a "new" information item. Simon's simulation results show a resemblance to frequencies exhibited in historical data, showing that his model provides considerable insight into how these empirical data are generated in their diverse environment.

It is valuable to note that version I of Simon's model is simply a special case of his version III model. If the parameter $\gamma$ is set to a constant 1, the weighted usage of each information item is equivalent to its accumulated number of uses. Thus the probability of it being chosen in the next selection process is proportional to its total number of uses so far, which is the second assumption in Simon's version I model.

## 3.5 Algorithms for Simon's Model of Usage

### 3.5.1 Three Significant Regions

It has been shown (Chen et al., 1993) that Simon's model provides a theoretical justification to Zipf's law and Lotka's law, which, in turn, describe the distribution in regions of the Pareto curve. At this point the regions of

the Pareto curve will be expressed as region I, region II, and region III. Region I is the region of the significant few. Region II is the middle class, and Region III is the region of the trivial many. There are limitations in what analytical studies can do because the analytical approach of Simon's model can only derive the "average behavior" of the Pareto curve.

Here it is important to realize that Simon's stochastic model also provides the computational experimentation for modeling the Pareto Principle (Chen, 1989). This approach allows us to go beyond the analytical methods to examine details as Simon's two assumptions are relaxed. It also enables us to study parameters under extreme conditions.

In this dissertation it is vitally important to begin computational experimentation with the Pareto Principle before extending the experimentation to consider modeling of concentrations in more than two groups.

### 3.5.2 Algorithms for Basic Models

The two basic versions of Simon's model can be easily programmed on a computer to simulate empirical data $(n_i, f(n_i))$, $i = 1, 2, \ldots, m$. Using notations earlier defined, the simulation algorithm consists of two steps (Simon and Van Wormer, 1963):

Step 1: For each kth selection, a random number, a is generated from the uniform distribution with

range 0 to 1. If $a \leq \alpha(k)$, $f(1,k) = f(1, k-1) + 1$ -- i.e., this is a "new" item and we add it to the "used once" category; otherwise we consider this item to be a "used" item and go to step 2 to find out how many times it has been used before.

Step 2: A random #, b, is drawn form the uniform distribution with range $1 \leq b \leq k$. Starting with $j = 1$, the cumulant of $j \cdot f(j,k-1)$ is computed and compared to b until an n is found such that $\sum_{j=1}^{n} j \cdot f(j,k-1) \geq b$. Then $f(n,k-1) - 1$, and $f(n+1, k) = f(n+1, k-1) + 1$. This is equivalent to saying that the kth item selected has been used n times already, now we move this item to the group where every item has been used n+1 times.

### 3.6 Algorithm for the Autoregressive Model

In the autoregressive model, the selection of the kth item is made in two stages:

Stage I: This stage determines whether this item has been selected previously. We first draw a random number, a, from a uniform distribution between 0 and 1. If $a \leq \alpha$, where $\alpha$ is a given constant, we add a "new" n(k)th item in our list of "used" items, where n(k) is the number of items that have been included in our list during the first k

selections/periods. We assign the kth usage to this new item, thus completing the selection process for the kth time period without going through stage II; otherwise go to stage II.

Stage II: For each item on our list, we keep track of two factors: the current usage of the item, and the current "growth potential" of the item usage. By the previous usage of the jth item at the end of the (k-1)th time period, $i_j(k-1)$, we mean the total number of times that the jth item has been selected up to that time. By growth potential of the item usage for the jth item at the end of the (k-1)th time period, $W_k(k-1)$, we mean a weighted sum of the previous selections for the item up to that time. The weights are assigned as follows: If we take as 1 the weight of the selection during period (k-1), then the weight of the selection during period (k-$\tau$) will be $\gamma^{(\tau-1)}$, where $\gamma$ is a proper fraction measuring the "decay" factor. That is, prior selection is assumed to create potential future use at a rate that falls off geometrically with the lapse of time since the prior selection occurred. We also keep track of the sum of the growth potentials for all items: $W(k-1) = \Sigma_j W_j(k-1)$. We draw another random number b from a uniform distribution between 0 and W(k-1), and assign the kth selection to the jth

item, where j is the smallest integer which satisfies the condition that the sum of the $\bar{w}_j(k-1) \geq b$. The result is that the probability that the selection will be assigned to any particular one of the existing items is proportional to the weight of that item.

### 3.7 Summary

In this chapter, the complexity of usage modeling has been discussed along with the difficulties in capturing certain empirical phenomena. The explanation Simon's approach to usage modeling, including the three versions of his model, gives insight into overall item usage behavior. The value of Simon's model in this research is in its adaptability to the modeling of program usage. Insight into the three versions of Simon's model provides us with a flexible tool for the modeling of program usage scenarios.

# CHAPTER 4 SIMULATION SET-UP

In this chapter, the general form of Simon's Autoregressive Model is applied in order to study the behavior of program usage in organizations. A simulation program was developed in order to generate usage patterns with varying parameters. The results show that patterns of usage can be predicted over time, thus capturing the dynamic nature of usage.

## 4.1 The Need for Computational Experimentation

It is necessary to begin the discussion of the experimentation performed by justifying the need for this type of work. Although research has shown that Simon's Model of usage provides a unifying model, there are limitations to what analytical studies accomplish. It is said that conventional analytical methods can only derive the "average behavior" of the distribution (Chen and Leimkuhler, 1986). Leimkuhler (1988) suggested that computational experimentation be used for modeling empirical phenomena, particularly in applications involving information systems design and problem solving. His theory was that computational experimentation enables researchers to study the distribution under "extreme conditions" and allows one to go beyond the analytical methods to examine details when the assumptions are relaxed (Leimkuhler, 1988; Neuts, 1986).

This is especially true when stochastic models admitting serial correlation have proved to be too complex to be solved explicitly in closed form for the equilibrium distribution (Ijiri and Simon, 1977).

The majority of the tools and software products available on the market today for storage are based on timing mechanisms. Very few incorporate actual program usage in making storage considerations. The incorporation of Simon's Version III Model makes the simulation experimentation imperative to this study. In order to capture the complexity of the dynamic behavior of Simon's Model of information accessing, simulation study is an appropriate measure.

**4.2 The Design of Experiment Used**

The first step in designing the simulation study to be performed is to develop a clear statement of the problem by soliciting information from concerned parties. As emphasized earlier, part of this research's purpose is to identify organizational issues of storage concern. Several organizations, considered to have large information systems departments, were interviewed. For privacy purposes these companies will not be named. Instead, it is sufficient to say that these companies included a national financial corporation, an international insurance company, two universities' information systems departments, and a power and light company. Questions were posed as to concerns for

cost in acquiring additional storage, costs and allocations for back-up and compression of files, and normal methodologies for collecting usage data. The insight given by these information systems departments gave a much clearer understanding of the phenomena occurring in these organizations. A common element in all cases was the tremendous cost incurred in acquisition of expanded storage capability. This issue of storage capacity has become more critical with the movement of many applications to network environments. Also, software applications running in Windows environments require much more storage capacity than previously required. The majority of daily production software is still run on mainframe systems. Therefore, the mounting number of programs and data files need to be kept in consideration regardless of the type of computer system used.

With the information gathered from the above mentioned organizations, the focus of this study was clear. There was a justification for examining the usage of programs in organizations. Usage patterns of software, like most items with usage concentrations, will certainly indicate the phenomena of the 80/20 Rule. That is, it is intuitive that a small number of programs are those that are used the majority of the time. Likewise, the vast majority of programs are used quite infrequently. It makes sense to give storage priority to those programs with the highest usage.

It is arguable that priority should be given based on criteria other than usage such as criticality, cost of development, user satisfaction, etc. For the sake of availability of criteria measures, only usage will be used in this study. All other criteria would require subjective estimation, and the purpose of the experimentation is to compare simulated results to actual data.

### 4.2.1 The Index Approach

At this point it is necessary to describe the index approach which was proposed by Chen and Leimkuhler (1986). This index approach forms the basis for the expression of simulated results and therefore warrants explanation.

In Kendall's (1960) study and 1763 papers published on operational research, he gave us the basis for the index approach. If the number of authors who have published n papers is computed, and the list arranged in ascending order of n, it would be found that n does not run consecutively at places. This is especially true when n is large. Also, it would be found that there are m different clusters of authors who publish the same number of papers, and $m \leq \max\{n\}$. To take into account the scatter of the larger values of n, Chen and Leimkuhler (1986) introduced an index $i = 1, 2, \ldots, m$, for the m successive observations of n and let $n_i$ denote the i-th nonzero value of n where $n_i < n_{i+1}$.

In order to apply the index approach to information usage, it is necessary to define the terms which are used in several different contexts of the approach.

$m$ = the maximum number of clusters of items with the same usage;

$n_i$ = the number of times an information item is used, $i = 1, 2, \ldots, m$;

$f(n_i)$ = the number of items used $n_i$ times;

$F(n_i)$ = the number of items used no less than $n_i$ times;

$r_i = \sum_{k=m-i+1}^{m} f(n_k)$ = the rank of item i ranked according to its usage;

$g(r_i) = n_{m-i+1}$ = the number of times an item with rank $r_i$ was used;

$G(r_i) = \sum_{k=m-i+1}^{m} n_k f(n_k)$ = the total number of usages for items ranking no greater than $r_i$;

$T = \sum_{i=1}^{m} f(n_i)$ = the total number of different items;

$N = \sum_{i=1}^{m} n_i f(n_i)$ = the total number of usages; and

$x_i = r_i/T$ = the fraction of total usage for the first i items;

$\theta_i = G(r_i)/N$ = the fraction of total usage for the first i items;

$\mu = N/T$ = the average usage per item.

Kendall's (1960) data is used to construct **Table 4.1** from raw data showing the significance of the aforementioned parameters.

For the purposes of this research we will be referring to computed values of these parameters used in the simulation model. Previous research by Chong (1993) used this data to explain several empirical phenomena. At this point we are mostly concerned with investigating the 80/20 Rule to show concentrations of usage for software items. Therefore, we use **Figure 4.1** in order to represent the 80/20 Rule. In Figure 4.1, $x_i$ and $\theta_i$ are plotted. The result shows that 80% of total usage was obtained with only approximately 26% of the total items. It is natural that we are more interested in those few items that are frequently used. At the same time, we cannot disregard the large number of items used infrequently; for, with some likelihood they could become part of the class of the frequently used. In the next section, the steps taken in order to capture this usage phenomenon are given.

### 4.2.2 The Design of the Simulation Program

As previously mentioned, Simon's model in its three versions can be incorporated into a computer simulation program. Therefore, the computer programs developed to

**Table 4.1 Empirical Laws Using Kendall's Data**

| i | $n_i$ | $f(n_i)$ | $n_i f(n_i)$ | $r_i$ | $G(r_i)$ | $x_i$ | $\theta_i$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 203 | 203 | 1 | 242 | 0.003 | 0.137 |
| 2 | 2 | 54 | 108 | 2 | 356 | 0.005 | 0.202 |
| 3 | 3 | 29 | 87 | 3 | 458 | 0.008 | 0.260 |
| 4 | 4 | 17 | 68 | 4 | 553 | 0.011 | 0.314 |
| 5 | 5 | 10 | 50 | 5 | 611 | 0.014 | 0.347 |
| 6 | 6 | 6 | 36 | 6 | 660 | 0.016 | 0.374 |
| 7 | 7 | 8 | 56 | 7 | 694 | 0.019 | 0.394 |
| 8 | 8 | 8 | 64 | 9 | 738 | 0.024 | 0.419 |
| 9 | 9 | 4 | 36 | 11 | 780 | 0.030 | 0.442 |
| 10 | 10 | 3 | 30 | 13 | 820 | 0.035 | 0.465 |
| 11 | 11 | 5 | 55 | 14 | 838 | 0.038 | 0.475 |
| 12 | 12 | 2 | 24 | 18 | 902 | 0.049 | 0.512 |
| 13 | 14 | 1 | 14 | 20 | 932 | 0.054 | 0.529 |
| 14 | 15 | 2 | 30 | 21 | 946 | 0.057 | 0.537 |
| 15 | 16 | 4 | 64 | 23 | 970 | 0.062 | 0.550 |
| 16 | 18 | 1 | 18 | 28 | 1025 | 0.076 | 0.581 |
| 17 | 20 | 2 | 40 | 31 | 1055 | 0.084 | 0.598 |
| 18 | 21 | 2 | 42 | 35 | 1091 | 0.095 | 0.619 |
| 19 | 22 | 2 | 44 | 43 | 1155 | 0.116 | 0.655 |
| 20 | 34 | 1 | 34 | 51 | 1211 | 0.138 | 0.687 |
| 21 | 49 | 1 | 49 | 57 | 1247 | 0.154 | 0.707 |
| 22 | 58 | 1 | 58 | 67 | 1297 | 0.181 | 0.736 |
| 23 | 95 | 1 | 95 | 84 | 1365 | 0.227 | 0.774 |
| 24 | 102 | 1 | 102 | 113 | 1452 | 0.305 | 0.824 |
| 25 | 114 | 1 | 114 | 167 | 1560 | 0.451 | 0.885 |
| 26 | 242 | 1 | 242 | 370 | 1763 | 1.000 | 1.000 |
| | | --- | ----- | | | | |
| m=26 | | T=370 | N=1763 | $\mu$=4.7649 | | | |

80/20: $\theta_i$ vs. $x_i$; % cumulative transactions vs. % cumulative holdings; $x_i$ is $r_i/T$ and $\theta_i$ is $G(r_i)/N$.

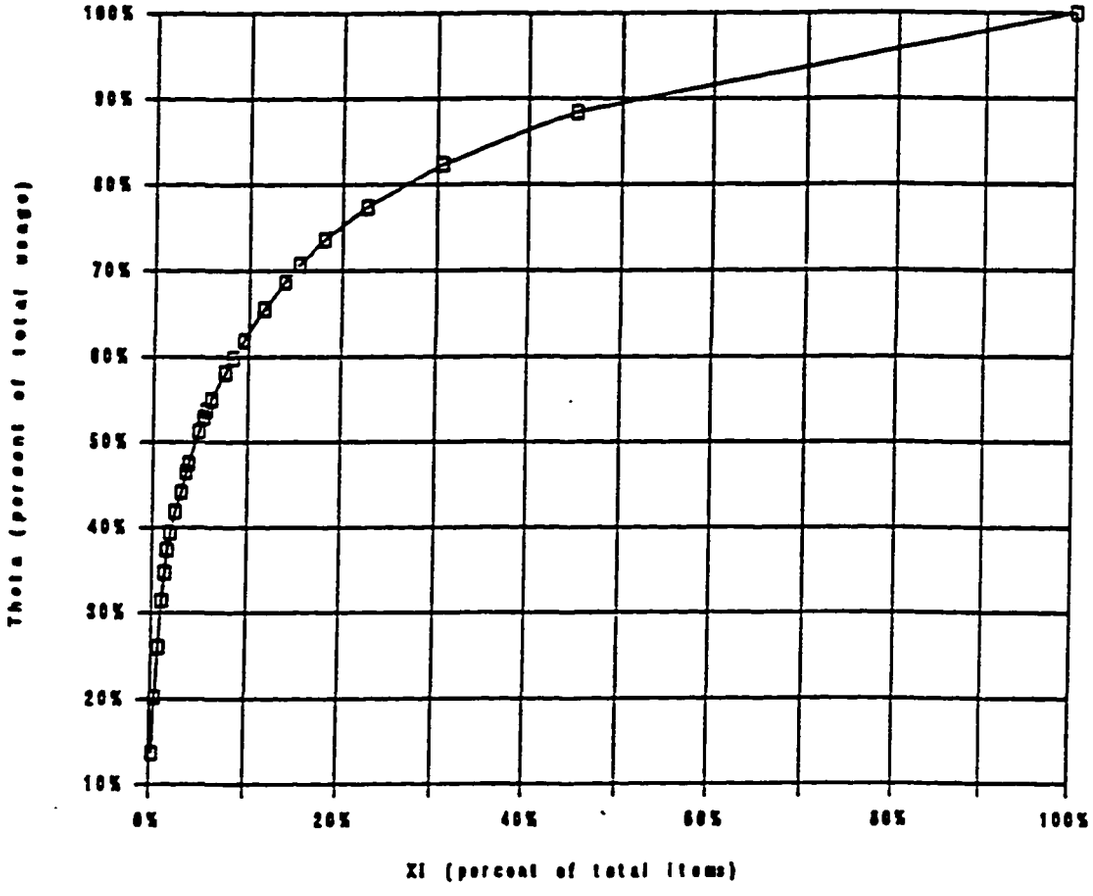


**Figure 4.1 80/20 Rule Using Kendall's Data**

perform the necessary simulation study were written in Turbo Pascal. The actual programs written for this dissertation were based on the previous programs written by Chong (1993) and Tong (1994). Considerable changes were necessary in order to make the earlier programs appropriate for the experimentation of this dissertation.

The previous programs were written in VS Pascal Release 2. The basic system used was an IBM mainframe under VM/CMS. There was a significant amount of CPU time needed, and the jobs were also executed in batch mode under MVS/ESA. Data from running the VS Pascal programs were downloaded into spreadsheet files. Much effort was taken in this dissertation in order to give the programs the ability to run on a personal computer. In the research of Chong (1993) and Tong (1994) the necessary simulation programs were not capable of running on a personal computer to achieve desired results. The arrays produced in the simulation quickly expended the primary memory allocated. Therefore, the simulation program for Simon's Model was written in this dissertation which made use of dynamic arrays. In using a dynamic array of pointers and a record structure to maximize key arrays, the availability of primary memory could be manipulated.

The original program developed by Tong (1993) was designed in order to generate desired usage for the three

versions of Simon's Model. The index approach was used in order to obtain m classes of items with different usage frequencies. Additional changes were necessary in order to enhance the simulation model into one that would give more than a static view of program usages. One of the major contributions added by this dissertation was the addition of cyclic evaluations in the simulation program. Previously, a simulation program for Simon's Model was generated in order to produce a distribution of usages for items after a given interval. The enhanced program generated in this dissertation added the ability to show the dynamic progression of the usages of individual programs. In the previous program it was impossible to keep a history of the distributions produced by individual items. The only capability was to produce a distribution at the end of simulation runs of varying lengths. There was no ability to track those same items in successive runs since identification numbers are randomly assigned to items at the beginning of each run.

The result of the revised program added a great deal of insight into the movement of software items throughout a desired number of usages. We now have the ability to see the changing rates at which programs' usages decline and increase in Simon's Version III Model. Actual simulation results are

given in Chapters 5 and 6. In **Appendix A**, a listing of the revised program can be seen.

Now we will discuss the necessary requirements for incorporating Simon's Model into a simulation program. The task of design began with the development of Simon's Version III Model. This model is the most applicable to the research being conducted, and also the most difficult to program.

Since stage two of the algorithm of Simon's Version III Model requires that some "used" records exist, we assign an initial condition that there are three records in a list at the beginning and each initial record has the same probability of being accessed in the next selection process (all three have weighted sum of accesses equal to 1 initially). The initial condition of records can be easily changed in the simulation program. This allows us to study the same phenomenon under different initial settings. However, computational experimentation has shown in general similar results under different initial conditions. This is consistent with Simon's original observations. Therefore, the general conclusion from our initial examination is that moderate changes in the initial conditions of list status do not appear to affect the equilibrium status (Simon and Van Wormer, 1963).

To reduce the variance of simulation results, different seed values of the random number generator can be selected

during each run. Also, runs of differing lengths were performed for a different number of cycles. A cycle is defined as the interval at which time the distribution of individual program usage is evaluated. Also, the program requires the input of the number of usages to be included in each cycle. Hence, a program run for 6 cycles of 20,000 usages implies that an assessment of program usage be given six times at intervals of 20,000 usages for a total of 120,000 usages. It was found in testing the program that the observations are less affected by the initial conditions if the total number of usages at the end of the desired number of cycles is at least 20,000.

### 4.2.3 The Choice of Factors and Levels

In testing the accuracy of the simulation program, it was necessary to choose the factors to be varied in the study and the levels at which the simulation runs would be made. The factors of interest in the simulation results are dependent on the values of $\alpha$ and $\gamma$. Therefore, initial simulation runs were performed for varying values of $\alpha$ and $\gamma$ where in each run the number of usages was held at N = 20,000. Previous test results showed that the area of the 80/20 curves is not affected by the total number of usages beyond this point.

In the simulation runs covering the range of possible values for $\alpha$ and $\gamma$, the results were compared. A summary of

the results is shown in **Table 4.2**. The simulation runs were performed ranging $\alpha$ in intervals of 0.10 starting at 0.10. For each value of $\alpha$, the value of $\gamma$ was also ranged in increments of 0.10. This test was done to simply to check for programming errors and consistency of results. Table 4.2 shows selected values over the testing interval in order to show differences in pairings of $\alpha$ and $\gamma$. At all levels it was interesting to observe the concentration of the 80/20 Rule and the number of programs produced.

It was also necessary to test program results for a varying number of cycles. The concentration of the 80/20 Rule should remain consistent regardless of the number of cycles being observed as the total number of programs increases. **Table 4.3** shows that the concentration of the 80/20 Rule did remain almost constant throughout successive cycles.

In designing this experiment, we were aware that varying values of $\alpha$ and $\gamma$ would produce different values for the response. Consequently, we are interested in identifying specific magnitudes of change and verifying ranges of uniformity. Therefore, a key focus of this study will be inspection for ranges of parameters in which "interesting" results occur. We know that a higher $\gamma$ means a slower rate

**Table 4.2 Summary of Simulation Observations**

| ALPHA | GAMMA | AREA | # OF PROGRAMS |
|---|---|---|---|
| 0.10 | 0.10 | 0.2593 | 643 |
| 0.10 | 0.50 | 0.3247 | 643 |
| 0.10 | 0.99 | 0.3747 | 643 |
| 0.20 | 0.10 | 0.2390 | 1193 |
| 0.20 | 0.50 | 0.2898 | 1193 |
| 0.20 | 0.99 | 0.3232 | 1193 |
| 0.50 | 0.10 | 0.1741 | 2977 |
| 0.50 | 0.50 | 0.1893 | 2977 |
| 0.50 | 0.99 | 0.1958 | 2977 |
| 0.99 | 0.10 | 0.0048 | 5942 |
| 0.99 | 0.50 | 0.0048 | 5942 |
| 0.99 | 0.99 | 0.0048 | 5942 |

** Note: In this table we show various areas of concentration of the 80/20 Rule for different combinations of $\alpha$ and $\gamma$. Also displayed are the number of programs which were produced from simulation runs each having 20,000 total usages.

**Table 4.3 Areas of Concentration and Total # of Programs**

**Area of Concentration of the 80/20 Rule For Cycle N**

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| .3102 | .3101 | .3125 | .3126 | .3152 |
| 6 | 7 | 8 | 9 | 10 |
| .3143 | .3136 | .3130 | .3118 | .3122 |

**Total Number of Programs After N Cycles**

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 1193 | 2420 | 3603 | 4835 | 5998 |
| 6 | 7 | 8 | 9 | 10 |
| 7191 | 8396 | 9601 | 10817 | 12001 |

of decay; i.e., a program's usage probability is little affected by its not being used. Thus, when $\gamma = 1.0$, there is no decay taking place at all, and the result should be identical to those of the version I of Simon's Model. On the other hand, a small $\gamma$ signifies that items that have not been used recently will possibly be neglected for a long to come, regardless how active they had been previously. A related interpretation is that previously inactive items do have a chance to become dominant in the future selection process even if this dominance is temporary. Therefore, it is plausible that a lower $\gamma$ should induce less concentration in its usage pattern.

Since $\gamma < 1$ reduces the usage concentration, and under most circumstances it is not plausible that $\gamma > 1.0$, (the lack of usage of an item increases the probability if its selection) it is intuitive that Simon's basic model with constant $\alpha$ provides the maximum concentration for a given $\alpha$. A lower value of $\gamma$ will decrease concentration. It was shown in the previous research of Chong (1993) that the 80/20 Rule held true if $\alpha \leq 0.20$. Since a lower $\gamma$ decreases concentration, it is plausible that a combination of $\alpha < 0.20$ and some proper $\gamma$ can also achieve the 80/20 Rule.

The aforementioned relationship between $\alpha$ and $\gamma$ shows that their interaction can produce observable results. For this reason we will attempt to identify other significant

behavior of combinations of $\alpha$ and $\gamma$. In **Table 4.4**, we observe the number of original items which remain in the top 50 ranking after 10 cycles of usage 1,000. In this table, $\alpha$ = 0.20, and $\gamma$ = 0.80. These observations are a foundation for future experimentation in Chapter 5.

## 4.3 Summary

In this chapter, we have discussed Simon's view for the need for computational experimentation in the study of information usage. The nature of Simon's model has permitted the development of the basic simulation program which will be used throughout the remainder of the dissertation. The design of this simulation program, which is unique to this dissertation, allows for the incremental capture of changing usage concentrations. The insight gained on the behavior of Simon's Autoregressive Model serves as the foundation for possible variations. The flexibility of Simon's model will permit the study of scenarios which are inherent to program usage. Thus, we will extend Simon's three versions by changing the functional forms of the entry of "new" programs and the decay of "old" programs.

**Table 4.4 Items Remaining in Top 50 After N Cycles**

| NEW ITEMS GENERATED IN CYCLE N | # OF ITEMS REMAINING IN TOP 50 AFTER N CYCLES | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | N=1 | N=2 | N=3 | N=4 | N=5 | N=6 | N=7 | N=8 | N=9 | N=10 |
| 1 | 50 | 23 | 13 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| 2 | | 27 | 18 | 9 | 7 | 5 | 4 | 4 | 3 | 2 |
| 3 | | | 19 | 13 | 10 | 9 | 7 | 6 | 6 | 3 |
| 4 | | | | 18 | 7 | 7 | 7 | 5 | 5 | 3 |
| 5 | | | | | 17 | 11 | 10 | 9 | 9 | 6 |
| 6 | | | | | | 10 | 9 | 9 | 8 | 6 |
| 7 | | | | | | | 6 | 5 | 5 | 5 |
| 8 | | | | | | | | 6 | 4 | 3 |
| 9 | | | | | | | | | 5 | 4 |
| 10 | | | | | | | | | | 5 |

# CHAPTER 5 MODEL VALIDATION AND SIMULATION RESULTS

## 5.1 Introduction

The purpose of this section is to validate the predictive power of Simon's model. An explanation of the use of Simon's Model in generating skew distributions was given in Chapter 3 of this dissertation. Here the appropriateness of the model is further demonstrated by comparing simulated results to real data.

## 5.2 The Need for Model Validation

One of the key contributions of this research has been the addition of a dynamic dimension to the use of Simon's model. In previous research (Chong, 1993; Tong, 1994) the program implementation of Simon's model did not capture the progression of change which occurs as the model is run for a desired number of usages. The previous value of the model was to capture the distribution of item usage at the end of a desired number of usages. Much work has been done in order to adapt the model to one that tracks the progression of individual software items at any number of intervals. That is, this research has given the ability to view an individual program's movement in ranking position throughout successive intervals of usage. Hence, changes in usage for individual programs can be examined; and we are able to predict these

changes through analysis of progressive program usage rankings.

In order to alleviate any skepticism as to the accuracy of the simulation results, a study of comparison was done with the program usage data of an actual organization. Before discussion of the data collection process is done, it is necessary to explain why other traditional methods of forecasting could not be applied to achieve the desired results.

### 5.2.1 The Inappropriateness of Other Forecasting Methods

If we desire to provide some form of predictive model for software usage, one might naturally suggest the use of a traditional method of business forecasting. There are several methods of forecasting which can immediately be deemed inappropriate due to their nature. For example, regression analysis presumes a relationship between an independent and dependent variable. For software, our desire is simply to predict usage. Usage cannot be shown to have any linear or curvelinear relationship with another variable.

We also cannot employ the method of time series analysis. In time series analysis data points are plotted in order to view trends and determine any cyclical or seasonal components to the data. The goal would be to make the data stationary and make a transformation to obtain "white noise." There is no way to isolate all possible trends for a

collection of thousands of programs. Time series analysis depends on the assumption that there are regular and repeating components interacting to produce a total series. Time series techniques are appropriate for forecasting variables that fluctuate in some stable pattern over time. In addition, software usage does not necessarily have observable characteristics which occur as a function of time. Instead, each individual program has a usage pattern dictated by its varying criteria.

If another method of forecasting could be suggested, it might be a method such as exponential smoothing. Exponential smoothing is a procedure for continually revising a forecast in the light of more recent experience. The smoothing constant A serves as the weighting factor. The actual value of A determines the extent to which the most current observation is to influence the forecasted value. When A is close to 1, the new forecast will include a substantial adjustment for any error that occurred in the preceding forecast. Conversely, when A is close to 0, the new forecast will be very similar to the old one. Therefore, the value assigned to A is the key to the analysis. One method of estimating A is an iterative procedure that minimizes the mean squared errors. The value of A producing the smallest error for a range of A values is chosen for use in generating

future forecasts. It is possible to adjust exponential smoothing for trends and seasonal variation.

Although exponential smoothing gives more flexibility than other forecasting techniques in estimating a changing variable such as usage, it is still not an extremely accurate measure. In fact, it is usually not as accurate as more sophisticated methods such as autoregressive techniques. Although exponential smoothing is applied to forecast large inventories, there is much effort involved in constantly adjusting weighting factors.

Another point which cannot be overlooked in evaluating traditional forecasting methods, is their inability to capture the overall behavior of a large collection of items with individual usages. Simon's model is the only mechanism which is based on an assessment of the full collection of data and provides the additional ability to track individual item usage. Exponential smoothing does not have the explanatory power of Simon's Model. Thus, we use Simon's Model which does not depend on a subjective weighting process and captures usage trends.

### 5.2.2 The Impact of Simon's Model

We will now show the effectiveness of Simon's model by displaying a portion of simulated results. In **Table 5.1** a listing of the results obtained through a simulation run are shown. In each column, a list of values is displayed. Each

**Table 5.1 - Top 50 Program Rankings For 10 Cycles**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 138 | 1907 | 2653 | 2653 | 2653 | 2653 | 2653 | 2653 | 2653 | 2653 |
| 62 | 138 | 2825 | 2825 | 2825 | 2825 | 2825 | 2825 | 2825 | 2825 |
| 973 | 2131 | 1907 | 3751 | 5086 | 5086 | 5086 | 5086 | 5086 | 5086 |
| 853 | 62 | 138 | 3859 | 5942 | 5942 | 5942 | 5942 | 5942 | 5942 |
| 41 | 973 | 2908 | 1907 | 3751 | 3751 | 3751 | 3751 | 3751 | 3751 |
| 886 | 853 | 2131 | 138 | 3859 | 3859 | 3859 | 3859 | 3859 | 3859 |
| 386 | 2058 | 3011 | 2908 | 1907 | 1907 | 1907 | 1907 | 1907 | 1907 |
| 518 | 41 | 62 | 4614 | 138 | 138 | 138 | 138 | 138 | 138 |
| 323 | 1858 | 973 | 2131 | 2908 | 2908 | 2908 | 9555 | 9555 | 9555 |
| 787 | 886 | 853 | 3011 | 5226 | 6981 | 6981 | 6981 | 6981 | 11257 |
| 591 | 386 | 3099 | 62 | 5018 | 5018 | 5018 | 2908 | 2908 | 2908 |
| 773 | 518 | 41 | 973 | 4614 | 5226 | 5226 | 5226 | 5226 | 6981 |
| 460 | 323 | 2058 | 4001 | 2131 | 4614 | 4614 | 5018 | 5018 | 5018 |
| 1028 | 1594 | 2613 | 853 | 3011 | 2131 | 2131 | 4614 | 4614 | 5226 |
| 681 | 2269 | 3551 | 3099 | 62 | 6186 | 7339 | 7339 | 1339 | 4614 |
| 750 | 1416 | 2526 | 41 | 5687 | 6409 | 6186 | 2131 | 2131 | 11707 |
| 874 | 1903 | 1858 | 2058 | 973 | 3011 | 3011 | 6186 | 6186 | 7339 |
| 636 | 787 | 886 | 2613 | 5566 | 6135 | 6409 | 6409 | 6409 | 2131 |
| 1062 | 2165 | 386 | 3551 | 4001 | 62 | 6135 | 3011 | 3011 | 6186 |
| 603 | 2027 | 518 | 2526 | 853 | 6500 | 7924 | 6135 | 6135 | 11229 |
| 887 | 1614 | 323 | 4380 | 4976 | 6244 | 8078 | 8078 | 10509 | 6409 |
| 699 | 591 | 1594 | 1858 | 3099 | 7046 | 8236 | 8236 | 8236 | 3011 |
| 1081 | 773 | 2269 | 886 | 41 | 5687 | 62 | 7924 | 7924 | 10884 |
| 1160 | 1335 | 2474 | 4290 | 2058 | 5566 | 6500 | 6500 | 10043 | 6135 |
| 352 | 1877 | 1416 | 386 | 5370 | 973 | 6244 | 6244 | 8078 | 10509 |
| 1013 | 1540 | 2412 | 3941 | 2613 | 853 | 8200 | 8200 | 8200 | 8236 |
| 214 | 1961 | 2583 | 3817 | 3551 | 4976 | 7046 | 7046 | 7046 | 7924 |
| 502 | 1364 | 3065 | 4459 | 1858 | 4001 | 5687 | 9037 | 9037 | 10043 |
| 118 | 460 | 2874 | 323 | 2526 | 3099 | 5566 | 62 | 62 | 8078 |
| 867 | 1872 | 1903 | 3832 | 4388 | 2058 | 7373 | 5687 | 9809 | 11874 |
| 262 | 2193 | 2541 | 518 | 886 | 5370 | 973 | 973 | 6500 | 6500 |
| 614 | 1489 | 787 | 1594 | 5462 | 41 | 853 | 5566 | 6244 | 6244 |
| 332 | 1028 | 2165 | 2269 | 4290 | 3551 | 4976 | 7373 | 5687 | 8200 |
| 657 | 2080 | 2027 | 1416 | 386 | 2613 | 4001 | 4001 | 973 | 7046 |
| 746 | 750 | 3036 | 2412 | 3941 | 2526 | 3099 | 853 | 5566 | 9037 |
| 1149 | 1231 | 773 | 2474 | 3817 | 4388 | 2058 | 4976 | 7373 | 62 |
| 529 | 681 | 1614 | 2583 | 4459 | 1858 | 5370 | 3099 | 4001 | 9809 |
| 276 | 2239 | 591 | 3065 | 323 | 4290 | 41 | 8810 | 853 | 5687 |
| 236 | 1651 | 877 | 2541 | 3832 | 386 | 3551 | 2058 | 4976 | 1373 |
| 114 | 1062 | 1540 | 3740 | 518 | 3941 | 2613 | 5370 | 8810 | 973 |
| 679 | 874 | 1335 | 2874 | 2269 | 886 | 2526 | 41 | 3099 | 5566 |
| 995 | 636 | 2790 | 4569 | 5838 | 5462 | 4388 | 2613 | 2058 | 4976 |
| 589 | 1586 | 1961 | 1903 | 1594 | 3817 | 1858 | 9119 | 5370 | 4001 |
| 508 | 603 | 1364 | 4442 | 5976 | 4459 | 386 | 3551 | 41 | 853 |
| 1107 | 887 | 460 | 787 | 5754 | 3832 | 3941 | 4388 | 2613 | 3099 |
| 442 | 699 | 1822 | 2027 | 2474 | 518 | 886 | 1858 | 9119 | 8810 |
| 302 | 1286 | 2560 | 3598 | 1416 | 323 | 5462 | 2526 | 3551 | 5370 |
| 1036 | 1081 | 1489 | 2165 | 2412 | 2269 | 4290 | 4290 | 4388 | 41 |
| 244 | 61 | 2495 | 1632 | 2128 | 224 | 3129 | 6087 | 7058 | 9002 |
| 256 | 1632 | 22 | 2008 | 5665 | 3487 | 7008 | 5251 | 582 | 2613 |

of these values represents the identification number of a program. Values within a column are arranged in rank order. That is, the first value in a column has the highest number of usages, the second has the second highest number of usages and so on. Successive columns show the movement of programs' usages at designated intervals. These intervals are established through keyboard input of the number of usages in a cycle. If a cycle is declared at five thousand usages, each column would then show the movement of programs after an interval of five thousand usages. For the example in Table 5.1 we can see that some programs hold their ranking positions through successive cycles while other programs' rankings decline sharply. In this particular example, we used a $\alpha = .20$ and $\gamma = .80$ in order to simulate the data.

In addition to the ability to track movement, the simulation program is also designed to use the index approach in order to keep a frequency distribution for program usage. A frequency distribution provides an easy way to identify programs of significant use. Simulation results also provide the concentration of the 80/20 rule. At the end of each cycle, the results show the percentage of programs which comprised 80 percent of the total program usage. It is worth noting here that the simulation program was revised in order to show a variety of usage concentrations such as 70/20/10. Varying usage concentrations will be used in later results.

## 5.3 The Collection of Data

In order to validate the predictive power of the dynamic version of Simon's model, it was necessary to find an organization which kept program usage data. Many organizations were approached, but most had no history of usage data. Fortunately, the necessary data was provided by a national financial corporation. For privacy purposes, the name of this firm will not be mentioned in this dissertation.

The organization was able to provide program usage data from their mainframe system. Their mainframe is a UNISYS computer with an operating system that produces daily logs. A sample of this data is shown in **Appendix B**.

In searching for program usage data, it was highly desirable to find data reflecting Local Area Network (LAN) usage. LAN usage data was not available from this organization. In fact, there were no organizations found that produced any type of log for LAN use. It is also necessary to add that the organization used was the only one found that kept any history of their daily usage logs.

The data shown in Appendix B is a small excerpt from a daily report. It was unfortunate that this data was not saved on the system. Therefore, the task of compiling the program usage data into a meaningful form was a quite arduous one.

The purpose in collecting this data was to have the ability to estimate the two Simon's model parameters, $\alpha$ and $\gamma$, for this organization. Much effort was taken in order to compile the firm's data into a frequency distribution.

When the search for this data began, the goal was to collect a large collection of data for consecutive usages. As mentioned earlier, all other organizations solicited kept no history of usage. The firm providing the data had a history of approximately thirteen months of program usage. The initial request was for two years of data, but a history of thirteen months was sufficient. This organization has a large program usage volume. Therefore, the firm being used provided more data than an average usage organization possessing a larger time frame of historical data. The purpose of the data collection was to perform the following tasks.

1) Use six months of data in order to estimate the values of $\alpha$ and $\gamma$ for this organization.
2) Use those values of $\alpha$ and $\gamma$ to run a simulation study showing the progression of program usage status over future program usages.
3) Compare the simulated results to the remaining five months of data, and hopefully show meaningful correlations.

Hence, this phase of data collection was very critical in testing the predictive power of our simulation model.

## 5.4 The Estimation of Model Parameters

### 5.4.1 Introduction

In this section the processes of estimating the critical values needed for Simon's model are described. Here the necessary assumptions described in Section 3.4.1 are restated thusly:

1) There is a constant probability, $\alpha$, that the (t + 1)st program used will be a new program that has not been used in the first t usages.

2) The probability that the (t+1)st program usage is one that has been used n times is proportional to $n \cdot f(n,t)$, where $f(n,t)$ is the number of distinct programs that have been used exactly n times in the first t program usages.

Therefore, the simulation mechanism is still valid in generating the usage patterns. Furthermore, the parameter of $\gamma$ in the autoregressive model represents the aging/decay factor as described by researchers (Kent, 1979 and Burrell, 1982).

### 5.4.2 The Organizational Data

In collecting the program usage data, the following fields were available: (1) average I/O time, (2) average memory, (3) size of code and data, and (4) usage count. Of this raw data, the only field necessary for validation of

Simon's model was the usage count. Each program name was listed with its usage count. It is necessary to note that these programs were comprised of those run on-line and in batch form. Due to this fact some program listings could be considered as subroutines or procedures called by other programs. Without the ability to make absolute distinctions in all cases, each item listing a usage count was considered as a program.

#### 5.4.2.1 Estimating N

As stated in Section 5.3, the first task that was necessary to be performed was the computation of the total number of program usages during the first available six months of data available. The daily logs provided usage totals, and summing these daily usage totals for six months gave N = 221,895 program usages. If we were interested in computing a usage count on an annual basis, N would be multiplied by 2. This would give us an average annual usage of N = 443,790. It was sufficient for the purposes of this study to use the actual N = 221,895 for six months of data.

#### 5.4.2.2 Estimating $\alpha$

In previous research performed by Chong (1993), techniques for estimating $\alpha$ were performed. Remember that in computing $\alpha$, the probability of a new entry, we are calculating the slope between points $X_{m-1}$ and $\theta_{m-1}$. It was proved in previous research that the value of $S_m = (1 - \theta_{m-1}/1$

- $X_{m-1}$) = 1/ $\mu$, where $\mu$ = N/T. Thus the value of $\alpha$ = T/N, where and T is the total number of different items and N is the total number of usages.

For the data collected in this study, T = 22,599 (the total number of programs) and N = 221,895 (the total number of program usages). Thus, T/N = 0.10185 ≈ 0.102. The computed value of T/N is approximately equal to the value of $\alpha$. The ability to approximate the value of $\alpha$ from T/N is an intuitive relationship. It is known that $\alpha$ is the probability of the entry of a new program. Therefore, the total number of program usages should be the product of N and $\alpha$.

#### 5.4.2.3 Estimating $\gamma$

The methodology used for computing the value of $\gamma$ for the organizational data is also the same as that utilized by Chong (1993). The process for estimating this parameter involved running several simulations with various values of $\gamma$ while holding N = 221,895 and $\alpha$ = 0.102. From these results 80/20 curves were generated. The usage concentrations at various levels of $\gamma$ were compared to the concentration observed from the collected data. It was evident that the appropriate value of $\gamma$ should be in excess of 0.999. Previous research in computing $\gamma$ for real data has shown that $s_1$ is not as effective in determining $\gamma$ as $s_m$ is in determining $\alpha$. With this in mind, we use $\log(s_1)$ to

assist us in estimating $\gamma$. It was necessary to run the simulation model for various values with $\gamma$ in excess of 0.999 while holding $\alpha$ and N at the constant computed rates. Since our $n_m$ = 21,426 and $\mu$ = 9.82, it was necessary to perform successive runs with varying values of $\gamma$. This methodology permitted successive computations of $s_1 = n_m/\mu$ in order to isolate the appropriate value of $\log(s_1)$. It was determined that the desired value of $\gamma$ would be one producing $\log(s_1)$ = 3.339. Hence, the value of $\gamma$ for the collected six months of program usages was found to be 0.99934.

### 5.5 Simulation Results

The next task at hand was to run the simulation model for a period of time and compare the simulated program usage to the actual usage results of the remaining data. In the following two sections, we discuss observations from both the real data and the simulated data.

#### 5.5.1 Findings from Real Data

In evaluating the second portion of the real data, there were several characteristics worthy of observation. The characteristics observed were as follows:

1) The concentration of the 80/20 rule
2) The number of programs whose usage remained in top ranking position throughout the total number of usages.
3) The movement in rank of items in the top 20 percent of usage.

The evaluations of these characteristics are presented in this section. The data being evaluated was examined over a period of four months. The organization providing the daily logs of program usage had two more months of data available beyond the four months, but there was a portion missing from the fifth month. In order to maintain integrity in continuous usage, these last two months were not used.

**Table 5.3** shows us the program usage patterns derived from the collected data. This table was constructed by means of a spreadsheet package. A spreadsheet provided the only feasible way to record programs by name, code, and usage in a categorical manner.

First, the concentration of the 80/20 rule was calculated. For simulated data, this concentration is computed for us. Since the real data was manually collected and put into a frequency distribution, the concentration was calculated using a spreadsheet program. For the real data it was found that 80% of the total program usage was performed by 14.45% of the programs. Conversely, 20% of the total program usage was performed by 85.5% of the programs.

Also, there were very few programs whose ranking position remained constant throughout the observation of usage. For the real data collected, there were only 7 programs which held ranking positions 1 through 8 throughout

**Table 5.3: Program Usage Distribution from Collected Data**

| i | ni | f(ni) | i | ni | f(ni) | i | ni | f(ni) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 7222 | 80 | 90 | 7 | 159 | 181 | 1 |
| 2 | 2 | 2082 | 81 | 91 | 6 | 160 | 182 | 1 |
| 3 | 3 | 551 | 82 | 93 | 1 | 161 | 183 | 2 |
| 4 | 4 | 346 | 83 | 95 | 4 | 162 | 184 | 3 |
| 5 | 5 | 215 | 84 | 97 | 5 | 163 | 185 | 2 |
| 6 | 6 | 381 | 85 | 98 | 1 | 164 | 186 | 3 |
| 7 | 7 | 286 | 86 | 100 | 5 | 165 | 187 | 3 |
| 8 | 8 | 223 | 87 | 101 | 3 | 166 | 188 | 1 |
| 9 | 9 | 266 | 88 | 102 | 1 | 167 | 190 | 2 |
| 10 | 10 | 301 | 89 | 103 | 3 | 168 | 191 | 1 |
| 11 | 11 | 218 | 90 | 104 | 5 | 169 | 192 | 1 |
| 12 | 12 | 302 | 91 | 106 | 5 | 170 | 193 | 2 |
| 13 | 13 | 95 | 92 | 107 | 6 | 171 | 194 | 2 |
| 14 | 14 | 193 | 93 | 108 | 6 | 172 | 195 | 3 |
| 15 | 15 | 357 | 94 | 109 | 5 | 173 | 196 | 1 |
| 16 | 16 | 17 | 95 | 110 | 5 | 174 | 198 | 2 |
| 17 | 17 | 10 | 96 | 111 | 4 | 175 | 199 | 2 |
| 18 | 18 | 15 | 97 | 112 | 3 | 176 | 200 | 3 |
| 19 | 19 | 14 | 98 | 113 | 3 | 177 | 201 | 1 |
| 20 | 20 | 12 | 99 | 114 | 5 | 178 | 202 | 3 |
| 21 | 21 | 10 | 100 | 115 | 4 | 179 | 203 | 2 |
| 22 | 23 | 15 | 101 | 116 | 5 | 180 | 204 | 1 |
| 23 | 24 | 18 | 102 | 117 | 4 | 181 | 205 | 3 |
| 24 | 25 | 7 | 103 | 118 | 3 | 182 | 206 | 1 |
| 25 | 26 | 2 | 104 | 119 | 3 | 183 | 207 | 1 |
| 26 | 27 | 8 | 105 | 120 | 2 | 184 | 209 | 1 |
| 27 | 28 | 9 | 106 | 121 | 1 | 185 | 210 | 1 |
| 28 | 29 | 7 | 107 | 122 | 3 | 186 | 211 | 1 |
| 29 | 30 | 10 | 108 | 123 | 2 | 187 | 212 | 1 |
| 30 | 31 | 1 | 109 | 124 | 2 | 188 | 213 | 2 |
| 31 | 32 | 14 | 110 | 125 | 1 | 189 | 214 | 1 |
| 32 | 33 | 13 | 111 | 126 | 3 | 190 | 215 | 2 |
| 33 | 34 | 1 | 112 | 127 | 4 | 191 | 216 | 1 |
| 34 | 35 | 1 | 113 | 128 | 3 | 192 | 217 | 1 |
| 35 | 36 | 15 | 114 | 129 | 2 | 193 | 219 | 1 |
| 36 | 37 | 17 | 115 | 130 | 2 | 194 | 220 | 2 |
| 37 | 38 | 8 | 116 | 131 | 5 | 195 | 221 | 2 |
| 38 | 39 | 1 | 117 | 132 | 4 | 196 | 222 | 1 |
| 39 | 40 | 6 | 118 | 133 | 3 | 197 | 223 | 1 |
| 40 | 41 | 12 | 119 | 134 | 4 | 198 | 224 | 1 |
| 41 | 42 | 14 | 120 | 135 | 5 | 199 | 225 | 2 |
| 42 | 43 | 1 | 121 | 136 | 3 | 200 | 226 | 1 |
| 43 | 44 | 9 | 122 | 137 | 3 | 201 | 227 | 1 |
| 44 | 45 | 11 | 123 | 138 | 4 | 202 | 228 | 2 |
| 45 | 46 | 7 | 124 | 139 | 3 | 203 | 229 | 1 |
| 46 | 47 | 10 | 125 | 140 | 3 | 204 | 230 | 1 |
| 47 | 49 | 6 | 126 | 142 | 3 | 205 | 231 | 2 |
| 48 | 50 | 5 | 127 | 143 | 4 | 206 | 232 | 1 |
| 49 | 51 | 8 | 128 | 144 | 3 | 207 | 234 | 1 |
| 50 | 52 | 5 | 129 | 145 | 1 | 208 | 235 | 1 |
| 51 | 54 | 7 | 130 | 146 | 2 | 209 | 237 | 1 |
| 52 | 55 | 16 | 131 | 147 | 3 | 210 | 238 | 1 |
| 53 | 56 | 1 | 132 | 148 | 3 | 211 | 239 | 1 |
| 54 | 57 | 6 | 133 | 149 | 2 | 212 | 240 | 1 |
| 55 | 58 | 1 | 134 | 151 | 3 | 213 | 241 | 2 |
| 56 | 60 | 1 | 135 | 152 | 3 | 214 | 242 | 1 |
| 57 | 61 | 6 | 136 | 153 | 2 | 215 | 243 | 2 |
| 58 | 62 | 1 | 137 | 155 | 3 | 216 | 244 | 1 |
| 59 | 63 | 12 | 138 | 156 | 2 | 217 | 246 | 1 |
| 60 | 64 | 8 | 139 | 157 | 1 | 218 | 247 | 1 |
| 61 | 65 | 9 | 140 | 159 | 1 | 219 | 248 | 1 |
| 62 | 66 | 13 | 141 | 160 | 1 | 220 | 250 | 1 |
| 63 | 67 | 5 | 142 | 161 | 2 | 221 | 251 | 1 |
| 64 | 68 | 12 | 143 | 162 | 1 | 222 | 252 | 2 |
| 65 | 69 | 10 | 144 | 163 | 1 | 223 | 253 | 1 |
| 66 | 70 | 1 | 145 | 165 | 2 | 224 | 255 | 1 |
| 67 | 72 | 15 | 146 | 166 | 1 | 225 | 256 | 1 |
| 68 | 73 | 2 | 147 | 168 | 1 | 226 | 257 | 1 |
| 69 | 75 | 1 | 148 | 169 | 3 | 227 | 258 | 1 |
| 70 | 76 | 7 | 149 | 171 | 1 | 228 | 262 | 1 |
| 71 | 78 | 9 | 150 | 172 | 2 | 229 | 266 | 1 |
| 72 | 79 | 4 | 151 | 173 | 1 | 230 | 268 | 1 |
| 73 | 80 | 5 | 152 | 174 | 1 | 231 | 279 | 1 |
| 74 | 81 | 6 | 153 | 175 | 2 | 232 | 284 | 1 |
| 75 | 82 | 12 | 154 | 176 | 1 | 233 | 307 | 1 |
| 76 | 83 | 5 | 155 | 177 | 1 | 234 | 706 | 1 |
| 77 | 86 | 3 | 156 | 178 | 2 | 235 | 2101 | 1 |
| 78 | 87 | 5 | 157 | 179 | 1 | 236 | 13858 | 1 |
| 79 | 88 | 1 | 158 | 180 | 4 | | | |

each successive month of usage. As stated earlier, the collected data was found to have a $\gamma = 0.99934$. At first glance, an aging factor of $\gamma = 0.99934$ seems to contradict what has been recognized in the literature as use of programs decaying rapidly with time. However, when we consider that this $\gamma$ represents the aging factor of one program, then we can calculate this aging factor for program usage to be $(0.99934)^N$. Therefore, after one year the probability of selection gained from the last selection only remains a very small portion of its original weight.

### 5.5.2 Findings from the Simulated Data

The purpose of this study of validation was to have the ability to compare real and simulated data. In order to accomplish this task, we generated simulated data using a value of N = 134,864 which was the same number of usages found in the real data. We also used the parameter values assessed from the previous six months of real data. The simulated data produced, using a $\alpha = 0.102$ and $\gamma = 0.99934$, provided output in the form of a frequency distribution. Frequency distribution results are produced after each cycle is performed. For the simulated data, we chose to view the cycles in increments of 16,858 usages. This provided 8 cycles of data to examine. In **Table 5.4** the frequency distribution produced after the eighth cycle is displayed. Amazingly, the concentration of program usage was very

**Table 5.4 - Program Usage Distribution for Simulated Data**

| i | ni | f(ni) | i | ni | f(ni) | i | ni | f(ni) |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 5799 | 80 | 80 | 4 | 159 | 185 | 1 |
| 2 | 2 | 2267 | 81 | 81 | 5 | 160 | 186 | 1 |
| 3 | 3 | 1213 | 82 | 82 | 3 | 161 | 189 | 1 |
| 4 | 4 | 672 | 83 | 83 | 2 | 162 | 190 | 2 |
| 5 | 5 | 401 | 84 | 84 | 4 | 163 | 192 | 1 |
| 6 | 6 | 336 | 85 | 85 | 7 | 164 | 193 | 2 |
| 7 | 7 | 294 | 86 | 87 | 4 | 165 | 196 | 1 |
| 8 | 8 | 268 | 87 | 88 | 8 | 166 | 197 | 2 |
| 9 | 9 | 239 | 88 | 89 | 1 | 167 | 198 | 1 |
| 10 | 10 | 133 | 89 | 90 | 3 | 168 | 199 | 3 |
| 11 | 11 | 121 | 90 | 91 | 5 | 169 | 200 | 2 |
| 12 | 12 | 126 | 91 | 92 | 2 | 170 | 202 | 1 |
| 13 | 13 | 111 | 92 | 93 | 3 | 171 | 203 | 1 |
| 14 | 14 | 51 | 93 | 94 | 7 | 172 | 204 | 1 |
| 15 | 15 | 71 | 94 | 95 | 3 | 173 | 205 | 1 |
| 16 | 16 | 65 | 95 | 96 | 5 | 174 | 207 | 2 |
| 17 | 17 | 57 | 96 | 97 | 7 | 175 | 208 | 3 |
| 18 | 18 | 55 | 97 | 99 | 1 | 176 | 211 | 2 |
| 19 | 19 | 49 | 98 | 101 | 2 | 177 | 213 | 1 |
| 20 | 20 | 43 | 99 | 102 | 4 | 178 | 214 | 1 |
| 21 | 21 | 44 | 100 | 103 | 2 | 179 | 216 | 1 |
| 22 | 22 | 43 | 101 | 104 | 2 | 180 | 218 | 1 |
| 23 | 23 | 46 | 102 | 105 | 2 | 181 | 221 | 2 |
| 24 | 24 | 29 | 103 | 106 | 1 | 182 | 224 | 1 |
| 25 | 25 | 33 | 104 | 107 | 1 | 183 | 225 | 1 |
| 26 | 26 | 33 | 105 | 108 | 3 | 184 | 229 | 1 |
| 27 | 27 | 22 | 106 | 109 | 4 | 185 | 232 | 1 |
| 28 | 28 | 30 | 107 | 110 | 1 | 186 | 233 | 1 |
| 29 | 29 | 31 | 108 | 111 | 4 | 187 | 238 | 2 |
| 30 | 30 | 24 | 109 | 113 | 1 | 188 | 239 | 1 |
| 31 | 31 | 21 | 110 | 114 | 2 | 189 | 245 | 1 |
| 32 | 32 | 25 | 111 | 115 | 1 | 190 | 250 | 1 |
| 33 | 33 | 27 | 112 | 116 | 2 | 191 | 257 | 1 |
| 34 | 34 | 19 | 113 | 118 | 4 | 192 | 258 | 1 |
| 35 | 35 | 22 | 114 | 119 | 2 | 193 | 260 | 1 |
| 36 | 36 | 19 | 115 | 120 | 1 | 194 | 261 | 1 |
| 37 | 37 | 21 | 116 | 121 | 1 | 195 | 263 | 1 |
| 38 | 38 | 24 | 117 | 122 | 3 | 196 | 269 | 1 |
| 39 | 39 | 19 | 118 | 123 | 1 | 197 | 270 | 1 |
| 40 | 40 | 14 | 119 | 124 | 4 | 198 | 271 | 1 |
| 41 | 41 | 17 | 120 | 125 | 2 | 199 | 274 | 1 |
| 42 | 42 | 16 | 121 | 126 | 4 | 200 | 275 | 1 |
| 43 | 43 | 19 | 122 | 127 | 1 | 201 | 288 | 1 |
| 44 | 44 | 21 | 123 | 128 | 2 | 202 | 297 | 2 |
| 45 | 45 | 9 | 124 | 129 | 1 | 203 | 298 | 1 |
| 46 | 46 | 11 | 125 | 130 | 6 | 204 | 300 | 1 |
| 47 | 47 | 12 | 126 | 133 | 1 | 205 | 306 | 1 |
| 48 | 48 | 9 | 127 | 135 | 2 | 206 | 308 | 1 |
| 49 | 49 | 13 | 128 | 138 | 3 | 207 | 309 | 1 |
| 50 | 50 | 13 | 129 | 139 | 1 | 208 | 310 | 1 |
| 51 | 51 | 10 | 130 | 140 | 1 | 209 | 327 | 1 |
| 52 | 52 | 11 | 131 | 141 | 2 | 210 | 330 | 1 |
| 53 | 53 | 14 | 132 | 142 | 3 | 211 | 336 | 1 |
| 54 | 54 | 7 | 133 | 143 | 5 | 212 | 344 | 1 |
| 55 | 55 | 1 | 134 | 144 | 2 | 213 | 348 | 1 |
| 56 | 56 | 14 | 135 | 146 | 3 | 214 | 351 | 1 |
| 57 | 57 | 8 | 136 | 147 | 1 | 215 | 357 | 1 |
| 58 | 58 | 6 | 137 | 148 | 2 | 216 | 361 | 1 |
| 59 | 59 | 4 | 138 | 149 | 2 | 217 | 373 | 1 |
| 60 | 60 | 9 | 139 | 152 | 2 | 218 | 377 | 1 |
| 61 | 61 | 3 | 140 | 153 | 2 | 219 | 394 | 1 |
| 62 | 62 | 8 | 141 | 154 | 3 | 220 | 403 | 1 |
| 63 | 63 | 8 | 142 | 156 | 1 | 221 | 413 | 1 |
| 64 | 64 | 8 | 143 | 157 | 2 | 222 | 484 | 1 |
| 65 | 65 | 7 | 144 | 159 | 1 | 223 | 545 | 1 |
| 66 | 66 | 6 | 145 | 160 | 1 | 224 | 560 | 1 |
| 67 | 67 | 7 | 146 | 161 | 1 | 225 | 636 | 1 |
| 68 | 68 | 6 | 147 | 162 | 1 | 226 | 645 | 1 |
| 69 | 69 | 6 | 148 | 164 | 1 | 227 | 683 | 1 |
| 70 | 70 | 6 | 149 | 166 | 3 | 228 | 1120 | 1 |
| 71 | 71 | 10 | 150 | 167 | 1 | 229 | 1171 | 1 |
| 72 | 72 | 8 | 151 | 168 | 1 | 230 | 2140 | 1 |
| 73 | 73 | 3 | 152 | 169 | 3 | 231 | 7229 | 1 |
| 74 | 74 | 8 | 153 | 172 | 1 | | | |
| 75 | 75 | 3 | 154 | 173 | 1 | | | |
| 76 | 76 | 5 | 155 | 177 | 1 | | | |
| 77 | 77 | 8 | 156 | 178 | 2 | | | |
| 78 | 78 | 6 | 157 | 180 | 1 | | | |
| 79 | 79 | 8 | 158 | 183 | 1 | | | |

similar to that of the real data. For the simulated data, it was found that 84.7% of the programs produced 20% of the total usage. Conversely, 15.3% of the programs produced 80% of the total usage. Comparing this concentration to the 80/14.45 distribution of the real data results initial results appear to show great consistency.

Another validating element of the comparison is found by the comparison of the number of programs that held top ranking throughout the total number of usages. For the simulated data 6 programs held top usage position for the eight cycles. This result shows great correlation to the real data considering that 6 of 13,547 total program held top ranking compared to 7 of 13,879 for the real data. Again this tends to show the effectiveness of the autoregressive version of Simon's Model in predicting usage.

For the simulated data it was obvious, as with the real data, that the vast majority of programs quickly declined out of significant usage status. Comparing the results of the real and simulated data, it is impossible to draw a direct correlation between programs that are real and simulated. In other words, we cannot specifically declare that a simulated program with a certain identification number is the same as a specific program in the real data. However, it is extremely valuable to use the simulated results in order to make observations about overall behavior of program assets.

Knowing that Simon's Autoregressive Model can accurately describe the behavior of programs in concentration, ranking and attrition rate, we will attempt to draw conclusions for organizations' program usage. The conclusions found will enable the implementation of organizational policy for storage decisions based on program usage.

# CHAPTER 6 EVALUATION OF SIMON'S MODEL FOR PROGRAM STORAGE

The purpose of this chapter is to show how some general observations from the study of Simon's Model can be applied for program storage decision-making. In addition, we are able to isolate dynamic usage scenarios prevalent in organizational program usage behavior. Simon's model provides us with a powerful mechanism for studying program usage scenarios. In this chapter, we adapt Simon's Model in a dynamic way by expressing $\alpha$ and $\gamma$ as functional values. This adaptation gives us new insight into the interrelation of program entry and program aging.

## 6.1 The Need for a Software Allocation Scheme

We have already emphasized the need that organizations have for better understanding the usage of their software assets. Many firms know which programs or data files are in continual or extensive use. In most cases, the software which is heavily used on a daily basis is a very small proportion of total program assets. This was shown in the simulated and real data of Chapter 5. In this chapter we are more concerned with those programs having usages which are not constant over time. That is, many programs in a firm are subject to a decay in usage rate or an increase in usage rate. The problem is that individual programs have their own

dynamic usage rates. There is no feasible way to predict the behavior of each individual program in an organization. Instead, it is more practical to provide a methodology for focusing on those programs whose changing usage is critical to storage allocation.

We have the ability to determine that software which falls into the category of the "trivial many." The results shown thus far indicate that in many firms at least 80% of programs fall into this category (Willet, 1994). Therefore, in most organizations a very large portion of programs can be eliminated from the group requiring maintenance attention.

It is our goal to concentrate on that portion of programs which fall into the "significant few." In fact, we can concentrate effort on a portion of the "significant few." It is known that in many organizations there are usually a very small portion of programs whose usage always remains in a top ranking position. It is reasonable to eliminate these programs from the group requiring monitoring since they show little to no decay in usage.

Once the group of programs of concern in the significant few are identified, it will be valuable to examine the changes in usage of these programs. In order for organizations to have the ability to establish a policy for better management of their storage assets, we must perform a careful inspection of the simulated data from Simon's Model.

That is, it is valuable to examine the differences in program usages for different combinations of $\alpha$ and $\gamma$. Much of this chapter will be dedicated to interpreting the simulation output produced by ranging $\alpha$ and $\gamma$ through all possible levels. Hopefully this will lead to insightful results showing the tendencies of usages within ranges of $\alpha$ and $\gamma$.

If an organization were to be presented with a policy derived from the knowledge of the effects of combinations of $\alpha$ and $\gamma$, they could in turn use this policy for making storage decisions. That is why we first approach utilizing Simon's Model results in order to give ISD personnel greater insight. With a better understanding of normal program usage behavior, organizations can develop policies based on easily understood criteria. Part of the true value in deriving a policy from the results of Simon's Model is the relative ease with which an organization can assess its own model parameters. Once model parameters are evaluated, a firm would be able to determine the category into which their software falls. Each category would provide a recommendation for the intervals at which the "significant few" should be assessed for usage.

## 6.2 Examination of Simulation Results

We began this portion of our research by observing the simulated data generated from our program discussed in the previous chapter. It was necessary to run the simulation

program numerous times in order to observe usage behavior for varying values of $\alpha$ and $\gamma$. In examining the simulation results, the following observations were of interest for varying values of the model parameters.

1) The total number of usages necessary in order for a software item to fall out of the top 50 ranking.
2) The number of software items which had a usages that did not change from cycle to cycle.
3) The steepness of decline or increase in ranking position for particular programs as usages progressed.

### 6.2.1 The Purpose of Simulation Observations

It is necessary to justify to selection of the above mentioned criteria for observation. Certainly, there are numerous observations that can be seen in any simulation output. The three criteria mentioned above were chosen due to their usefulness in creating a recommendation for organizations.

The most difficult criterion to choose was the appropriate level for observing changes in ranking status. One might ask the question, "Why track the ranking position of any particular software item?" The reason for evaluating this change in ranking position was to assess the speed with which items change in usage rate. It was decided that a valuable use of this evaluation would be to view the change

in rank of an item which began in a high ranking position. The number of usages that it would take for this item to decline significantly in ranking would be indicative of the decline of other items. In fact, observations of the changing usages of software items indicated that items beginning in a lower position actually decrease more rapidly. If a manager were to use the change in ranking position of items in order to make an assessment of all software, a lower ranking item would not serve as a good indication. In fact, most of the lower ranking items are used very few times. It is more reasonable to track an item which began at a high ranking position. We are interested in how long it will take a heavily used item's usage to decline until it reaches the point of becoming obsolete.

The level of the fiftieth ranking was chosen for several reasons. First, we want to observe the change in ranking position of the first program whose usage is not constant. There is no need to observe a program if its ranking position remains the same. The next question involves the degree to which the change should be observed. One might desire to track this program until it drops out of the significant few or until it drops out of the top 10%. If we wait for the usage of a top ranking program to change with this magnitude, then all other programs would have changed more drastically. Also, tracking a software item beyond the point of the

fiftieth ranking position would require considerable simulation time for certain combinations of $\alpha$ and $\gamma$. For these reasons, the fiftieth ranking was chosen as the critical value for observation.

The reasons for not tracking software items of almost constant usage have already been explained. In addition it is intuitive to observe the steepness in change of a software item. As with any graphical method, the results provided are more concise and understandable. That is why we will graphically shown the trend in movement of ranking position.

### 6.2.2 Observations from the Simulation Study

The purpose of this section is to determine the implications of various combinations of $\alpha$ and $\gamma$ for organizations. We know, in general, the value of $\alpha$ refers to the entrance rate of new items while $\gamma$ refers to the decay or aging rate of old items. It is interesting to observe the interaction of these two forces. Ultimately, we hope to find regions or classes in which the usage behavior of software items is similar. That is, it is desirable to find some continuity in behavior with changes occurring at certain values of $\alpha$ and $\gamma$.

It was determined earlier in this chapter that we are interested in the point at which an item of initial high usage drops out of the top fifty ranking. We are also interested in the graphical presentation of the change in

usage of the certain items. Therefore, we ran the simulation program in increments of 0.10 for both $\alpha$ and $\gamma$ and observed the results.

### 6.2.3 Results from Varying $\alpha$ and $\gamma$

We will first discuss the observations from the simulation program. It is worth noting that this particular simulation study computes the top fifty ranking of programs as well as showing the frequency distribution of programs by usage. The simulation program is written in versions which show the usage concentration in A, B, and C categories and according to the 80/20 Rule. At this point it is sufficient to study the results using the 80/20 Rule. The results were as follows for varying values of $\alpha$ and $\gamma$.

1) When the value of $\alpha$ was less than approximately 0.4, the number of items remaining at a constant usage was dependant on the value of $\gamma$. As $\gamma$ increased up to the point of 0.999, only one program had non-declining usage. When $\gamma$ increased above 0.999 it appeared that many items held their top ranking position.
2) When the value of $\alpha$ increased above 0.4 different observations were found. The patterns were still dependent on changing values of $\gamma$. In fact, for $\gamma < 0.999$ and $0.4 \leq \alpha \leq 0.9$ there were no items which were observed to have a constant usage. With

$\gamma \geq 0.999$ only an average of three programs had constant usage.

3) It was also interesting that $\alpha > 0.9$ produced similar results regardless of the value of $\gamma$. At such a high entry rate, there will be a constant change in item ranking position. In fact the frequency distribution for $\alpha > 0.9$ revealed a very small number of classes. This is a reasonable observation since almost all items are used only once.

A summary of findings regarding stationarity of ranking position follow:

| Number of Programs With Constant Usage | | $\gamma$ | |
|---|---|---|---|
| | | $< 0.999$ | $\geq 0.999$ |
| $\alpha$ | $< 0.4$ | One program top ranking | Six programs top ranking |
| | $0.4 \leq \alpha \leq 0.9$ | No items with top ranking | Three programs top ranking |
| | $\geq 0.9$ | No items plus max usage = 5 | No items plus max usage = 4 |

These results tell us which program to monitor in tracking the change in rank to position 50. Hence, in each category we are excluding the number of programs indicated from usage examination. The next program in rank in each category is the one which was tracked to position 50.

### 6.2.4 Graphical Results of Changing Rank Positions

We want to determine how many usages were required in order for an item to drop below a rank of 50 in order to make a recommendation to IS management personnel. This study will give a conservative estimate of the point at which a reassessment of program storage status should be made. For example, if it takes 100,000 total usages before the program being observed drops to position 50, IS management would be recommended to re-evaluate the programs being stored on hard disk every 100,000 usages. Certainly, there is no way to determine if the re-evaluation will be cause for action. Management may find that there is still sufficient hard disk storage available. On the other hand they may also find that the vast majority of programs are fading out of usage and should be considered for an alternate storage medium. In any case, the re-evaluation should be a function of the rate at which program usage declines. This is only to be determined by the interactions of $\alpha$ and $\gamma$.

Below are some graphical results showing the re-evaluation points various values of $\alpha$ and $\gamma$. It was discovered that the re-evaluation points changed within the six regions found in the previous section. Selected values were chosen from each region in order to show the amount of time required to reach position 50 was approximately the same

in each region. **Figure 6.1** and **Figure 6.2** show the rate at which a program declines in usage for $\alpha < 0.4$ and $\gamma < 0.999$. The results show that in both cases approximately 200,000 usages over all programs were required in order to reach position 50. **Figure 6.3** and **Figure 6.4** show results from the region of $\alpha < 0.4$ and $\gamma > 0.999$. In this case re-evaluation must be made after 150,000 total usages. Moving on to the next region where $0.4 \leq \alpha \leq 0.9$ and $\gamma < 0.9$ we observe that the decline rate is approximately linear. Also, the decline is at a more rapid rate. These results are shown in **Figure 6.5** and **Figure 6.6**. The re-evaluation here is made after 75,000 total usages. We also have the case where $0.4 \leq \alpha \leq 0.9$ and $\gamma \geq 0.999$. Here the decline is much more rapid, and the re-evaluation should be made after only 30,000 usages. These results are shown in **Figure 6.7**. In the last two cases where $\alpha > 0.9$, the re-evaluation should be almost constant. With a large value of $\alpha$ we not only see extremely rapid decline in usage, but we also see many items moving up in usage ranking. With such a high probability of an item being a new and then decaying quickly, it would be very difficult for an organization to monitor their storage. The only redeeming quality of these two cases is that they would be very rare.

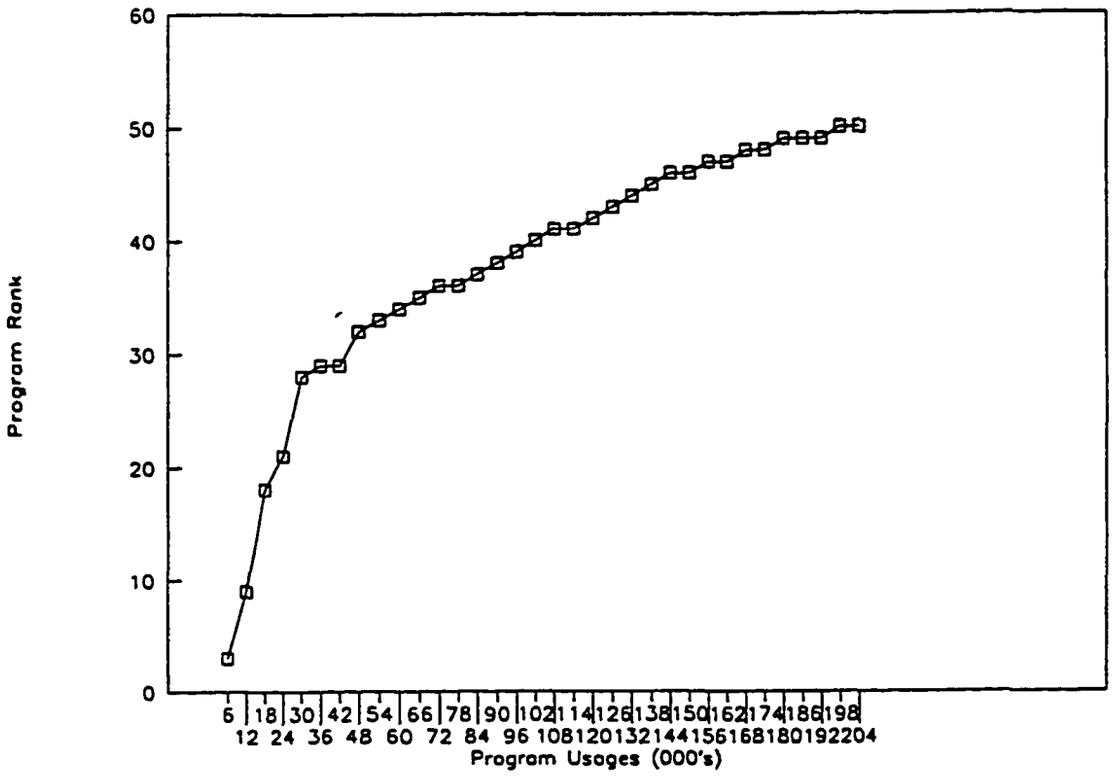


**Figure 6.1 Program Rank Changes (Alpha = 0.1 and Gamma = 0.5)**

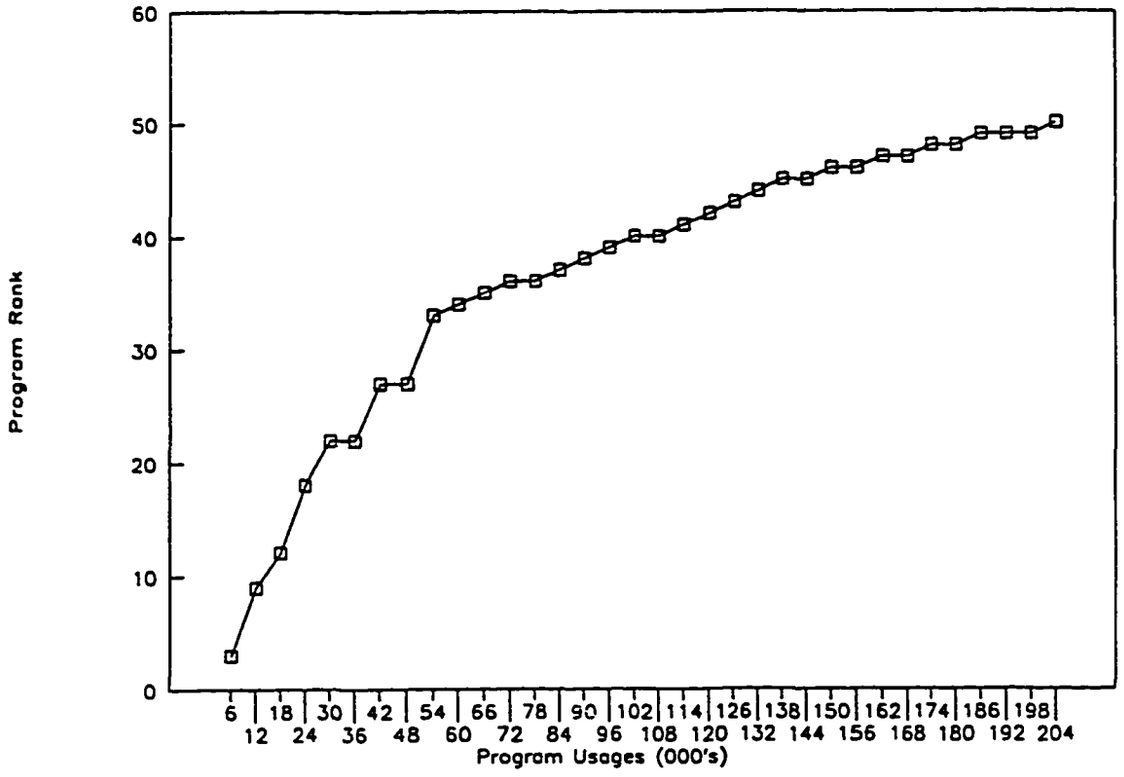


**Figure 6.2 Program Rank Changes (Alpha = 0.3 and Gamma = 0.5)**

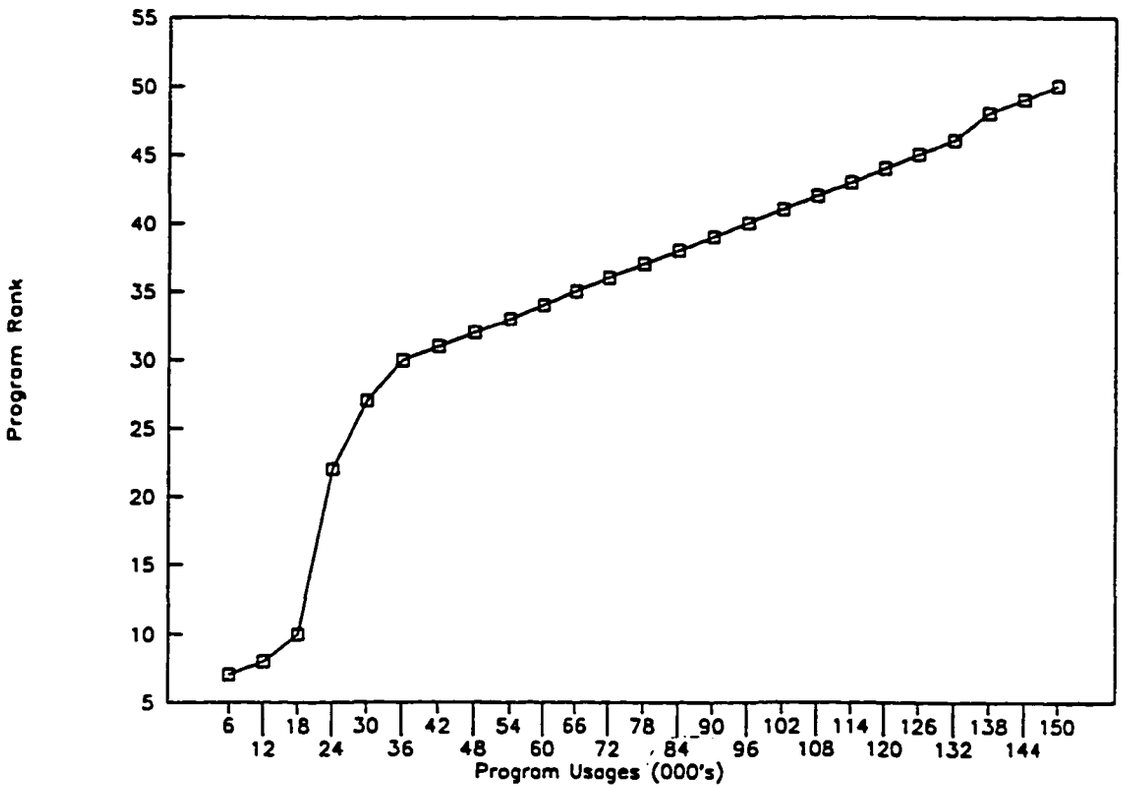


**Figure 6.3 Program Rank Changes (Alpha = 0.1 and Gamma = 0.999)**

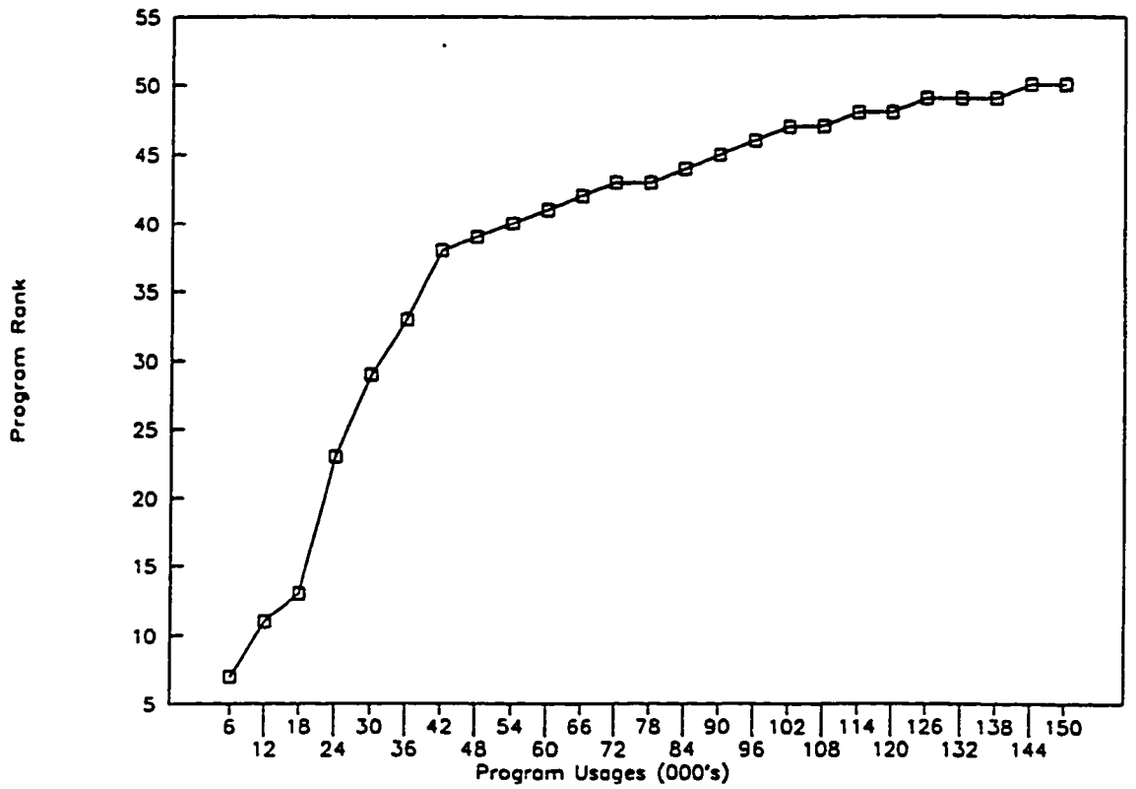


**Figure 6.4 Program Rank Changes (Alpha = 0.3 and Gamma = 0.999)**

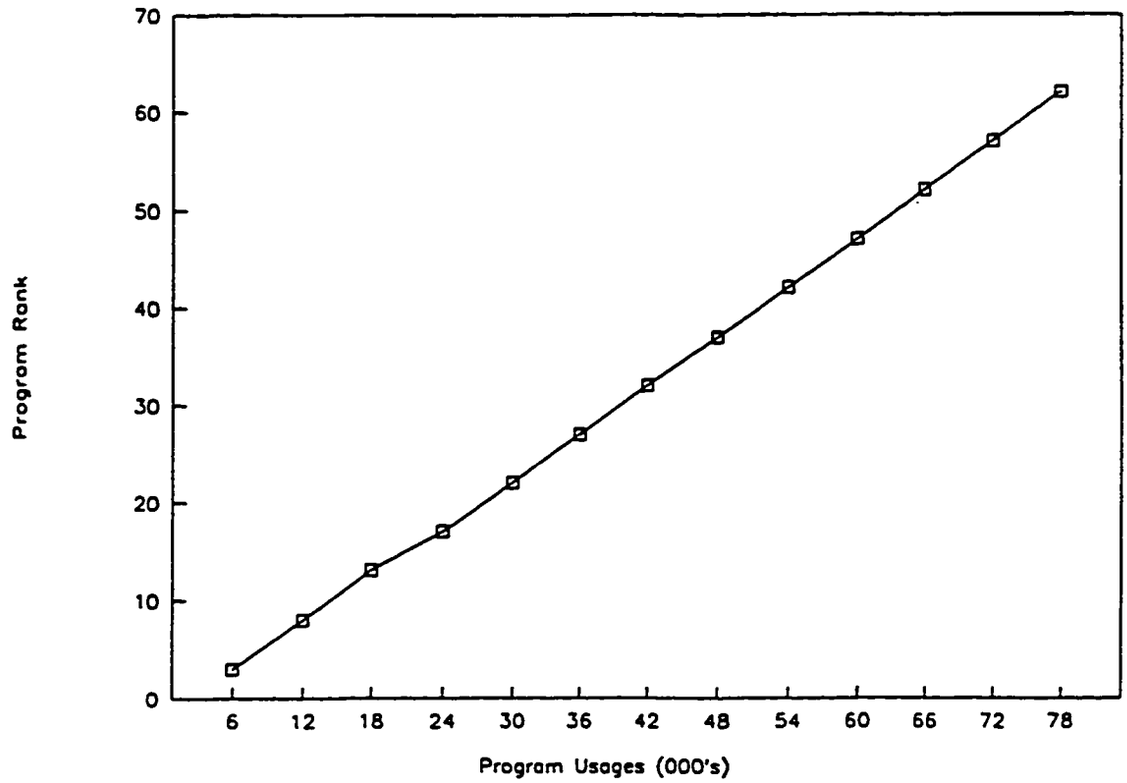


**Figure 6.5 Program Rank Changes (Alpha = 0.4 and Gamma = 0.5)**

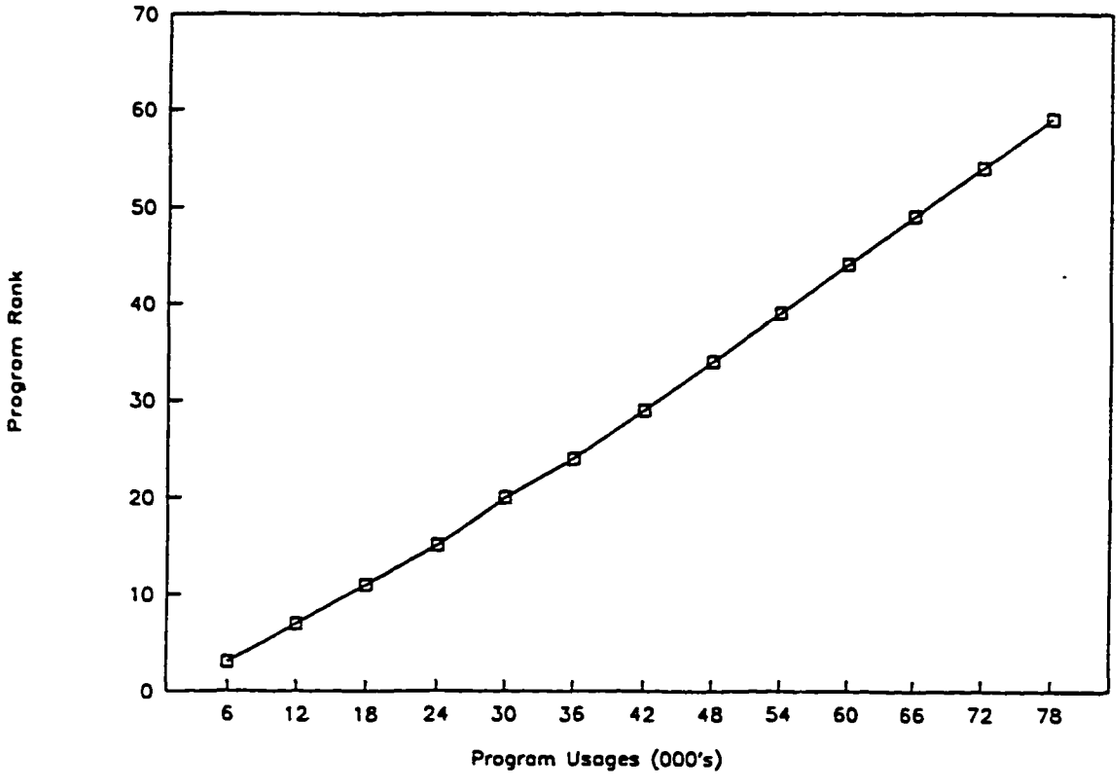


**Figure 6.6 Program Rank Changes (Alpha = 0.4 and Gamma = 0.9)**

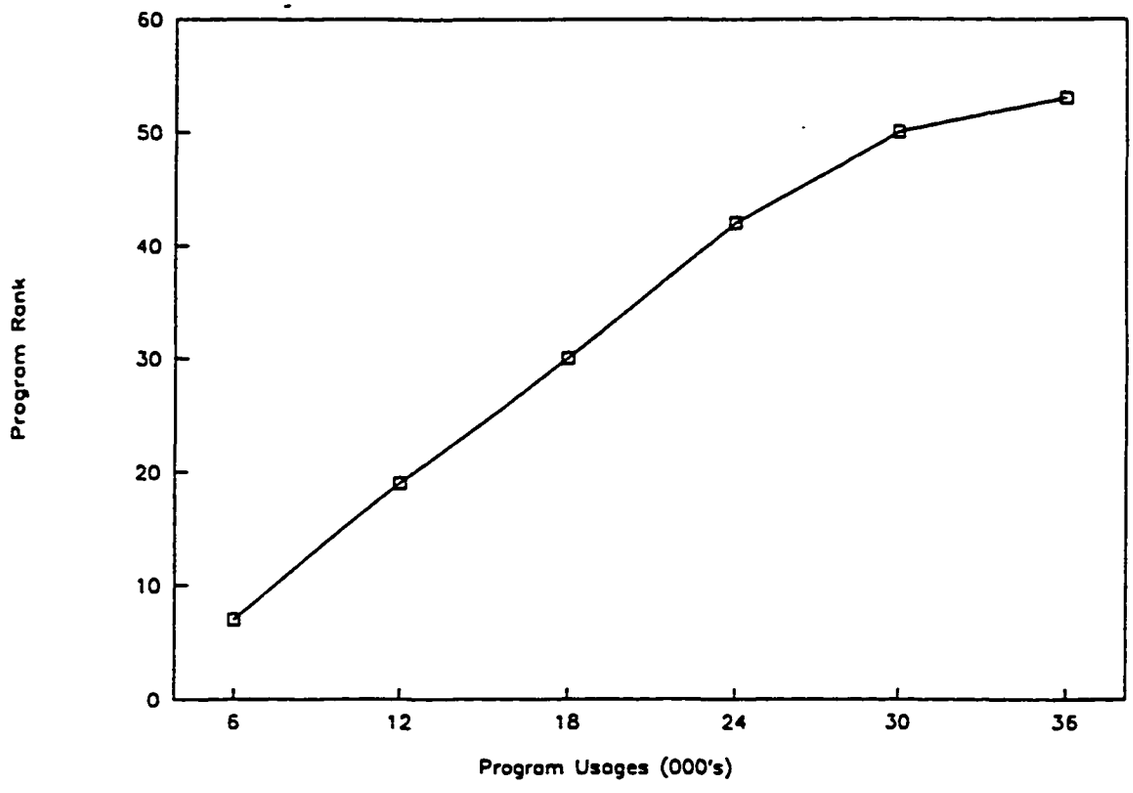


**Figure 6.7 Program Rank Changes (Alpha = 0.4 and Gamma = 0.999)**

Hence, we can make the following organizational recommendations for storage monitoring based on our simulated results.

| Number of Total Usages Between Storage Re-evaluation | | $\gamma$ | |
|---|---|---|---|
| | | $< 0.999$ | $\geq 0.999$ |
| $\alpha$ | $< 0.4$ | 200,000 | 150,000 |
| | $0.4 \leq \alpha \leq 0.9$ | 75,000 | 30,000 |
| | $\geq 0.9$ | Constant Monitoring | Constant Monitoring |

## 6.3 Organizational Implications

The implication of these results for organizations needing ISD storage decision-making are very important. It was mentioned in Chapter 1 of this dissertation that most organizations do not monitor the usage of their software and data assets. Without a monitoring policy there is no way to make informed decisions regarding usage dependent storage capacity. The proposed methodology in this chapter provides organizations with a very simple rule of thumb that has been tested and is only dependent on the two parameters of Simon's Model. We have shown that it is fairly easy to estimate the parameters of $\alpha$ and $\gamma$ from operating system logs. Also, the recommendations for re-evaluation points may be thought of as intervals for inspection. In this respect it is a

maintenance policy. As with any maintenance policy, the purpose is to prevent critical situations which are potentially costly. It is desirable to establish preventative measures for their ability to outperform ad hoc measures in reactive situations.

## 6.4 Capturing the Dynamic Nature of Program Usage

Thus far, we have studied program usage using Simon Autoregressive Model. It has been assumed in these previous runs that the parameters of $\alpha$ and $\gamma$ are constant. This assumption was based on the fact that the data collected for model validation showed values of $\alpha$ and $\gamma$ that remained approximately constant. Analysis was performed on these parameters at various intervals of the collected usage data. The ability to use constant values of $\alpha$ and $\gamma$ thus far has been due to the nature of the organization. Also, we only had the capability to examine program usage data for a relatively short interval of time.

### 6.4.1 Modeling Dynamic Program Usage

It would add much value to the scope of this research, if we were able to model an intuitive program usage scenario. Consider the situation which occurs when a new technology is developed or an organization has developed a new transaction processing system. In this situation, the entry rate of new programs ($\alpha$) is increasing. At the same time, the increasing entry rate will certainly have and effect on the aging of old

programs ($\gamma$). In fact, it is the increasing entry of new items which causes the old items to change more rapidly in usage. In order to view this phenomenon in a more practical sense, consider the case where an individual uses a particular application package or program suite. If a new program package is developed with superior features, then the individual may be enticed into switching packages. In this scenario, the increasing entry rate of new programs caused other programs to become obsolete. Thus, the effect was a decrease in $\gamma$.

The goal of this research has been to effectively apply Simon's Model in order to study dynamic program usage. If dynamic usage patterns can be identified, then they will be helpful in developing appropriate storage allocation schemes. With this goal in mind, we proceed in our formulation of Simon's Model with dynamically changing parameters of $\alpha$ and $\gamma$.

### 6.4.2 The Determination of Model Parameters

A critical factor in applying Simon's Version III Model is the choice of parameters $\alpha$ and $\gamma$. Simon explained in his book (Ijiri and Simon, 1977) that $\alpha$ is a critical factor in his model and should always be greater than zero. However, in certain circumstances, it is possible that the entry of "new" items will cease, and all items will then be "used" ones. This would imply that $\alpha$ would be equal to zero.

In the research of Tong (1994), self-organizing heuristics were studied using Simon's Model. In his study, access frequency distributions were examined, and $\alpha(k)$ was defined as a linear function that decreases a certain amount whenever a "new" record is being accessed. Assuming that the maximum number of records in a list is N, the formal expression of the linear function was defined as: $\alpha(k) = 1 - n(k-1)/N$, where $k = 1, 2, \ldots, R$; $n(k-1) = 0, 1, 2, \ldots, N$ where $n(k-1)$ is the number of "used" records in the list after k-1 accesses. Also, R was the assumed maximum number of accesses.

In this study, we define our own functional form of $\alpha(k) = SQRT[n(k-1)/N] + \alpha_0$, where $n(k-1)$ = the number of programs used in k-1 accesses; N = the maximum number of programs; and $\alpha_0$ = the initial input value of $\alpha$. We must also note that $k = 1, 2, \ldots, R$ where R is defined as the maximum number of accesses.

As previously discussed, we may consider the scenario when an increase in $\alpha$ has an affect on the value of $\gamma$. We have defined $\alpha$ to be a function which increases more rapidly than a linear function between the values of 0 and 1. If the value of $\gamma$ decreases as a result of $\alpha$ increasing, it is appropriate to have $\gamma$ decrease linearly. That is, the effect on $\gamma$ should not be as dramatic since it is a result of the change in $\alpha$. Therefore, we define $\gamma(k) = 1 - n(k-1)/N$, which

is the linearly decreasing function used in the work of Tong (1994).

Our simulation program is designed to begin with three programs in the list. Also, we restrict the value of $\alpha$ so that it does not grow larger than one. Likewise, the value of $\gamma$ is kept from declining below the value of zero.

In performing a simulation of this scenario, the value of N was chosen to be set to 15,000. Recall that in this dissertation Simon's Model has been written in order to view the simulation in a series of cycles. The program is given an input value for the number of program usages which comprise a cycle. In addition, the number of cycles must be given as an input value. Between cycles, we are able to inspect the changes in usage of the various programs. Hence, the length of the simulation run is defined as number of cycles X usages per cycle.

### 6.4.3 Observations from Simulated Results

For our simulation study, we observe the results from running the program for 5 cycles of length 1,000. With both $\alpha$ and $\gamma$ changing as a function of n(k-1) and N, the number of possible combinations for observation is very large. Therefore, initial parameter values were chosen in order to reasonably model a storage situation and enable us to view $\alpha$ and $\gamma$ at a spectrum of values. We input the starting value for $\alpha$ at 0.10. The observations from the simulated data are

summarized as follows. In **Table 6.1** an overview of observations is given. We inspect the changing values of $\alpha$ and $\gamma$, the number of programs accumulated at the end of each cycle and the number of frequency classes. Our observations show us that the largest increase in the value of $\alpha$ is between cycles 1 and 2. The value of $\alpha$ continues to increase, but at a decreasing rate. One of the things that we are interested in observing is the change in ranking position of the various program through the given number of cycles. In **Table 6.2**, we begin by summarizing the change in rank of the items which began in the top 10 positions. The movement of these 10 items can be interpreted more easily in **Figure 6.8**. In **Table 6.3** we provide a comprehensive view of the changes in the top 50 program usage ranking for our five cycles.

It is also interesting for us to examine the concentration of usages in the various cycles. From these distributions, it is possible to view the percentage of programs which comprised 80 percent of the total usage. This measure is that which we refer to as the concentration of the 80/20 Rule. As we progressed through the cycles, the concentration of the 80/20 Rule progressed as follows: 0.215, 0.306, 0.333, 0.365, 0.384. Thus, a decreasing value of $\gamma(k)$ caused the usages to be spread out among more items.

**Table 6.1 Summary of Observations**
**($\alpha$ Increasing / $\gamma$ Decreasing)**
**N = 15,000 and Cycle Length = 1,000**

| | ALPHA | GAMMA | ENDING # OF PROGRAMS | # OF FREQUENCY CLASSES |
|---|---|---|---|---|
| Cycle 1 | 0.204 | 0.989 | 163 | 20 |
| Cycle 2 | 0.266 | 0.973 | 412 | 30 |
| Cycle 3 | 0.318 | 0.953 | 712 | 35 |
| Cycle 4 | 0.365 | 0.930 | 1050 | 37 |
| Cycle 5 | 0.409 | 0.904 | 1433 | 40 |

**Table 6.2 Ranking Position Changes of Original Top 10 Programs ($\alpha$ Increasing and $\gamma$ Decreasing)**

| Program ID # | Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 |
|---|---|---|---|---|---|
| 2 | 1 | 1 | 1 | 1 | 1 |
| 4 | 2 | 4 | 4 | 4 | 4 |
| 1 | 3 | 2 | 2 | 2 | 2 |
| 5 | 4 | 3 | 3 | 3 | 3 |
| 6 | 5 | 5 | 5 | 5 | 5 |
| 10 | 6 | 6 | 8 | 8 | 9 |
| 11 | 7 | 8 | 12 | 14 | 16 |
| 7 | 8 | 15 | 23 | 30 | 38 |
| 36 | 9 | 12 | 18 | 24 | 30 |
| 9 | 10 | 22 | 33 | 39 | 52 |

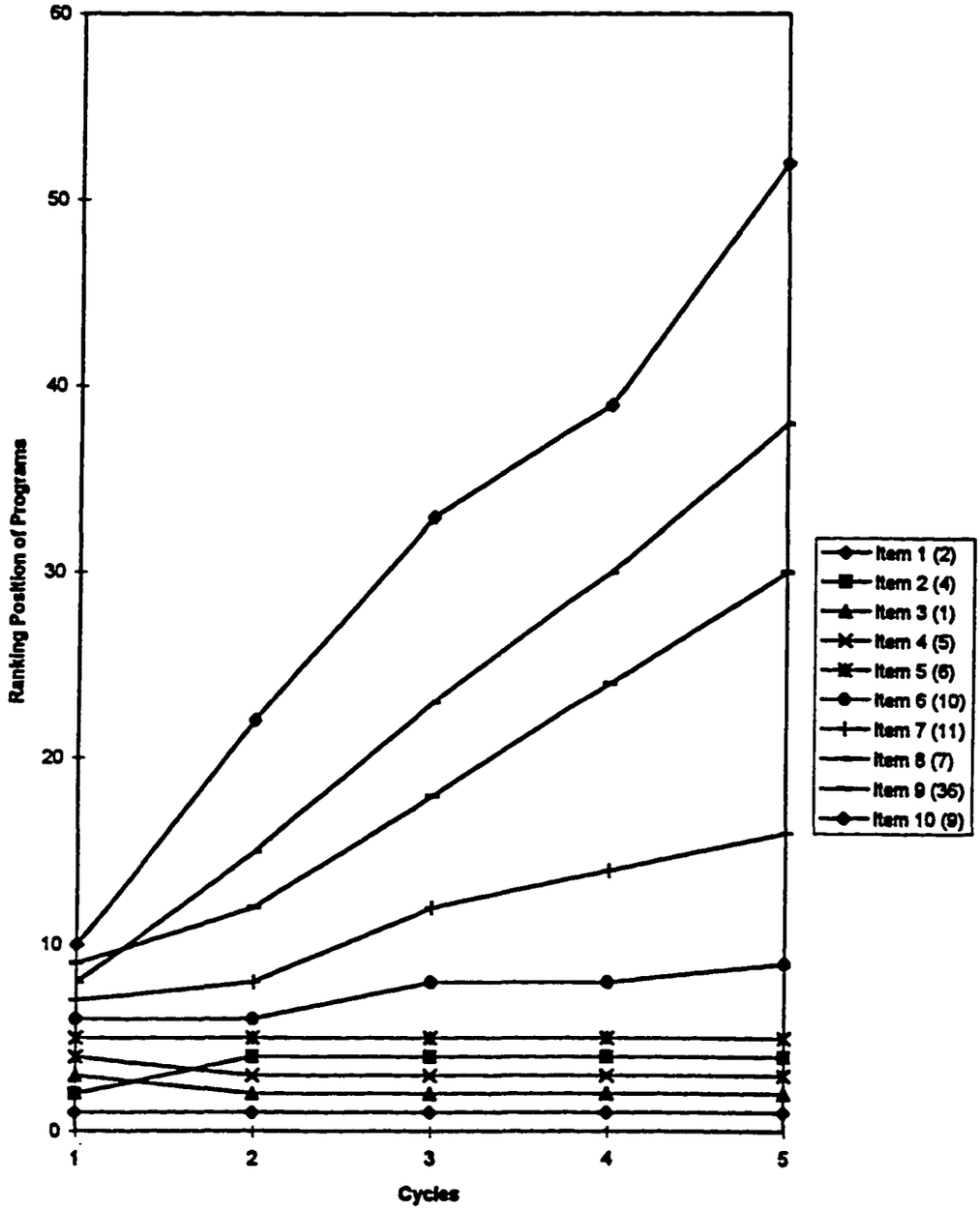


**Figure 6.8 Change in Position of Top 10 Items (Changing $\gamma$)**

**Table 6.3 - Top 50 Program Rankings For 5 Cycles**

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 |
| 4 | 1 | 1 | 1 | 1 |
| 1 | 5 | 5 | 5 | 5 |
| 5 | 4 | 4 | 4 | 4 |
| 6 | 6 | 6 | 6 | 6 |
| 10 | 10 | 433 | 433 | 433 |
| 11 | 190 | 542 | 542 | 542 |
| 7 | 11 | 10 | 10 | 1293 |
| 36 | 40 | 586 | 586 | 10 |
| 9 | 284 | 190 | 190 | 586 |
| 3 | 195 | 294 | 294 | 190 |
| 70 | 36 | 11 | 719 | 294 |
| 16 | 247 | 247 | 694 | 719 |
| 40 | 16 | 40 | 11 | 1150 |
| 12 | 7 | 455 | 247 | 694 |
| 15 | 242 | 284 | 683 | 11 |
| 85 | 19 | 195 | 40 | 995 |
| 14 | 343 | 36 | 455 | 247 |
| 30 | 367 | 399 | 785 | 785 |
| 8 | 3 | 617 | 811 | 811 |
| 105 | 70 | 683 | 284 | 683 |
| 144 | 9 | 16 | 195 | 40 |
| 42 | 144 | 7 | 769 | 455 |
| 19 | 12 | 242 | 36 | 284 |
| 133 | 276 | 498 | 399 | 1034 |
| 72 | 148 | 19 | 617 | 1079 |
| 52 | 205 | 343 | 995 | 769 |
| 31 | 89 | 367 | 16 | 195 |
| 33 | 155 | 362 | 950 | 1256 |
| 26 | 288 | 528 | 7 | 36 |
| 28 | 294 | 70 | 841 | 1243 |
| 17 | 171 | 421 | 498 | 399 |
| 45 | 350 | 9 | 242 | 617 |
| 83 | 107 | 3 | 367 | 1206 |
| 54 | 228 | 694 | 854 | 16 |
| 89 | 279 | 144 | 19 | 950 |
| 99 | 15 | 350 | 343 | 1126 |
| 50 | 355 | 12 | 421 | 7 |
| 107 | 33 | 390 | 9 | 242 |
| 108 | 313 | 276 | 3 | 841 |
| 121 | 255 | 604 | 362 | 498 |
| 39 | 133 | 443 | 945 | 343 |
| 78 | 290 | 205 | 528 | 367 |
| 29 | 200 | 148 | 70 | 1045 |
| 73 | 263 | 171 | 1018 | 854 |
| 101 | 143 | 155 | 276 | 19 |
| 47 | 314 | 582 | 711 | 1376 |
| 55 | 330 | 107 | 144 | 3 |
| 84 | 238 | 355 | 350 | 362 |
| 23 | 85 | 288 | 12 | 1139 |

If we examined this gradual change graphically, we would recognize a slight change in the shape of the skew distribution moving from the northwest to the southeast. These graphical results are displayed in **Figures 6.9, 6.10, 6.11, 6.12** and **6.13**. The changing concentrations provide us with insight for storage allocation. If the concentration of the 80/20 Rule is increasing while the "new" programs are still entering, the maximum number of allowable programs for a particular storage medium will eventually be reached. For this reason it is desirable to explore storage allocation algorithms which perform favorably under the condition of dynamic program usage.

For comparative purposes, we ran the simulation program with the same initial values for $\alpha$, N, number of cycles and cycle length, but held $\gamma$ to be constant at 0.999. This was the initial value for $\gamma$ in our previous run. We observed that programs did not "age" as quickly when $\gamma$ was decreasing. These results are shown in **Table 6.4** and **Figure 6.14**. This implies that a constant value of $\gamma$ may allow bounds for storage allocation decision-making which are slightly more relaxed than for a dynamic $\gamma$ situation.

## 6.5 Conclusions From Simulation Study

In this chapter, we have used Simon's model in order to study program usage behavior in a variety of ways. We compared the results for varying values of $\alpha$ and $\gamma$ when these

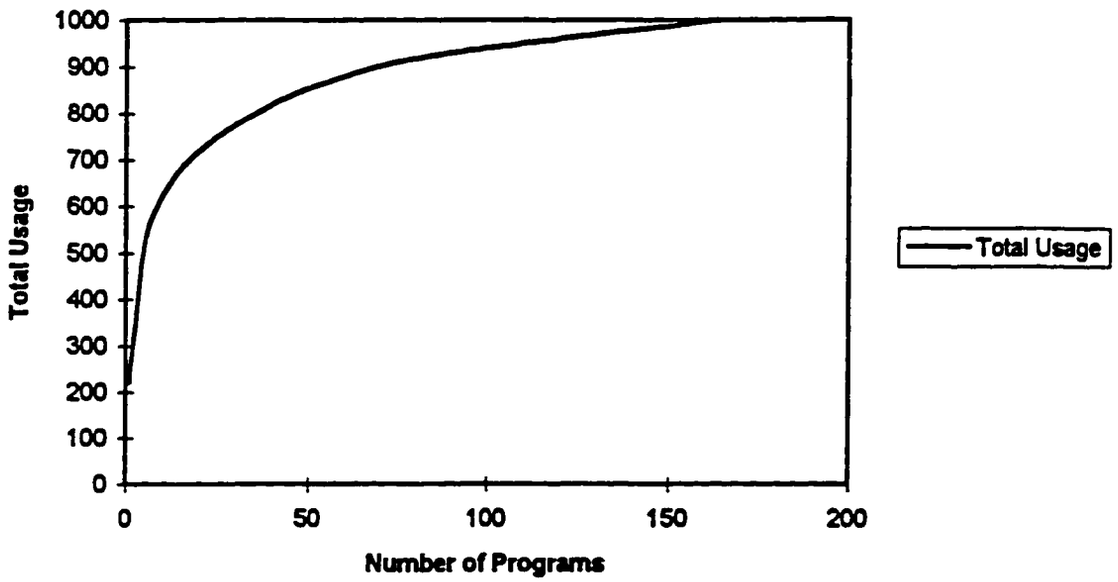


**Figure 6.9 Cycle 1 Cumulative Usage**

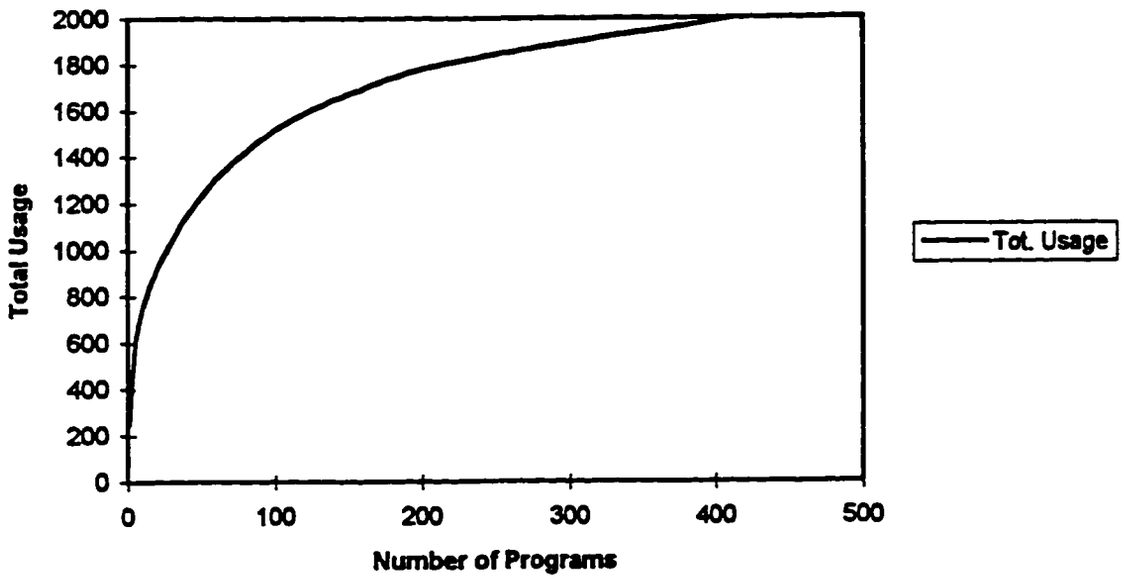


**Figure 6.10 Cycle 2 Cumulative Usage**

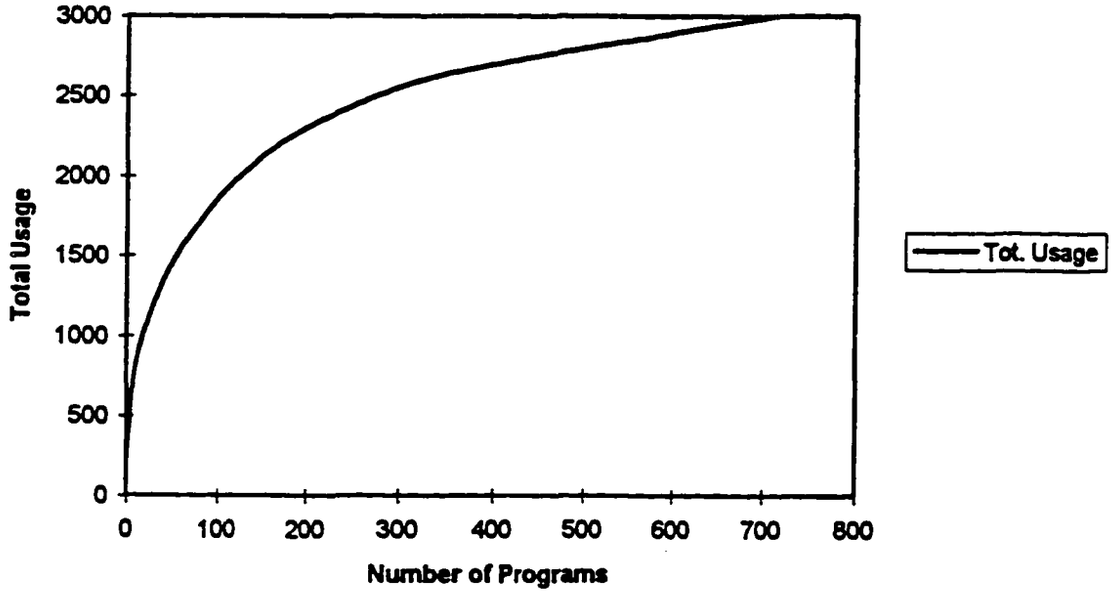


**Figure 6.11 Cycle 3 Cumulative Usage**

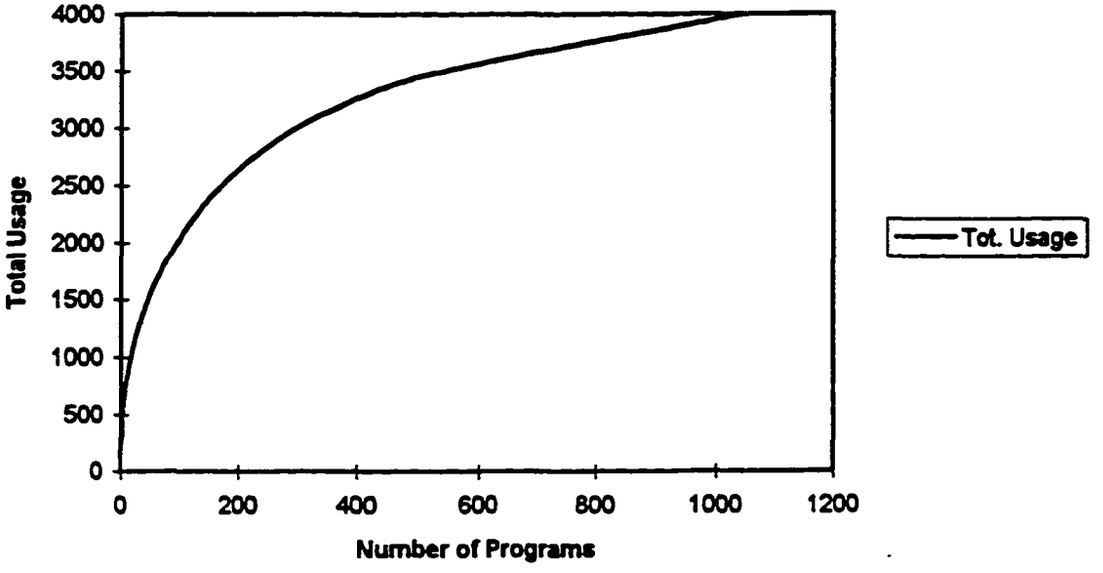


**Figure 6.12 Cycle 4 Cumulative Usage**

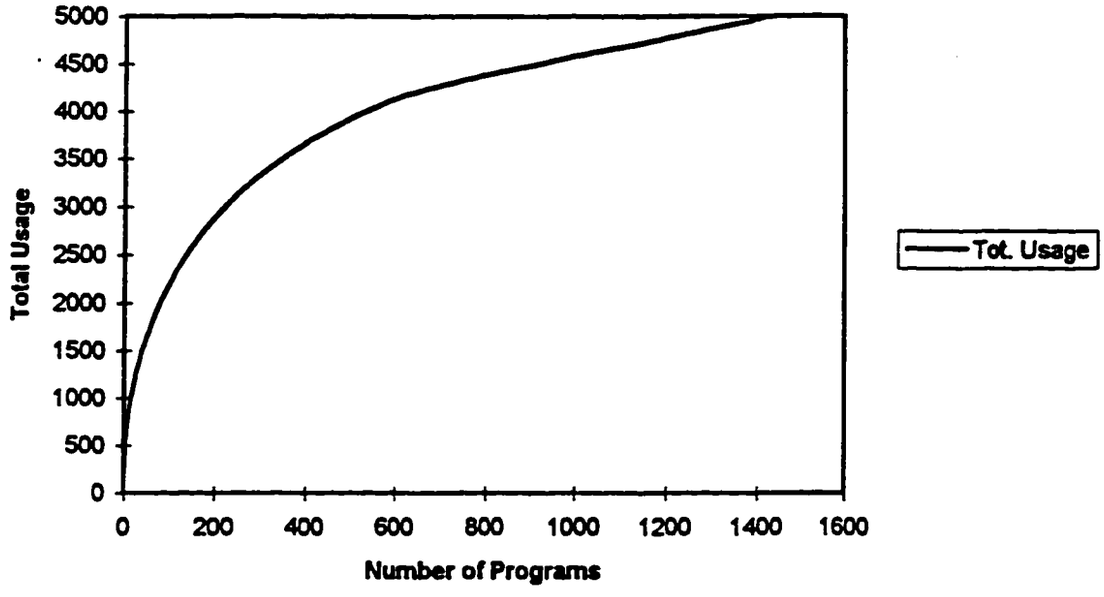


**Figure 6.13 Cycle 5 Cumulative Usage**

**Table 6.4 Ranking Position Changes of Original Top 10 Programs ($\alpha$ Increasing and $\gamma$ Constant)**

| Program ID # | Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 |
|---|---|---|---|---|---|
| 4 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 |
| 15 | 3 | 3 | 3 | 3 | 3 |
| 1 | 4 | 4 | 4 | 4 | 4 |
| 87 | 5 | 5 | 5 | 5 | 5 |
| 13 | 6 | 11 | 14 | 18 | 18 |
| 30 | 7 | 6 | 7 | 7 | 8 |
| 5 | 8 | 15 | 19 | 25 | 29 |
| 75 | 9 | 7 | 11 | 13 | 16 |
| 19 | 10 | 16 | 20 | 26 | 30 |

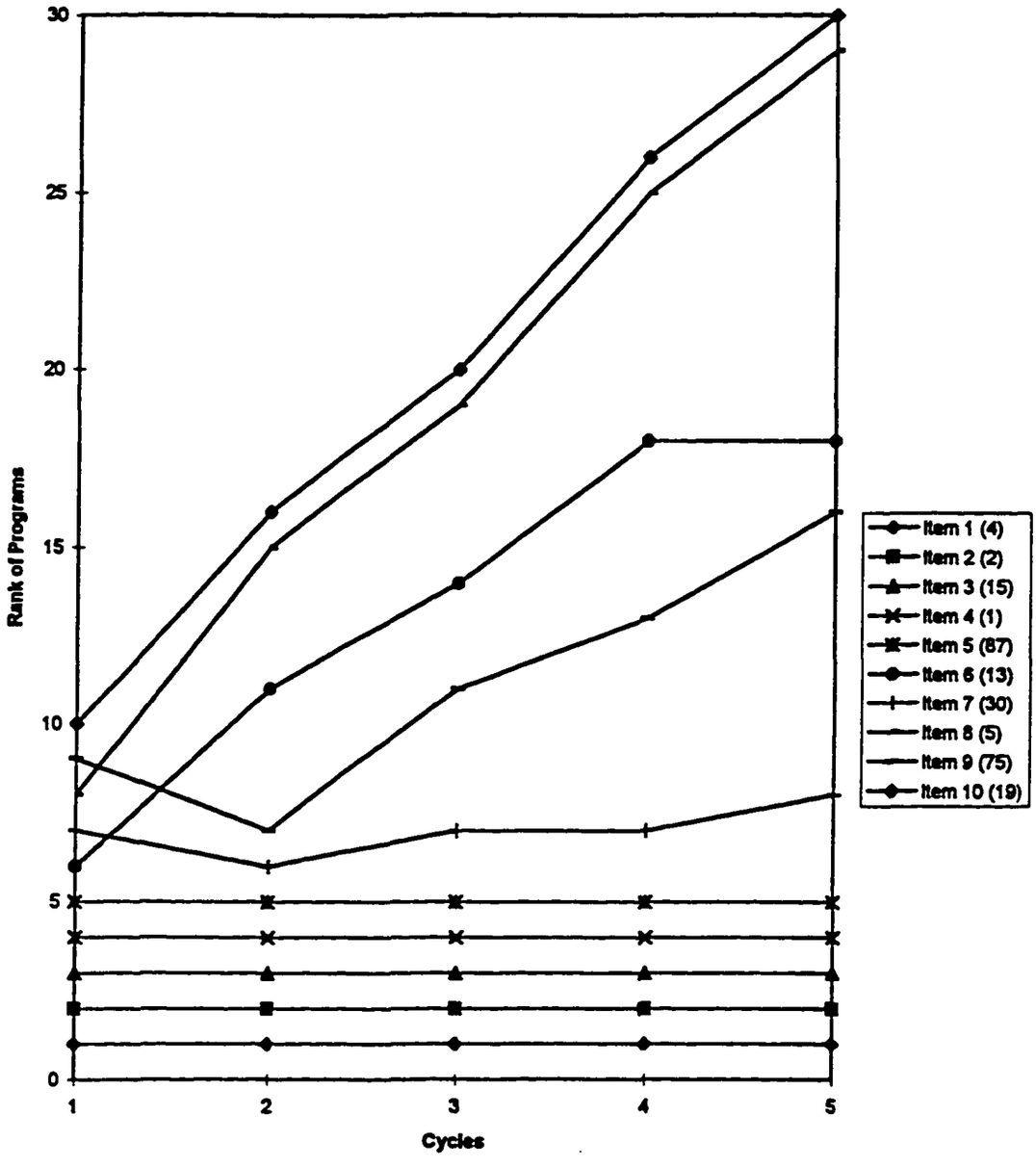


**Figure 6.14 Change in Position of Top 10 Items (Constant $\gamma$)**

parameters remain constant. Based on our observations, a scheme was developed which presents the recommended intervals for organizational evaluation of program usage. If an ad hoc policy is currently being used for allocation decisions, a recommendation based on the differences in $\alpha$ and $\gamma$ provides a better estimate for appropriate intervals for action.

Much effort was also taken in order to model a real world scenario for dynamic changes in program usage. Viewing the entry of "new" programs as a factor in the aging of "old" programs, we revised our simulation model in order to make $\alpha$ and $\gamma$ functions of the total number of usages and maximum number of programs. Our results showed that this view provides reasonable results while demonstrating the need for the incorporation of dynamic usages in storage allocation decisions. In the following chapter we will show the cost implications for using these dynamic usage frequencies in an established algorithm for program storage allocation.

# CHAPTER 7 A MINIMUM COST MODEL FOR HIERARCHICAL STORAGE ALLOCATION USING DYNAMIC USAGE FREQUENCIES

## 7.1 Introduction

The purpose of this chapter is to show the implications of the use of dynamic usage frequencies in storage allocation algorithms. In Chapter 2, we investigated various algorithms developed for the purpose of optimizing storage in multi-level storage systems. In the models developed by Ramamoorthy and Chandy (1970) and P. P. Chen (1973) the assumption of static usage frequency was made. An in-depth description of these models was provided in Chapter 2 of this dissertation. Both models are optimization models which serve to either minimize storage cost or minimize average access time. The focus of Chen's model (Chen, 1973) was on the placement of files using various types of storage devices that minimizes cost. In the work of Ramamoorthy and Chandy (1970), the model involved the placement of programs and their associated data in a storage hierarchy to minimize access time. In this chapter, we develop a cost-oriented optimization model for the assignment of programs to various levels of storage. In the following section, an explanation of the problems associated with optimization of program storage is given.

## 7.2 The Optimal Program Storage Hierarchy Problem

The model developed here is aimed at the placement of programs, but it is also applicable to the placement of data. We have chosen to concentrate on program storage in order to accommodate the fact that programs consist of a variable number of blocks which are assigned to a single storage medium. If data files are considered in the storage hierarchy, then other assumptions can be made. Among these assumptions is the fact that data can be divided into equal size blocks, and blocks of the same file may be stored on different devices in the memory hierarchy.

### 7.2.1 Assumptions of the Model

The following is a list of assumptions which are made for this model including the notations used.

1. Let M denote the total number of devices in the storage hierarchy.
2. For each device j, the cost per block is $c_j$ for devices j = 1, 2, ..., M. It is assumed that $c_j$ is a constant value which is known. Also, it is proportional to the efficiency with which the program can be retrieved from storage. That is, a program stored on the hard disk will have a higher associated cost per block than a program which is stored on a magnetic tape. Figures which estimate the current cost of storage types may also be expressed in cost/megabyte. Ghandeharizadeh et

al. (1994) made reference to the current costs associated with storage options. **Figure 7.1** shows the trade-off in service time versus cost. Many organizations use a resource accounting formula to accumulate individual storage charges. A current comparison of media on a dollar-per-megabyte and performance basis is given in **Figure 7.2**.

3. Let L denote the total number of programs, where each is stored in a file consisting of $N_i$ blocks for i = 1, 2, ..., L. Assume that the blocks of a program cannot be separated from the program file on different media. Namely, blocks of a program file cannot be divided between several devices.
4. Suppose that $x_{ij}$ = 1 if program i is assigned to device j or 0 if program i is not assigned to device j. Thus,

$$\sum_{j=1}^{M} x_{ij}=1$$

5. Let $f_i$ = the reference frequency for the program i per unit time. Here, our time unit will be expressed in terms of cycles where a cycle is defined as the number of total usages over which we will observe a distribution of frequencies. Thus, the total request rate for device j is

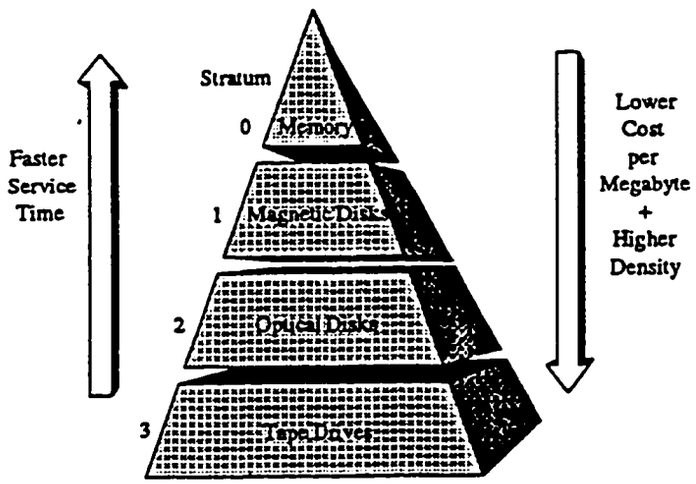


**Figure 7.1 Tradeoffs in Hierarchical Storage Systems**

| Medium | Cost/Megabyte | Access Time |
|---|---|---|
| Solid State | $60 - $100 | < 3 ms |
| RAID | $2 - $10 | 9 - 20 ms |
| Hard Drive | $0.80 - $2 | 9 - 20 ms |
| Optical (single platter) | $1 - $4 | 50 - 100 ms |
| Optical (jukebox) | $0.40 - $1 | 25 - 30 sec |
| Tape (single) | $0.40 - $2 | 30 sec-3 min |
| Tape (autoloader) | $0.05 - $1 | 1 - 5 min |

**Figure 7.2 Cost of Storage Media**

$$\lambda_j = \sum_{i=1}^{L} f_i x_{ij}$$

We assume here that a single program is allocated to only one device, and any usage frequency profile ($f_i$) will reflect only the usage history since the previous allocation.

6. It is assumed that one program is transferred per input/output request. The service time for each device is presumed to be a random variable which varies accordingly for input/output requests due to the electromechanical nature of storage devices. Thus request service time is assumed to be exponentially distributed with mean $1/\mu_j$ ($\mu_1 \geq \mu_2 \geq \ldots \geq \mu_M > 0$). In order to prevent the queue length for requests from growing without bound, it is required that

   $$\lambda_j < \mu_j$$

   where $\mu_j$ is the service rate of device j. Namely, the overall arrival rate to a device must be less than the device service rate.

7. It is also necessary to define $BS_j$ which is the maximum allowable number of blocks which can be stored on a device j. This value is a constant which is assigned as blocks of storage are added.

### 7.2.2 The Problem Formulation

With the following assumptions stated above, a model for allocation of programs to various levels of storage while minimizing cost can be stated as follows:

Minimize Cost

$$\sum_{j=1}^{M} c_j \sum_{i=1}^{L} N_i X_{ij} \qquad (7a)$$

Subject To

$$\sum_{j=1}^{M} x_{ij}=1 \qquad i = 1, \ldots, L \qquad (7b)$$

$$\sum_{i=1}^{L} N_i x_{ij} \leq BS_j \qquad j = 1, \ldots, M \qquad (7c)$$

$$\sum_{i=1}^{L} f_i x_{ij} \leq \mu_j \qquad j = 1, \ldots, M \qquad (7d)$$

### 7.2.3 A Program Allocation Example

Suppose that we have two devices (M = 2). Assume that we have five program files (L = 5) and we want to allocate them between the two devices. Assume that the cost of the

two storage devices are $c_1$ = \$4/block and $c_2$ = \$2/block. Let the table below summarize the system.

| i | Ni | fi |
|---|---|---|
| 1 | 40 | 17 |
| 2 | 20 | 9 |
| 3 | 30 | 5 |
| 4 | 40 | 5 |
| 5 | 75 | 13 |

In this table,

$i$ = index of programs;

$N_i$ = the block size of each program;

$f_i$ = the usage frequency of program i

We assume here that the block sizes of the individual programs are known, and that the usage frequencies of the programs are known. It is worth noting here that the assumption of known frequency is a limitation, and the use of Simon's Model will be applied in order to study this cost model under the assumption of dynamic frequency.

Suppose for this problem, parameters $\mu_1$ = 30 and $\mu_2$ = 20 which are the service rates for each storage device. The variables $x_{ij}$ with i = 1, ..., L; and j = 1, ..., M are defined as follows:

$$x_{ij} = \begin{cases} 1 & \text{if program i is stored on device j} \\ 0 & \text{otherwise} \end{cases}$$

Thus, we have defined this model as an allocation problem. In addition, we have capacity constraints on the two storage devices defined by total block size (BS1 and BS2). For this problem, we will let $BS_1 = 200$ and $BS_2 = 500$.

We can now express our optimization model for this example as

Minimize Cost = 4*40*x11 + 2*40*x12 + 4*20*x21 + 2*20*x22 +
4*30*x31 + 2*30*x32 + 4*40*x41 + 2*40*x42 +
4*75*x51 + 2*75*x52

Subject To

x11 + x12 = 1

x21 + x22 = 1

x31 + x32 = 1

x41 + x42 = 1

x51 + x52 = 1

40*x11 + 20*x21 + 30*x31 + 40*x41 + 75*x51 ≤ 200

40*x12 + 20*x22 + 30*x32 + 40*x42 + 75*x52 ≤ 500

17*x11 + 9*x21 + 5*x31 + 5*x41 + 13*x51 ≤ 30

17*x12 + 9*x22 + 5*x32 + 5*x42 + 13*x52 ≤ 20

In order to solve this linear programming problem, LINDO/PC Release 5.0 was used. The initial results were as follows.

| Optimal Objective Function Value (Cost) = 640 |
|---|

| Variable | Value |
|---|---|
| x11 | 1 |
| x12 | 0 |
| x21 | 0 |
| x22 | 1 |
| x31 | 0 |
| x32 | 1 |

| Variable | Value |
|---|---|
| x41 | 0 |
| x42 | 1 |
| x51 | 1 |
| x52 | 0 |

These results indicate that programs 1 and 5 were allocated to device 1, and programs 2, 3 and 4 were allocated to device 2. Implementation of this algorithm would then cause these programs to be moved to their appropriate locations in order to optimize cost. Intuitively, one might think that files will always be allocated to the cheaper storage device in order to minimize cost. In this model program access is constrained by the service time requirements which would ensure that some files are stored on the faster and more expensive storage medium.

At this point we have solved this allocation problem using a single usage frequency profile. The contention of this dissertation is that program usage frequencies are not static over time. As shown in Chapter 6, there are many occurrences such as change in technology, changing

organizational needs and new program production which cause program usage behavior to be dynamic. We have shown that this dynamic behavior can be captured through the use of Simon's Model with changing parameter values of $\alpha$ and $\gamma$. With the insight that we have gained through the dynamic study of Simon's Model, we can adapt the optimization model presented above. We will call this model the Dynamic Reallocation Model With Cost Minimization. The process for implementing this model is shown in **Figure 7.3**. For the example that we are currently using, we will show the effect that changing usage frequency has on the model.

In this example, we started out with 5 programs and a total usage of 49. Therefore, we can estimate $\alpha \approx 0.102$. We are assuming here that the scenario stated in Chapter 6, where $\alpha$ is increasing and $\gamma$ is decreasing is valid. That is, we are observing the changing usage distribution under the assumption that new programs are "arriving" which in turn causes a "decay" in the usage of the existing programs. Using Simon's Autoregressive model with increasing $\alpha$ and decreasing $\gamma$, we observe the frequency distribution when $\alpha = 0.20$. The frequency data follows in which an initial allocation of programs has been made according to the original optimal solution.

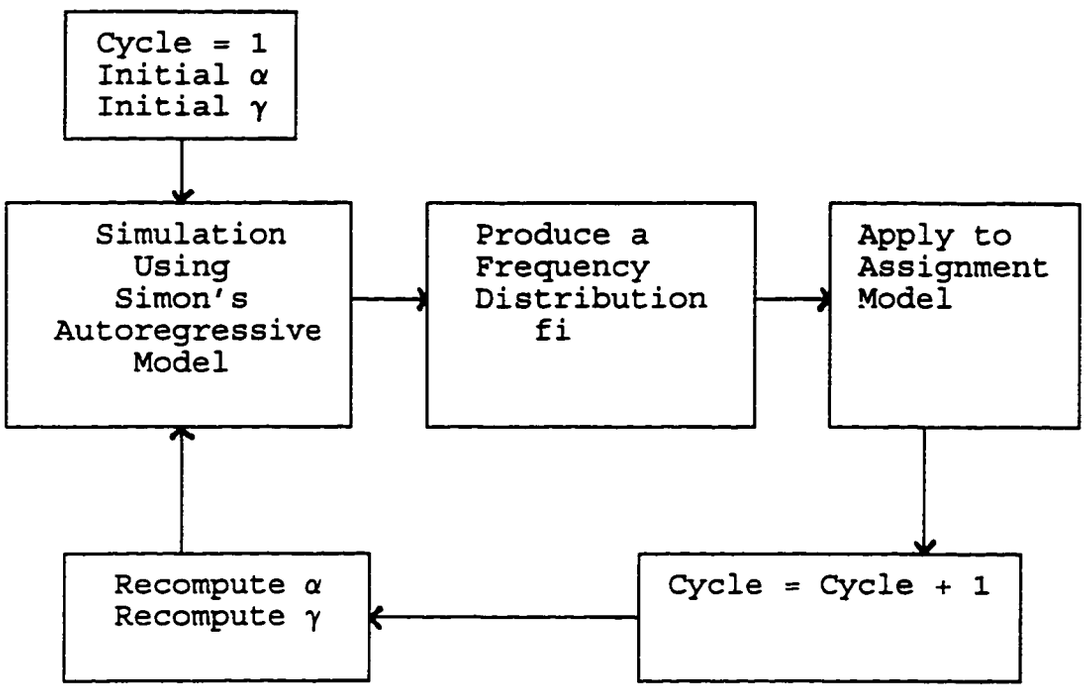


**Figure 7.3 Dynamic Reallocation Model with Cost Minimization**

| i | Ni | fi |
|---|---|---|
| 1 | 40 | 10 |
| 2 | 20 | 6 |
| 3 | 30 | 4 |
| 4 | 40 | 3 |
| 5 | 75 | 5 |
| 6 | 30 | 4 |
| 7 | 30 | 6 |
| 8 | 30 | 2 |

Using this new usage distribution our model is updated as follows.

Minimize Cost = 4*40*x11 + 2*40*x12 + 4*20*x21 + 2*20*x22 +
4*30*x31 + 2*30*x32 + 4*40*x41 + 2*40*x42 +
4*75*x51 + 2*75*x52 + 4*30*x61 + 2*30*x62 +
4*30*x71 + 2*30*x72 + 4*30*x81 + 2*30*x82

Subject To

x11 + x12 = 1

x21 + x22 = 1

x31 + x32 = 1

x41 + x42 = 1

x51 + x52 = 1

x61 + x62 = 1

x71 + x72 = 1

x81 + x82 = 1

$$40*x11 + 20*x21 + 30*x31 + 40*x41 + 75*x51 + 30x61 + 30x71 + 30x81 \leq 200$$

$$40*x12 + 20*x22 + 30*x32 + 40*x42 + 75*x52 + 30x62 + 30x72 + 30x82 \leq 500$$

$$10*x11 + 6*x21 + 4x31 + 3x41 + 5x51 + 4*x61 + 6*x71 + 2*x81 \leq 30$$

$$10*x12 + 6*x22 + 4x32 + 3x42 + 5x52 + 4*x62 + 6*x72 + 2*x82 \leq 20$$

Note that in this example, the increasing value of $\alpha$ has caused the addition of 3 new programs. We observe the changes in allocation which occur when the model is adjusted for the entry of these new programs. The model will allocate them to the appropriate storage medium in order to produce the minimum cost.

The results of this optimization run gave an increased optimal objective function value (cost) = $770. This implies that the addition of the 3 new programs increased the cost by (770 - 640) or 130 dollars. It is interesting to compare these results to those which would have been incurred if the model did not account for all dynamic changes. If that were the case, the three new programs would have simply been stored on device 1 which is the more expensive storage medium. We make this assumption since most organizations will choose to store new programs on the fastest medium.

With the addition of these new programs, and no change made to the allocation of existing programs, the total cost incurred would have been $1000. This cost was computed by adding the cost of the initial allocation to the cost of allocating 3 new files to device 1. Thus, for this small example the saving in cost was $1000 - $770 = $230.

### 7.2.4 Example Considering Further Reallocation

We will now consider a second example which considers a different value for the entry rate, $\alpha$. In this example we let $\alpha$ = .40. Our goal here is to show that there will still be a cost improvement as $\alpha$ increases and $\gamma$ decreases. We generated a frequency distribution according to these parameters. The table below summarizes this system.

| i | Ni | fi |
|---|---|---|
| 1 | 40 | 9 |
| 2 | 20 | 5 |
| 3 | 30 | 4 |
| 4 | 40 | 3 |
| 5 | 75 | 3 |
| 6 | 30 | 3 |
| 7 | 30 | 5 |
| 8 | 30 | 1 |
| 9 | 20 | 2 |
| 10 | 20 | 3 |

| i | Ni | fi |
|---|---|---|
| 11 | 20 | 2 |
| 12 | 20 | 2 |
| 13 | 20 | 1 |
| 14 | 20 | 1 |
| 15 | 20 | 1 |
| 16 | 20 | 1 |
| 17 | 20 | 1 |
| 18 | 20 | 1 |
| 19 | 20 | 1 |
| 20 | 20 | 1 |

Given this data, we define our model as follows.

Minimize Cost = 4*40*x11 + 2*40*x12 + 4*20*x21 + 2*20*x22 +
4*30*x31 + 2*30*x32 + 4*40*x41 + 2*40*x42 +
4*75*x51 + 2*75*x52 + 4*30*x61 + 2*30*x62 +
4*30*x71 + 2*30*x72 + 4*30*x81 + 2*30*x82

4*20*x91 + 2*20*x92 + 4*20*x101+ 2*20*x102+
4*20*x111+ 2*20*x112+ 4*20*x121+ 2*20*x122+
4*20*x131+ 2*20*x132+ 4*20*x141+ 2*20*x142+
4*20*x151+ 2*20*x152+ 4*20*x161+ 2*20*x162+
4*20*x171+ 2*20*x172+ 4*20*x181+ 2*20*x182+
4*20*x191+ 2*20*x192+ 4*20*x201+ 2*20*x202

Subject To

x11 + x12 = 1

x21 + x22 = 1

x31 + x32 = 1

x41 + x42 = 1

x51 + x52 = 1

x61 + x62 = 1

x71 + x72 = 1

x81 + x82 = 1

x91 + x92 = 1

x101+ x102 = 1

x111+ x112 = 1

x121+ x122 = 1

x131+ x132 = 1

x141+ x142 = 1

x151+ x152 = 1

x161+ x162 = 1

x171+ x172 = 1

x181+ x182 = 1

x191+ x192 = 1

x201+ x202 = 1

40*x11 + 20*x21 + 30*x31 + 40*x41 + 75*x51 +
30*x61 + 30*x71 + 30*x81 + 20*x101 + 20*x111 +
20*x121 + 20*x131 + 20*x141 + 20*x151 + 20*x161
+ 20*x171 + 20*x181 + 20*x191 + 20*x201 ≤ 200

40*x12 + 20*x22 + 30*x32 + 40*x42 + 75*x52 +
30*x62 + 30*x72 + 30*x82 + 20*x102 + 20*x112 +
20*x122 + 20*x132 + 20*x142 + 20*x152 + 20*x162
+ 20*x172 + 20*x182 + 20*x192 + 20*x202 ≤ 500

9*x11 + 5*x21 + 4x31 + 3x41 + 3x51 + 3*x61 +
5*x71 + x81 + 2*x91 + 3*x101 + 2*x111 + 2*x121 +
x141 + x151 + x161 + x171 + x181 + x191 +x201 ≤ 30

9*x12 + 5*x22 + 4x32 + 3x42 + 3x52 + 3*x62 +
5*x72 + x82 + 2*x92 + 3*x102 + 2*x112 + 2*x122 +
x142 + x152 + x162 + x172 + x182 + x192 +x202 ≤ 20

Solving this problem, we obtain an optimal objective function value of $1450. As in the first problem, we compute the results if no reallocation had been done since the last run. The total cost would be estimated at $1730. Therefore, the cost saving is $280. We must also note that at this point our Lindo output shows that the capacity constraints for the service rates have zero slack. Therefore, we have reached a maximum for the number of program usages which can be handled by this system.

### 7.3 Cost Implications for Varying $\alpha$ and $\gamma$

The examples presented in the above sections have demonstrated that cost savings can be realized with the consideration of dynamic usage frequencies. In this section, we observe the effect that sensitive changes in $\alpha$ and $\gamma$ have on the magnitude of cost savings. It is necessary to observe these changes for a full range of variation in $\alpha$ and $\gamma$.

In order to perform a cost analysis, the simulation program with an increasing $\alpha$ and decreasing $\gamma$ rate was used. It was stated earlier that the combination of an increasing $\alpha$ and decreasing $\gamma$ represents an intuitive scenario of program usage behavior. Therefore, we will continue this cost analysis based on that same scenario.

The table below summarizes the results. The optimization model was run at six intervals varying $\alpha$ and $\gamma$. For each of these runs, two devices were assumed (M = 2).

| $\alpha$ | $\gamma$ | # of Prog | Opt. Cost (Dynamic) | Opt. Cost (Static) | Total Cost Saving | Cost Saving per Prog |
|---|---|---|---|---|---|---|
| .10 | .95 | 3 | $100 | $200 | $100 | $33.33 |
| .41 | .90 | 6 | $180 | $300 | $120 | $20 |
| .59 | .76 | 12 | $360 | $540 | $180 | $15 |
| .72 | .62 | 20 | $660 | $1020 | $360 | $18 |
| .84 | .46 | 28 | $1040 | $1500 | $460 | $16.43 |
| .95 | .28 | 37 | $1480 | $2040 | $640 | $15.13 |

These results show that the total cost saving continued to increase as the values of $\alpha$ increased. It is noted that in general the cost saving per program decreased slightly as the total number of programs increased. This is an intuitive result. As the total number of programs increases, the limit for the total block capacity is being reached. When the capacity for both devices is reached, there will be no more savings to realize until the capacity is expanded.

### 7.4 Conclusions on the Dynamic Reallocation Model with Cost Minimization

Although these examples are very simplistic, we have demonstrated the implications of dynamic usage frequency. Viewing a program allocation situation as dynamic will not only affect the cost but also affect the placement of programs under some service time constraint.

The model which has been proposed in this section, is supported by the work of Ramamoorthy and Chandy (1970) and P. P. Chen (1973), although this model does have significant differences from those developed by the aforementioned authors. We have not taken into consideration mean response time constraints for individual requests, nor have we considered the possibility of allocating blocks of one file to different storage media.

The results of this chapter confirm, in a limited way, the fact that improvements in cost can be made if files are allocated according to changing usage frequencies. Here we have only studied a range of 3 to 37 programs for brevity in the model formulation. Naturally, a problem of this type which models a real world situation could grow to become quite large. Although there are methodologies for solving large problems of this type, we are only concerned here with the implications of the formulation.

It is the eventual purpose of this study to automate the allocation of programs to different storage media through the insight on program behavior gained through Simon's Model. The first step will be accomplished by organizations' willingness to study their usage frequencies. Through this type of effort, it will be possible to isolate changes in $\alpha$ and $\gamma$ corresponding to real world situations.

# CHAPTER 8 CONCLUSIONS AND FUTURE RESEARCH

In this dissertation, we have addressed a storage allocation problem which is found in large organizations. Although many methodologies exist for the allocation of data to different storage media, it has been found that organizations still use manual or ad hoc techniques for the storage of their programs. Several interesting results have been found in studying this problem.

First, we have shown that Simon's model of information usage is a valuable tool for studying the dynamic behavior of program usage frequencies. Considerable effort was taken in order to show the validity of this model and the inappropriateness of other conventional methods in forecasting program usage. Although Simon's Model has previously been used in order to model other scenarios such as library holdings, self-organizing heuristics, and buffer replacement algorithms, several changes have been made in order to make the model appropriate for our situation. The simulation study, using Simon's Model, was programmed to allow the viewing of usage concentrations and frequency distributions in a progressive manner. This provided a way to view the incremental changes in usage while varying the model parameters.

Secondly, the dissertation addressed the need to incorporate dynamic usage frequencies into the modeling process. Traditional methodologies for program storage decision-making considered only a static view of usage frequencies. This study viewed this assumption as a limiting condition. It was found that a dynamic version of Simon's Autoregressive Model could be formulated in order to model program usage scenarios. This involves representing the entry rate of "new" programs as an increasing function which in turn causes an increase in the "decay" of old programs. In using Simon's Model to study the nature of program usage in organizations, a scheme for evaluation and assessment was proposed.

Lastly, this dissertation provided a minimum cost model for hierarchical storage allocation using dynamic usage frequencies. This cost model was specially formulated for the allocation of programs. It was very interesting to study this cost optimization problem using our simulation results. This model is theoretical in the sense that the behavior of program usage has only been viewed according to the scenario of increasing $\alpha$ and decreasing $\gamma$. The model developed in this dissertation should be viewed as an initial attempt in the development of a dynamic reallocation model. It is possible in our scenario to predict the behavior of $\alpha$ and $\gamma$ by examining simulation results when these parameters

increase and decrease, respectively. Results show that the total number of programs will continue to increase as long as $\alpha$ continues to increase, and the "aging" of old programs is more dramatic if $\alpha$ is increasing. Our results have also shown that as $\alpha$ and $\gamma$ change, the usage concentrations will not remain stable. Intuitively, changing usage concentrations would imply changing usage profiles.

With these considerations in mind, we have provided a theoretical model which is adaptable to organizations through the capability of computing $\alpha$ and $\gamma$. The value of knowing the changes in these parameters can have an influence on the type of allocation scheme used. We have shown that our allocation model gives superior results to those in which only a static program usage profile is made. Therefore, we can presume that this allocation model would work well in other organizational situations where $\alpha$ is increasing and $\gamma$ is decreasing. What we do not yet know is the magnitude of cost differentiation under extreme cases of parameter variation. Different scenarios of the influence of $\alpha$ and $\gamma$ on each other should be explored in order to make adjustments to the allocation model. For instance, $\alpha$ decreasing and $\gamma$ increasing may require an allocation model which is subject to a different set of constraints.

Thus, it is appropriate to summarize the steps that an organization would take in order to implement the results of this dissertation.

1. Utilize the program usage frequency data which is available to the organization but is commonly unexamined. Through the methodologies discussed in this research, an organization has the ability to compute the value of two critical parameters. That is, the value of the entry rate ($\alpha$) of new programs and the value of the decay rate ($\gamma$) of old programs can be estimated.
2. Use the decision rule developed in Chapter 6 of this dissertation in order to categorize the organization's overall usage behavior. This will enable the organization to better understand the degree of dynamic behavior inherent in their collection of programs.
3. With the insight gained from the above two steps, an organization would have the ability to begin to isolate program usage scenarios. Simon's model provides much flexibility in studying the behavior of usage frequencies. It has been shown in this dissertation that Simon's model can be adapted to study a variety of usage situations which are observed.
4. Once an organization has determined the specific dynamics of their organization's program usage, an allocation model can be further explored. The data

generated from Simon's model can be used to evaluate the performance of various allocation algorithms. In this manner, many allocation algorithms could possibly be eliminated if their performance is poor under certain dynamic usage situations.

It is hopeful that the insight gained through this dissertation will lead to future research contributions. In particular there is the need to extend this study to the hierarchical storage of data. In designing and managing archival storage, the selection of data to be archived to a secondary storage media has great impact on the performance of database applications. Therefore, our initial findings concerning program allocation can be adjusted and extended to data storage management. Literature (Silberschatz et al., 1991) describes the next generation of database applications as having little in common with today's data processing databases. The use of multimedia and complex objects will cause databases to become tremendous in size. Thus, the algorithms currently used for DBMS operations may become obsolete. For these reasons, the study of organizational information scenarios using Simon's Model will continue due to its flexibility and insight in studying usage phenomena.

## REFERENCES


Abele, D., "SIDF Finds Common Ground for LAN Backup," *Network World*, Vol. 11, Iss. 39, September 26, 1994, pp. 48.

Boehm, B. W., "Improving Software Productivity," *Computer*, September 1987, pp. 43-57.

Boyd, D., "Implementing Mass Storage Facilities in Operating Systems," *Computer*, Vol. 11, Iss. 2, February, 1978, pp. 40-45.

Brisse, M., "Hierarchical Storage Management," Intergraph Corporation Report, January, 1996.

Burrell, Q. L., "A Simple Stochastic Model for Library Loans," *Journal of Documentation*, Vol. 36, 1982, pp. 115-132.

Busse, T., "SMART Monitors LAN Software Use," *Infoworld*, Vol. 16, Iss. 19, May 9, 1994, p. 52.

Chen, P., "Optimal File Allocation in Multi-level Storage Systems," *AFIPS Conference Proceedings*, Vol. 4, 1973, pp. 277-282.

Chen, P. and G. Mealy, "Optimal Allocation of Files with Individual Response Time Requirements," *Proceedings of the Seventh Annual Princeton Conference on Information Sciences and Systems*, Princeton University, March 1973.

Chen, Y. S., "Analysis of Lotka's Law: The Simon-Yule Approach," *Information Processing & Management*, Vol. 25, No. 5, 1989, pp. 527-544.

Chen, Y. S., "An Exponential Recurrence Distribution in the Simon-Yule Model of Text," *Cybernetics and Systems: An International Journal*, Vol. 19, 1988, pp. 521-545.

Chen, Y. S., P. P. Chong, and M. Y. Tong, "Theoretical Foundation of the 80/20 Rule," *Scientometrics*, Vol. 28, No. 2, 1993, pp. 183-203.

Chen, Y. S., and F. F. Leimkuhler, "Booth's Law of Word Frequency," *Journal of the American Society for Information Science*, Vol. 41, No. 5, 1990, pp. 387-388.

Chen, Y. S., and F. F. Leimkuhler, "A Relationship Between Lotka's Law, Bradford's Law, and Zipf's Law," *Journal of the American Society for Information Science*, September 1986, pp. 307-314.

Chong, P. P., "On Information Usage Modeling," Ph.D. Dissertation, Louisiana State University, 1993.

Chu, C. H., and Y. C. Chu, "Computerized ABC Analysis: The Basis for Inventory Management," *Computers & Industrial Engineering*, Vol. 13, Nos. 1-4, 1987, pp. 66-70.

Chu, W. and H. Operbeck, "Program Behavior and the Page Fault Frequency Replacement Algorithm," *IEEE Computer*, November, 1976, pp. 29-38.

Ferrill, P., "Frye's Expanded Network Management Utilities Really Shine," *Infoworld*, Vol. 16, Iss. 24, 1994, pp. 88-90.

Flores, B. E., D. L. Olson, and V. K. Dorai, "Management of Multicriteria Inventory Classification," *Mathematical Computer Modeling*, Vol. 16, No. 12, 1992, pp. 71-82.

Flores, B. E., and D. C. Whybark, "Implementing Multiple Criteria ABC Analysis," *Engineering Costs and Production Economics*, Vol. 15, 1988, pp. 191-195.

Flores, B. E., and D. C. Whybark, "Know Your ABC," *Management Decision*, Vol. 26, No. 3, 1985.

Ghandeharizadeh, S., D. Ierardi and R. Zimmermann, "Management of Space in Hierarchical Storage Systems," USC Department of Computer Science Technical Paper, September 29, 1994.

Goodhue, D. L., J. A. Quillard, and J. F. Rochart, "Managing the Data Resource: A Contingency Perspective," *MIS Quarterly*, 12(2), June 1988, pp. 12-17.

Iida, J., "Stretching Your Storage Capacity," *Communications Week*, Iss. 563, June 26, 1995, pp. 73-75.

Ijiri, Y., and H. A. Simon, *Skew Distributions and the Sizes of Business Firms*, North-Holland, 1977.

Imagery Software, "Hierarchical Storage Management," Market White Paper, July 1995.

Kay, R. L., "What's the Meaning of This?!" *Computerworld*, October 17, 1994, pp. 89-93.

Kendall, M. G., "The Bibliography of Operations Research," *Operations Research Quarterly*, Vol. 11, 1960, pp. 31-36.

Kent, A., *Use of Library Materials: The University of Pittsburgh Study*, New York: Marcel Dekker, 1979.

Kim, Jin-Soo, "Usage-dependent Information Systems Design," Ph.D. Dissertation, Louisiana State University, 1990.

Korgeniowski, P., "Desperately Seeking Storage Solutions," *Datamation*, August 15, 1994, pp. 62-64.

Lancaster, F. W. and J. L. Lee, "Bibliometric Techniques Applied to Issues of Management: A Case Study," *Journal of the American Society for Information Science*, 36(6): pp. 389-397, 1985.

Leimkuhler, F. F., "On Bibliometric Modeling," *Informetrics*, 1988, pp. 97-104.

Lum, V., M. Senko, C. Wang and H. Ling, "A Cost Oriented Algorithm for Data Set Allocation in Storage Hierarchies," *Communications of the ACM*, Vol. 18, No. 6, June, 1975, pp. 318-322.

Mangold, G., "HSM and Backup Products Differ by Design," Avail Systems White Market Paper, August 8, 1994.

Martin, J., *Programming Real Time Computer Systems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1965.

Martin, D. and G. Estrin, "Models of Computations and Systems," *Journal of the Association for Computing Machinery*, Vol. 14, 1967.

Merenbloom, P., Software Monitoring and Metering ... Do It Right or Pay," *Infoworld*, Vol. 15, Iss. 17, April 26, 1993, p. 47.

Morgan, H. and D. Levin, "Optimal Program and Data Locations in Computer Networks," *Communications of the ACM*, Vol. 20, Iss. 5, May 1977, pp. 315-322.

Nash, K. S., "Keeping Pace with CASE Philosophy," *Computer World*, August 17, 1992, pp. 81-83.

Neuts, M. F., "An Algorithmic Probabilist's Apology," in *The Craft of Probabilistic Modeling*, Springer-Verlag, New York, 1986.

Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 3rd Ed., McGraw-Hill, Inc., New York, 1992.

Ramamoorthy, C., "Discrete Markov Analysis of Computer Programs," *Proceedings ACM 20th National Conference*, ACM Pub. P-65, 1965, pp. 386-391.

Ramamoorthy, C. and K. Chandy, "Optimization of Memory Hierarchies in Multiprogrammed Systems, *Journal of the Association for Computing Machinery*, Vol. 17, No. 3, July 1970, pp. 426-445.

Reich, D., "Page Fault Model of Staging for Mass Storage Volumes," *IBM Research Report*, RC 7430, November 30, 1978.

Reinhardt, A., "Managing Storage," *BYTE*, June, 1994.

Ronen, B. and I. Spiegler, "Information as Inventory: A New Conceptual View," *Information and Management*, Vol. 21, 1991, pp. 239-247.

Ryan, A., "Kick the Hard Disk Habit: Add Hierarchical Storage," *Datamation*, Vol. 40, Iss. 23, pp. 62-64.

Sanders, R., "The Pareto Principles: Its Use and Abuse," *The Journal of Services Marketing*, Vol. 1, No. 2, Fall 1987, pp. 37-40.

Silberschatz, A., M. Stonebraker and J. Ullman, "Database Systems: Achievements and Opportunities," *Communications of the ACM*, Vol. 34, No. 10, October, 1991.

Simon, H. A., *Models of My Life*, Basic Books (Harper Collins), 1991.

Simon, H. A., "On Judging the Plausibility of Theories," in *Logic, Methodology and Philosophy of Sciences*, Vol. III, Amsterdam: North-Holland, 1968.

Simon, H. A., "On a Class of Skew Distribution Functions," *Biometrika*, Vol. 42, 1955, pp. 425-440.

Simon, H. A., and T. A. Van Wormer, "Some Monte Carlo Estimates of the Yule Distribution," *Behavior Science*, Vol. 1, No. 8, 1963, pp. 203-210.

Smith, A., "Analysis of the Optimal Look-ahead, Demand Paging Algorithms," *SIAM Journal of Computing*, Vol. 5, Iss. 4, December 1976, pp. 743-757.

Smith, A., "Bibliography on Paging and Related Topics," *Operation Systems Review*, Vol. 12, Iss. 4, October 1978, pp. 39-56.

Smith, A., "Long Term File Reference Patterns and Their Application to File Migration Algorithms," *IEEE TSE*, 1981.

Stritter, E. "File Migration," Stanford Computer Science Report, STAN-CS-77-594, Ph.D. Dissertation, January, 1977.

Swanson, E. B., "The Information Loop as a General Analytic View," *Information and Management, Vol. 20, 1991, pp. 37-47.*

Tong, Y. M., "A Self-Adaptive Database Buffer Replacement Scheme," Ph.D. Dissertation, Louisiana State University, 1994.

Willet, S., "Running Out of Room?" *Infoworld*, Vol. 16, Iss. 31, pp. 49-52.

Yourdon, E., *Decline and Fall of the American Programmer*, Englewood Cliffs, NJ: Prentice Hall, 1992.

Zunde, P., "Empirical Laws and Theories of Information and Software Sciences," *Information Processing and Management*, Vol. 20, No. 1-2, 1984, pp. 5-18.

Zimmerman, G. W., "The ABC's of Vilfredo Pareto," *Production and Inventory Management*, 3rd Qtr. 1975, pp. 1-9.

**APPENDIX A**

```
Program Simonc(Output);
{$M 65520,0,655360}
Uses Crt,DOS;
(*********************************************************)

(*       Simon's Third Model with Serial Correlation     *)


(*              CONSTANT ENTRY RATE (ALPHA)              *)
(*      INCREASING ENTRY RATE WITH DECREASING GAMMA      *)




Const
   (*   R = 20000; *) (* NUMBER OF ITERATIONS PER CYCLE *)
   (*   RC = 1;    *) (* NUMBER OF CYCLES EACH RUN *)
      MAX = 16000;     (* MAXIMUM NUMBER OF Programs EACH
                          CYCLE *)
      MAXINDEX = 800; (* MAXIMUM RANK INDEX NUMBER *)
   (*   ALPHA = 0.20; *)   (* PROBABILITY OF NWE ENTRY *)
   (*   GAMMA = 0.85;   *)(* GEOMETRIC RATE OF DIE OUT *)

      SEED1 = 123;
      SEED2 = 12345;
      SEED3 = 78901;
      SEED4 = 965431;

      SEED = SEED1;    (* Random number Seed *)

      MinReal = 2.9E-39;
Label 99,100;
Type
      Prog_Rec = Record
                   Fid:Integer;
                   S:Integer;
                   W:Real
                 End;
      PRecord =^Prog_Rec;
      PArray=Array[1..Max] of PRecord;
      AMAXI = ARRAY[0..MAXINDEX] OF Longint;
      STR80=String[80];
```

```
Var
      NMAX:Integer;
      R,RC:Integer;
      Ratio,Alpha,Alpha0,Gamma:Real;
      Over,Change:Boolean;
      PUnit:PArray;
      A,B,CUMULANT:Real;
      nk:INTEGER;          (* No. of Programs currently *)
      aa:INTEGER;          (* Executions allocated initially
*)
      I,J,TEMP:INTEGER;    (* Loop counter and Temp Var *)
      c,ST,T:LONGINT;         (* cycle # and asste range
allocated *)
      K:Longint;           (* aggregated asset size *)
      MINREAL1:REAL;
      WEIGHTSUM:Real;      (* weight Sum of All Programs *)
      OutFile:Text;
      Hours1,Minutes1,Seconds1,Hundredths1:Word;
      Hours2,Minutes2,Seconds2,Hundredths2:Word;
(*********************************************************)
Procedure ShowTime;
Var Minutes,Time1,Time2:Real;
Begin (* ShowTime *)

Time1:=(Hours1*3600.0)+(Minutes1*60.0)+Seconds1+(Hundredth
s1/100);

Time2:=(Hours2*3600.0)+(Minutes2*60.0)+Seconds2+(Hundredth
s2/100);
   Minutes:=(Time2-Time1)/60;
    Writeln(OutFile,'    Execution    Time    for
Program=',Minutes:8:2,' minutes.')
End; (* ShowTime *)
(*********************************************************)
Procedure Page(S:Str80);
Begin
   Clrscr;
   Gotoxy(20,5); Writeln(S);
   Gotoxy(20,6); Writeln('Simulation in Progress');
   Gotoxy(20,7); Writeln('Your Patience Required');
   (* Repeat
      Until KeyPressed *)
End;
```

```
(*************************************************************)
Procedure Initialize;
(*                                                           *)
(*  Initialize number of Programs with size and weight       *)
(*                                                           *)
Var
   i:Integer;
Begin  (* Initialize *)
   GetTime(Hours1,Minutes1,Seconds1,Hundredths1);
   TextBackGround(Blue);
   TextColor(Yellow);
   Assign(OutFile,'AlpSimon.out');
   Rewrite(OutFile);
   RandSeed:=Seed;
   A:=Random;
   MinReal1:=100*MinReal;


   Repeat
      GOTOXY(20,5);
      Write('Enter Value of NMAX Less than or equal to
',MAX,':');
      Readln(NMAX)
   Until (NMAX>0) and (NMAX<=MAX);

   Repeat
      GOTOXY(20,7);
      Write('Enter Value of Alpha:');
      Readln(Alpha)
   Until (Alpha>0) and (Alpha<1);



(*   Repeat
      GOTOXY(20,9);
      Write('Enter Value of Gamma:');
      Readln(Gamma)
   Until (Gamma>0) and (Gamma<1); *)

   Repeat
      GOTOXY(20,11);
      Write('Enter number of iterations:');
      Readln(R)
   Until R>0;

   Repeat
      GOTOXY(20,13);
      Write('Enter number of cycles:');
      Readln(RC);
```

```
    Until RC>0;

    nk:=3;

    (* Alpha := 1 - nk/NMAX; *)
    Ratio:=nk/NMAX;
    Gamma:=1-Ratio;

    For i:=1 to 3 Do
       Begin
          New(PUnit[i]);
          PUnit[i]^.S:=i;
          PUnit[i]^.W:=1;
          PUnit[i]^.Fid:=i
       End;

    WEIGHTSUM :=0;
    AA := 0;

    For i:= 1 to nk Do
       Begin
          AA := AA + PUnit[i]^.S;
          WEIGHTSUM := WEIGHTSUM + PUnit[i]^.W
       End;

    For i:=(nk+1) to Max Do
       PUnit[i]^.W:=0;
    End;
(*********************************************************)
Procedure GetA80(Var F,G:AmaxI; IndexNow:Integer);

(* Calculate The concentration Measure of *80/20 Rule *)

Var i:Integer;
   A80:Real;

Begin (* GetA80 *)
   A80 := 0;
   For I:= 1 to IndexNow Do
      A80 := A80 + (G[I]+G[I-1])*(F[I]-F[I-1]);

   A80:=A80/(2*T*nk) - 0.5;

   Writeln(OutFile,' ');
   Writeln(OutFile,' ');
   Writeln(OutFile,' AREA OF CONCENTRATION OF 80/20 RULE
==== ', A80:6:4);
   Writeln(OutFile,' ');
   Writeln(OutFIle,'Mu===',T/nk:10:4);
```

```
   Writeln(OutFile,' ');
   Writeln(OutFIle,'M===',IndexNow:4);
   Writeln(OutFile,' ')
End; (* GetA80 *)
(*********************************************************)
Procedure SortOut( Nof:Integer);
(* Output current firm size and weight information in
variuos formats *)

Var i,j,Temp:Integer;
    PTemp:PRecord;
    CF,CG:AMAXI;
    IndexNow:Integer;
    RTemp:Real;
Begin (* SortOut *)
   Writeln(OutFile,' ');
   Writeln(OutFile,' ');
   Writeln(OutFile,'                                   *** PROGRAM
EXECUTIONS AND WEIGHT LIST ***   ');
   Writeln(OutFile,' ');
   Write(Outfile,'- - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - -');
   Writeln(OutFile,'- - - - - - - - - - - ');
   Writeln(Outfile,'          IN ENTRY ORDER
         IN DESCENDING');
   Write(Outfile,'- - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - -');
   Writeln(OutFile,'- - - - - - - - - - - ');

 (*  For i:=1 to Nof Do
     Fid[i]:=i;  *)
                                  (* Keep Track of Initial
Firm Id *)
   For i:= 1 to (Nof-1) Do       (* Sort Firm Based on Size
*)
      For j:=i+1 to Nof Do
         If PUnit[j]^.S > PUnit[i]^.S Then
            Begin (* Swap *)
               PTemp := PUnit[i];       (* Swap Pointers *)
               PUnit[i]:=PUnit[j];
               PUnit[j]:=PTemp;

            End;
      (* Keep Output up to first 50 Programs *)
      If Nof > 50 Then
         j:=50
      Else
         j:=Nof;
      (* Output the final Programs *)
```

```
      For i:= 1 to j Do
         Begin
            With PUnit[i]^ Do
               Begin

Write(Outfile,'S(',i:3,')=',S:5,';W(',i:3,')=',W:7:4);
                  Write(Outfile,'      S(',Fid:5,') =');
                    Writeln(Outfile,PUnit[i]^.S:5,';
W(',Fid:5,')=',
                                        PUnit[i]^.W:7:4)
               End
         End;
      (* Page(' Output for final Programs Done'); *)
      Writeln(OutFile,' ');
      Writeln(OutFile,' ');
      Writeln(OutFile,' ');
      Write(Outfile,'RANK    USAGE     # OF  PROGRAM  TOTAL
WEIGHT    ');
      Writeln(OutFile,'INC. P. IF NOT NEW');
      Write(Outfile,'----       ----          ------------
------------    ');
      Writeln(OutFile,'------------------ ');
      J:=1;
      Temp:=1;
      Rtemp:=PUnit[1]^.W;
      CG[0]:=0;
      CF[0]:=0;
      For i:=2 to Nof Do
         Begin (* i loop *)
            If PUnit[i-1]^.S > PUnit[i]^.S Then
               Begin
                  Write(Outfile,j:3,' ',PUnit[i-1]^.S:6,'
 ',Temp:8,RTemp:16:4);
                  If RTemp > (MinReal*WeightSum) Then
                     Writeln(OutFile,RTemp/WeightSum:16:4)
                  Else
                     Begin
                        RTemp:=0;
                        Writeln(OutFile,RTemp:16:4)
                     End;
                  CG[j]:=CG[j-1]+PUnit[i-1]^.S*Temp;
                  CF[j]:=CF[j-1]+Temp;
                  j:=j+1;
                  Temp:=1;
                  RTemp:=PUnit[i]^.W
               End
            Else
              Begin
                 Temp:=Temp+1;
```

```
                RTemp:=RTemp+PUnit[i]^.W
              End
          End; (* i loop *)
       Write(Outfile,j:3,'     ',PUnit[i-1]^.S:6,'
',Temp:8,RTemp:16:4);
       Writeln(OutFile,RTemp/WeightSum:16:4);
       CG[j]:=CG[j-1] + PUnit[Nof]^.S*Temp;
       CF[j]:=CF[j-1] + Temp;
       GetA80(CF,CG,j);
End; (* SortOut *)
(*********************************************************)
Procedure CleanUp(nk:Integer);
Var i:Integer;
Begin (* CleanUp *)
   For i:=1 to nk Do
      Dispose(PUnit[i])
End; (* CleanUp *)

(*********************************************************)
Begin (* Main *)
   Change:=True;
   Over:=False;
   ClrScr;
   Initialize;
   Alpha0:=Alpha;
   Page(' Initialization Done');
   Writeln(OutFile,'       ');

Writeln(OutFile,'=========================================
============== ');
   Writeln(OutFile,'       ');
   Writeln(OutFile,'       ALPHA = ',ALPHA:9:8,'      GAMMA
= ',GAMMA:5:4);
   Writeln(OutFile,'       ');
   Writeln(OutFile,'       ');
   Writeln(OutFile,'   CYCLE SIZE = ',R:6,'; # OF CYLCE EACH
RUN=',RC:2);
   Writeln(OutFile,'       ');
   Writeln(OutFile,'       ');
   Writeln(OutFile,'            *** INITIAL CONDITIONS ***
');
   Writeln(OutFile,'       ');
   Writeln(OutFile,'       ');
   Writeln(OutFile,'         RANDOM # GENERATOR SEED = ',
SEED:6);
   Writeln(OutFile,'       ');
   Writeln(OutFile,'       ');
   Writeln(OutFile,'     NUMBER OF PROGRAMS=   ', NK:3,'
SOFTWARE EXECUTIONS =',AA:4);
```

```
   Writeln(OutFile,'        ');
   Writeln(OutFile,'        ');

Writeln(OutFile,'========================================
==============');

   st := AA + 1;     (* Assign Initial starting asset *)
   T:= 0;

   For c:= 1 to rc Do
      Begin (* C Loop *)
         GotoXY(20,8);
         Writeln('Cycle Number=',c:2);
         T := T + R; (* Loop start / End # of this cycle *)
         For k := st to T Do
            Begin
               If ((K mod 2000)=0) Then
                  Begin
                    Gotoxy(20,9+(k div (2000*c)));
                    Writeln('Cycle  Count=',K:4,'  out  of
',c*R:5)
                  End;
               A := Random;  (* Random number from 0 to 1 *)
(*************************************************************)
(*
               *)
(*           New Algorithm introduced to define Alpha and
Gamma as          *)
(*         functions of Ratio = nk/NMAX
               *)
(*         Alpha becomes a decreasing function of Ratio
               *)
(*          While Gamma becomes an increasing function of
Ratio             *)
(*
               *)
(*************************************************************)
          (*  New statements introduced next  *)
               If Change Then
                  Begin
                     (* Ratio:=nk/(20*NMAX); *)
                     Ratio:=nk/NMAX;
                     (*  Alpha :=Alpha*(1 - Ratio);  *)
                     Gamma:=1-Ratio;
                     (* Gamma:=(1-Gamma)*Ratio+Gamma;*)
                     Alpha:=Sqrt(Ratio)+Alpha0;
                     If Alpha >= 1 Then
                        Over:=True;
                     Change:=False;
```

```
                End;
             If Over Then Goto 99;
             If A <= Alpha Then
                Begin       (* Allocate new program *)
                   For i:=1 to nk Do
                      If PUnit[i]^.W > MinReal1 Then (*
Prevent Data Underflow *)
                         PUnit[i]^.W:=PUnit[i]^.W*Gamma;
                   nk := nk + 1;
                   Change:=True;
                   If nk > MAX Then
                      Begin
                          ClrScr;
                          Writeln(' Overflow in the
                          number of programs with
                          nk=',nk:1);
                          Writeln('Press Return Key to
                          end program');
                          Readln;
                          Exit
                       End;
                   New(PUnit[nk]);
                   PUnit[nk]^.s:=1;
                   PUnit[nk]^.W:=1;
                   PUnit[nk]^.Fid:=nk;
                End
             Else
                Begin (* else *)
                   B := Random*WeightSum;
                              (* Standardized B *)
                   Cumulant := 0;
                   j := 0;
                   Repeat
                      j := j + 1;
                      CumuLant := Cumulant + PUnit[j]^.W;
                   Until Cumulant >=B;
                   For i:= 1 to nk Do
                      If PUnit[i]^.W > MinReal1 Then
                                    PUnit[i]^.W    :=
PUnit[i]^.W*Gamma;
                   PUnit[j]^.W := PUnit[j]^.W + 1;
                   PUnit[j]^.S := PUnit[j]^.S + 1
                End;  (* else *)
             (*Writeln('WeightSum=',WeightSum:10:5);
             Readln;  *)
             If Gamma >0 Then
                WeightSum := WeightSum*Gamma + 1;
          End; (* K Loop *)
       99:Gotoxy(20,10+(k div 2000));
```

```
        Writeln('Cycle Loop ',C:2,' complete with ',R:5,'
iterations.');
        Writeln(OutFile,'Cycle Loop ',C:2,' complete with
',R:5,' iterations.');
        Writeln(OutFile,'Alpha=',Alpha:9:8,' NMax=',NMax:3,'
nk=',nk:1,' ','Gamma=',Gamma:9:8);
        Writeln(OutFile,'        ');

Write(Outfile,'----------------------------------------
------------');
        Writeln(OutFile,'-------- ');
        Write(Outfile,'    CURRENT  PERIOD  ENDING  AFTER
EXECUTIONS ALLOCATED =  ');
        Writeln(OutFile,T:6);
        Writeln(OutFile,'       ');
        Writeln(OutFile,'       CURRENT  TOTAL  NUMBER  OF
PROGRAMS = ',NK:5);

Write(Outfile,'----------------------------------------
------------');
        Writeln(OutFile,'-------- ');
        If Over Then
           Writeln(OutFile,'      SIMULATION HALTED SINCE
ALPHA >=1');
        Gotoxy(20,12+(k div 2000));Writeln('Sort Out in
Progress.');
        Writeln(OutFile,^L);
        SortOut(NK);
        ST := 1 + T;
        ClrScr;
        Page(' Cycle Done');
        If Over then goto 100
    End; (* For  c Loop *)
    100:GetTime(Hours2,Minutes2,Seconds2,Hundredths2);
    ShowTime;
    Close(OutFile);
    CleanUp(nk);
End. (* Main *)
```

ALL SMFII ANALYSIS FOR SYSTEM #1285
RANGE: 00:01:00 10/30/94 - 23:30:00 10/30/94

MOST HEAVILY USED PROGRAMS

| TASK NAME | AVERAGE PROCESSOR TIME | AVERAGE I/O TIME | AVERAGE MEMORY (CODE+DATA) | COUNT |
|---|---|---|---|---|
| *LIBRARY/MAINTENANCE | 0.2 | 0.7 | 27368.2 | 13 |
| BLDIST/ENACOPY | 0.0 | 0.1 | 1018.9 | 13 |
| *SYSTEM/JOBFORMATTER | 0.1 | 0.1 | 14978.5 | 12 |
| *OBJECT/BLDIST/NOTIFIER | 0.8 | 0.6 | 20186.0 | 8 |
| (CHARGE)DMSUPPORT/BLCHARGEDB ON BANDL | 0.5 | 0.0 | 19690.5 | 6 |
| *SYSTEM/SDASUPPORT | 0.1 | 7.0 | 39415.6 | 5 |
| *SYSTEM/HWERRORSUPPORT | 0.0 | 0.0 | 728.3 | 4 |
| *SYSTEM/LOGANALYZER | 67.4 | 27.6 | 45147.5 | 4 |
| (CHARGE)BLCHARGEDB | 1.3 | 1064.5 | 456964.0 | 3 |
| (PROD)CCNTROLCARD | 0.4 | 0.2 | 30052.7 | 3 |
| (PROD)TOOLS/OBJ/SDSANAL ON UCFO2 | 5.4 | 3.8 | 11005.0 | 3 |
| *OBJECT/BLPACK/DIRINTERFACE | 0.0 | 0.0 | 6030.5 | 2 |
| *SYSTEM/SMFII/QUERY | 57.1 | 50.0 | 95618.5 | 2 |
| (BLDIST)OBJECT/SOURCE/BLDIST/PAGECT ON BANDL | 2.0 | 0.2 | 7995.0 | 1 |
| (BLSCHED)OBJECT/SCHED/SCHEDULER ON BANDL | 10.4 | 1.6 | 20486.0 | 1 |
| (BLSCHED)OBJECT/SCHED/UPDATE ON BANDL | 6.2 | 2.3 | 95044.0 | 1 |
| (CHARGE)OBJECT/BLCHARGE/BILLER ON BANDL | 11722.1 | 6233.3 | 135718.0 | 1 |
| (CHARGE)OBJECT/BLCHARGE/COMSEXTRACT ON BANDL | 0.1 | 27.6 | 11929.0 | 1 |
| (CHARGE)OBJECT/BLCHARGE/LOGGER ON BANDL | 142.8 | 84.7 | 33369.0 | 1 |
| (CHARGE)OBJECT/BLCHARGE/ONLINELOGGER ON BANDL | 110.1 | 54.2 | 17417.0 | 1 |
| (CHARGE)OBJECT/BLPACK/DIRINTERFACE ON BANDL | 0.1 | 0.1 | 6201.0 | 1 |
| (CHARGE)OBJECT/RELEASETRAIL ON BANDL | 0.2 | 0.1 | 2003.0 | 1 |
| (PROD)KEYEDIO/LIBRARY906 | 0.0 | 0.1 | 19765.0 | 1 |
| (PROD)NRB/OBJ/XGEN/W500SATF ON UCFO2 | 1963.5 | 2.9 | 36324.0 | 1 |
| (PROD)NRO/OBJ/XGEN/ENTITY ON UCFO2 | 2.7 | 3.1 | 19943.0 | 1 |
| (PROD)SRC/OBJ/VOICELINK3 ON UCFO2 | 165.3 | 3.3 | 14435.0 | 1 |
| *POKE | 0.1 | 0.1 | 2275.0 | 1 |

## VITA

The author was born in Thibodaux, Louisiana in 1966 and was raised in the town of Napoleonville, Louisiana. She is the younger daughter of Dr. and Mrs. Raymond Folse and is the wife of Robert F. Cope, III. She graduated from Edward Douglas White Catholic High School in 1984. Subsequently she graduated from Nicholls State University in December of 1987, receiving a bachelor of science degree in Computer Science and Applied Mathematics. Upon completion of her undergraduate degree she attended Tulane University from Fall 1988 to Spring 1989 to pursue graduate work in mathematics. After that period of time she was employed by Pan American Life Insurance Company in the Group Actuarial Division. In Summer 1990, she returned to graduate school at Louisiana State University in order to pursue a doctorate in business administration majoring in Information Systems and Decision Sciences in the area of Management Information Systems. She has been employed by Southeastern Louisiana University since Fall 1993 as a Visiting Assistant Professor in the Department of Management and teaches Management Information Systems and Business Statistics.

## DOCTORAL EXAMINATION AND DISSERTATION REPORT

**Candidate:** Rachelle Folse Cope

**Major Field:** Business Administration (Information Systems and Decision Sciences)

**Title of Dissertation:** Dynamic Storage Allocation Using Simon's Model of Information Usage

**Approved:**

Dr. Ye-Sho Chen
**Major Professor and Chairman**

Dr. Daniel Fogel
**Dean of the Graduate School**

**EXAMINING COMMITTEE:**

Dr. Ye-Sho Chen

Dr. Kwei Tang

Dr. Ishwar Murthy

Dr. Peter Kelle

Dr. Dennis Webster

**Date of Examination:**

December 15, 1995