27
12-16-76
2508 to NTIS

UCID- 17251

Lawrence Livermore Laboratory

THE SIMULATION OF THE PERFORMANCE OF A COLLECTION OF
STORAGE SPACE ALLOCATION ALGORITHMS

Joseph C. Nichols

September 23, 1976

This is an informal report intended primarily for internal or limited external distribution. The opinions and conclusions stated are those of the author and may or may not be those of the laboratory.

Prepared for U.S. Energy Research & Development Administration under contract No. W-7405-Eng-48.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

THE SIMULATION OF THE PERFORMANCE OF A COLLECTION OF
STORAGE SPACE ALLOCATION ALGORITHMS

ABSTRACT

NOTICE
This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research and Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Storage algorithms for reserving and freeing contiguous blocks of storage space were investigated. A simulation program was constructed and run for each method using several different distributions of the parameters that characterize the requests for storage space. The performance of a noncontiguous storage algorithm was simulated for comparison with the other methods. The effect of disk flaws on the performance of several algorithms was also simulated. The simulation runs showed that the algorithm currently in use at LLL performs best for theoretical distributions of file size and life but that it is not best-suited for the distributions of size and life drawn from the samplings of the LLL machines. The results did not indicate a better suited choice for the LLL data because samples from different LLL systems resulted in different rankings among the various algorithms.

INTRODUCTION

LLL operates four computers with a time-sharing system, LTSS. Each computer system consists of a CDC 7600 computer and two disk units totalling 176M 60-bit words in addition to other I/O devices, tapes, printers, etc. LTSS is a time-sharing system, controlling access with user identifiers and time allocation and providing only basic services such as file cataloging and I/O services. All other services, e.g. compiling, editing, and batch process control, as well as application programs are provided by files containing the executable instructions and are kept on the disks.

The time-sharing scheme is priority scheduling of both memory space and cpu time. Each program can be outbid as it runs and replaced in memory by a program running at a higher priority. For this reason, each program run under LTSS must create a disk file into which its memory image can be saved. This file is called the drop file.

LTSS allocates disk space with a "best fit" algorithm, looking for the smallest hole that will contain a requested file. When a file located next to

a hole is destroyed, the hole is enlarged to encompass the file's space. When a file surrounded by other files is destroyed, a hole is created the size of the file. Because users frequently do not destroy files after use, files are purged after they remain unused for a period of time. Two lifetimes are provided and are tied, for convenience, to the access attributes of the file. In 1975, read-write files were allowed 4 hr of disuse and read-only files were allowed a 24-hr lifetime. Determined users employed the self-recall facility of LTSS to produce programs that recalled themselves every 30 to 90 min to "use" the files and thereby defeat the purpose of the purge. Ordinarily however, these underhanded programs were used infrequently.

During the summer of 1975, the number of cataloged files on the 7600 systems increased rapidly until about noon. As a consequence, it became difficult to use the computers because of the combination of active programs begun early in the day and leftover files that had been forgotten by their owners. Two system changes were initiated to combat this situation.

The LTSS was modified to facilitate changing the purge periods. These modifications were originally installed as system parameters that could be changed only by patching techniques. The 1975 changes allowed for a separate purge time for each disk unit that could be set from the operator's privileged terminal.

The system had been modified earlier in 1975 to allow repacking of files on disks. (This change had been resisted because productive use of the computer is not possible during the repack, a process requiring 10 to 15 min per logical disk unit of about 40,000,000 60-bit words.) It was noted that when the disks were effectively full (no holes as large as maximum memory size), there typically were ten to twenty thousand free sectors. It was hoped that repacks would allow enough space for productive use until the purge algorithm freed more space.

Experience proved that the repack routine provided two or three hours of use, until the disk reverts back to the pre-repack condition. This "swiss cheese" effect provided the reasons for both the variable purge feature in the system and this study.

A relationship is hypothesized between file size and its expected life, based on the operating rules for LTSS. Each process requires a drop file if it is expected to run more than a few milliseconds. Many processes output a few lines to a terminal and terminate within a few seconds. These processes typically require less than 20 sectors of memory. Long running processes

usually operate on large amounts of data and are more likely to require large memory sizes for efficiency. These large processes tend to generate sizable files of data for later use or to be staged to an off-line hardcopy device.

LTSS provides a method for a file to recall itself after a specified time limit. Additionally, it makes available the contents of several system tables to user-initiated processes. We wrote a program to utilize these features in collecting data for input to this simulation.

OUTLINE OF THE SIMULATION PROCEDURE

Each request for storage space for a given file is characterized by the outcome of three random variables; the length of time between successive requests (this integer will be called TIME, T); the number of sectors that are requested to store the given file (this integer will be called SIZE, S); and the length of time that the file will remain in the storage space (this integer will be called LIFE, L). Similar simulations¹ have been conducted with the assumption that the above random variables are independent. However, the data obtained through monitoring the LLL system indicate that for the particular variety of jobs run at LLL, S and L are not independent. Similar conclusions have been reached at other centers.^{2,3}

There are 12 simulation programs available, ten of which we will discuss here. These ten are WORSTFIT, FIRSTFITL, FIRSTFIT3, FIRSTFIT4, BESTFIT2, BESTFIT3, BESTFIT4, PAGES, BUDDY1, and FIBONACCI. A more detailed description of these algorithms is presented in the following section.

To compare the behavior of the algorithms under various different assumptions about the distributions of T, S, and L, we considered the following two classes of distributions.

Case 1: Theoretical Distributions

T: Exponential distribution (independent of S, L)

$$\Pr (T < t) = 1 - e^{-\lambda t},$$

where $t \geq 0$ and λ is the expected number of arrivals per unit time (i.e., mean of $T = \frac{1}{\lambda}$).

S: Geometric distribution (independent of T, L)

$$\Pr (S < j) = 1 - (1-p)^j,$$

where mean of $S = \frac{1}{p}$.

L: Exponential distribution with a mean of $\frac{1}{M}$ (independent of S, T). These distributions for CASE 1 and the selection of parameters λ , P, and M are discussed and used by Purdom, Stigler, and Cheam¹ and were chosen here for comparison purposes.

Case 2: Distribution Based on Data Collected at LLL

Data was collected at LLL by monitoring three CDC 7600's. This monitoring produced empirical distributions for the random variables L and S, presented in tabular form. These tables were converted to display a cumulative probability distribution function for each of the random variables L and S. The random variable T was taken to be constantly equal to unity.

The initial condition of the storage area was determined by specifying the total number of free sectors available, together with the size and locations of all of the storage regions that are not initially available. For instance, any collection of files of known size and life may be specified to exist at any desired location before the simulation begins. Such files might exist, for instance, immediately after the completion of a repack. In addition, specific areas of the storage unit may be declared permanently unavailable. This permits the simulation of disk flaws in the case of a disk storage unit. storage unit.

Three random numbers are then generated to determine one value for each of the random variables T, S, and L. The clock is advanced by time T and an attempt is made to store the file characterized by size S and life L, according to the given algorithm.

All files currently stored are aged by the outcome of time T and the space containing any file with a negative age is returned to free storage. Any adjacent blocks of free storage are combined to form contiguous blocks of free storage, separated only by those files that are still alive. For the Buddy Boy algorithm and the Fibonacci algorithm, only those adjacent free blocks that are "buddies" are recombined.^{4,5} The simulation continues until a specified number of requests for storage space has been satisfied. The final condition of the storage space and the following statistics describing the behavior of the algorithm are then calculated.

- 1) XMSU = TNR/RA. The mean space utilization ratio, i.e., the ratio of the total number of requests (TNR) to the number of such requests that have been accepted (RA). This measure can be misleading at times. For instance, under the CASE 1 type distributions with mean T = 3571, mean L = 100,000, and mean S = 7, the simulation was allowed to run until 1600 requests had been accepted. The BUDDY1 algorithm had XMSU = 1.376 while the PAGES algorithm had XMSU = 1.463, indicating that BUDDY1 rejected 602 files while PAGES rejected 740 files. PAGES "should" have the lowest value of XMSU, in general, but what must have happened was that PAGES successfully stored a few large files of long lifetimes while BUDDY1 rejected these files and stored many small files with relatively short lifetimes.
- 2) The number of holes (i.e., the number of contiguous sections of free storage space separated by assigned sections) available at the termination of the simulation.
- 3) Total number of free sections available at the termination of the simulation.
- 4) Average size of holes available at the termination of the simulation.
- 5) The dispersion D at the end of the simulation.

$$D = \sum_{\text{sum over all holes}} \frac{1}{\text{size of each hole}}$$

Statistics 4 and 5 are meaningless for the Buddy Boy and the Fibonacci algorithms since adjacent blocks of free storage can be combined for these algorithms only if the blocks are "buddies."

- 6) The length of time T that the simulation must run for the given algorithm to successfully accept a predetermined number of requests.

DESCRIPTION OF THE ALGORITHMS CONSIDERED

In the following, (S,L) represents the outcome of the random variable S (size) and the random variable L (life).

Worstfit Algorithm (WORSTFIT)

(S, L) is stored in the first S sector of the largest contiguous free section of the storage space.

Firstfit Algorithm (FIRSTFITL)

(S,L) is stored in the first S sector of the first section of the storage space that contains at least S consecutive free sectors.

Bestfit Algorithm (BESTFIT2)

(S,L) is stored in the first S sector of the smallest section of contiguous free storage space that contains at least S free sectors.

Buddy Boy Algorithm (BUDDY1)

The total storage space is initially 2^N sectors for some integer N. The free block of size 2^N is divided in half into two smaller blocks each of size 2^{N-1} (buddies) if the initial request for storage space is for less than 2^{N-1} free sectors. One of the blocks of size 2^{N-1} is again divided in half into two buddies, if necessary, and this process continues until an integer K is found with the properties $S \leq 2^K$, a free block of size 2^K exists, and K is minimal with respect to the first two properties. When free blocks are recombined, only those blocks that are buddies may be recombined.

Fibonacci Algorithm (FIBONACCI)

This is similar to the Buddy Boy algorithm. However, each block allocated has a size in sectors which is a Fibonacci number. A free block is not split in half as in the Buddy Boy algorithm but, rather, is split into two smaller blocks the sizes of which are determined by the two previous Fibonacci numbers; i.e., $S_{i-2} + S_{i-1} = S_i$. The Fibonacci sequence used progresses as follows: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144... .

A Noncontiguous Algorithm (PAGES)

An integer (MIN) is initially specified and determines the smallest size block that may be allocated. A request for storage space is satisfied if and only if the total existing free storage space can be broken up into "PAGES" of size $\geq$ MIN, each being a section of contiguous free storage. This algorithm was considered because with MIN = 1, the external fragmentation problem (the "swiss cheese" effect) disappears completely. Thus, a measure of the fragmentation properties of a given algorithm can be obtained by comparing its performance to the performance of this noncontiguous algorithm.

Firstfit Algorithm with Size Distinction (FIRSTFIT3)

An integer (CRITIC) representing a critical file size is a parameter of this algorithm. The value of this integer is established initially and the algorithm proceeds as follows. If S is less than CRITIC, the Firstfit search for free storage space begins at Location 0 and proceeds to successively larger addresses in the storage space. If S is greater than CRITIC, the search begins at location M where M is the address of the last word in the storage space and proceeds to successively smaller addresses in the storage space. In either case, (S,L) is stored in the first section of free storage space encountered that contains at least S free sectors.

Firstfit Algorithm with Life Distinction (FIRSTFIT4)

This algorithm is identical to FIRSTFIT3 with the exception that the life L of the file is compared to CRITIC to determine where the search begins.

Bestfit Algorithm with Size Distinction (BESTFIT3)

A "flaw" of size 1 is initially placed at some address in the storage space. CRITIC is also established initially. Thus there are two parameters that must be established for this algorithm. If S is less than CRITIC, the algorithm proceeds as a Bestfit algorithm with the restriction that (S,L) may only be stored in locations with addresses smaller than the address of the flaw. Similarly if S is greater than CRITIC, (S,L) may also be stored in a contiguous block where all addresses are greater than the address of the flaw.

Bestfit Algorithm with Life Distinction (BESTFIT4)

This algorithm is the analog to BESTFIT3 with the segregation made on the basis of the life L of the file rather than on its size.

SIMULATION PROGRAMS

There are 12 simulation programs available: WORSTFIT, FIRSTFITFL, FIRSTFIT3, FIRSTFIT4, BESTFIT2, BESTFIT3, BESTFIT4, PAGES, BUDDY1, FIBONACCI, BESTFITK, and FIRSTFITK. The latest version of each can be found with the ID number 999996 under the ELF directory .NICK. The name of the file in which each program is kept is the same as the name of the program.

Programs 1 through 10 have been described in this paper with respect to the particular allocation algorithm used by that program. All programs use the distributions of both CASE 1 and CASE 2. In CASE 1, the particular distribution used is controlled for all programs by subroutine GENERATE. Thus, only GENERATE needs to be changed to run a given simulation under different input distributions. GENERATE returns one value for each of the three random variables that characterize each request for storage space. Programs BESTFITK and FIRSTFITK both use the distributions suggested in Ref. (6). BESTFITK is merely a Besfit algorithm and FIRSTFITK is simply a Firstfit algorithm.

Programs 1 through 10 can be driven by the distributions of CASE 2 by replacing the subroutine GENERATE with the subroutine GENERATE2. This subroutine returns values for each of the three random variables that characterize each request for storage space according to the distributions by CASE 2, the empirical data collected at LLL. The time between requests (TINA) is constant and equal to unity.

In addition, the first time GENERATE2 is called, the data generated by monitoring the LLL 7600's is entered in the form of two arrays, Y(14, 9) and X(13). Each *row* of the Y array represents the distribution of file lives (L) for a fixed value of file size in the form of a cumulative distribution function. Likewise, each *column* of the Y array represents the distribution of files sizes (S) for a fixed value of file life in the form of a cumulative distribution function. The X array is the distribution of file sizes (S) in the form of the cumulative distribution function. A random number is generated and the X array is searched to determine a file size. A second random number is then generated and the row of the Y array corresponding to the above determined file size is searched to determine a file life.

The CASE 2 distributions were used to drive the simulation program under two conditions for all simulations: the number of requests (BIGNUM) was 10,000 and the time between requests (TINA) was 1.0.

Algorithm		SIZE	LIFE	ADDRESS
BESTFIT2:	Initial T matrix;	4096	-1	0
BESTFIT3:	Initial T matrix;	1365	-1	0
		1	9999999	1365
		2730	-1	1366

CRITIC was set equal to 25. NHALF was changed so it had a variable disk address. NHALF was an input value rather than a "set" 2048. We tried several values of NHALF with CRITIC equal to 25 to see what address gave the best results.

BESTFIT4:	Initial T matrix:	2048	-1	0
		1	9999999	2048
		2047	-1	2049

CRITIC was set equal to 120.

BUDDY1:	Initial T matrix:	4096	-1	0
FIRSTFIT3:	Initial T matrix:	4096	-1	0

CRITIC was set equal to 25.

FIRSTFIT4:	Initial T matrix;	4096	-1	0
------------	-------------------	------	----	---

CRITIC was set equal to 120.

FIRSTFITL:	Initial T matrix;	4096	-1	0
------------	-------------------	------	----	---

PAGES:	Initial T Matrix;	4096	-1	0
--------	-------------------	------	----	---

Sectors per page was set equal to 30.

WORSTFIT:	Initial T matrix;	4096	-1	0
-----------	-------------------	------	----	---

Definitions of the various input parameters follow.

BIGNUM The number of requests that must be accepted before the simulation can terminate.

XMT Mean value of the random variable T, the time to the next request. Logically, XMT is expressed in units of 10^{-2} sec i.e., if XMT = 1000, the mean value of T is 1000 sec or 3.6 hr.

XML Mean value of the random variable L, the life of the file, in the same units as XMT.

XMS Mean value of the random variable S, the size of the file, in units of sectors.

NT Total number of sectors initially available.

LAST Total number of holes plus files initially placed in the storage unit. In any case, $LAST \geq 1$.

MAXFIB The number of Fibonacci numbers that apply to the particular simulation being run, i.e., if $NT = 144$, then the Fibonacci sequence must be 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 so $MAXFIB = 11$. This parameter applies only to the Fibonacci algorithm.

MIN The minimum number of sectors per page for the algorithm PAGES.

CRITIC An integer representing the critical file size.

K1 Applies only to BESTFIT3 and BESTFIT4. Its meaning is obvious from its only use with DIMENSION T(256,3): $NFINISH = 1Q8SCH(T, K1, 768, 1, NHALF) - (K1 - 1)$ such that the third column of the T matrix is searched for the value NHALF. However, because 1Q8SCH assumes a one-dimensional array, some adjusting is necessary.

BESTFIT3 and BESTFIT4 do not work well in their present configuration. By placing a false flaw initially in the T matrix, the algorithm is doomed immediately, especially if T has only 256 elements. If T were increased to 2000 elements, these two algorithms might be more useful.

The only other initial specification is the initial state of the storage space, the matrix T. The first column of T specifies the sizes of the blocks. The second column of T determines the initial state of each block, (i.e., negative LIFE means hole; large positive LIFE 9999999 means flaw or permanently stored file). The third column of T is the address, beginning at 000, of the block described in columns 1 and 2.

RESULTS OF THE SIMULATIONS

Tables 1 through 9 show the results of the simulation programs for various input specifications. Conclusions follow concerning the performance of the storage allocation algorithms with respect to the several runs conducted.

- The performance of an algorithm is not independent of the distribution of the three random variables used to characterize each request for storage space.
- For the distributions considered in CASE 1, the algorithm BESTFIT2 consistently performed better than any other algorithm that was required to find contiguous storage space.
- With one disk flaw placed at the middle address of the storage space of 256 sections, approximately 25 more attempts are needed to successfully allocate 1600 storage requests (See Table 9).
- The rankings of the performance of the algorithms with respect to the Mean Storage Utilization Ratio (XMSU) were identical for the two different sets of data obtained by monitoring the R-machine. The rankings for the two sets of data obtained by monitoring the U-machine were quite similar although not identical. However, the algorithm rankings corresponding to data from the U-Machine are significantly different from the rankings corresponding to data obtained from the R-machine.

ACKNOWLEDGEMENTS

Leslie Froisland, Dan Ries, and Don Emery contributed much to the preparation of this paper. I am grateful to them and to Carl Schneider and Patricia Lien of TID for polishing the language and format of this report.

REFERENCES

1. P. W. Purdom, S. M. Stigler, and T. Cheam, "Statistical Investigation of Three Storage Allocation Algorithms," *BIT* 11, 187 (1971).
2. B. H. Margolin, R. P. Parmelee, and M. Schatzoff, "Analysis of Free-Storage Algorithms, " *I.B.M. Systems J.* 10, 283 (1971).
3. D. S. Hirschberg, "A Class of Dynamic Memory Allocation Algorithms," *CACM* 16, 615 (1973).
4. J. A. Hinds, "An Algorithm of Locating Adjacent Storage Blocks in the Buddy System," *CACM*, 18, 221 (1975).
5. B. Cranston and R. Thomas, "A Simplified Recombination Scheme for the Fibonacci Buddy System," *CACM* 18, 331 (1975).
6. D. E. Knuth, *The Art of Computer Programming*, Vol. I, (Addison Wesley, Reading, Mass., 1973) 2nd ed. This book provides a general description of the storage space allocation problem and discusses the basic algorithms simulated in our study.

Table 1. Simulation results for CASE 1 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	Parameters
PAGES	1.04187	0	2	88	44	Minimum number of sectors per page = 1 (Number of files stored noncontiguously = 87)
BESTFIT2	1.06437	0	3	87	29	
FIRSTFIT4	1.06937	0	3	74	24	CRITIC = 50,000
FIRSTFIT3	1.06937	0	3	74	24	CRITIC = 17
FIRSTFIT1	1.06937	0	3	74	24	
WORSTFIT	1.0906	0	4	54	13	
BUDDY1	1.1312	0	5	76	15	
BESTFIT4	1.1669	1	2	103	51	CRITIC = 50,000
BESTFIT3	1.2681	1	3	86	28	CRITIC = 17
						BIGNUM = 1600
						XMT = 33333
						XML = 100000
						XMS = 17
						Total number of cells = 128

Table 2. Simulation results for CASE 1 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	Parameters
BESTFIT2	1.25937	0	5	12	2	
FIRSTFITF1	1.26875	0	9	27	3	
FIRSTFIT3	1.27188	0	10	39	3	CRITIC = 4
FIRSTFIT4	1.2762	0	8	19	2	CRTIC = 50,000
WORSTFIT	1.35312	0	13	74	5	
BUDDY1	1.37625	0	7	27	3	
PAGES	1.4625	0	4	7	1	Minimum number of sectors per page = 1 (Number of files stored noncontiguously = 452)
						BIGNUM = 1600 XMT = 3571 XML = 100000 XMS = 7 Total number of cells = 128

Table 3. Simulation results for CASE 1 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	Parameters
PAGES	1.00062	0	2	88	44	Minimum number of sectors per page = 1 (Number of files stored noncontiguously = 15)
BESTFIT2	1.00625	0	2	126	63	
FIRSTFIT1	1.00688	0	2	121	60	
FIRSTFIT3	1.00937	0	2	121	60	CRITIC = 11
WORSTFIT	1.02187	0	3	107	35	
BUDDY1	1.03062	0	2	96	48	
						BIGNUM = 1600 XMT = 33333 XML = 100000 XMS = 11 Total number of cells = 128

Table 4. Simulation results for CASE 1 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	Parameters
PAGES	1.45375	0	3	66	22	Minimum number of sectors per Page = 1 (Number of files stored noncontiguously = 303)
BESTFIT2	1.5075	0	3	82	27	
FIRSTFIT3	1.52438	0	1	82	82	CRITIC = 45
WORSTFIT	1.53625	0	2	65	32	
FIRSTFITF1	1.55625	0	3	48	16	
BUDDY1	1.6425	0	1	64	64	
						BIGNUM = 1600 XMT = 33333 XML = 100000 XMS = 45 Total number of cells = 128

Table 5. . . Simulation results for CASE 2 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	D
WORSTFIT	1.9600	0	19	196	10	1.8625
FIRSTFIT3	1.9636	1	10	95	9	1.0679
FIRSTFIT4	1.9700	1	20	235	11	1.8679
FIRSTFITF1	1.9724	0	18	206	11	1.7500
BESTFIT2	1.9842	0	15	236	15	1.2250
PAGES	2.0337	1	22	175	7	5.9165
BESTFIT4	2.3567	1	8	1185	148	0.4366
BUDDY1	2.8461	0	3	96	32	0.0937
BESTFIT3	2.9748	1	7	95	13	0.6500

R-Machine
63 Samples

Table 6. Simulation results for CASE 2 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	D
WORSTFIT	1.9957	0	20	306	15	1.5167
FIRSTFIT3	2.0158	0	11	136	12	0.9958
FIRSTFIT4	2.0372	0	14	146	10	1.4167
FIRSTFITF1	2.0384	0	12	116	9	1.2667
BESTFIT2	2.0611	0	17	316	18	1.3101
PAGES	2.0969	0	15	146	9	1.8839
BESTFIT4	2.3486	1	12	1025	85	0.8076
BUDDY1	2.9838	0	5	160	32	0.1562
BESTFIT3	3.1259	1	11	375	34	0.566
						R-Machine 86 Samples

Table 7. Simulation results for CASE 2 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	D
BESTFIT4	1.5301	1	11	325	29	0.8713
FIRSTFIT4	1.5691	0	32	646	20	2.3101
FIRSTFITF1	1.5736	0	18	516	28	1.2321
FIRSTFIT3	1.5752	0	20	536	26	1.0593
BESTFIT2	1.5757	0	24	656	27	1.5858
WORSTFIT	1.6764	0	61	1166	19	3.7667
BESTFIT3	1.6909	1	15	255	17	1.1262
PAGES	1.6914	0	29	396	13	4.4374
BUDDY1	1.8825	0	7	352	50	0.1562
						U-Machine 42 Samples

Table 8. Simulation results for CASE 2 distributions.

Algorithm	XMSU	No. of Flaws	No. of Holes	Total Space	Average Hole Size	D
FIRSTFIT3	1.8372	0	11	286	26	0.5792
BESTFIT2	1.8435	0	14	316	22	0.8217
FIRSTFITF1	1.8473	0	21	296	14	1.77
FIRSTFIT4	1.8466	0	22	306	13	1.8167
WORSTFIT	1.9091	0	44	696	15	3.2
PAGES	1.9725	0	30	276	9	5.622
BESTFIT3	2.6522	1	14	595	42	0.9855
						U-Machine 76 Samples

Table 9. Comparison of the performance of the FIRSTFIT Algorithm with no disk flaw to the performance of the FIRSTFIT algorithm with only one disk flaw and a similar comparison for the BESTFIT algorithm.

Algorithm	XMSU	No. of Flaws*	TNR	Parameters
FIRSTFITL	1.24625	0	1994	BIGNUM = 1600 XMT = 8333 XML = 100000
FIRSTFITFL	1.2618	1	2019	XMS = 24 Total number of cells = 256
FIRSTFITFL	1.06937	0	1711	BIGNUM = 1600 XMT = 33333 XML = 100000 XMS = 17 Total number of cells = 128
BESTFIT2	1.06437	0	1703	BIGNUM = 1600 XMT = 33333
BESTFIT2	1.07812	1	1725	XML = 100000 XMS = 17 Total number of cells = 128

* Flaws were placed at the middle address of the storage space.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Energy Research & Development Administration, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately-owned rights.

NOTICE

Reference to a company or product name does not imply approval or recommendation of the product by the University of California or the U.S. Energy Research & Development Administration to the exclusion of others that may be suitable.

Printed in the United States of America

Available from

National Technical Information Service

U.S. Department of Commerce

5285 Port Royal Road

Springfield, VA 22161

Price: Printed Copy \$; Microfiche \$2.25

Page Range	Domestic Price	Page Range	Domestic Price
001-025	\$ 3.50	326-350	10.00
026-050	4.00	351-375	10.50
051-075	4.50	376-400	10.75
076-100	5.00	401-425	11.00
101-125	5.50	426-450	11.75
126-150	6.00	451-475	12.00
151-175	6.75	476-500	12.50
176-200	7.50	501-525	12.75
201-225	7.75	526-550	13.00
226-250	8.00	551-575	13.50
251-275	9.00	576-600	13.75
276-300	9.25	601-up	*
301-325	9.75		

*Add \$2.50 for each additional 100 page increment from 601 to 1,000 pages:
add \$4.50 for each additional 100 page increment over 1,000 pages.

Technical Information Department

LAWRENCE LIVERMORE LABORATORY

University of California | Livermore, California | 94550