CMU-CS-81-114

A Study of File Sizes and Lifetimes

M. Satyanarayanan

DEPARTMENT OF COMPUTER SCIENCE

CARNEGIE-MELLON UNIVERSITY

April 1981

Abstract

An investigation of the size and lifetime properties of files on the primary computing facility in the Department of Computer Science at Carnegie-Mellon University is presented in this paper. Three key issues are examined: the effect of migration on file characteristics, the effect of file type on file characteristics, and the correlation between file sizes and lifetimes. Analytical models that fit the observed data are derived using two alternative techniques.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

Table of Contents

1. Introduction

The performance of a file system depends strongly on the characteristics of the files stored in it. This paper discusses the collection, analysis and interpretation of data pertaining to files in the computing environment of the Computer Science Department at Carnegie-Mellon University (CMU-CSD). The information gathered from this work will be used in a variety of ways:

1. As a data point in the body of information available on file systems.

2. As input to a simulation or analytic model of a file system for a local network, being designed and implemented at CMU [1].

3. As the basis of implementation decisions and parameters for the file system just mentioned.

4. As a step toward understanding how a user community creates, maintains and uses files.

2. Data Collection

2.1. The Environment

The data used in this paper was obtained on a Digital Equipment Corp. PDP-10 Model KL-10 processor [5] with 1 Mword of primary and eight 200 Mbyte disk drives, running the TOPS-10 operating system [8]. This machine has been the main computational resource of the CMU-CSD for the past five years. Towards the end of this period, a number of other machines were added to this environment. Though the machine used for this study is now off-loaded by those machines, it continues to play a very important role and is still heavily used. Consequently, I believe that the data presented here is a good reflection of the file usage characteristics of this community.

2.2. The File System

In the TOPS-10 operating system, every file has a 6-character *file name* and a 3-character *file extension*, and is a member of exactly one *directory*. The file extension indicates the nature of the contents of a file. For example, a Pascal program source would have the extension PAS, while its relocatable object module would have the extension REL. An installation-dependent number of extensions are regarded as "standard" extensions. Though system and user programs often make assumptions about a file based on its extension, there is no mechanism for validating or guaranteeing these assumptions. In practice, it is extremely rare that a standard extension is used for non-standard purposes. A quarter of the files examined had non-standard extensions; such files were ignored for those parts of this study that discriminated on the basis of file type. File names, unlike extensions, have no system-wide significance and were not examined.

A file consists of a sequence of fixed-length *blocks*, which are the units of addressability on the disks. Each block consists of 128 36-bit words. The last block in a file may be only partially written; such blocks were regarded, in this study, as whole blocks. The size of a file is limited only by the amount of secondary storage available. Unlike some file systems, such as OS/VS2 for the IBM 370 [4], a user does not have to specify the maximum size of a file at the time of its creation.

The operating system maintains, for each file, information regarding its size, its owner, the date it was last written, the date it was last accessed and its physical storage map. This information may be obtained by queries from user programs to the operating system.

In the environment in which this study was done, a manual file migration scheme is used to relieve the paucity of disk space. Every month, the operations staff runs a program which copies onto magnetic tape, and deletes from disks, those files which have neither been written nor read in the preceeding three months. Files so migrated may be restored to disk at the request of their owners; in practice, very few such requests are received. Each user has a file named MIGRAT.DIR to which the migration program appends details of every file of that user it migrates. The union of a user's current directory and his MIGRAT.DIR entries constitutes the set of all files created, but not deleted, by that user.

2.3. The Collection Technique

The files in this study fall into two classes: *current files* and *migrated files*. Data for both classes were obtained without any modifications to the operating system. A vendor-supplied utility program which creates a file containing details of every other file in the system was used to obtain data on current files. Data on migrated files was obtained by examining the MIGRAT.DIR file of every user in the system. For both clases the data extracted was organized as a 3-dimensional array with logarithmic age histogram buckets on one dimension, logarithmic size histogram buckets on another dimension, and the set of standard file extensions on the third. This array was created once each for current and migrated files, recorded in a file, and used as a database for software written to answer questions such as "What is the distribution of file sizes for current files with ages in *a given range* and with *a given set of extensions*." Table 6-1 shows an example of the output for one such query.

It should be noted that the data gathered by this method is a snapshot of the file system at one point in time. To examine the temporal behavior of the file properties described here, one would have to take snapshots spaced apart in time and compare the data from each.

2.4. The Quantities Measured

Probably the three most common questions asked about any file are:

1. "What does it contain?"

2. "How big is it?"

3. "How old is it?"

To the designer of a file system, the first question is probably only of marginal relevance. In any case, a precise answer to it requires a complete specification of the contents of a file! Specifying the extension of a file answers this question at one level of granularity. One outcome of this study is, therefore, a histogram of file extensions for any cross-section of the set of files examined. Figure 6-1 shows such a histogram. The integers on the abscissa are mappings from the set of extensions to integers, as defined by Tables 6-2 and 6-3.

The size distribution of files is a crucial factor in deciding many of the file system parameters. The size of a file, measured in blocks, is one of the two quantities of primary interest in this study.

The other important quantity is the age of a file. "Age" is usually understood to mean the interval between the creation of a file and the instant of data collection. However, the original date of creation of a file is not maintained by TOPS-10; only the dates of last modification and last access are available. The difference between these two dates is a measure of the usefulness of the current data in the file. This quantity, the useful lifetime of a file, is the second quantity of interest in this study. For brevity, "lifetime" will mean "useful lifetime" in the rest of this paper. Fortuitously, it is the lifetime of a file, not its chronological age, which is important in the design of file migration algorithms. Further, the file system design described in [1] and referred to in Section 1 is predicated on the assumption that the lifetime of files is short — this study was conducted, in part, to verify this assumption.

3. Data Interpretation

3.1. General Observations

A total of about 36,000 current files and 50,000 migrated files were examined in this study. About 99% of the files examined had sizes less than 1000 blocks and lifetimes less than 2000 days. Both size and lifetime are discrete variables, with minimum values of 1 block and 1 day respectively. However, for ease of data interpretation and analytical approximation, both variables are treated as continuous variables.

Even a cursory examination of the data reveals some interesting facts. As Figure 6-2 indicates, the size

distribution is skewed towards small sizes: 50% of the files are less than 5 blocks long and 95% of them are less than 100 blocks long. Figure 6-3 shows that the lifetime distribution is also skewed towards the low end, though not as sharply as the size distribution. Nearly 30% of the files have lifetimes of one day and 50% of them have lifetimes less than 30 days. The rest of the data analysis discusses three questions:

1. Are the properties of migrated files different from those of current ones?

2. Does the type of a file affect its properties?

3. Does the size of a file influence its lifetime?

3.2. Effect of Migration

Figure 6-4 compares the size distributions of current and migrated files. Except at the very low end, there is virtually no difference between the curves. At the low end, there are fewer migrated files than current files. I conjecture that this phenomenon is due to the following: a large number of very short files are created by system programs. Text editors and mail servers are two examples of programs which create short auxiliary files which are used only once. These files are automatically deleted by the programs which created them when they are run a second time, or by users when they run out of disk quotas. Such files are unlikely to remain both unaltered and undeleted for a period of time long enough to qualify them for migration. Consequently, small files are likely to form a smaller fraction of the migrated population than the current population.

Figure 6-5 shows that migrated files tend to have shorter lifetimes than current files. To see why this is so, consider how a long-lifetime file gets migrated. It would have to get created, then read (but not written) frequently for a long time and then all accesses to it would have to stop for a period long enough for it to qualify for migration. The only obvious files that meet these criteria are the successive versions of commonly used system or user programs. The infrequency of generation of such files leads to the fact that there are fewer long-lifetime files in the migrated population than in the current population.

The rest of this paper discusses only current files. Unless otherwise specified, the comments about current files also hold for migrated files with, perhaps, slightly different absolute numbers.

3.3. Effect of File Type

Since it is located in a research-oriented, academic environment, the machine on which this study was conducted is used primarily for two activities: document preparation and program development. Nearly half the files examined were created in conjunction with one of these two activities: program sources files, program object files, document processor input files, and document processor output files. This section

examines the characteristics of these four classes. The remaining half of the files was highly fragmented, with no clearly identifiable, large classes. Detailed study, discriminating on the basis of file type, of that set of files is unlikely to yield any fresh insights.

Figure 6-6 shows the effect of file type on file size. Object files and document processor output files tend to have much larger sizes than source files and document processor input files. The size characteristics of the entire population resembles that of source and document files.

Figure 6-7 shows the effect of file type on file lifetimes. Document processor files tend to have much shorter lifetimes than program files. I believe that this is due to the fact that once a document is complete, people tend to read the hard copy rather than the machine-readable copy. Important programs, on the other hand, tend to be used many times after they are debugged. Certain program source files are read long after they are debugged; for example, useful macro definitions are often included in other programs.

Table 3-1 summarizes the important characteristics of different file types. Probably the most important lesson to be learned in this section is that the type of activities engaged in by a user community strongly influences the size and lifetime properties of the files created by it. Files in a commercial data processing environment or a fusion research center can be expected to exhibit markedly different characteristics from those reported here.

Type of File	Number	File Size		File LifeTime	
		Mean	Std Dev	Mean	Std Dev
Program Sources	4010	21.84	47.63	363.6	731.3
Object Files	3474	53.99	116.3	414.6	681.4
Doc. Proc. Input	7085	29.28	70.95	137.5	322.7
Doc. Proc. Output	872	61.6	111.04	45.2	207.9
Entire Population	35652	23.89	66.83	238.9	531.9

Table 3-1: Effect of File Type on File Sizes and Lifetimes

3.4. Size/Lifetime Correlation

How does the size of a file affect its lifetime? Intuitively, one would expect large files to exhibit longer lifetimes than small files. Since the environment contains no large, frequently-modified databases, the most likely type of large files are infrequently-modified databases such as the one used in this study, or frequently-used and rarely-modified system programs such as compilers and editors. Small files, on the other hand, are likely to be temporary files of various sorts, or files associated with use-once-and-throw-away programs.

Figure 6-8 shows the lifetime distribution of files, with size as a parameter[1]. Suprisingly, the curves indicate that large files tend to have shorter, not longer, lifetimes than small files. The largest average lifetime is, in fact, that of 1-block files! Table 3-2 summarizes the information in Figure 6-8. At this point in time, I have no convincing explanation to offer for this counter-intuitive observation. One possibility is, of course, that the large databases in the system are modified far more frequently than I was led to believe. Another possibility is that the anomaly is a purely temporary phenomena, since the data is a snapshot of the file system. Repeating this study after a few months or a year will reveal whether this is indeed the case.

Size of File	Number	File LifeTime	
		Mean	Std Dev
1 block	8745	264.8	633.1
10 blocks	762	231.4	512.8
99 to 100 blocks	207	170.5	308.8
401 to 500 blocks	101	123.1	344.5
901 to 1000 blocks	13	120.2	240.6

Table 3-2: Effect of Size on Lifetime

4. Analytic Approximation

4.1. General Discussion

My aim in investigating analytic approximations to the size and lifetime distributions was twofold:

- To obtain a simple and computationally efficient means of generating random size and lifetime variables.
- To see if a model useful in analytic performation evaluations could be postulated for file sizes and lifetimes.

A Markovian model is analytically the most tractable [3]. At least to a first approximation, the process of generating files seems Markovian: the size and lifetime of a file one creates is independent of the files one has created in the past. For these two reasons, this study restricted its attention to Markovian models.

The simplest Markovian model is an exponential distribution [3]. If the size distribution is exponential with mean M, the probability that a random file has a size less than X is given by $1 - e^{-X/M}$. Both the mean and standard deviation of such a distribution are equal to M. Unfortunately, almost all the size and lifetime

[1] There are too few files of size 500 blocks or more to obtain a smooth cumulative distribution function; a discrete function is therefore shown for such files.

distributions observed have standard deviations between two and three times that of the corresponding means. This implies that a simple exponential model is certain to be unsuitable.

A hyperexponential model is a Markovian model which can exhibit coefficients of variation (i.e., ratio of standard deviation to mean) greater than unity. A k-stage hyperexponential consists of k simple exponentials with means M_1, M_2, ..., M_k, weighted so that they have probabilities α_1, α_2.....α_k of being chosen. Figure 4-1 shows such a model.

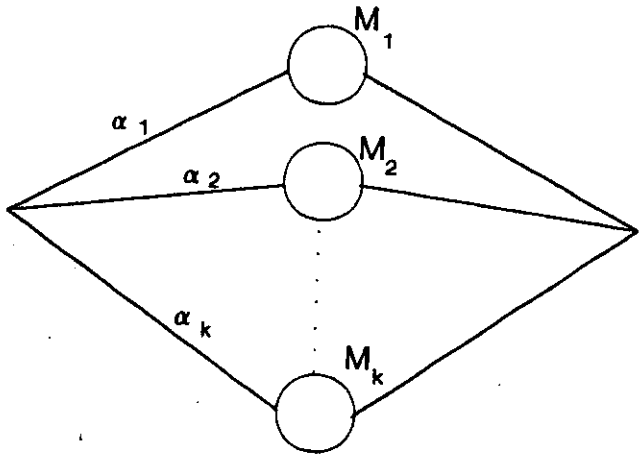


Figure 4-1: A k-Stage Hyperexponential Server

To generate a value for the random variable represented by this model, one proceeds in two steps:

1. With probability α_i, select one of the k stages.

2. Generate one value from an exponential distribution of mean M_i.

Further details on hyperexponential distributions can be found in [3, 2]. Each of the k stages can be viewed as being one population class with a simple exponential distribution. There is no guarantee, however, that such classes correspond to any clearly identifiable types of files. To fit a hyperexponential model to empirical data one needs to determine the number of stages, k, the means M_i, and the probabilities α_i. The next two sections discuss two alternative approaches for estimating these parameters.

4.2. The Moment Matching Method

In this method we try to find a hyperexponential model whose first few moments match the corresponding moments of the empirical distribution. If the empirical distribution was truly hyperexponential, one could find a model with all its moments matching. Otherwise the model is only an approximation to the empirical distribution.

The p^{th} moment of a k-stage hyperexponential is related to its parameters (α's and M's) by the following relationship:

$$\alpha_1 M_1^p + \alpha_2 M_2^p \ldots\ldots \alpha_k M_k^p = (p^{th} \text{ moment})/p!$$

This is easily derived using the moment generating function technique [3]. By using an iterative solution technique on 2k-1 such equations and the constraint $\alpha_1 + \alpha_2 + \ldots + \alpha_k = 1$, one can solve for the 2k unknowns, α_1 to α_k and M_1 to M_k.

Figure 6-9 compares the empirical size distribution of current files with a 2-stage hyperexponential fit. The first three moments of these two curves are identical. The two curves differ by no more than 0.05 at all points except at the very low end. Figure 6-10 shows the distribution of lifetimes of current files versus a 2-stage hyperexponential fit. Clearly the fit is not as good as for file sizes, especially at the low end, where the hyperexponential grossly underestimates the empirical distribution.

Adding more stages to the hyperexponential, thereby matching more moments, yielded negligible improvements in the fit. Figure 6-11, for example, compares the empirical file size distribution with two and three stage hyperexponential fits. The latter two curves are virtually indistinguishable! The moment matching technique is thus only of limited usefulness in analytically approximating the empirical data presented here.

4.3. A Heuristic Approach

The basis of this technique is that a simple exponential has its mean equal to its standard deviation. Starting from the low end, one examines successively larger initial segments of the empirical distribution until one finds a segment with its mean close to its standard deviation. This segment is represented as one stage of a hyperexponential. Its mean, M, is the mean of the segment and its probability of selection, α, is the fraction of the total population in that segment. This initial segment is removed from the distribution, and the procedure is repeated on the rest of the distribution. The procedure terminates when the entire distribution has been approximated.

The above procedure is a heuristic rather than an algorithm because judgement has to be used in deciding when the mean of a segment is close enough to its standard deviation. In practice, good results were obtained when the mean and standard deviation were within 25% of each other. Closer examination of this procedure, to see if it can be used as the basis of an algorithm, is one possible extension of the work presented here.

Figure 6-12 shows the fit of a 3-stage hyperexponential obtained by this method to the empirical distribution of file sizes. The fit is indeed excellent throughout the range of the curves.

- The unit of allocation for file sizes is a track, which is about 25 times larger than the unit of size used in this study.
- The definition of lifetime used here differs from the age-related quantities measured in their study.
- Their observations span a year and their analysis includes an examination of the time behavior of the file characteristics.

6. Summary

Keeping in mind the constraints discussed earlier, the results of this study may be summarized as follows:

- Most files are very small.
- Most files have short lifetimes. However, some files have significantly longer lifetimes.
- The type of a file class strongly affects the properties of that class.
- Larger files tend to have shorter lifetimes; i.e., the data within them tends to remain unaltered for shorter lengths of time.
- A hyperexponential is a very good model of the file size distribution and a reasonable model of the lifetime distribution.

Acknowledgements

I would like to thank George Robertson for his advice and encouragement throughout the course of this research, Louis Monier for helping me devise the iterative solution mentioned in Section 4.2, and Ivor Durham, whose PLOT program enabled me to produce the graphs in this document with a minimum of effort. Of course, I am also indebted to all the users, past and present, of the CMU-10A who unwittingly contributed to the data presented here!

Figure 6-13 shows the fit of a 3-stage hyperexponential to the lifetime distribution. The fit is not as good as the file size fit, but it is certainly better than that obtained by matching moments. At the very low end, near 1 day, the approximation grossly underestimates the actual curve. The only way to remove this anomaly was to treat files of age 1 as a class by themselves and assign a constant probability to them. This constant distribution causes no problems in random variable generation, but it does violate the Markovian assumption thereby making analytical solutions harder. One could get over this difficulty by viewing the constant distribution as a distribution with mean 1 and extremely small standard deviation. Such a distribution could be realized using an Erlang distribution [3; 2], thus retaining the Markovian property of the model. Figure 6-14 shows the effect of this modification — as expected, the fit at the very low end is much better.

Below about 500 days, Figure 6-14 shows that the hyperexponential lifetime curve is consistently above the empirical curve. Whether this is merely an outcome of the curve fitting procedure or an observation of significance is not clear. If it is of significance, the curves tell us that the lifetime of files is longer than that predicted by a Markovian generation process. There are, in fact, certain processes that might give rise to such a phenomenon. The most commonly used programming languages and document processor in this environment support separate compilation and inclusion of external sources files (called "require" files). During the course of debugging, attention is typically focussed on one such file and modifications are made to it. However, each compilation results in all the related files being accessed, thereby lengthening their lifetimes. The lifetime of such files is no longer independent of all other files, thus violating the Markovian assumption.

5. Limitations and Extensions

In using the data and analysis presented in this paper, a file system designer should bear in mind the conditions under which they hold:

1. The data was gathered from one machine (the most heavily used one) in an environment with many machines. It would be instructive to examine the file usage patterns on the other machines and compare them with the one presented here. Such a comparison would have to take into account the fact that both the hardware and software on those machines is different from that on the machine discussed here.

2. The data is a profile of the file system at one point in time. The time-varying behavior of the system remains to be studied.

3. The data was obtained in an academic environment and may not be directly extendable to other environments. Stritter [7] and Smith [6] discuss the collection and analysis of data on IBM/370s at the Stanford Linear Accelerator Center. Besides the obvious differences in environment and machines, their work differs from mine in the following ways:

    - They examine only text editor files.

References

[1] Accetta, M., Robertson, G., Satyanarayanan, M. and Thompson, M.
The Design of a Network-Based Central File System.
Technical Report CMU-CS-80-134, Department of Computer Science, Carnegie-Mellon University, August, 1980.

[2] Allen, A.O.
Probability, Statistics, and Queueing Theory.
Academic Press, 1978.

[3] Kleinrock, L.
Queueing Theory Vol.1: Theory.
John Wiley & Sons, 1975.

[4] International Business Machines Corp.
OS/VS2 MVS JCL.
Order No. GC28-0692-4, May 1979.

[5] Digital Equipment Corp., Maynard, Mass.
DecSystem10 Hardware Reference Manual.
DEC-10-XSRMA-A-D.

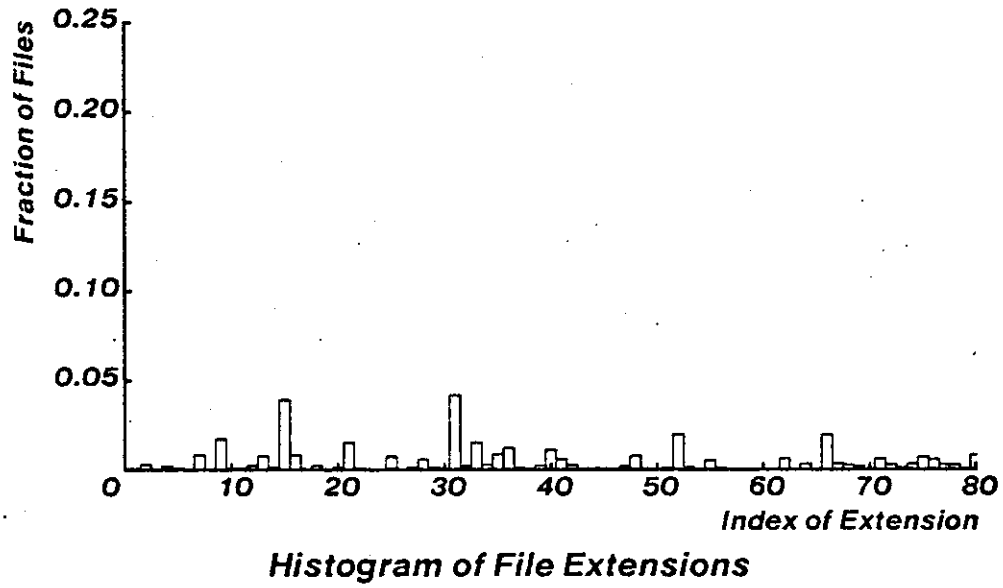
[6] Smith, A.J.
Long Term File Reference Patterns and Their Application to File Migration Algorithms.
Technical Report, University of California, Berkeley, 1978.

[7] Stritter, E.P.
File Migration.
PhD thesis, Department of Computer Science, Stanford University, January, 1977.

[8] Digital Equipment Corp., Maynard, Mass.
DecSystem10 Operating Systems Command Manual, DEC-10-OSCMA-A-D.
May 1974.

```
; Data Source Identification: Combined histogram for DSKB, DSKC and TEMP on CMUA on 14,15-JAN-81.
;
; The files contained no Migrated files
; Population Selection Parameters:
;     LoSizeValue =            1  HiSizeValue =         1000
;     LoSizeIndex =            1  HisSizeIndex =          28
;     LoAgeValue =             1  HiAgeValue =          5000
;     LoAgeIndex =             1  HisAgeIndex =           32
; Extension mode specified was ALLBUTUFD
; Total number of relevant files =        35652
;
; Moment 1 of Size  =  2.3889931E+01
; Moment 2 of Size  =  6.0366638E+03
; Moment 3 of Size  =  2.4054326E+06
; Moment 4 of Size  =  1.5510999E+09
; Moment 5 of Size  =  1.0626644E+12
;
; Moment 1 of Age   =  2.2819419E+02
; Moment 2 of Age   =  3.3500745E+05
; Moment 3 of Age   =  9.6468970E+08
; Moment 4 of Age   =  3.6899178E+12
; Moment 5 of Age   =  3.6545613E+16
;
;
; Data points: Size Limit, Cumulative Fraction
;
          1,    2.4528778E-01
          2,    3.5052731E-01
          3,    4.2118254E-01
          4,    4.7523280E-01
          5,    5.1828789E-01
          6,    5.5808930E-01
          7,    5.9045776E-01
          8,    6.1805789E-01
          9,    6.4330192E-01
         10,    6.6467519E-01
         20,    7.8256478E-01
         30,    8.3689553E-01
         40,    8.7445304E-01
         50,    8.9950071E-01
         60,    9.1683495E-01
         70,    9.2861549E-01
         80,    9.3851677E-01
         90,    9.4625827E-01
        100,    9.5206439E-01
        200,    9.7983281E-01
        300,    9.8816335E-01
        400,    9.9276337E-01
        500,    9.9569632E-01
        600,    9.9725120E-01
        700,    9.9806462E-01
        800,    9.9868170E-01
        900,    9.9963637E-01
       1000,    1.0000000
```

Table 6-1: Sample Output: Size Distribution of Current Files



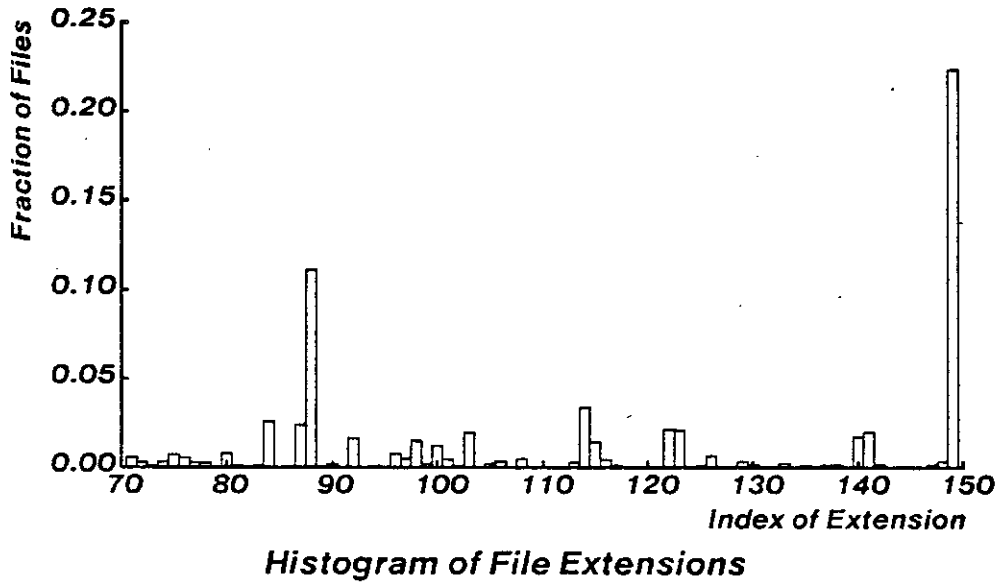


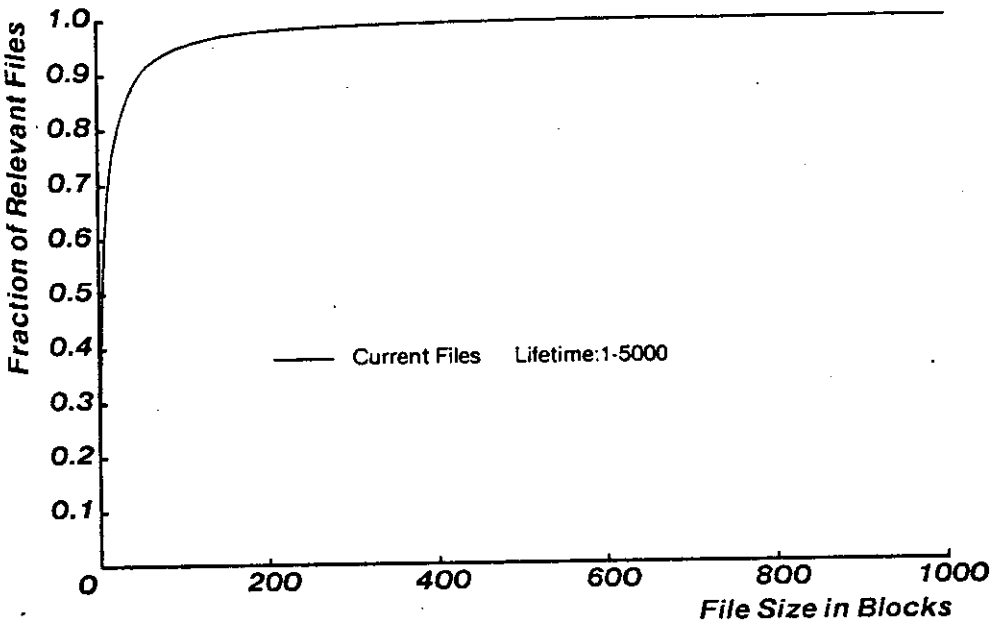
Figure 6-1: Histogram of Current File Extensions

Index	Extension	Purpose
0	A68	Algol-68 source file
1	ABS	Nonrelocatable program; output from PDP-11 assemblers/compilers
2	ADA	ADA source file for Intermetrices semantic analyzer
3	ALG	Algol-10 source file
4	APL	Saved APL workspace
5	ASM	Output from the PQCC TCOL assemblers, ASM or SPASM
6	ATR	Simula attribute file (companion to .REL file)
7	AUX	Auxiliary file from Scribe
8	BAI	Composite .SM1 files created by BAIL
9	BAK	Backup file from TECO or FINE
10	BAS	BASIC source fil
11	BBD	A bulletin board for BBOARD
12	BH	Input file to BH
13	BIB	Use-bibliography input to Scribe
14	BIN	PDP-11 binary file (executable)
15	BLI	BLISS-10, -11, -16, -32 and -36 source files
16	BOX	Mail.Box and $MAIL$.BOX are the only known appearances
17	CBL	COBOL input file
18	CCL	Stored commands for lots of different kinds of programs
19	CFL	Concept Font Load file (BILOS)
20	CHK	Renamed nnnCHK.TMP file from SOS
21	CMD	Stored command line for COMPILE, LOAD, EXECUTE, etc.
22	CNG	SHEPHERD change log
23	CRF	Output from an assembler or compiler for input to CREF program
24	CTR	ISP simulator COUNTER file
25	DAT	FORTRAN data file
26	DEB	Trace output from /DEBUG switch in BH
27	DFN	Input to PQCC TCOL assembler (data structure definition files)
28	DFS	require file for PUB
29	DIC	output file from BLSDIC, or dictionary of SPELL
30	DIP	IC package definitions for SUDS
31	DIR	As in MIGRAT.DIR, list of files migrated by CMU failsafe
32	DIS	Distribution file for MAIL or RDMAIL
33	DOC	Documentation file
34	DST	Alternate form of DIS; RDMAIL distribution file
35	ERR	error log file from Scribe
36	EXE	Single executable file (.SAV, .HGH/.LOW or .SHR/.LOW)
37	F4	FORTRAN source (Old DEC Fortran)
38	FAI	FAIL assember source file
39	FAS	FASLAP file, output of MACLisp compiler
40	FIN	Temporary file created by FINE (editor)
41	FON	Font description for Scribe
42	FTN	FORTRAN source (new DEC Fortran)
43	GDB	ISP Parse Tree
44	GRD	Grade input files to the grader program
45	GSI	Scribe output to the GSI photocomposer
46	GST	Graphics (GDP-II) Character set
47	HGH	High segment of non-sharable two-segment program
48	HLP	Help file
49	HYP	Scribe hyphenation dictionary
50	ICX	ISP Simulator ICONNECT file
51	IMG	Bit map (Image mode) file for XGP
52	INI	As in SWITCH.INI, user configuration file for programs
53	ISP	ISPS or ISPL source file
54	KSL	Text representation of character set for XGP
55	KST	Character set for XGP
56	LAP	Output from LISP compiler (CMU/UCI Lisp)
57	LBK	Backup of overwritten .LSP file (written by LISP)
58	LBL	Libraries for Bliss-16, -32, -36
59	LCK	SHEPHERD file lock file
60	LDA	LINK11 binary output file (alleged wierd format)
61	LEX	Lexicographic output from Scribe
62	LIB	Scribe database definition file
63	LIS	Listing file from Bliss-32
64	LMD	Stored commands for the Hydra linker
65	LNK	Output from PDP-11 linkers
66	LOG	Log file (output file) from BATCH
67	LOW	Low segment of sharable or non-sharable two-segment program
68	LPT	line printer file (from lots of programs, including Scribe)
69	LRC	Control file for AT
70	LSD	Printable listing of DIP definitions for SUDS drawing program.
71	LSP	LISP source (CMU/UCI Lisp)
72	LST	Reserved for listings produced by compilers
73	M00	CROSS assembler input files
74	M11	Macro-11 source

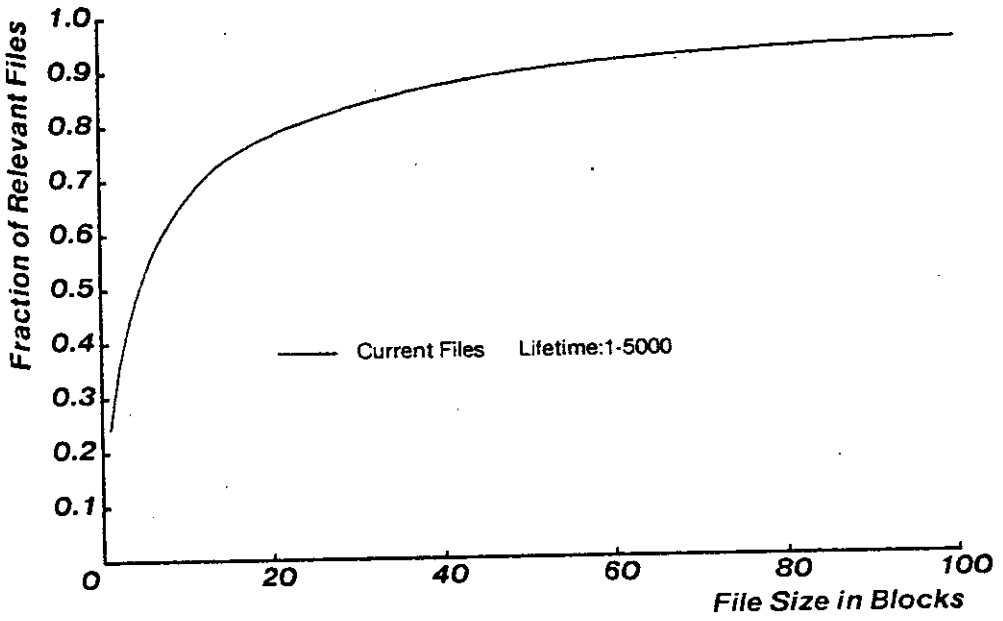
Table 6-2: Mappings of Extensions 0 to 74

Index	Extension	Purpose
75	MAC	Macro-10 input file
76	MAK	Scribe database for document type ()
77	MAP	Link map from various linkers, including LINK, LINKER, LINK11
78	MAS	MASTER file, many little files in one big one (SUBFIL)
79	MBK	Backup MCL files created by SLURP
80	MCL	MACLisp source file
81	MDU	List of modules in SHEPHERD tree
82	ME	As in DELETE.ME, the default file name FINE starts up with
83	MEX	Medusa executable file
84	MIC	MIC control file
85	MID	Source file for MIDAS, MIT assembler
86	MLI	MLISP source
87	MSG	RDMAIL format file (RD file.MSG or RD file)
88	MSS	input file to Scribe
89	MUM	MUMBLE source file (high level language for K.map microcode)
90	NAM	As in N.NAM, list of people for the N program
91	NWS	SHEPHERD News file
92	OBJ	Relocatable PDP-11 program
93	OCX	ISP Simulator OCONNECT file
94	OLR	Previous .LRC file from AT
95	OPR	Installation instructions for a program or system
96	OTL	outline file from Scribe
97	P11	Intermediate output from Bliss-11. NEVER use as a source file
98	PAS	Output from Hydra linker; binary page images
99	PIC	Pascal source file
100	PLN	"Picture" output control file for BH
101	PNT	"Plan" files, read by FINGER if you are not logged in.
102	POD	Diablo output file from Scribe
103	PRE	Dover (Press) files
104	PRS	BH intermediate (PReSort) files
105	PRT	SUDS parts-list file
106	PUB	source file for PUB
107	Q	As in FTP.Q, the list of SMLFL commands to the QNET mailer
108	QED	A Queued mail message to be sent by the QNET mailer
109	QNI	A LISP.INI file for QACLSP, the Quick-Loading MACLSP
110	QNT	REMIND queue entry file (written on REMIND server area)
111	RAP	Output file from SMECO; list of new wires to wrap
112	RDP	Output from the Ada parser (read by Ada semantic analyzer)
113	REF	Scribe reference environment description
114	REL	relocatable output from assembler, compiler, etc.
115	REQ	require file for Bliss-10 and/or Bliss-11
116	RLS	require files or Bliss-16, -32, and -36
117	RPT	ISP Simulator REPORT file
118	RPY	REMIND reply file (written on REMIND server area)
119	RSM	Require Summary (REQUIR program)
120	RUN	Output file from SMECO; list of wire runs; input to WW machine
121	S12	Binary or text file from SIX12 SAVE/LOAD or STORE/RECALL
122	SAI	SAIL compiler source file
123	SAV	Single-segment executable core image
124	SEL	Selection control file for BH
125	SHP	uCode output from CMICRO assembler
126	SHR	High segment of two-segment sharable program
127	SIM	Simula source file
128	SM1	BAIL output from SAIL compiler
129	SNO	SNOBOL or SITBOL input file
130	SPX	A drawing created by the SPACS drawing program
131	SRT	intermediate file from BH
132	SUB	SHEPHERD list of submodules within a module
133	SUD	RANDOM FILES ASSOCIATED WITH SUDS
134	SWF	Simulated virtual memory file written by PMMLIB
135	SYS	System control file, root of SHEPHERD tree
136	TCL	TCOL output from Ada semantic analyzer; also, other kinds of
137	TEC	In PROFIL.TEC, the Teco initialization file read for a user.
138	TFO	Scribe typewriter-font description file
139	TIM	Output file from the Bliss Timer Package
140	TMP	Innumerable kinds of temporary files of all sorts
141	TXT	Random text file not created by document production system
142	TYP	Output file from TYPER program
143	UFD	User File Directory (always found on [1,1])
144	UNF	UnFasl file; commentary produced by MACLisp compiler
145	UNR	Output file from SMECO; list of old wires to unwrap
146	USR	List of users of a system for SHEPHERD
147	XGO	XGP output file from PUB, Scribe, others
148	XXY	Uninteresting standard extensions
149	ZZZ	Non-standard extensions

Table 6-3: Mappings of Extensions 75 to 149

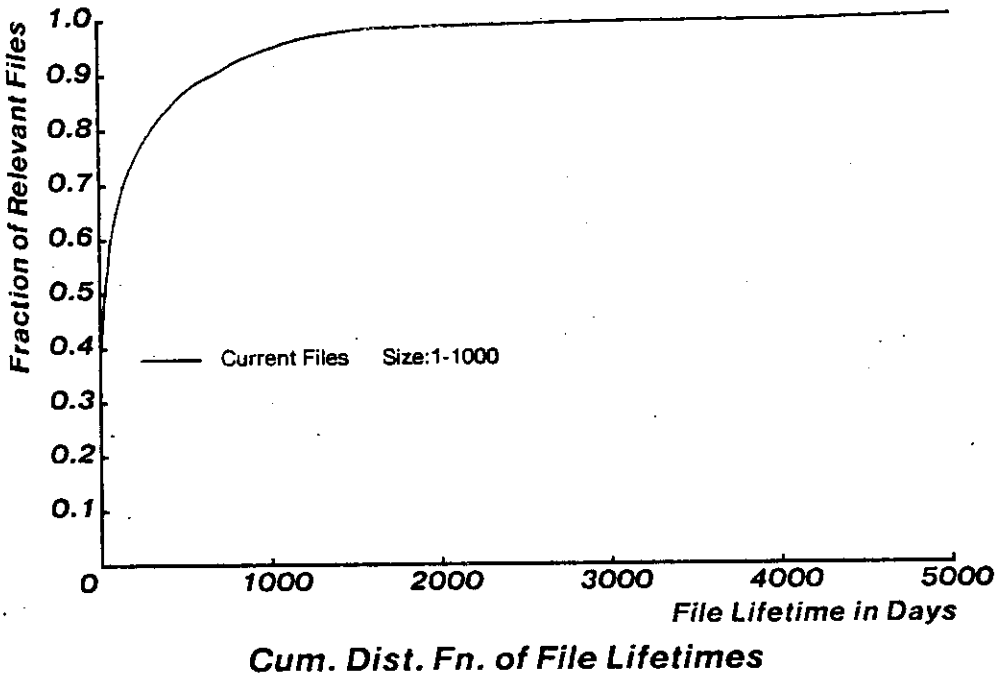


Cum. Dist. Fn. of File Sizes



Cum. Dist. Fn. of File Sizes

Figure 6-2: Size Distribution of Current Files



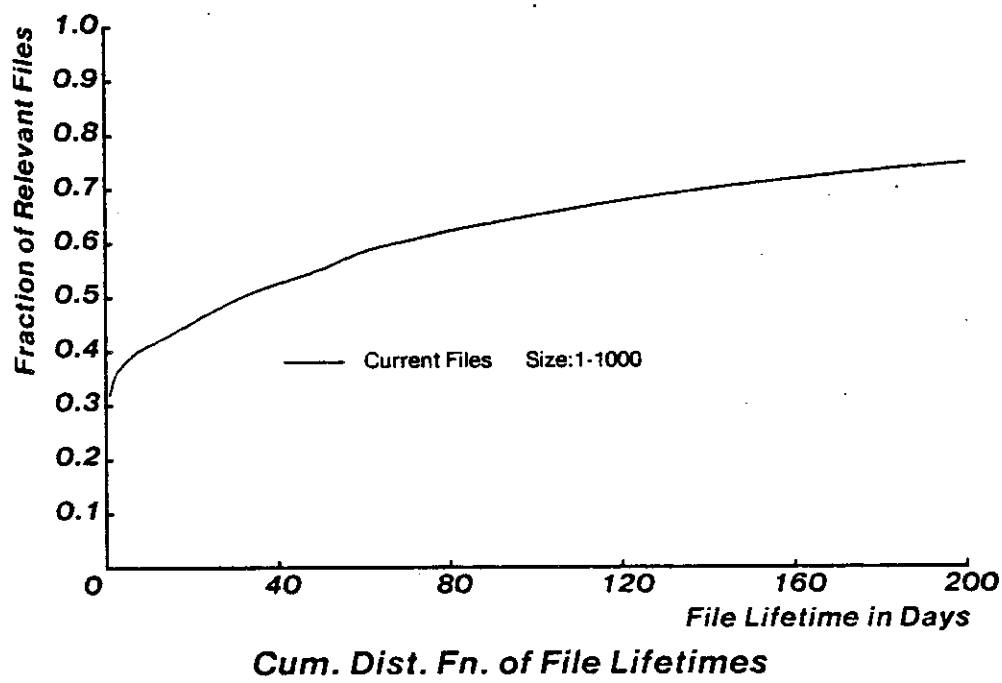
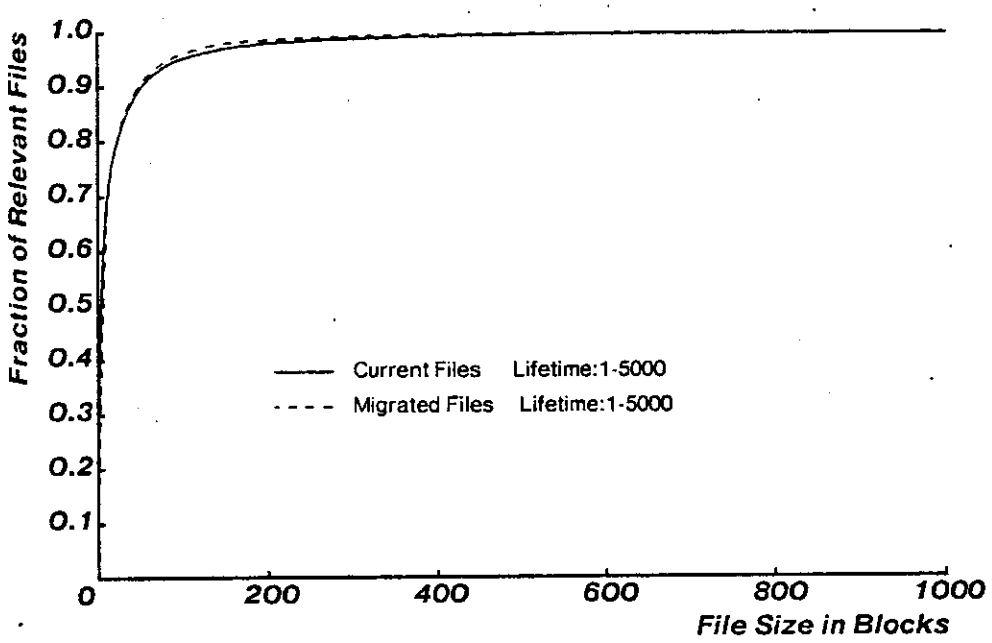


Figure 6-3: LifeTime Distribution of Current Files



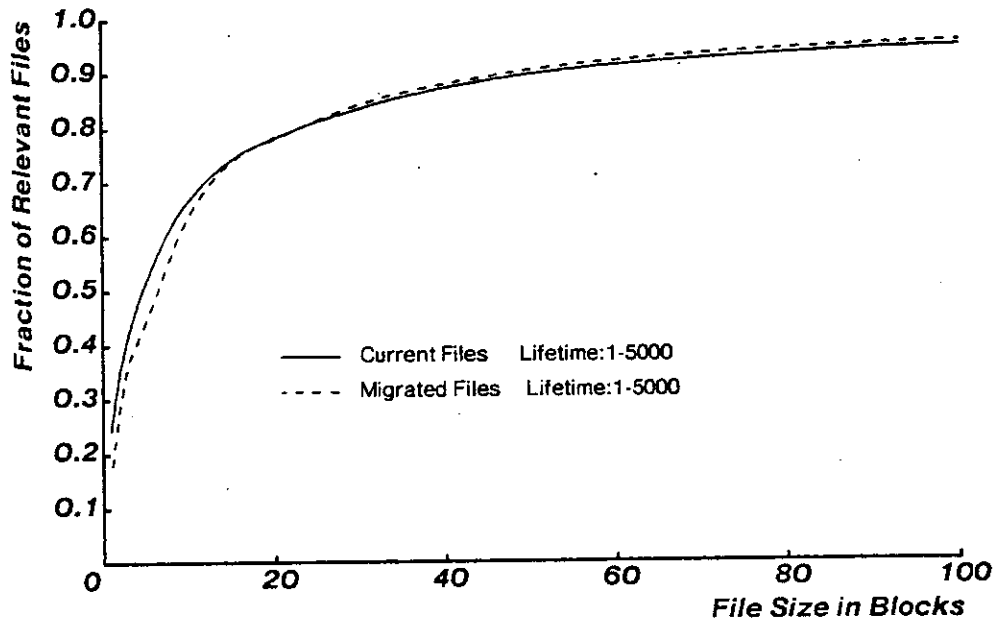
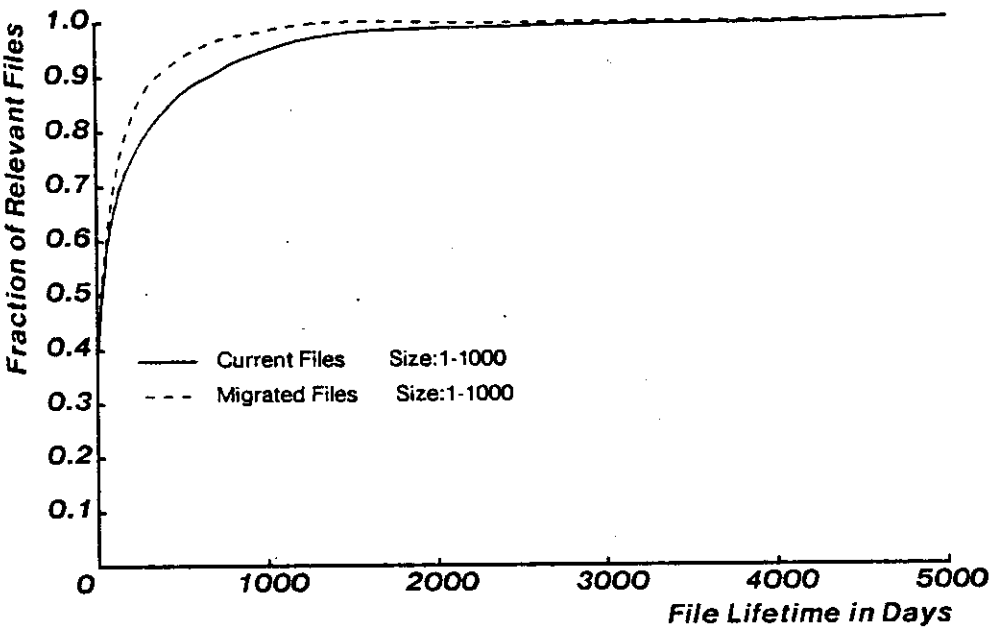
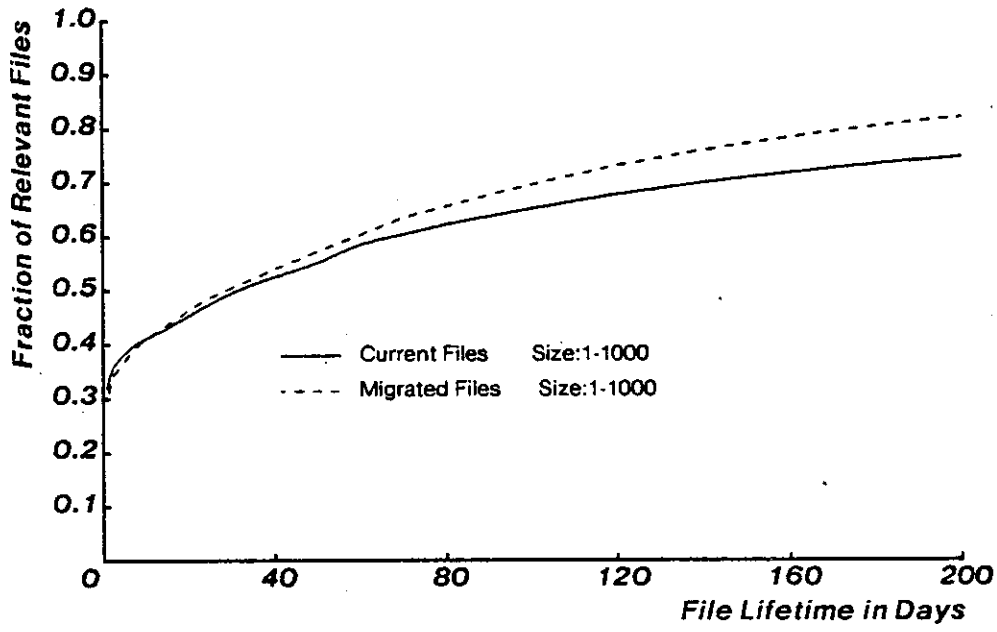


Figure 6-4: Effect of Migration on File Size

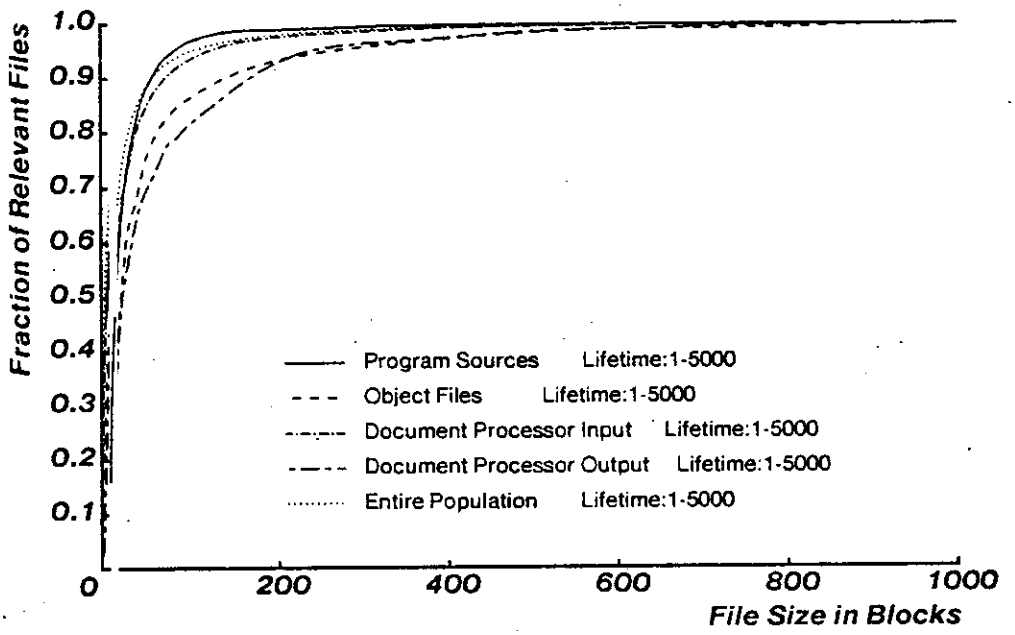


Cum. Dist. Fn. of File Lifetimes

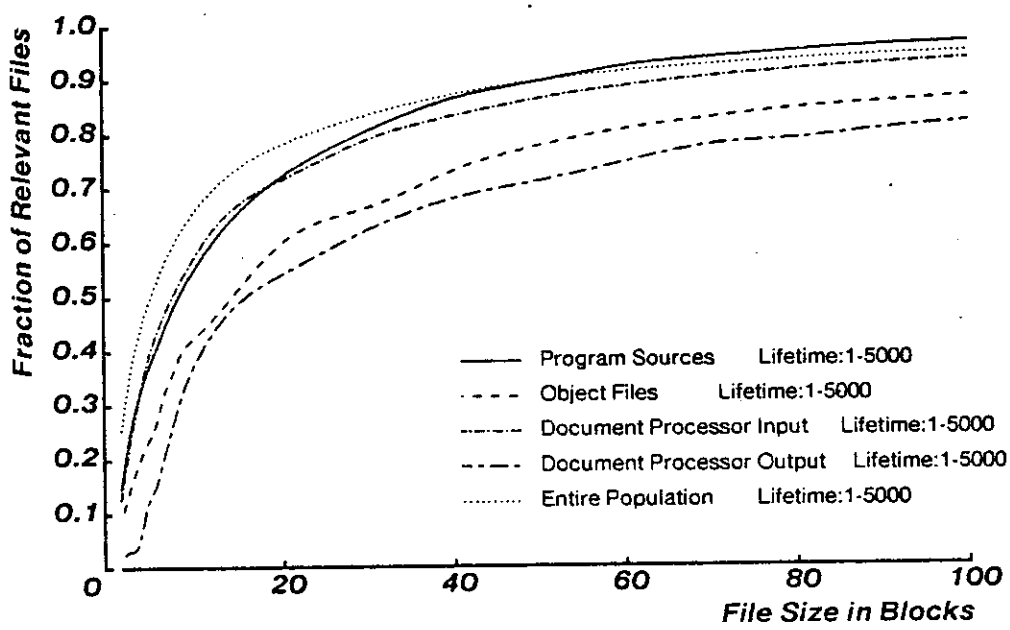


Cum. Dist. Fn. of File Lifetimes

Figure 6-5: Effect of Migration on File Lifetime

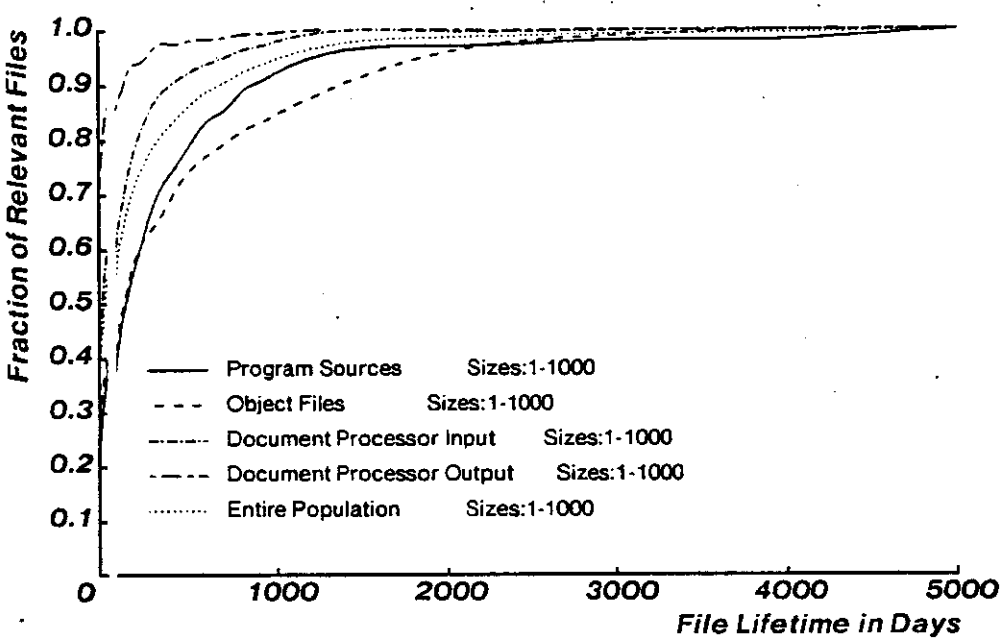


Cum. Dist. Fn. of File Sizes

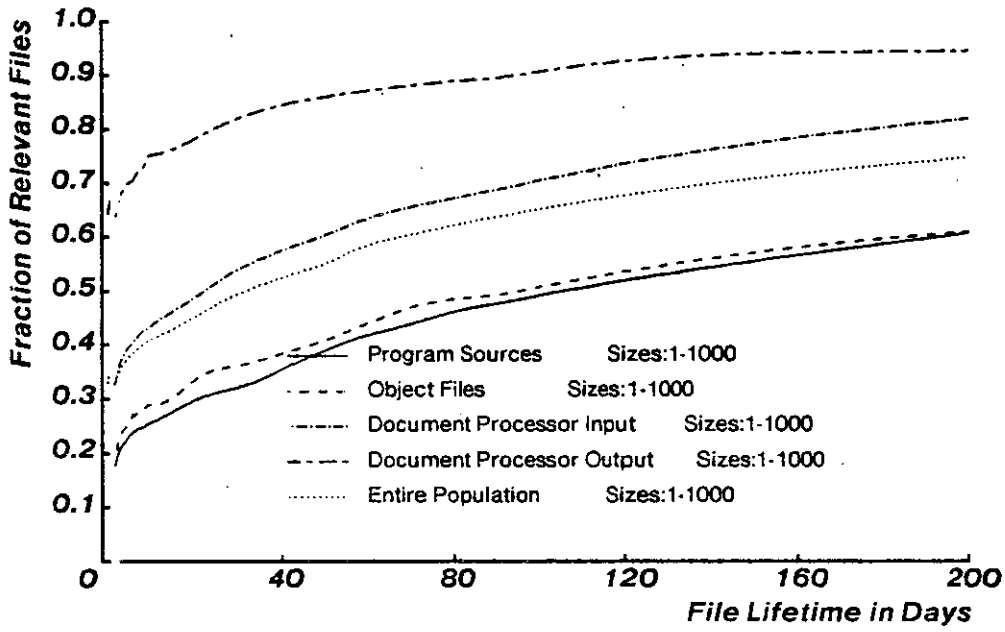


Cum. Dist. Fn. of File Sizes

Figure 6-6: Effect of File Type on Size

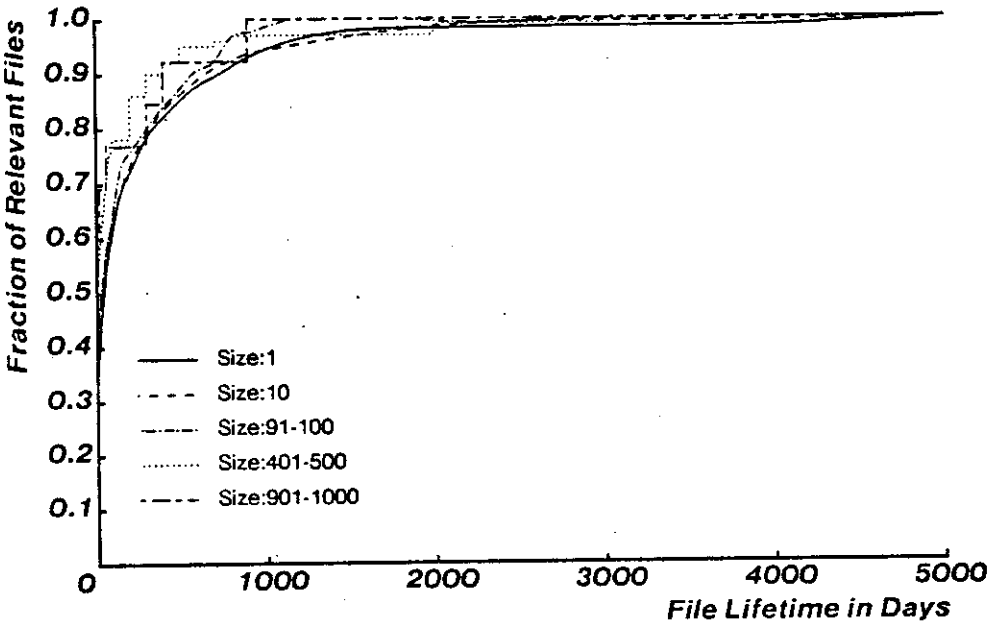


Cum. Dist. Fn. of File Lifetimes



Cum. Dist. Fn. of File Lifetimes

Figure 6-7: Effect of File Type on Lifetime



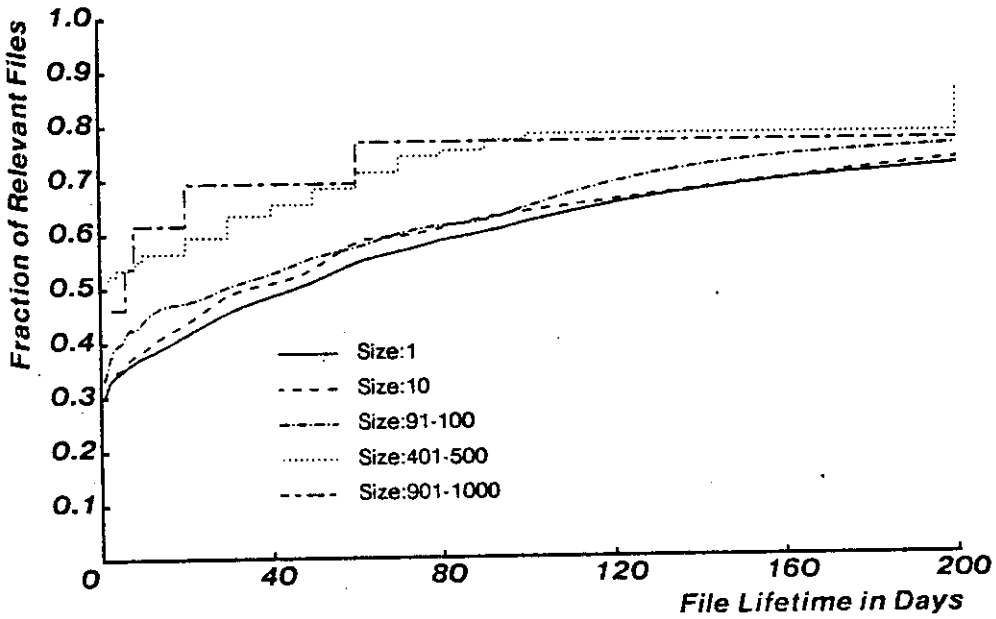
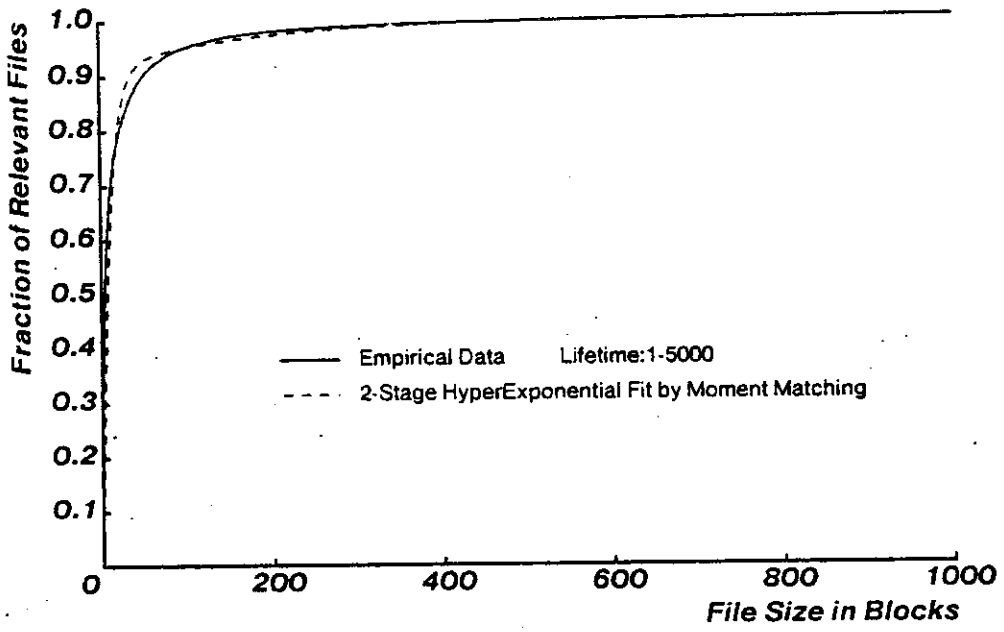


Figure 6-8: Effect of Size on Lifetime



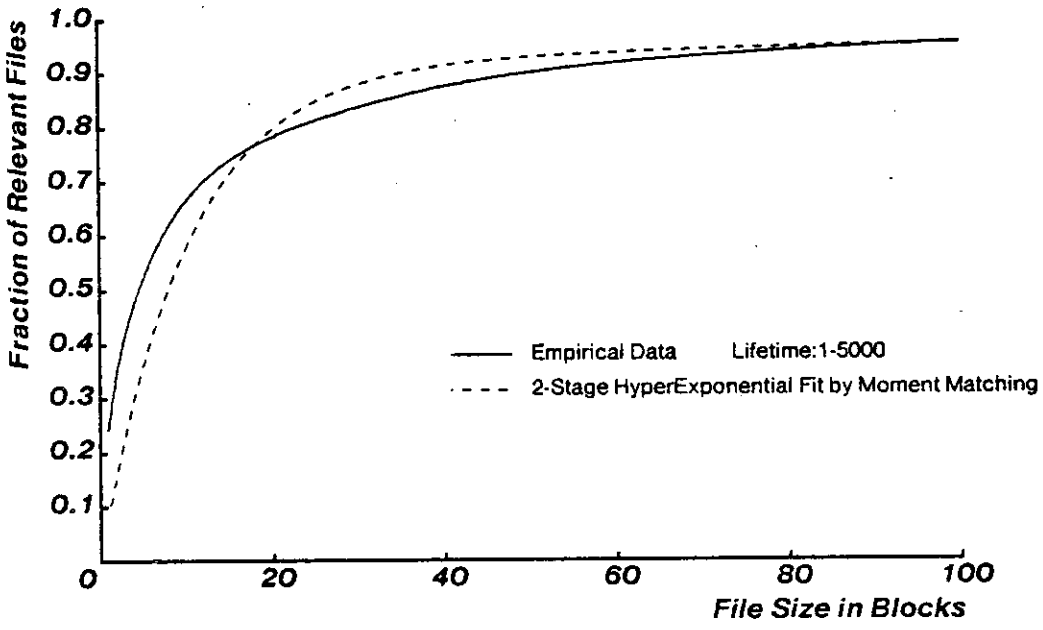
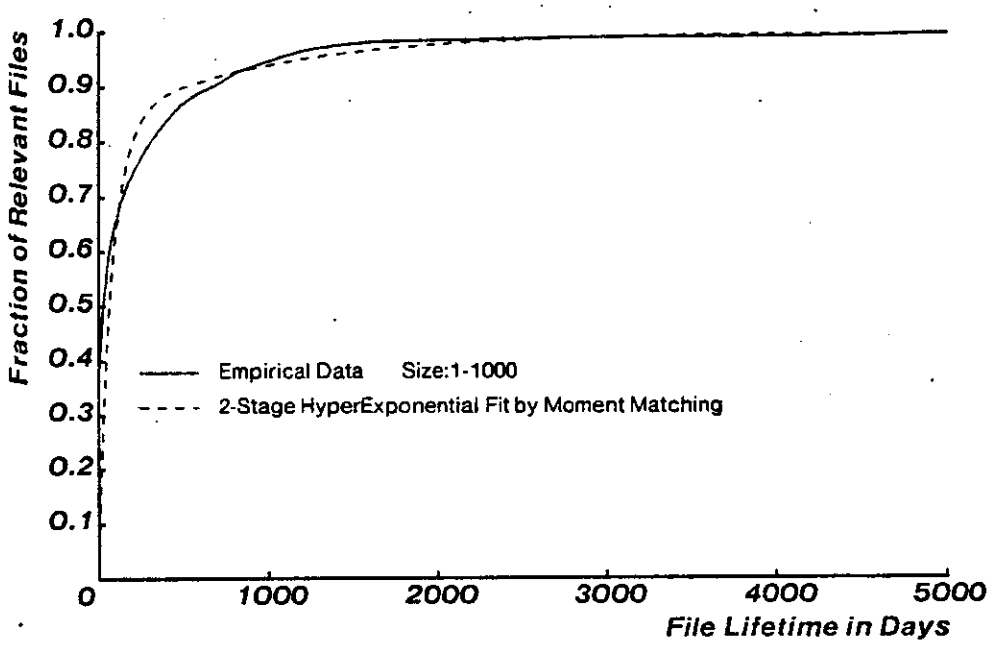


Figure 6-9: 2-Stage HyperExponential Fit for File Size



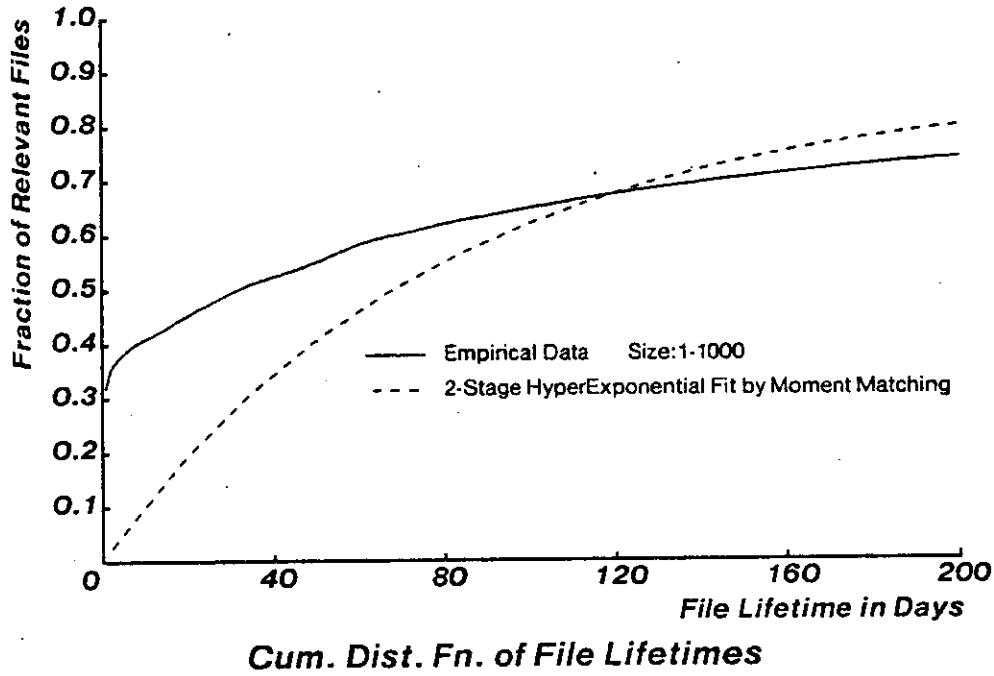
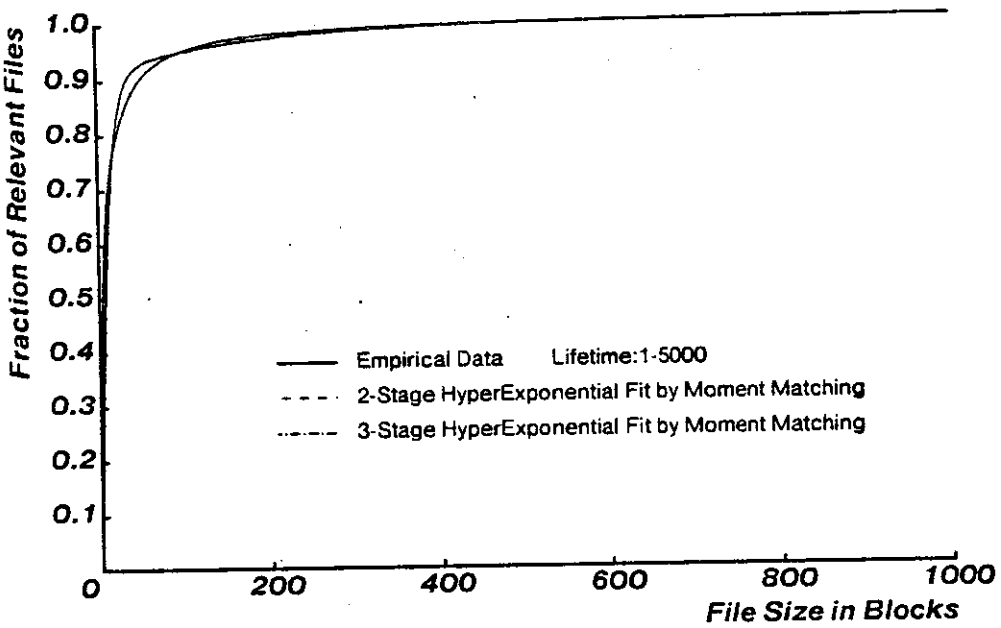
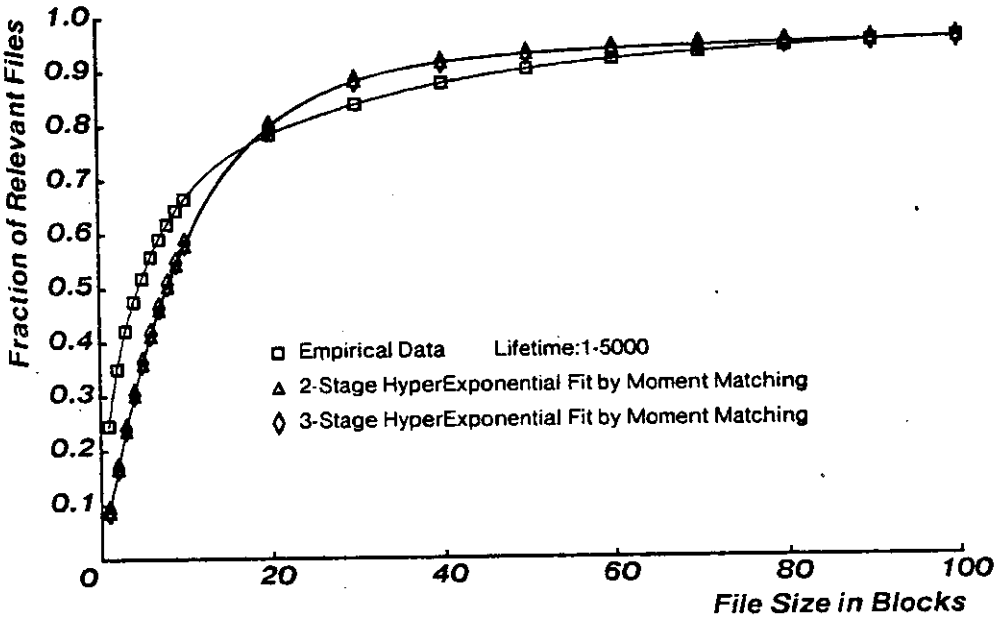


Figure 6-10: 2-Stage HyperExponential Fit for File Size

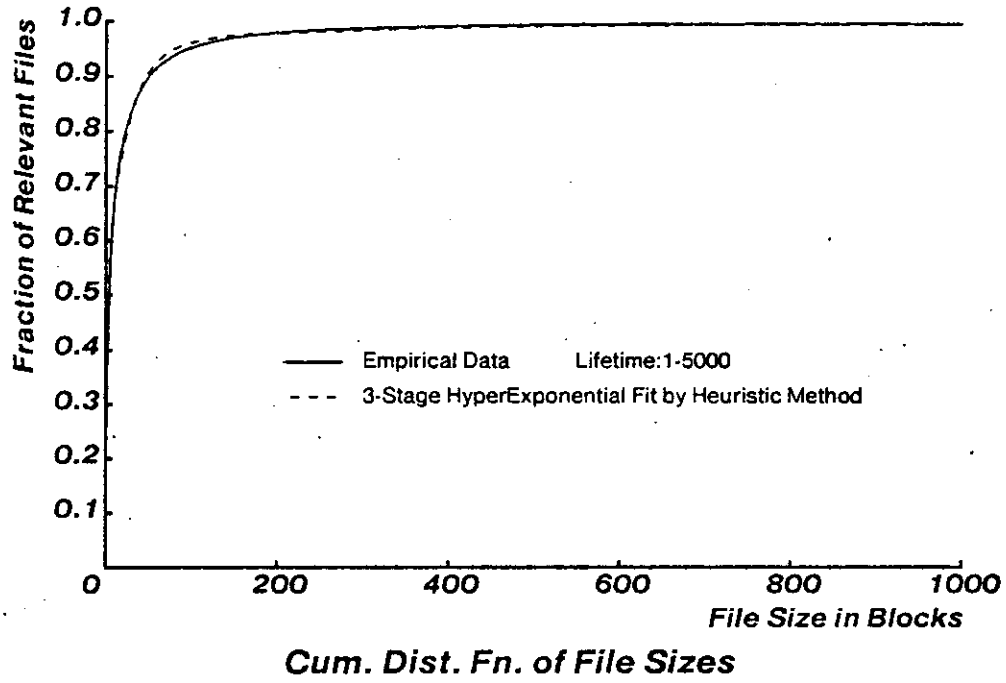


Cum. Dist. Fn. of File Sizes



Cum. Dist. Fn. of File Sizes

Figure 6-11: 2- and 3-Stage HyperExponential Fits for File Size



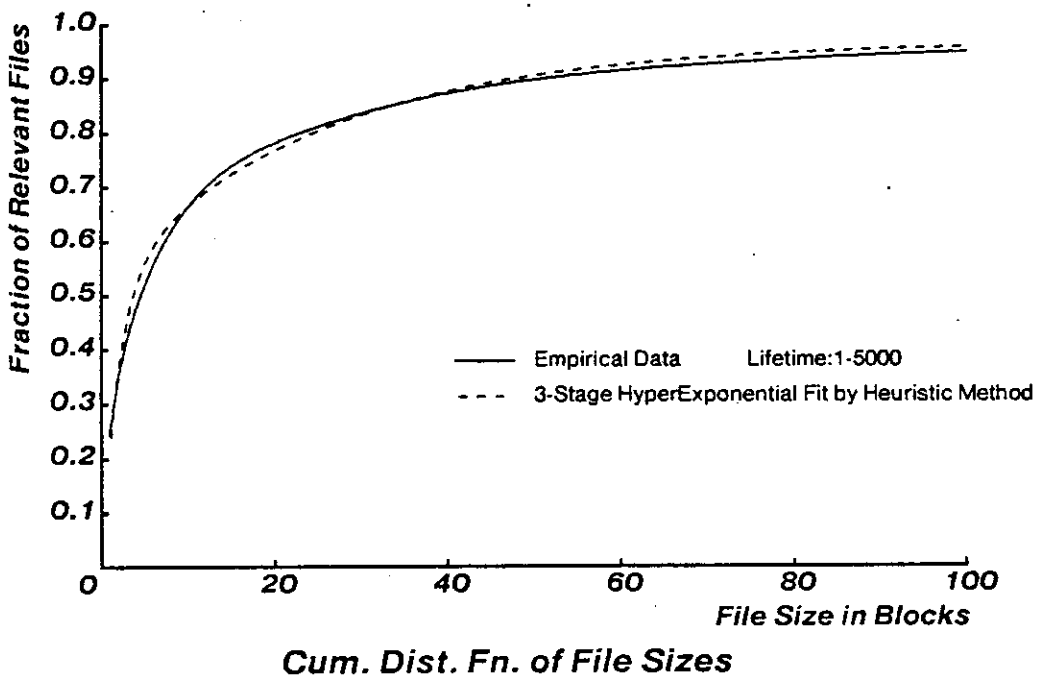
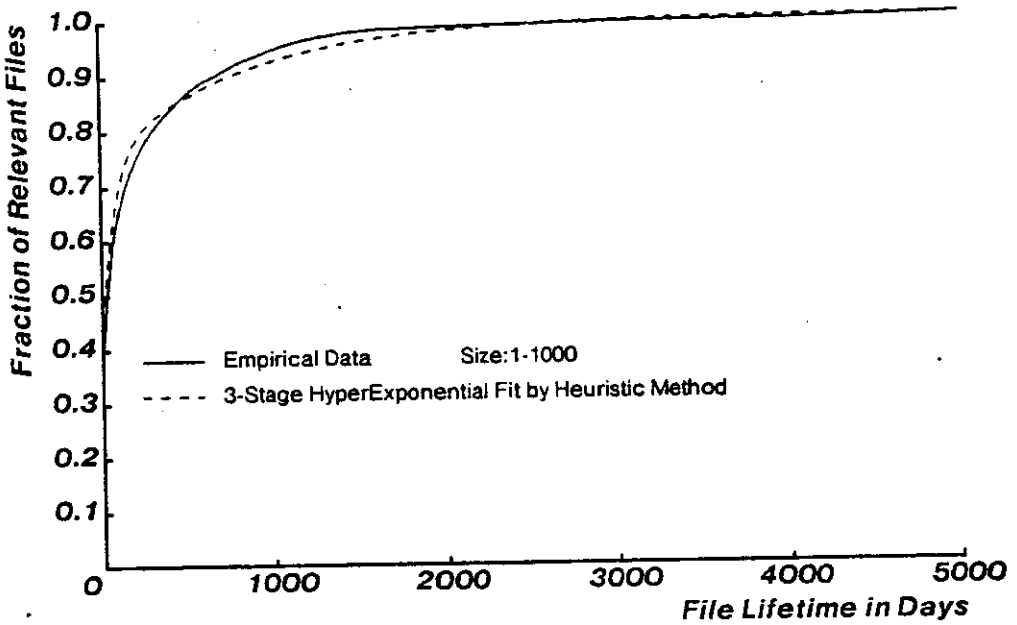


Figure 6-12: Heuristic 3-Stage HyperExponential Fit for File Size



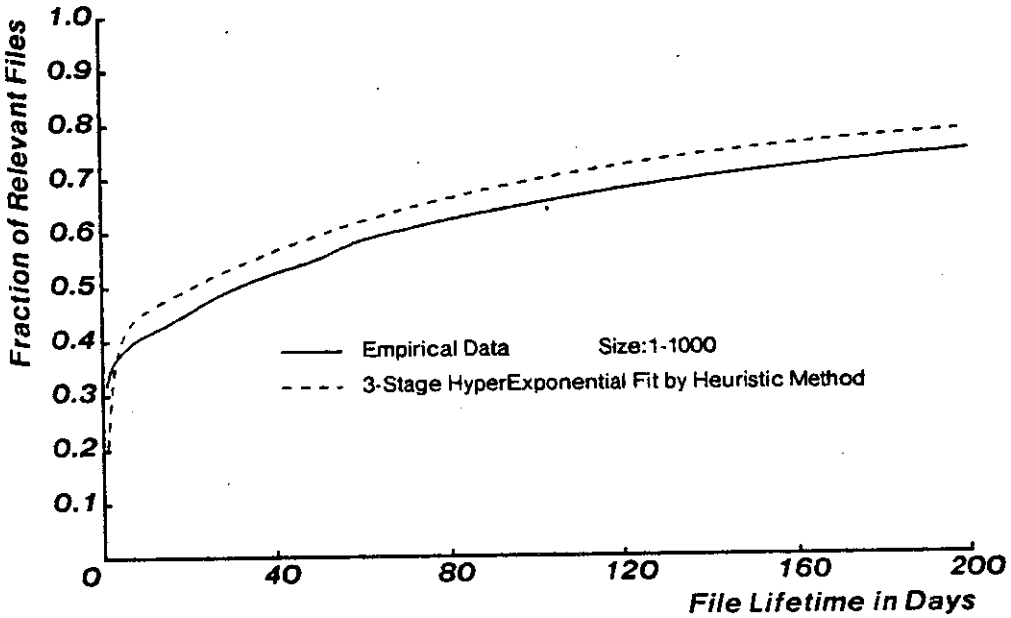


Figure 6-13: Heuristic 3-Stage HyperExponential Fit for File Lifetime

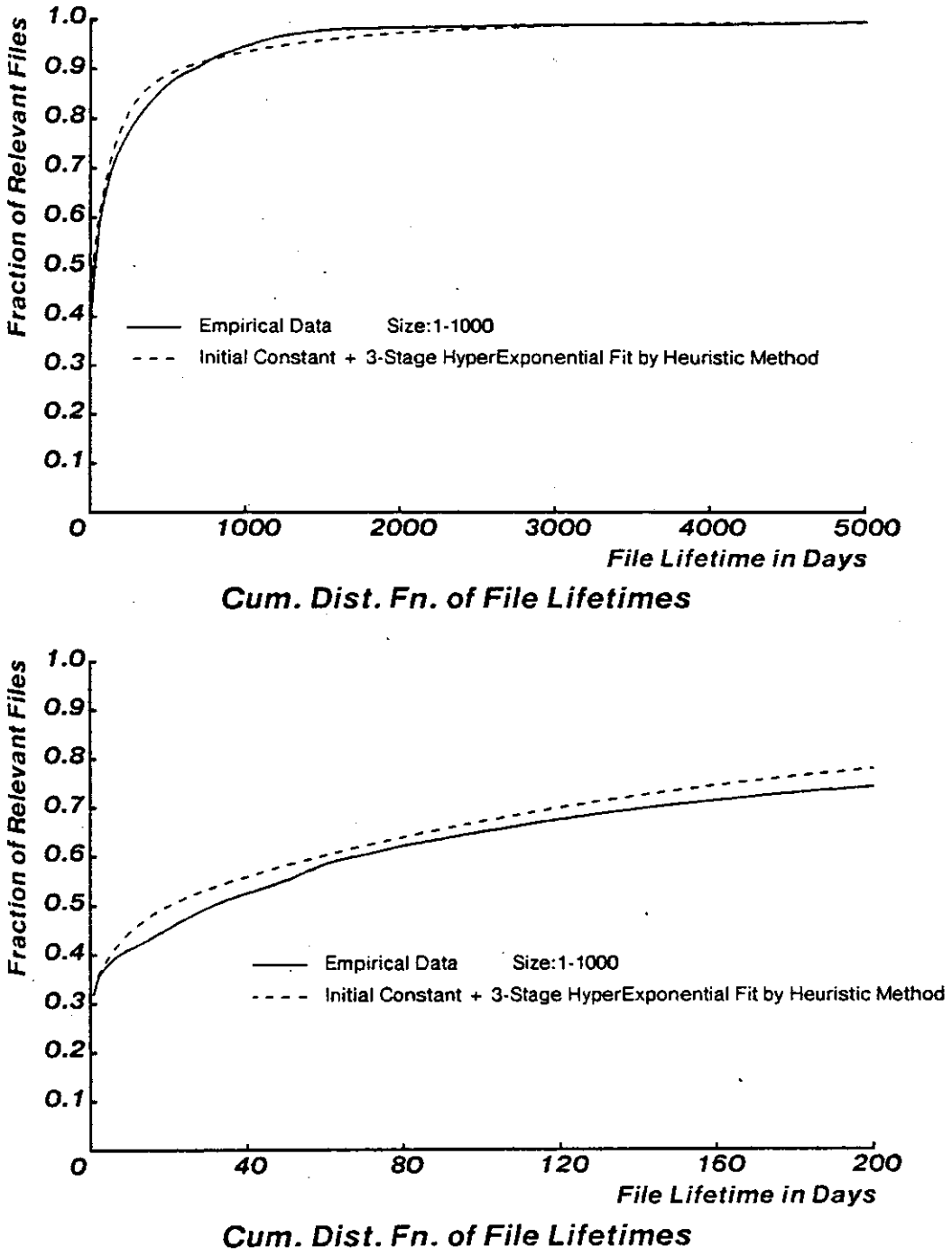


Figure 6-14: Heuristic Constant + 3-Stage HyperExponential Fit for File Lifetime