STAN-CS-73-353 SU-SEL-73-012

Random Arrivals and MTPT Disk Scheduling Disciplines

by

Samuel H. Fuller

August 1972

Technical Report No. 29

This work was conducted while the author was a
Hertz Foundation Graduate Fellow at Stanford
University, Stanford, Calif., and was partially
supported by the Joint Services Electronics
Program, U.S. Army, U.S. Navy, and U.S. Air
Force under Contract N-00014-67-A-0112-0044.
Computer time was provided by the Atomic Energy
Commission under Contract AT-(0403) 515.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY • STANFORD, CALIFORNIA

STAN-CS-73-353 SEL 73-012

RANDOM ARRIVALS AND MTPT DISK SCHEDULING DISCIPLINES

by

Samuel H. Fuller

August 1972

Technical Report No. 29

DIGITAL SYSTEMS LABORATORY

Dept. of Electrical Engineering Dept. of Computer Science

Stanford University

Stanford, California

This work was conducted while the author was a Hertz Foundation Graduate Fellow at Stanford University, Stanford, Calif., and was partially supported by the Joint Services Electronics Program, U.S. Army, U.S. Navy, and U.S. Air Force under Contract N-00014-67-A-0112-0044. Computer time was provided by the Atomic Energy Commission undes Contract AT-(04-3)515.

RANDOM ARRIVALS AND MTPT DISK SCHEDULING DISCIPLINES

ABSTRACT

This article investigates the application of minimal-total-processing-time (MTPT) scheduling disciplines to rotating storage units when random arrival of requests is allowed. Fixed-head drum and moving-head disk storage units are considered and particular emphasis is placed on the relative merits of the MTPT scheduling discipline with respect to the shortest-latency-time-first (SLTF) scheduling discipline. The data presented are the results of simulation studies. Situations are discovered in which the MTPT discipline is superior to the SLTF discipline, and situations are also discovered in which the opposite is true.

An implementation of the MTPT scheduling algorithm is presented and the computational requirements of the algorithm are discussed, It is shown that the sorting procedure is the most time consuming phase of the algorithm.

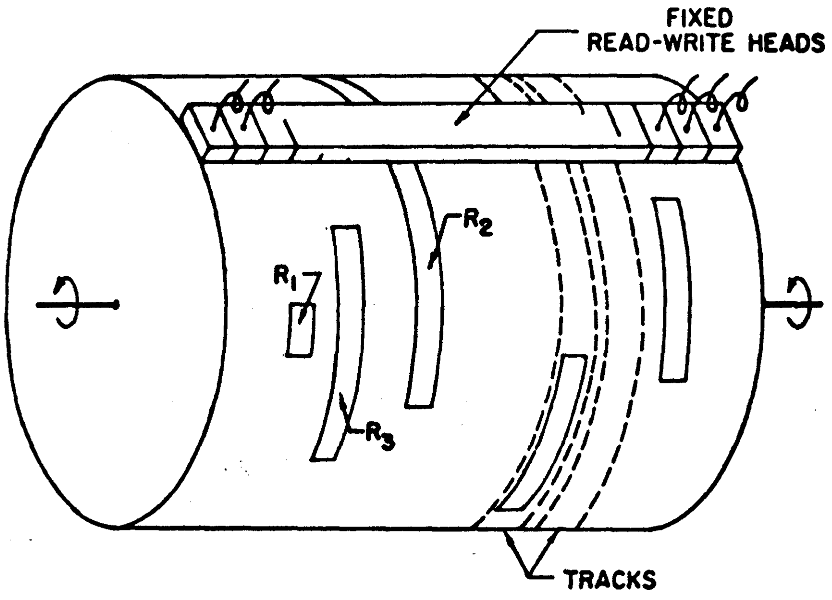
TABLE OF CONTENTS

LIST OF FIGURES

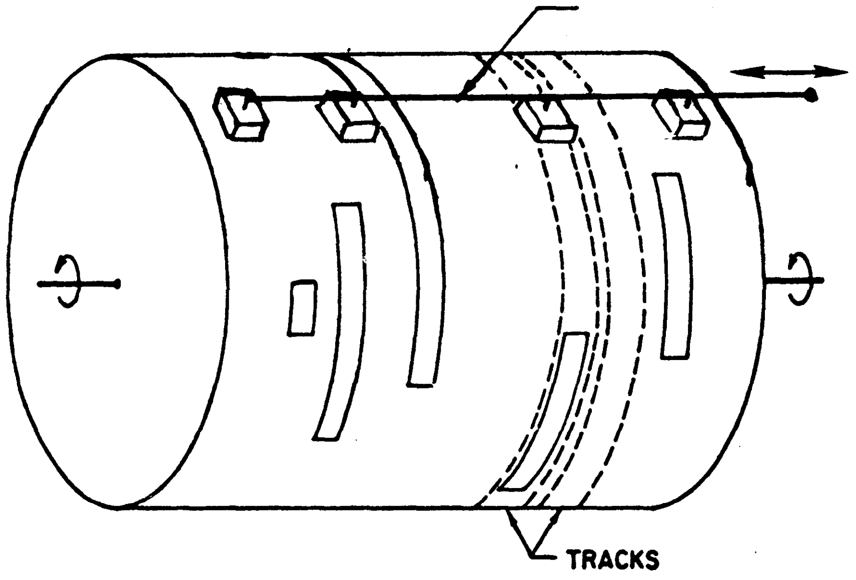
1. Introduction

This article looks at the practical implications of the drum scheduling discipline introduced in Fuller [1971]. The scope of this paper will include the classes of rotating storage devices shown in Fig. 1.1. Let the device in Fig. 1.1(a) be called a fixed-head file drum, or just fixed-head drum; the essential characteristics of a fixed-head drum is that there is a read-write head for every track of the drum's surface and consequently there is no need to move the heads among several tracks. Furthermore, the drum in Fig. 1.1(a) allows information to be stored in blocks, or records, of arbitrary length and arbitrary starting addresses on the surface of the drum. Physical implementations of a fixed-head file drum may differ substantially from Fig. 1.1(a); for instance, a disk, rather than a drum may be used as the recording surface, or the device may not rotate physically at all, but be a shift register that circulates its information electronically.

The other type of rotating storage unit that will be studied here is the moving-head file disk, or simply moving-head disk, the only difference between a moving-head disk and a fixed-head drum is that a particular read-write head of a moving-head disk is shared among several tracks, and the time associated with repositioning the read-write head over a new track cannot be ignored. A set of tracks accessible at a given position of the read-write arm is called a cylinder. Figure 1.1(b) shows the moving-head disk implemented as a moving-head drum, but this is just to simplify the drawing and reemphasize that 'fixed-head drum' and 'moving-head disk' are generic terms and are not meant to indicate a specific physical implementation.



(a) A fixed-head drum storage unit.



(b) A moving-head drum (disk) storage unit.

Figure 1.1. Storage units having rotational delays.

The analysis and scheduling of rotating storage units in computer systems has received considerable attention in the past several years [cf. Denning, 1967; Coffman, 1969; Abate et al., 1968; Abate and Dubner, 1969; Teorey and Pinkerton, 1972; Seaman et al., 1966; Frank, 1969]. In these papers, first-in-first-out (FIFO) and shortest-latency-time-first (SLTF) are the only two scheduling disciplines discussed for fixed-head drums or intra-cylinder scheduling in moving-head disks; in [Fuller, 1971] however, a new scheduling discipline is introduced for devices with rotational delays, or latency. This new discipline finds schedules for sets of I/O requests that minimize the total processing time for the sets of I/O requests. Moreover, if we let N be the number of I/O requests to be serviced, the original article presents a minimal-total-processing-time (MTPT)* scheduling algorithm that has a computational complexity on the order of NlogN, the same complexity as an SLTF scheduling algorithm.

Several other articles have been written since the MTPT scheduling discipline was originally presented, and they develop upper bounds and asymptotic expressions for differences between the SLTF and MTPT scheduling disciplines [Stone and Fuller, 1971; Fuller, 1972b]. Like the original paper, however, these articles address the combinatorial, or static, problem of scheduling a set of I/O requests; new requests are

* The algorithm was called an optimal drum scheduling algorithm in the original article, but this article refers to the algorithm as the minimal-total-processing-time (MTPT) drum scheduling algorithm. This name is more mnemonic and recognizes that other drum scheduling algorithms may be optimal for other optimality criteria.

not allowed to arrive during the processing of the original set of I/O requests. Although the MTPT scheduling discipline can always process a set of I/O requests in less time than the SLTF scheduling discipline, or any other discipline, we cannot extrapolate that the MTPT discipline will be best in the more complex situation when I/O requests are allowed to arrive at random intervals. On the other hand, even though the SLTF discipline is never as much as a drum revolution slower than the MTPT discipline when processing a set of I/O requests [Stone and Fuller, 1971], we are not guaranteed the SLTF discipline will take less than a drum revolution longer to process a collection of I/O requests when random arrivals are permitted.

Unfortunately, the analysis of the MTPT scheduling discipline presented in the previous articles does not generalize to MTPT scheduling disciplines with random arrivals. Moreover, attempts to apply techniques of queueing theory to MTPT schedules has met with little success. For these reasons, this article presents the empirical results of a simulator, written to investigate the behavior of computer systems with storage units having rotational delays [Fuller, 1972A].

Another important question not answered by the earlier papers is what are the computational requirements of the MTPT scheduling algorithm? Although the MTPT scheduling algorithm is known to enjoy a computational complexity on the order of NlogN, where N is the number of I/O requests to be scheduled, nothing has been said about the actual amount of computation time required to compute MTPT schedules. MTPT scheduling ' disciplines will be of little practical interest if it takes NlogN seconds to compute MTPT schedules, when current rotating storage devices have periods of revolution on the order of 10 to 100 milliseconds. No obvious, unambiguous measure of computation time exists, but this article will present the computation time required for a specific implementation of the MTPT scheduling algorithm, given in the Appendix, on a specific machine, an IBM 360/91.

The next section, Sec. 2, discusses the implementation of the MTPT scheduling algorithm that will be used in this article and presents the computation time required by this algorithm, and Sec. 3 introduces two modifications to the original MTPT algorithm. Section 4 shows the results of using the SLTF and MTPT scheduling disciplines on fixed-head drums where a range of assumptions are made concerning the size and distribution of I/O records, Section 5 continues to present the results of the simulation study but considers moving-head disks. We will see situations with fixed-head drums and moving-head disks, where the MTPT disciplines offer an advantage over the SLTF discipline; and the converse will also be seen to be true in other situations. The ultimate decision as to whether or not to implement a MTPT discipline for use in a computer system will depend on the distribution of record sizes seen by the storage units as well as the arrival rate of the I/O requests; the discussion in the following sections will hopefully provide the insight necessary to make this decision.

2. An Implementation of the Original MTPT Drum Scheduling Algorithm

In this section we will try to add some quantitative substance to the significant, but qualitative, remark that the MTPT drum scheduling algorithm has an asymptotic growth rate of $N \log N$.

An informal, English, statement of the original MTPT scheduling algorithm is included in the Appendix, along with a well-documented copy of an implementation of the MTPT scheduling algorithm, called MTPTO. This implementation of the MTPT algorithm has been done in conjunction with a larger programming project, and as a result two important constraints were accepted. First, MTPTO is written to maximize clarity and to facilitate debugging; the primary objective was not to write the scheduling procedure to minimize storage space or execution time. Secondly, the algorithm is written in FORTRAN because this is the language of the simulator with which it cooperates [Fuller, 1972A]. A glance at MTPTO, and its supporting subroutines: FINDCY, MERGE, and SORT, shows that a language with a richer control structure, such as ALGOL or PL/I, would have substantially simplified the structure of the procedures.

The results of this section were found with the use of a program measurement facility, called PROGLOOK, developed by R. Johnson and T. Johnston [1971]. PROGLOOK periodically* interrupts the central processor and saves the location of the instruction counter. The histograms of this section are the results of sampling the program counter as MTPTO is repetitively executed, and then the number of times the program counter is caught within a 32 byte interval is plotted as a function of the interval's starting address.

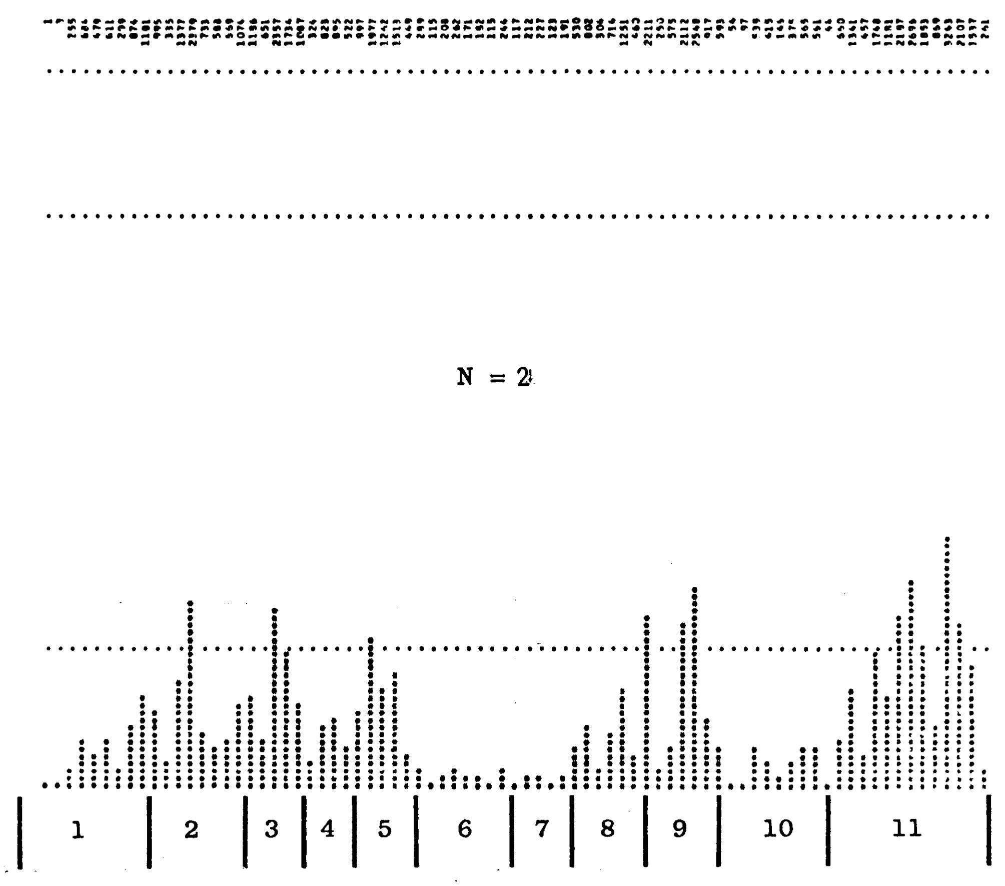
Figure 2.1 is the result of PROGLOOK monitoring MTPTO as it schedules N requests where N = 2, 3, 4, 6, 8, 10, 13, and 16. The abscissa of all the histograms is main storage locations, in ascending order and 32 bytes per line, and the ordinate is the relative fraction

* For all the results described here, PROGLOOK interrupted MTPTO every 500 microseconds.

Legend for Computation Time Histograms of Figure 2.1

1. Initialize data structures.
2. Find f_δ.
3. Redefine record endpoints relative to f_δ.
4. Construct the minimal cost, no-crossover permutation.
5. Find the membership of the cycle of ψ'.
6. Transform the permutation ψ' to the permutation ψ^o.
7. Transform the permutation ψ^o to the single-cycle permutation ϕ^o.
8. Construct MTPT schedule from ϕ^o.
9. Subroutine FINDCY: find cycle in which specified node is a member.
10. Subroutine MERGE: merge two cycles.
11. Subroutine SORT: an implementation of Shellsort.

N = 2



N = 3

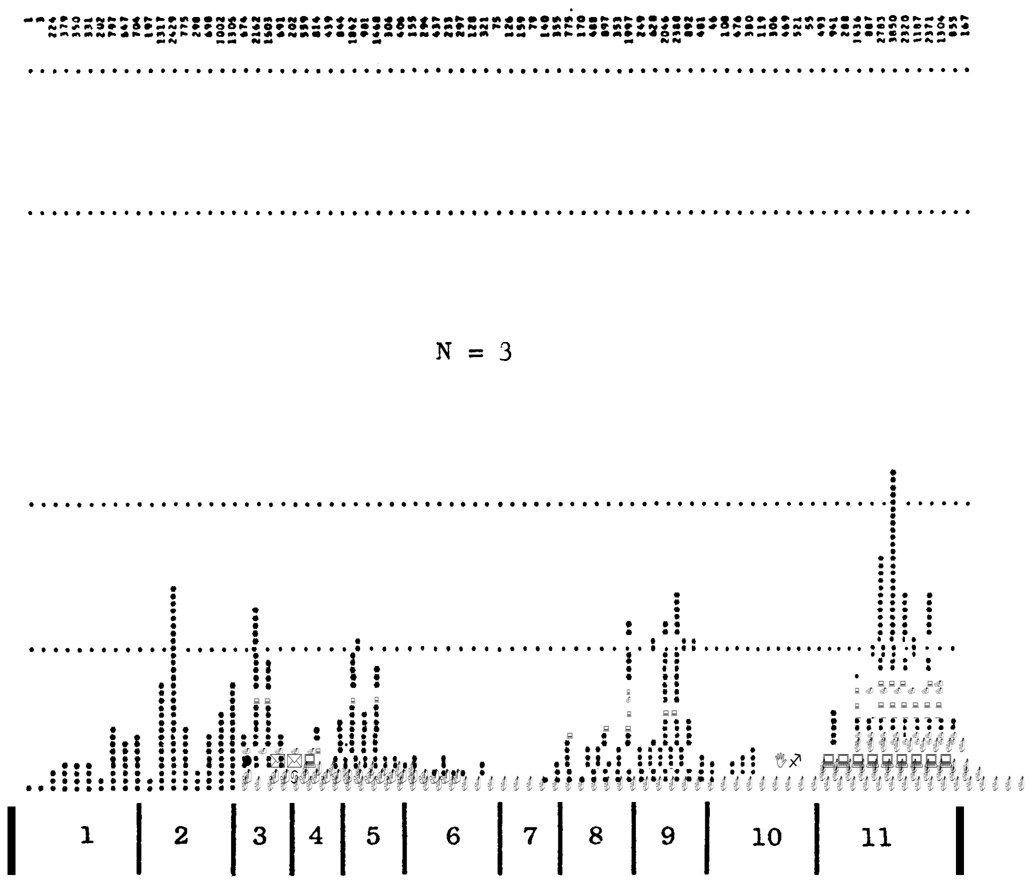
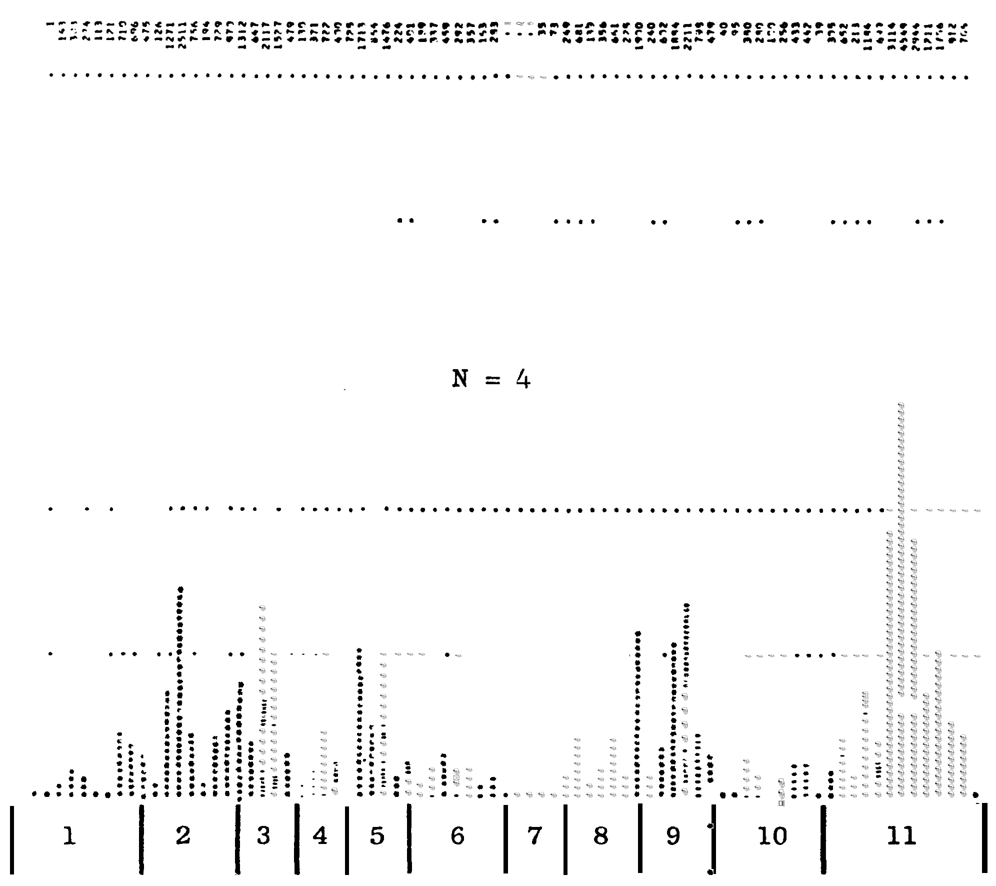


Figure 2.1. (continued)



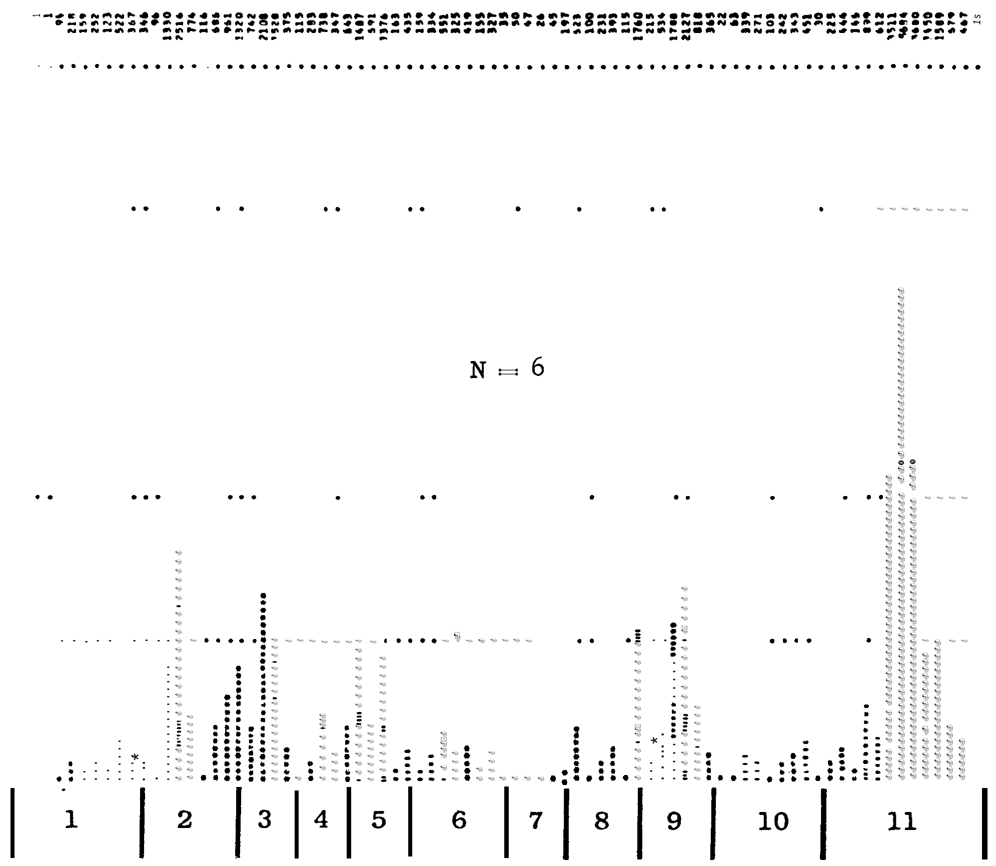
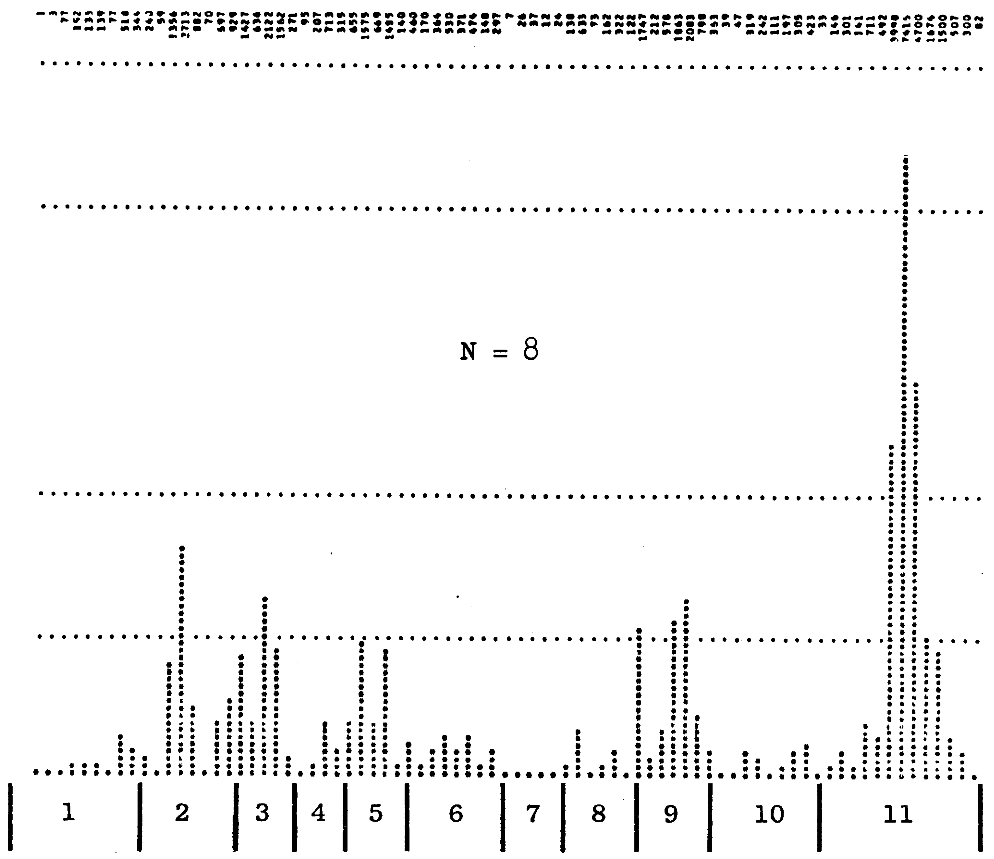


Figure 2.1. Computation time histograms for the MTPTO algorithm.



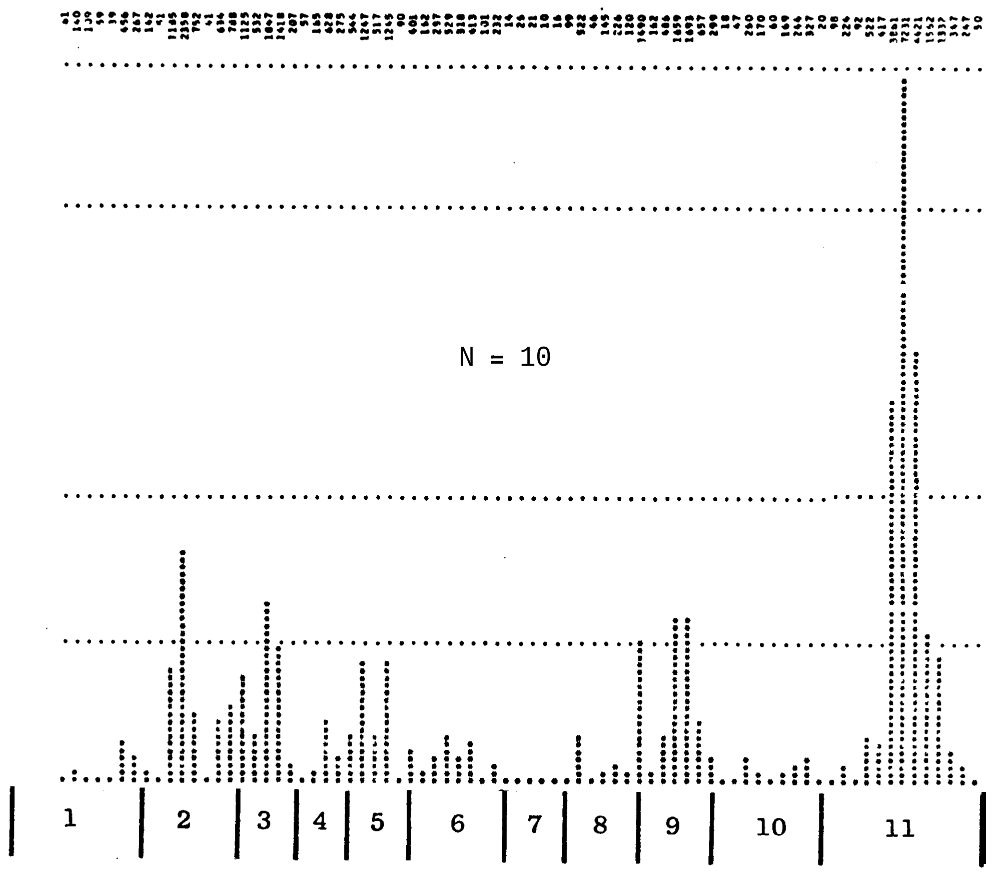
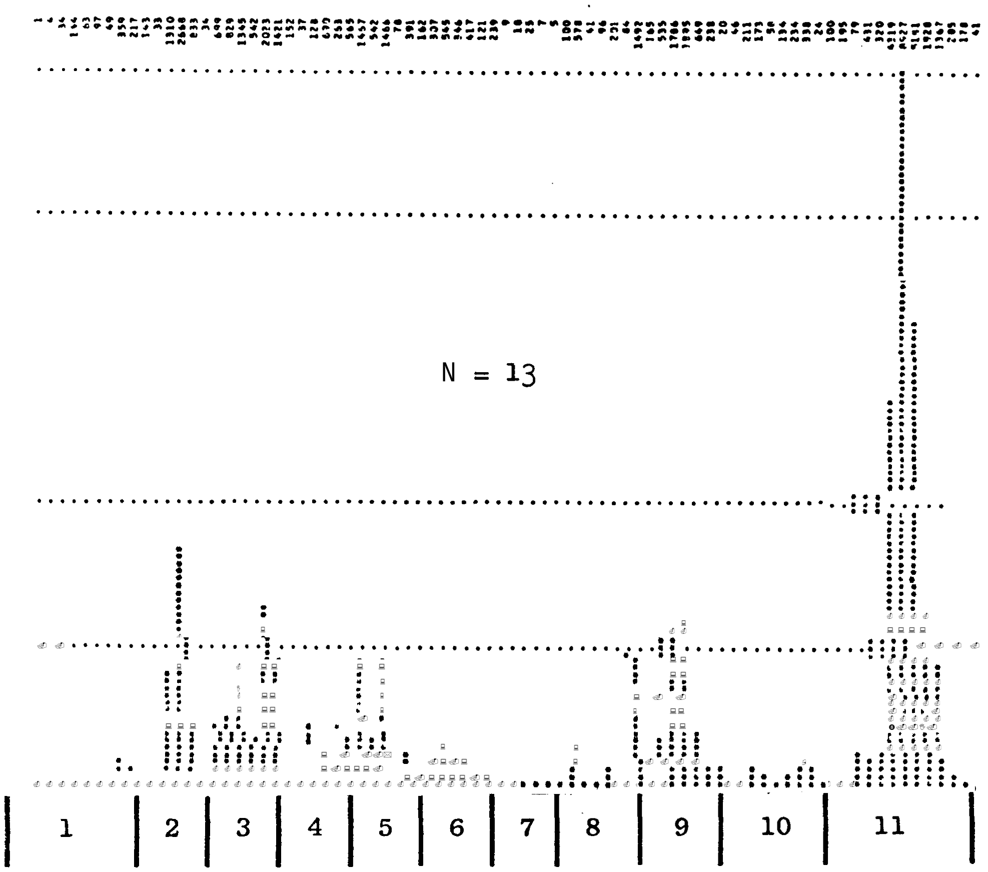


Figure 2.1. (continued)



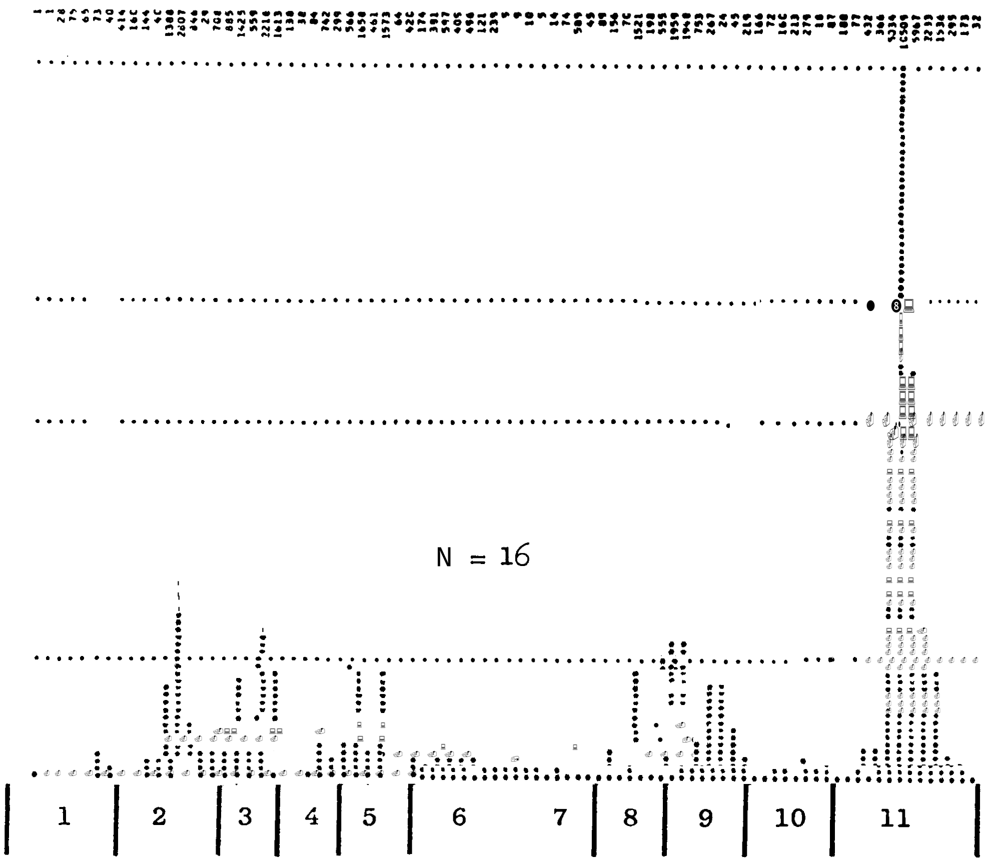


Figure 2.1. (continued)

of time MTPTO spends in an interval in order to schedule a set of N requests. The scales of the histograms in Fig. 2.1 are selected so that an interval whose computation time grows in direct (linear) proportion to N will remain a constant height in all the histograms. Figure 2.1 illustrates that only the sort procedure is growing at a rate perceptively faster than linear; for N in the range of 2 to 16 the rest of MTPTO experiences a linear, or less than linear, growth rate.

The particular sorting algorithm used in MTPTO is Shellsort [Shell, 1959; Hibbard, 1963] because, for most machines, Shellsort is the fastest of the commonly known sorting algorithms for small N [Hibbard, 1963]. If MTPTO is regularly applied to sets of records larger than 10, quick-sort, or one of its derivatives [Hoare, 1961; van Emden, 1970 A,B] may provide faster sorting. Whenever the algorithm is used for N more than three or four, Fig. 2.1 indicates that initially sorting the starting and ending addresses of the I/O requests is the most time consuming of the eleven major steps in MTPTO.

The upper curve of Fig. 2.2 is the expected execution time of MTPTO as a function of queue size. Note that it is well approximated by

$$100N + 50 \quad \text{microseconds}$$

for $N < 8$. For $N \geq 8$ the sorting algorithm begins to exhibit its greater than linear growth rate. The lower curve in Fig. 3.2 is the expected execution time for MTPTO minus the time it spends sorting; it can be approximated by

$$50N + 50 \text{ microseconds.}$$

The curves of Fig. 2.2 suggest an implementation of MTPTO might maintain a sorted list of the initial and final addresses of the I/O

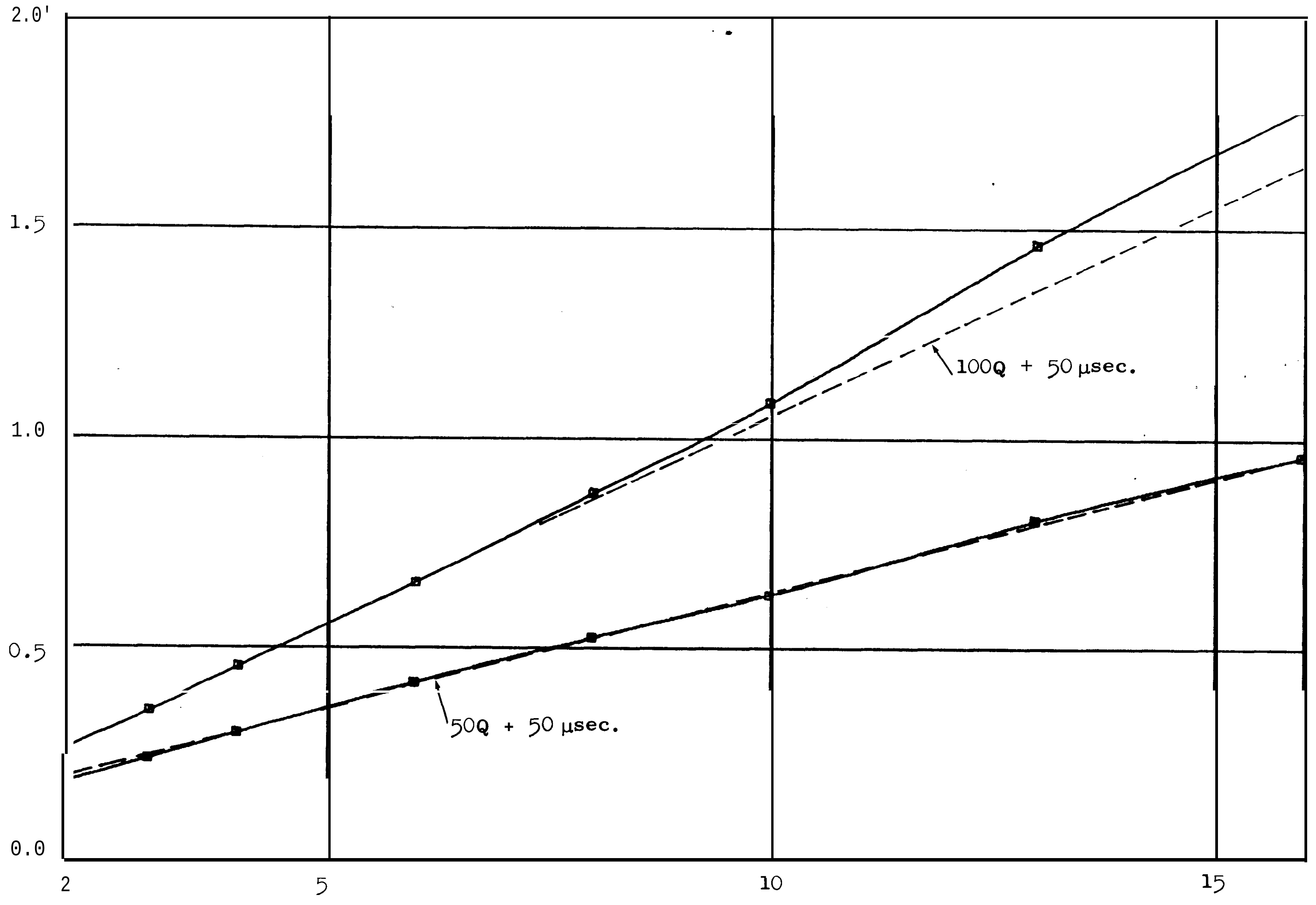


Figure 2.2. Computation time of MTPTO and MTPTO without sort phase, as a function of queue size.

requests at all times: techniques exist to add, or delete, I/O requests from the list in O(log N) steps [Adel'son-Vel-skiy and Landis, 1962]. Then a schedule could be found more quickly after it is requested since there would be no need to execute the costly sorting step.

Figure 2.2 can only be used as a rough guide to execution times since it is very sensitive to implementation. In particular, these statistics were collected on an IBM 360/91 and they must be appropriately scaled for processors that are slower, or faster. The algorithm is implemented in FORTRAN and has been compiled by the IBM FORTRAN H compiler* [IBM, 1971A]. An examination of the machine instructions produced by this compiler indicates a factor of 2 or 4 could be gained by a careful implementation in machine language. Furthermore, the starting and final values of the I/O requests are represented as double precision, floating point numbers, and practical implementations of MTPTO would very likely limit the starting and ending addresses to a small set of integers, 128 or 256 for example.

3. Two other MTPT scheduling algorithms

The original MTPT drum scheduling algorithm whose implementation was just discussed in the previous section, is not the only MTPT scheduling algorithm that may be of practical significance; for example, consider Fig. 3.1. Application of the MTPTO scheduling algorithm shows the schedule it constructs is

$$4,\ 3,\ 5,\ 1,\ 2. \qquad (3.1)$$

* During compilation, the maximum code optimization was requested, i.e.

```
// EXEC FORTHCLG,PARM.FORT='OPT=2'
```

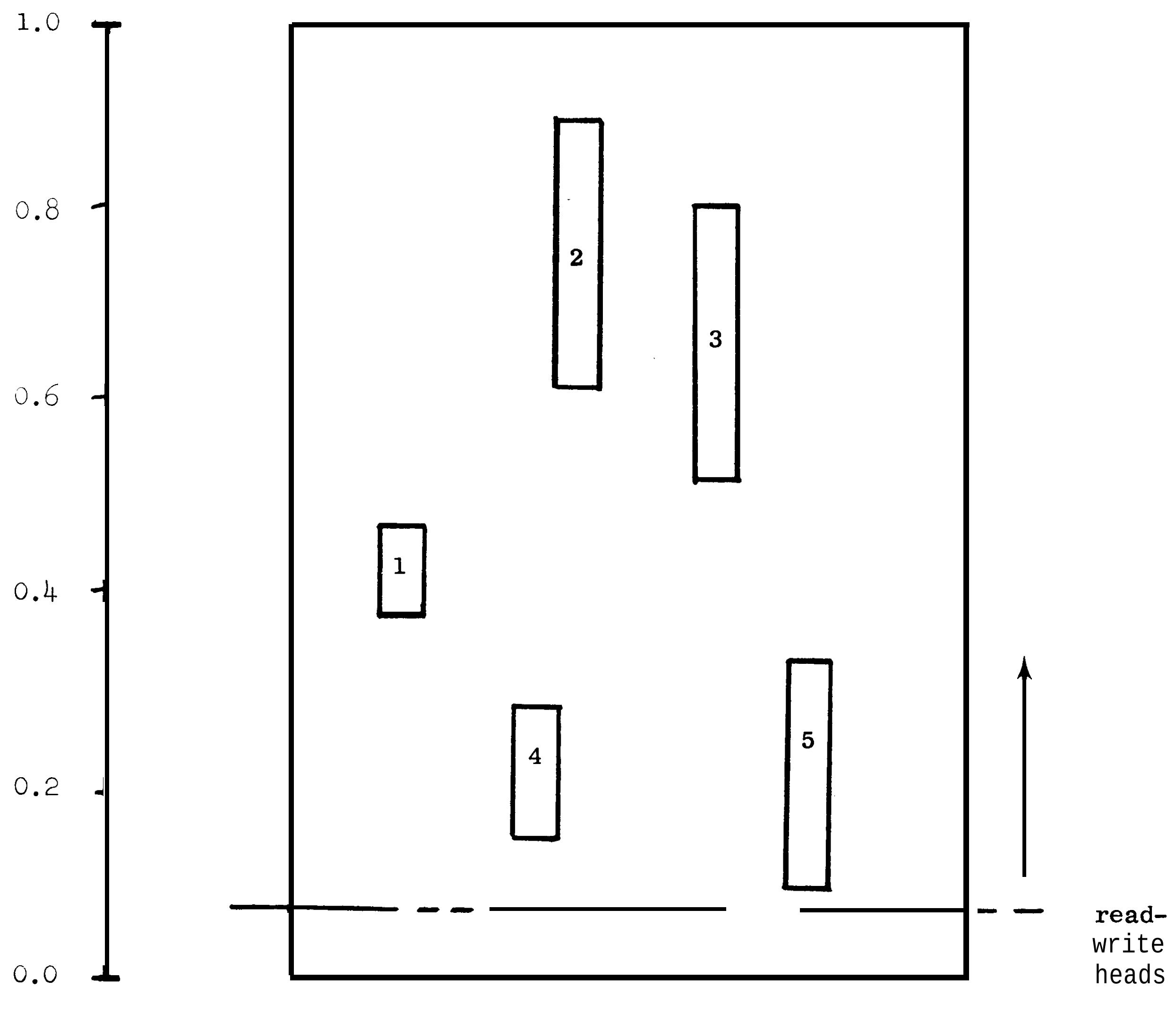


Figure 3.1. An example with four MTPT sequences.

This is a MTPT schedule, but then so are the sequences

5, 3, 4, 1, 2; (3.2)

4, 1, 3, 5, 2; and (3.3)

5, 1, 3, 4, 2. (3.4)

The only properties that can confidently be stated about all the MTPT schedules is that they require the same amount of processing time to service a particular set of I/O requests, and the last record that they process is the same.

Two of the MTPT sequences for the example of Fig. 3.1 share a distinct advantage over the MTPT sequence constructed by MTPTO. The last two sequences process record 1 on the first revolution while the sequence constructed by MTPTO, as well as the second sequence, overlook record 1 on the first revolution, even though they are latent at the time, and process it on the second revolution. Any reasonable measure of drum performance will favor the last two MTPT sequences over the first two.

Although MTPTO is the only MTPT scheduling algorithm that has been studied in detail and known to enjoy a computational complexity of NlogN, the above example indicates that other MTPT algorithms may be of interest. For this reason, two other MTPT scheduling algorithms have been implemented and are listed following the MTPTO algorithm in the Appendix,

The MTPT1 procedure corrects the deficit in the MTPTO procedure just illustrated; MTPT1 uses MTPTO to find a MTPT sequence and then traces through the schedule looking for records, like record 1 in our example, that can be processed at an earlier revolution without disturbing the processing of any of the other records. No claim is made here that MTPT1 is an NlogN process, it is used here to indicate how much better improved MTPT algorithms can be expected to be over MTPT0.

The third MTPT algorithm studied here, MTPT2, is what might be called the shortest-latency-time-first MTPT scheduling algorithm. Like MTPT1 it used MTPT0 to find a MTPT sequence for the I/O requests currently in need of service. Then, it sees if the first record in the MTPT sequence is closest to the read-write heads, if it is not it deletes the record with the shortest potential latency from the set of requests, applies the MTPT0 algorithm to the remaining I/O requests and checks if this new sequence is a MTPT sequence by comparing its processing time to the processing time of the MTPT0 sequence for the N requests. If not, it continues searching for the nearest record that starts a MTPT sequence. As in the case for the MTPT1 algorithm, the MTPT2 algorithm is not an NlogN process, the purpose of discussing it here is to see how the MTPT2 scheduling discipline compares with the other MTPT disciplines, as well as the SLTF discipline. In the example of Fig. 3.1, sequence (3.4) is the MTPT2 sequence and (3.3) is the MTPT1 sequence.

4. Random Arrivals and Fixed-Head Drums

We will now compare the performance of the MTPT0, MTPT1, MTPT2, and SLTF scheduling disciplines when they are used on a fixed-head drum (Fig. 1.1(a)) and I/O requests are allowed to arrive at random points in time. Before proceeding with the results, however, some discussion is needed to clarify the models of drum and record behavior that are used in the simulations.

As successive I/O requests arrive for service, some assumptions must be made about the location of the starting and ending addresses of the new I/O request. In at least one real computer system, it is reasonable to model the starting addresses of successive I/O requests as independent random variables uniformly distributed around the circumference of the drum, and to model the length of the records as exponentially distributed random variables with a mean of about one third of the drum's circumference [Fuller and Baskett, 1972].

The other assumption made here is that the arrival of I/O requests form a Poisson process. In other words, the inter-arrival time of successive I/O requests are independent random variables with the density function

$$f(t) = \lambda e^{-\lambda t}, \qquad \lambda > 0 \text{ and } t > 0.$$

A more realistic assumption might be to assume that the drum is part of a computer system with a finite degree of multiprogramming on the order of 4 to 10. So little is known about the relative merits of SLTF and MTPT disciplines, however, it is prudent to keep the model as simple as possible until we have a basic understanding of these scheduling disciplines.

Several other minor assumptions must be made, and at each point an attempt was made to keep the model as simple as possible. The time required to compute the scheduling sequence is assumed to be insignificant, the endpoints are allowed to be real numbers in the interval $[0,1)$, the period of revolution of the drum will be assumed constant and equal to τ, no distinction is made between reading and writing on the drum, and no attempt is made to model the time involved in electronically switching the read-write heads.

A number of different measures of drum performance are reasonable. In this section, however, three measures will be used: expected waiting time, the standard deviation of the waiting time, and expected duration of drum busy periods. I/O waiting time will be defined here in the common queueing theory sense; that is, the time from the arrival of an I/O request until that I/O request has completed service. Let a drum be defined by busy when it is not idle, in other words the drum is busy when it is latent as well as when it is actually transmitting data.

These three measures of performance will be shown as a fraction of ρ, where ρ is the ratio of the expected record transfer time to the expected interarrival time. Use of the normalized variable ρ assists in the comparison of simulations with records of different mean lengths and always provides an asymptote at $\rho = 1$. In the figures in this section, ρ is shown from 0 to .75. The statistics of drum performance for $\rho > .75$ blow up very fast, and moreover the expense required to run simulations of meaningful precision for large ρ outweighed the possible insight that might be gained. Observed p's for operational computer systems are commonly in the range of .1 to .5 [cf. Bottomly, 1970].

The precision of the summary statistics of the following simulations is described in detail in [Fuller, 1972A]. All the points on the graphs in this article represent the result of simulation experiments that are run until 100,000 I/O requests have been serviced; this number of simulated events proved sufficient for the purposes of this article. The sample mean of the I/O waiting times, for example, are random variables with a standard deviation less than .002 for $\rho = .1$ and slightly more than .1 for $\rho = .75$.

The corresponding statistics for the expected duration of busy intervals are

$$\sigma_{\bar{x}} = .005 \quad \text{for } \rho = .1,$$

$$\sigma_{\bar{x}} = .1 \quad \text{for } \rho = .55.$$

The variability of the simulation points is often hard to see, but plots of the residuals at the bottom of the graphs often show the experimental error.

All the graphs in this section are really two sets of curves. First, they show the measure of performance as a function of ρ for the four scheduling disciplines studied: MTPT0, MTPT1, MTPT2, and SLTF; and then on the same graph the difference between SLTF and each of the three MTPT disciplines is shown. The residual curves more clearly demonstrate the relative performance of the scheduling disciplines than can be seen from directly studying the original curves. Some of the curves, particularly the residual curves, do not go all the way to the right hand side of the graph; this is simply because it was felt that the marginal gain in insight that might be obtained from the additional points did not justify the additional cost.

Figure 4.1 shows the mean I/O waiting times for a fixed-head drum servicing record with lengths drawn from an exponential distribution with a mean of 1/2, i.e. $\mu = 2$ and density function

$$f(t) = \mu e^{-\mu t}, \qquad t > 0.$$

Figure 4.1 displays an unexpected result, the SLTF and MTPT2 curves lie directly on top of each other to within the accuracy of the simulation. MTPT0 and MTPT1 perform progressively poorer than MTPT2 and SLTF as the arrival rate of I/O requests is increased. MTPT0, MTPT1, and MTPT2 show increasingly smaller mean waiting times; this is consistent with the observation that MTPT0 is an 'arbitrary' MTPT schedule while MTPT2, and to a lesser extent MTPT1, look at several MTPT schedules in the process of deciding how to sequence the set of I/O requests. We will see in all the figures that follow in this section, and the next, that MTPT0, MTPT1, and MTPT2 consistently perform in the same order of performance shown in Fig. 4.1. The observation that MTPT0 and MTPT1 are poorer scheduling disciplines than the SLTF disciplines for heavily loaded drums is not too surprising. It is very rare for large ρ that all the current requests will be processed before any new request arrives. When an additional request does arrive, a new MTPT sequence must be calculated and the non-robust nature of the MTPT0 algorithm suggests there will be little resemblance in the two sequences.

Figure 4.2 shows the standard deviation of the I/O waiting time for a fixed-head drum and records with lengths exponentially distributed with $\mu = 2$, i.e. the same situation as Fig. 4.1. As in Fig. 4.1, the SLTF and MTPT2 sequences behave very similarly except that the MTPT2 curve is below the SLTF by a small, but distinct, amount, indicating that the MTPT2 discipline, while providing the same mean waiting time exhibits a smaller variance, or standard deviation, than does the SLTF discipline.

Figures 4.3 and 4.4 show the mean waiting time for drums with records having exponentially distributed records lengths with means of 1/3 and 1/6 respectively. These figures reinforce our general impressions from Fig. 4.1. The relatively poor performance of the MTPT0 and MTPT1 disciplines becomes more pronounced as the mean record size decreases; this follows from the observation that the number of MTPT sequences, for a given ρ, increases as μ increases. We can see this by

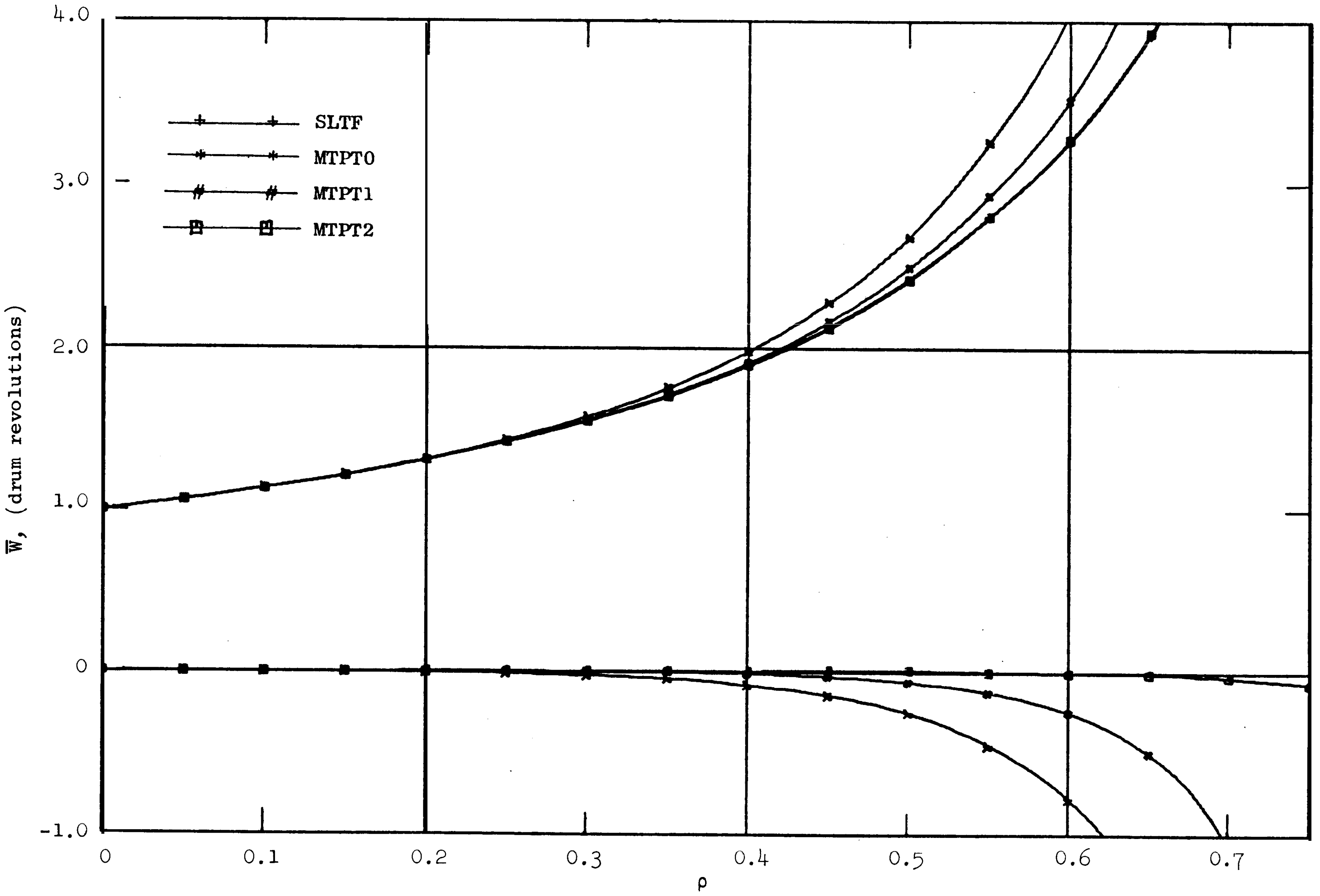


Figure 4.1. The expected waiting time when the records are exponentially distributed records with $\mu = 2$.

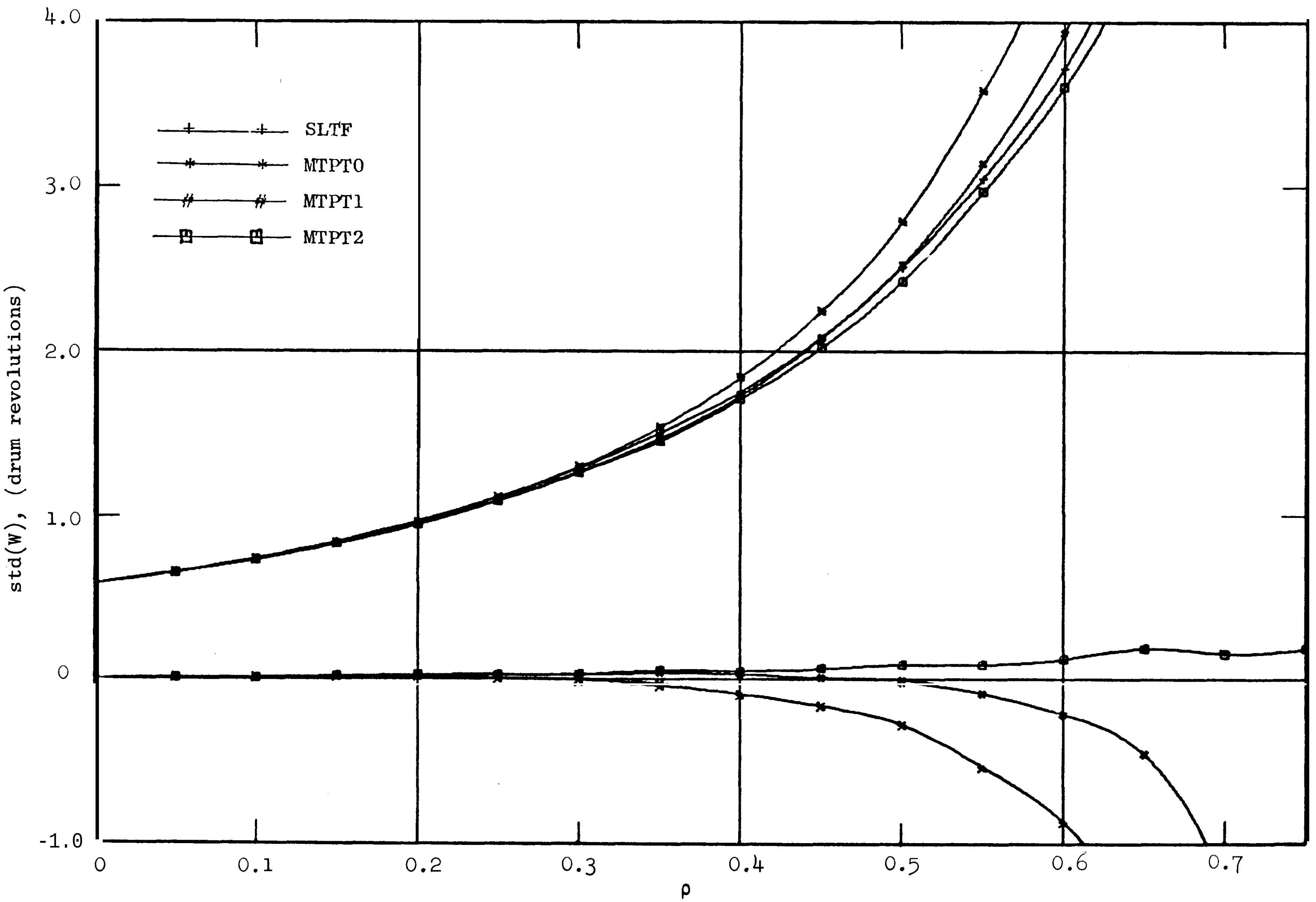


Figure 4.2. Standard deviation of the waiting time for exponentially distributed records with $\mu = 2$.

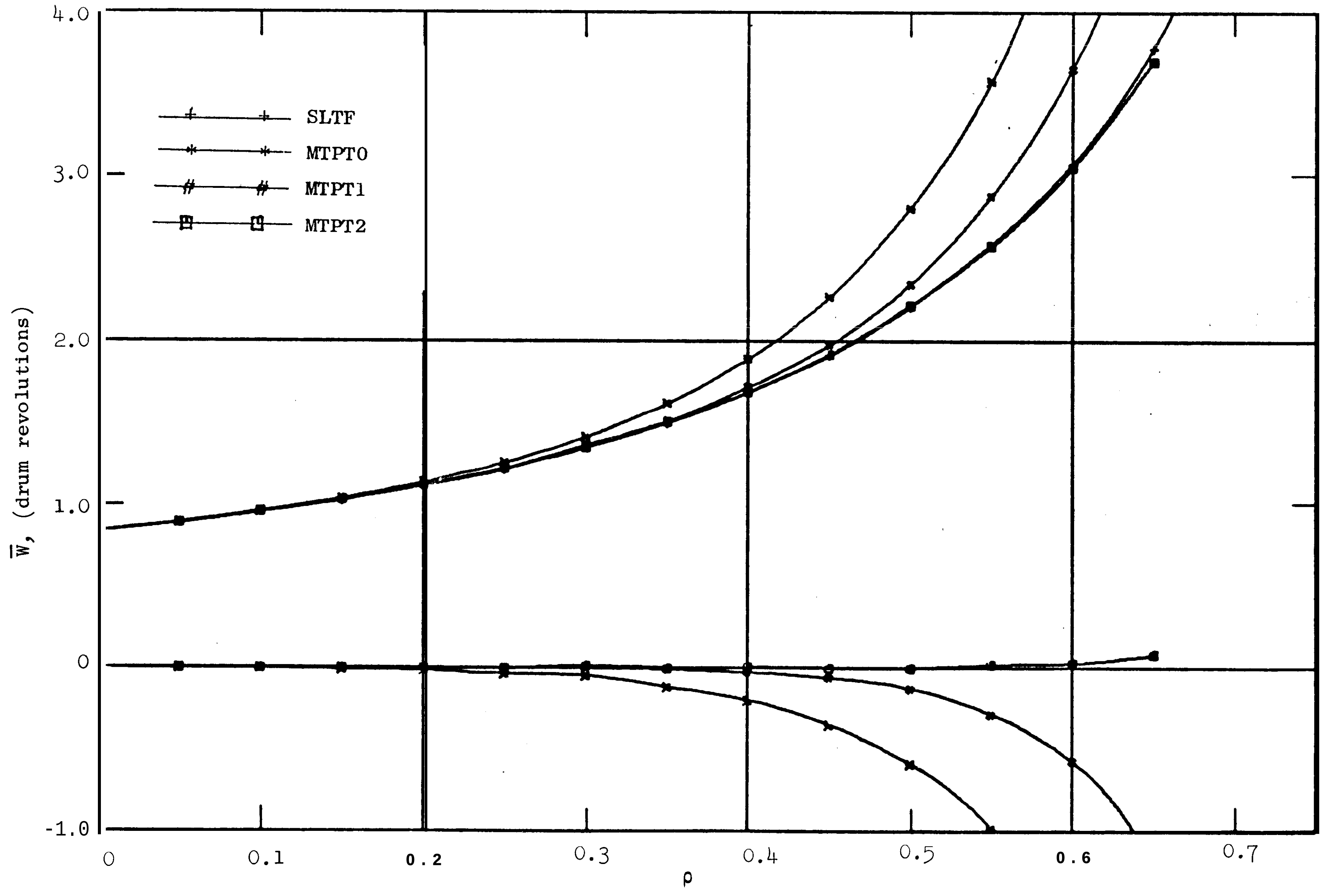


Figure 4.3. The expected waiting time when the records are exponentially distributed with $\mu = 3$.

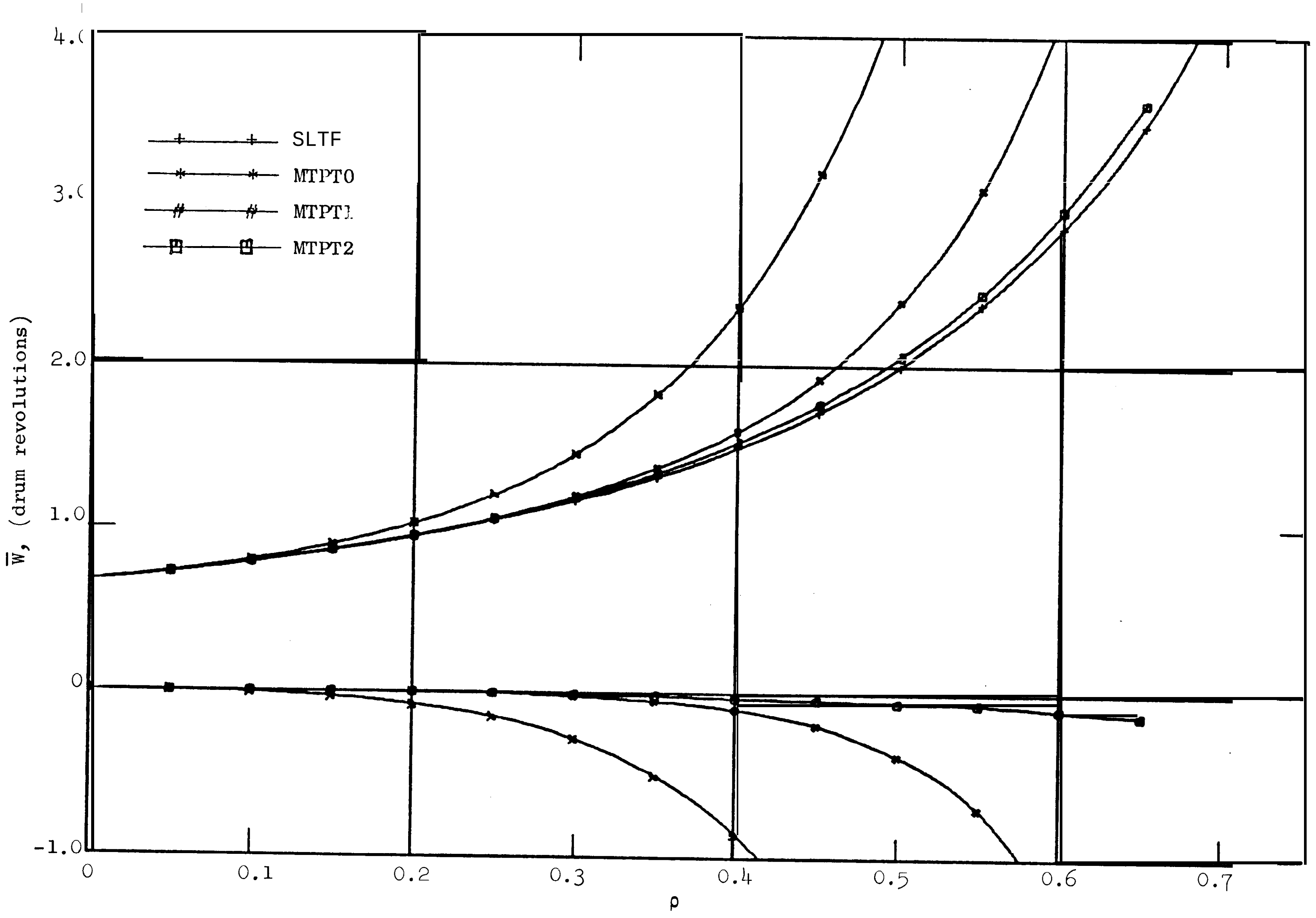


Figure 4.4. The expected waiting time when the records are exponentially distributed with $\mu = 6$.

applying Little's formula, $\bar{L} = \lambda\bar{W}$, or equivalently, $\bar{L} = \rho\bar{W}\mu$. Hence, the mean queue depth for the corresponding ρ coordinate in Figs. 4.1 and 4.4 is three times deeper in Fig. 4.4 than in Fig. 4.1.

A disturbing aspect of Fig. 4.4 is that the MTPT2 sequence is slightly worse than the SLTF sequence, a change from the identical performance indicated in Figs. 4.1 and 4.3. The difference is too large to be dismissed as a result of experimental error in the simulation; these two disciplines were simulated a second time, with a different random number sequence, and the same difference was observed. The standard deviation of the I/O wait times whose means are shown in Figs. 4.3 and 4.4 are essentially identical to Fig. 4.2 with the same trend exhibited in the mean; the difference in the MTPT0 and MTPT1 curves, with respect to the SLTF and MTPT2 curves, becomes increasingly pronounced as the mean record size is decreased.

Figures 4.5-4.8 explore the relative merits of the four scheduling disciplines along another direction. Figures 4.5 and 4.6 show the performance of a drum with record lengths uniformly distributed from zero to a full drum revolution, and Figs. 4.7 and 4.8 show the performance of a drum with record exactly 1/2 of a drum revolution in length. Figures 4.1, 4.5, and 4.7 show the mean I/O waiting time for drums with records that all have a mean of 1/2, but have variances of 1/4, 1/12, and 0 respectively. This set of three curves clearly shows that as the variance of the record sizes is decreased, the relative performance of the MTPT sequences improves with respect to the SLTF discipline.

The standard deviation of the waiting times for uniformly distributed record lengths, Fig. 4.6, and constant record lengths,

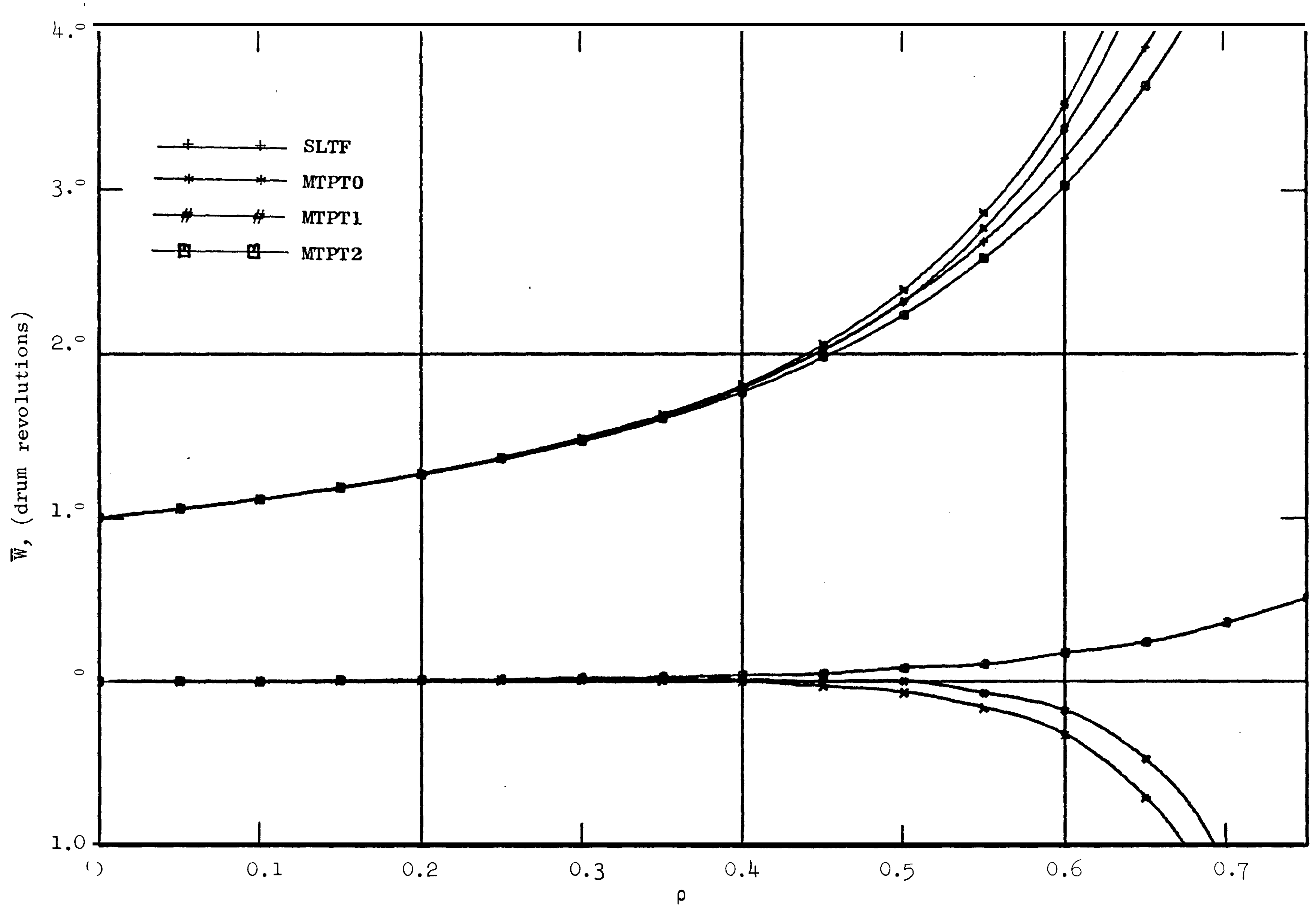


Figure 4.5. The expected waiting time when the records are uniformly distributed from zero to a full drum revolution.

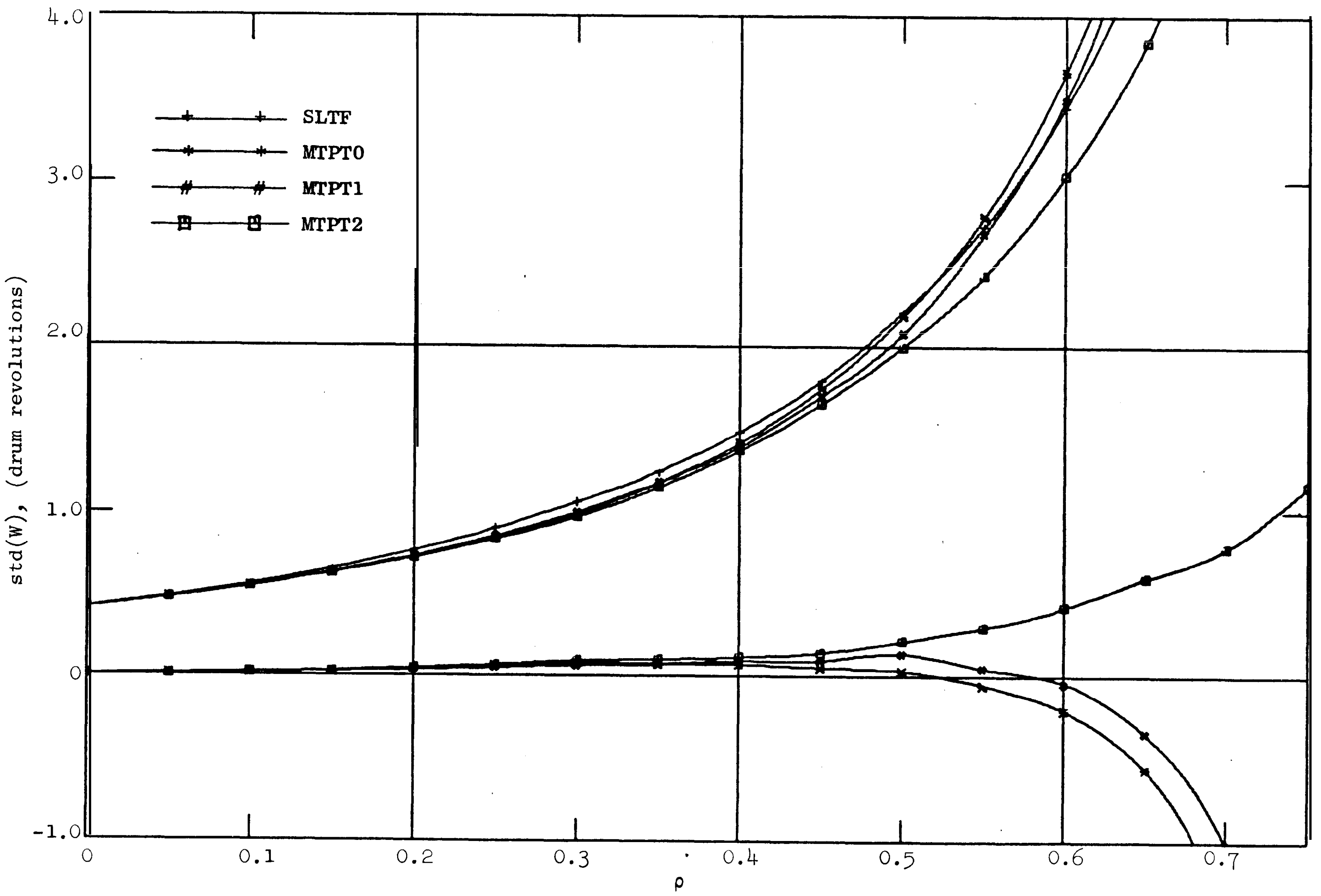


Figure 4.6. The standard deviation of the waiting time when the records are uniformly distributed between zero and a full drum revolution.

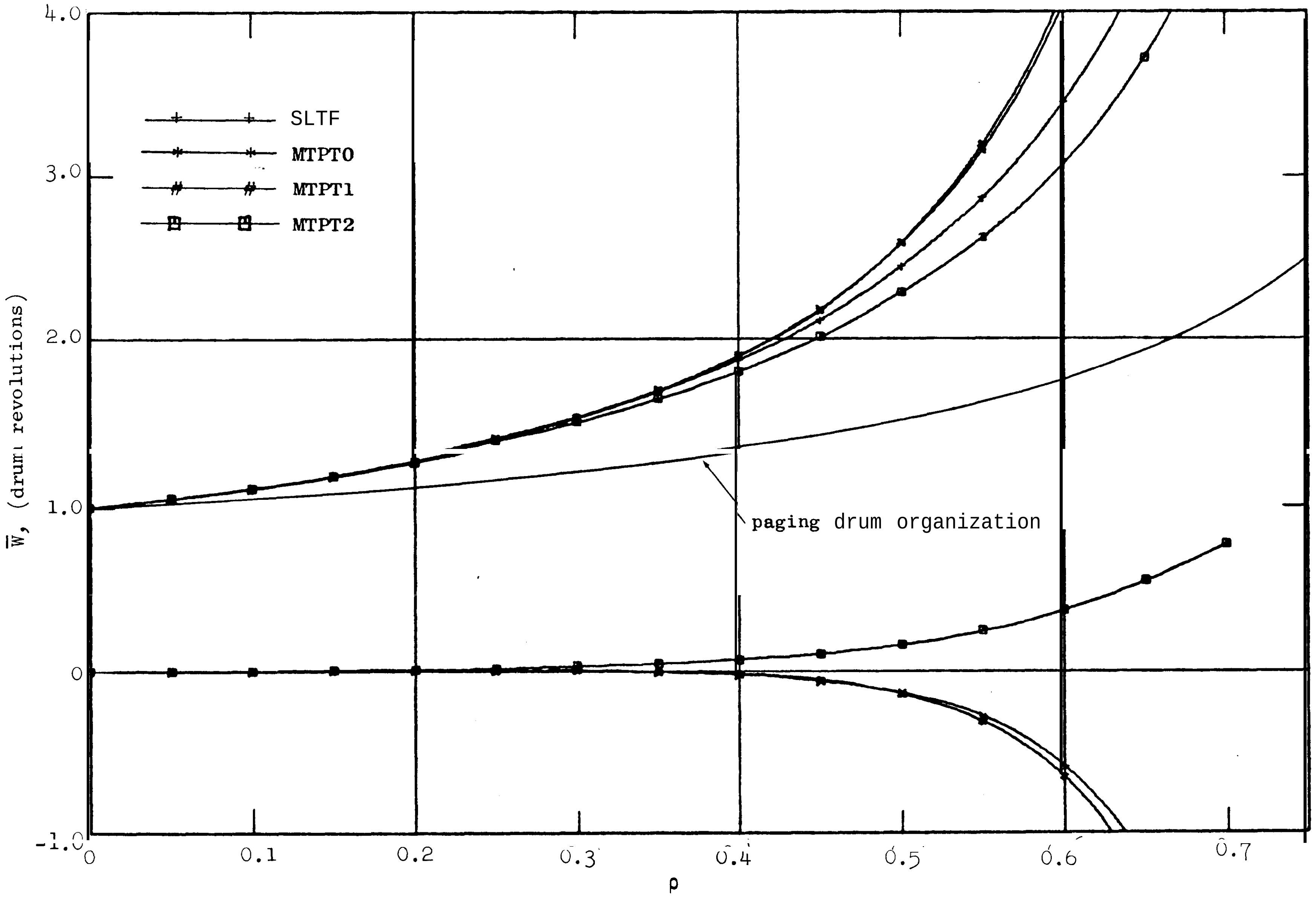


Figure 4.7. The expected waiting time when all the records are 1/2 the drum's circumference in length.

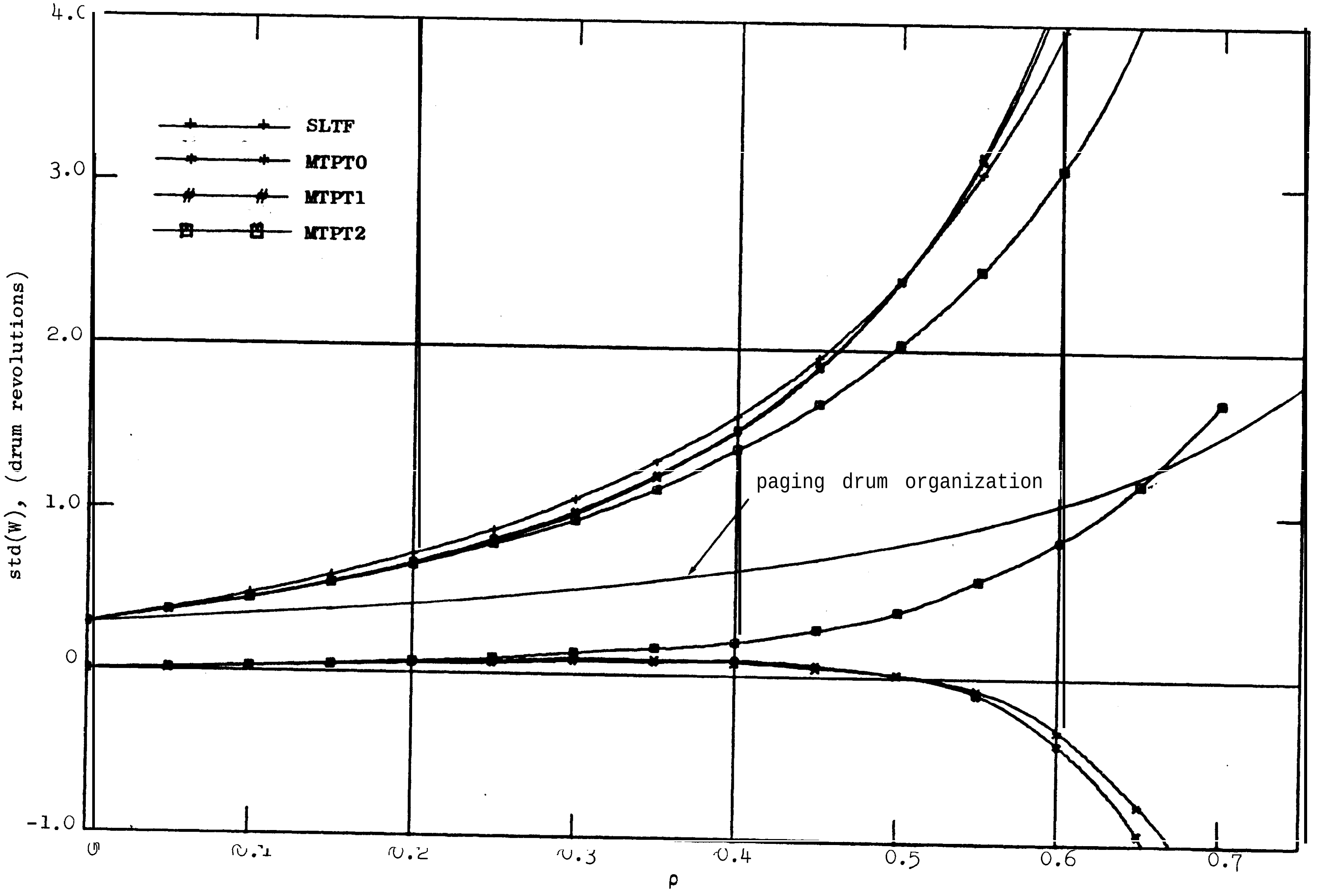


Figure 4.8. The standard deviation of the waiting time when all the records are 1/2 the drum's circumference in length.

Fig. 4.7, show an even more impressive improvement in the MTPT schedules as the variance of the record lengths are decreased. Clearly the advantage of MTPT disciplines is enhanced as the variation in the lengths of records is decreased.

Figures 4.8 and 4.9 include another smooth curve as well as the four curves already discussed. These curves show the mean and standard deviation of a drum organized as a paging drum, with 2 pages per track. There is no need to simulate a paging drum since Skinner [1967] and Coffman [1969] derived the exact formula for the mean waiting time and Fuller [1972C] derive@ the standard deviation. The paging drum shows a pronounced improvement over any of the four scheduling disciplines discussed in this article, and if a drum is only going to service fixed-size records, Figs. 4.7 and 4.8 indicate the pronounced advantages in organizing the drum as a paging drum.

Figures 4.9 and 4.10 illustrate another measure of drum performance, the mean drum busy interval. Since a MTPT scheduling discipline minimizes the total processing time of the outstanding I/O requests, it might be suspected the MTPT disciplines will minimize the drum's busy periods even when random arrivals are allowed. Figure 4.9 shows the mean drum busy interval for a drum with exponentially distributed records, $\mu = 2$. The result is surprisingly close to what we might have guessed from previous combinatorial observations [Fuller, 1972B]. We see that the expected difference between the MTPT discipline and the SLTF when no random arrivals are allowed, approached the mean value of the records' length, modulo the drum circumference, as N gets large. In other words, for exponentially distributed records, with $\mu = 2$ and the drum circumference defined to be unity, the mean record length, modulo 1, is .3435.

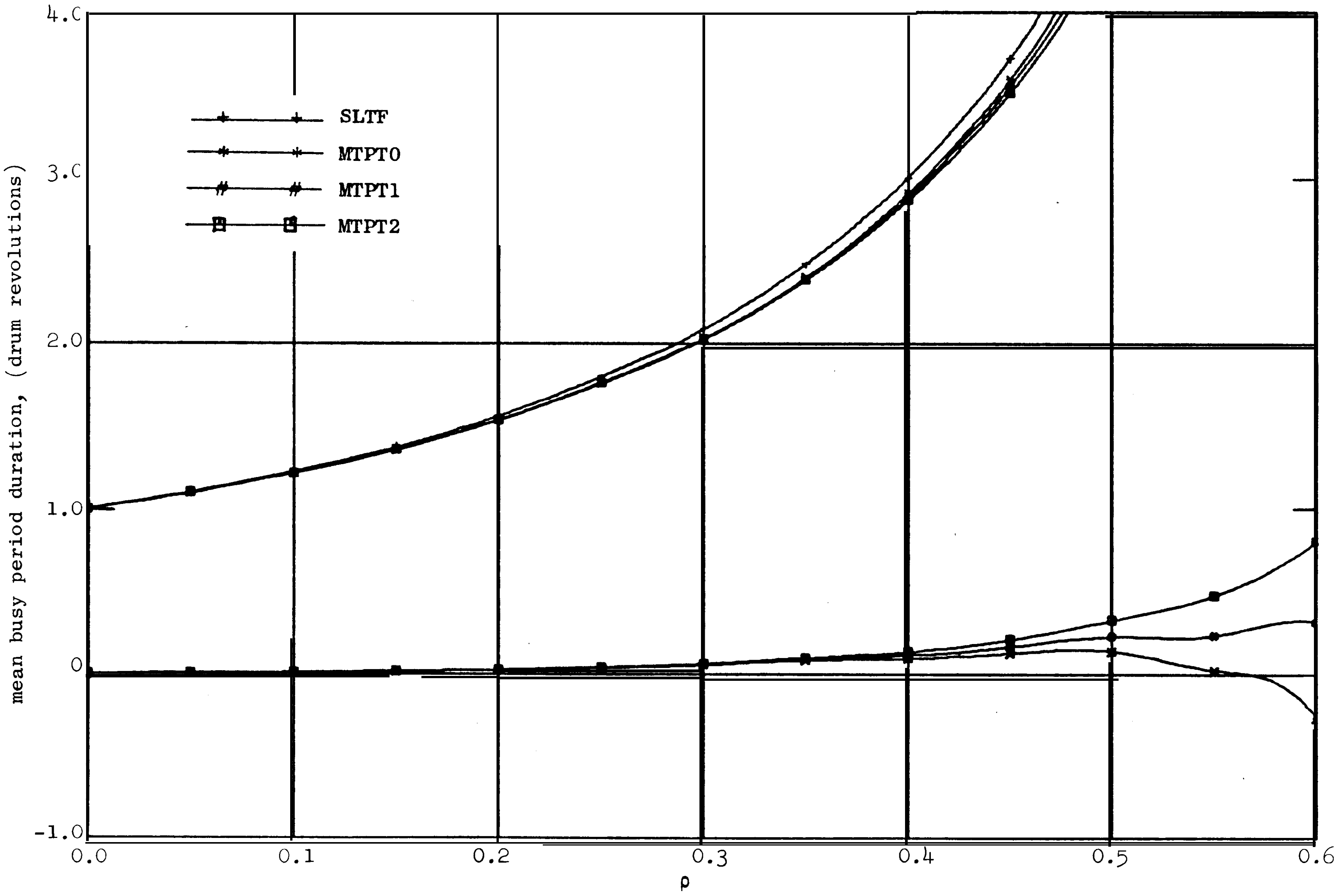


Figure 4.9. The mean duration of the busy intervals when the records are exponentially distributed with $\mu = 2$.

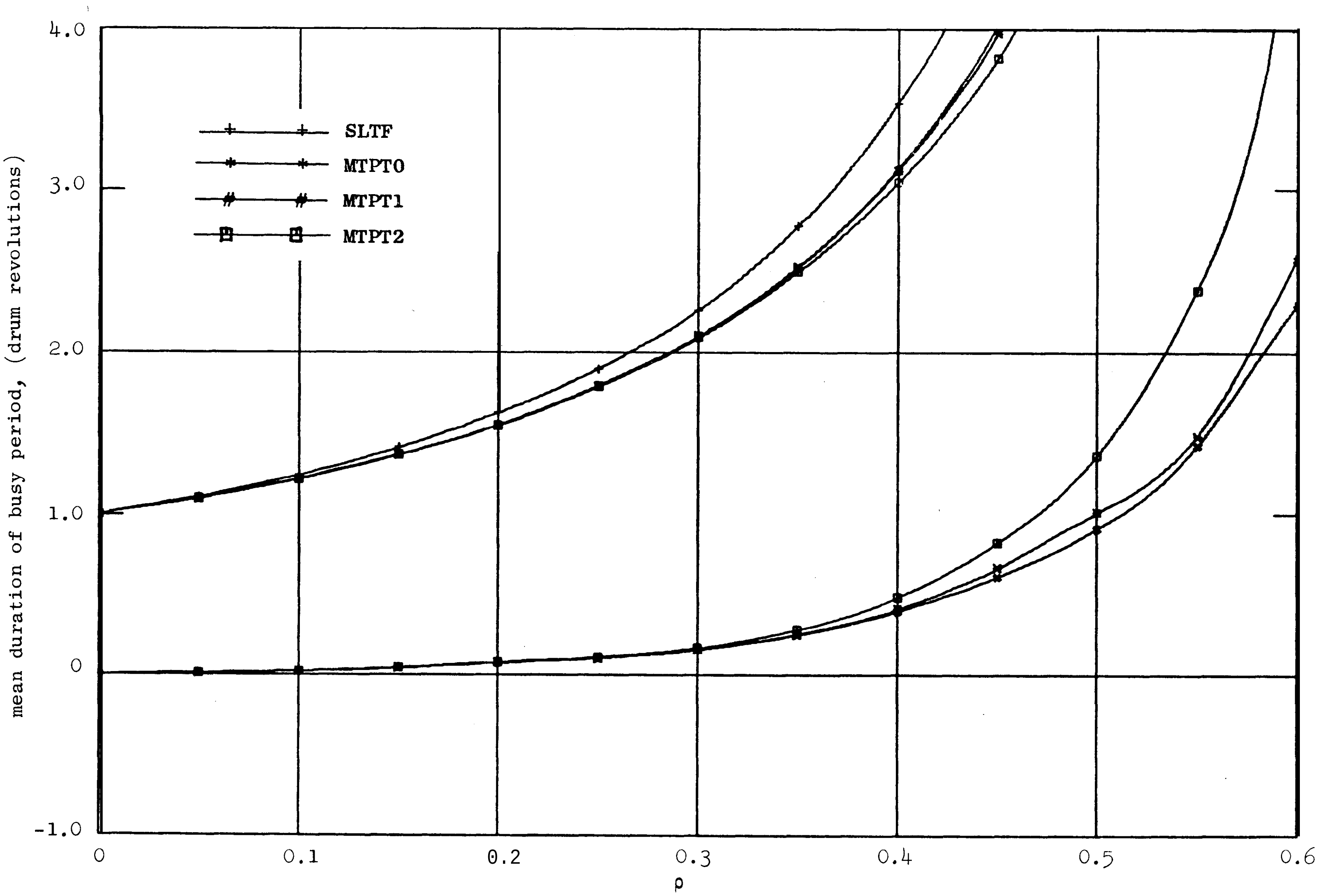


Figure 4.10. The mean duration of the busy intervals when the records are all 1/2 the drum's circumference in length.

For fixed-size records, the mean record length, modulo a drum revolution is still 1/2. Both Figs. 4.9 and 4.10 show that the best of the MTPT disciplines, MTPT2, and the SLTF discipline are approaching a difference of the expected record size, modulo the drum's circumference.

5. Random Arrivals and Moving-Head Disks

A storage device even more common than a fixed-head drum is the moving-head disk, or drum, schematically depicted in Fig. 1.1(b). For the purposes of this article, the only difference between a moving-head disk and a fixed-head drum is that a single read-write head must be shared among several tracks, and the time required to physically move the head between tracks is on the same order of magnitude of a drum revolution, and hence cannot be ignored even in a simple model, as was the analogous electronic head-switching time in the fixed-head drum.

Before proceeding with the results of this section a few more comments must be made on the simulations in order to completely specify conditions leading to the results of this section. Some assumption must be made concerning the time to reposition the head over a new cylinder. Let AC be the distance, in cylinders, that the head must travel, then the following expression roughly models the characteristics of the IBM 3330 disk storage unit [IBM, 1971B]:

$$\text{seek time} = 0.6 + .0065\ \text{AC}\ . \qquad (5.1)$$

Our unit of time in Eq. (5.1) is a disk (drum) revolution, and in the case of the IBM 3330, the period of revolution is 1/60 of a second. The relative performance of the four scheduling disciplines of this article is insensitive to the exact form of Eq. (5.1) and replacing Eq. (5.1) by

$$\text{seek time} = 1 + .07\ AC,$$

which approximates the IBM 2314 [IBM,1965] does not change any of the conclusions of this section.

A decision has to be made concerning the inter-cylinder scheduling discipline. Although an optimal disk scheduling discipline might integrate the intra-queue and inter-queue scheduling disciplines, in this article they will be kept separate. The inter-queue scheduling discipline chosen for this study is called SCAN, [Denning, 1967] (also termed LOOK by Teorey and Pinkerton [1972]). SCAN works in the following way: when a cylinder that the read-write heads are positioned over is empty, and when there exists another cylinder that has a non-empty queue, the read-write heads are set in motion toward the new cylinder. Should more than one cylinder have a non-empty queue of I/O requests the read-write heads go to the closest one in their preferred direction; the preferred direction is simply the direction of the last head movement. This inter-cylinder discipline is called SCAN because the read-write heads appear to be scanning, or sweeping, the disk surface in alternate directions.

SCAN gives slightly longer mean waiting times than the SSTF (shortest-seek-time-first) inter-cylinder scheduling discipline. However, from Eq. (5.1) we see the bulk of the head movement time is not a function of distance, and SCAN has the attractive property that it does not degenerate, as SSTF does, into a 'greedy' mode that effectively ignores part of the disk when the load of requests becomes very heavy [Denning, 1967].

An I/O request arriving at a moving-head disk has a third attribute, in addition to its starting address and length, the cylinder from which it is requesting service. In the spirit of keeping this model as simple as possible, we will assume that the cylinder address for successive I/O records are independent and uniformly distributed across the total number of cylinders. While no claim is made that this models reality, like the Poisson assumption it simplifies the model considerably and allows us to concentrate on the fundamental properties of the scheduling disciplines. Furthermore, the results of this section are shown as a function of the number of cylinders on the disk, where we let the number of cylinders range from 1 (a fixed-head disk) to 50. Conventional disk storage units have from 200 to 400 cylinders per disk but for any given set of active jobs, only a fraction of the cylinders will have active files. Therefore, the results of this section for disks with 5 and 10 cylinder is likely to be a good indication of the performance of a much larger disk that has active files on 5 to 10 cylinders.

Finally, in all the situations studied here, the records are assumed to be exponentially distributed with a mean of 1/2. This assumption is both simple and realistic and the observations of the previous section for other distributions of record lengths indicates the sensitivity of this assumption.

Figure 5.1 is the mean I/O waiting time for the SLTF, MTPT0, MTPT1 and MTPT2 scheduling disciplines for the disk model just described; the numbers of cylinders per disk include 1, 2, 5, 10, 25 and 50. Note the abscissa is now labeled in arrivals per disk revolution (λ) rather than ρ, and the curves for one cylinder are just the curves of Fig. 4.1, and are included here for comparative purposes. Figure 5.1 shows quite a different result than seen for fixed-head drums of the last section. As the number of cylinders increases, the MTPT disciplines show more and

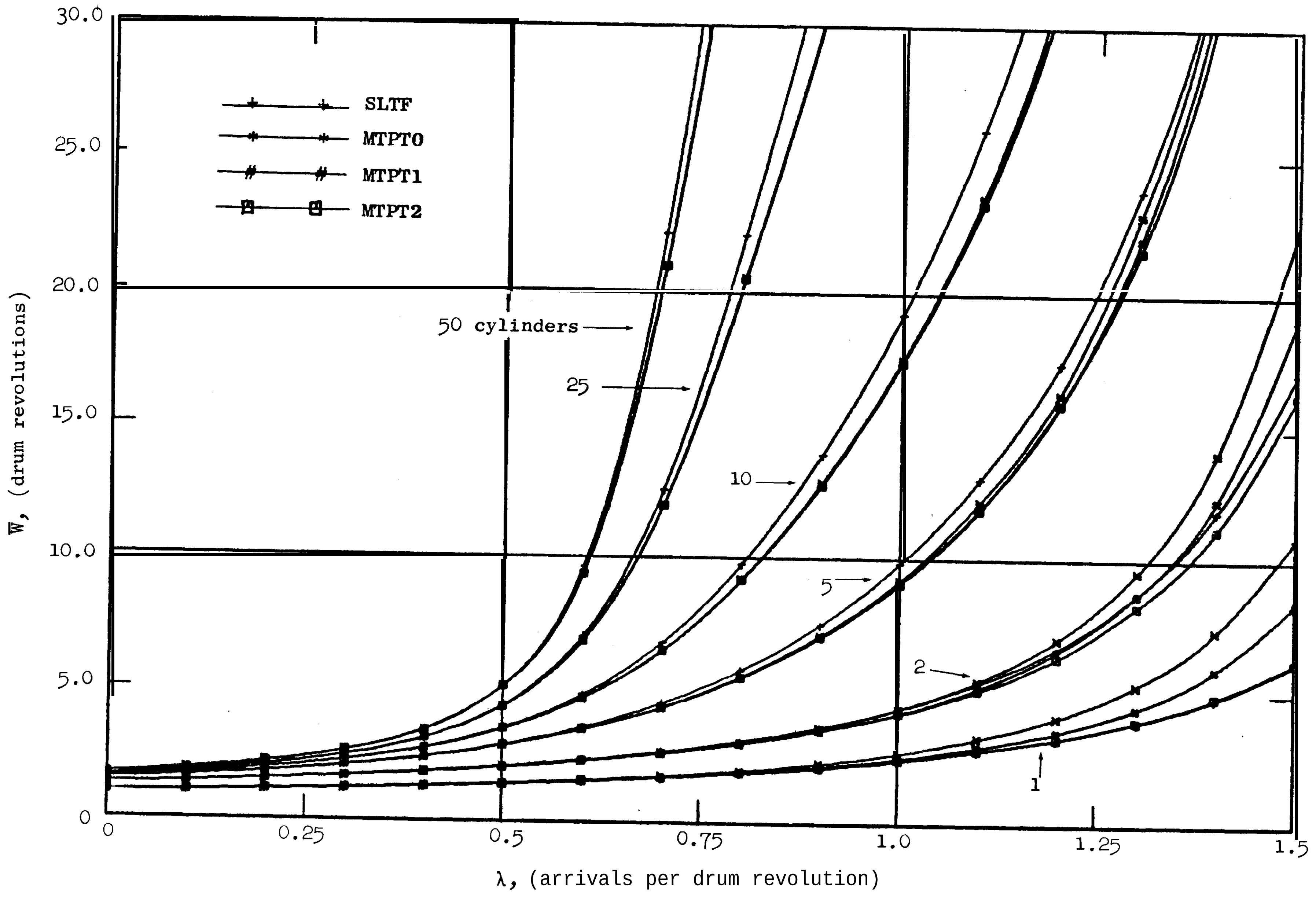


Figure 5.1. The expected waiting time of moving-head disks.

more of an advantage over the SLTF scheduling discipline and also the difference between the MTPT disciplines decreases as the number of cylinders increases.

The reasons for the results seen in Fig. 5.1 are straightforward. The waiting time on an I/O request is made up of three types of intervals: the time to process the I/O requests on other cylinders, the time to move between cylinders, and the time to service the I/O request once the read-write heads are positioned over the I/O request's own cylinder. By their definition, MTPT disciplines minimize the time to process the set of I/O requests on a cylinder and hence minimize one of the three types of intervals in an I/O request's waiting time. The chance a new I/O request will arrive at a cylinder while the MTPT schedule is being executed is minimized since a new request will only go to the current cylinder with a probability of 1/(number of cylinders). All three MTPT disciplines process the I/O requests on a cylinder in the same amount of time, barring the arrival of a new request, and so the difference in expected waiting times between the three implementations can be expected to go to zero as the number of cylinders increases.

Figure 5.2 shows the difference between each of the MTPT disciplines and the SLTF discipline; for clarity the residual curves for one cylinder are not included in Fig. 5.2. Figure 5.3 shows the residuals of Fig. 5.2 divided by the mean I/O waiting time for the SLTF discipline. In other words, Fig. 5.3 is the fractional improvement that can be expected by using the MTPT disciplines instead of the SLTF disciplines. Normalizing the residuals in this way shows a phenomenon not obvious from the first two figures; as the number of cylinders increases, the fractional improvement becomes relatively independent of the number of

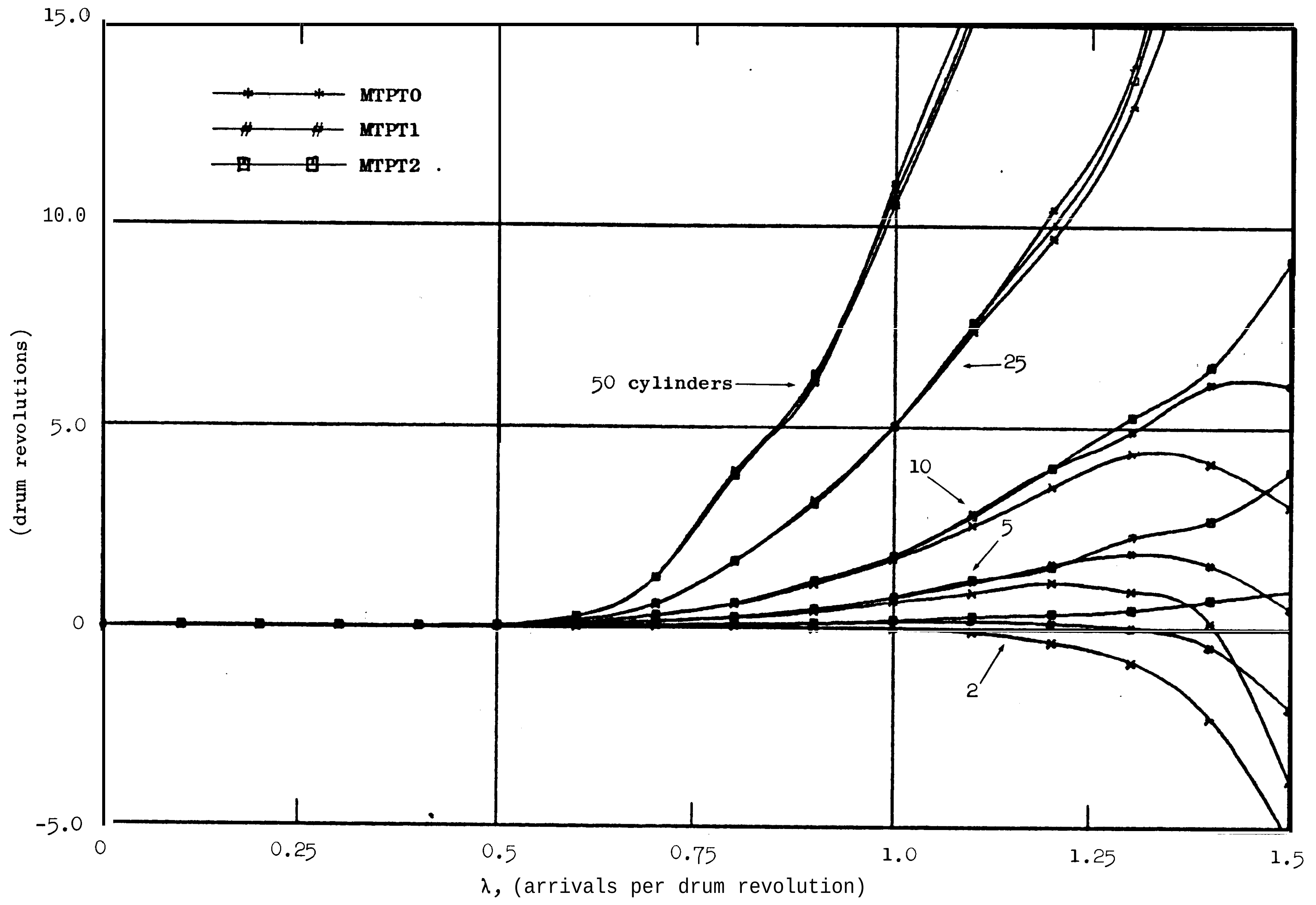


Figure 5.2. The difference between the expected waiting time of a moving-head disk when using the SLTF discipline and a MTPT discipline.

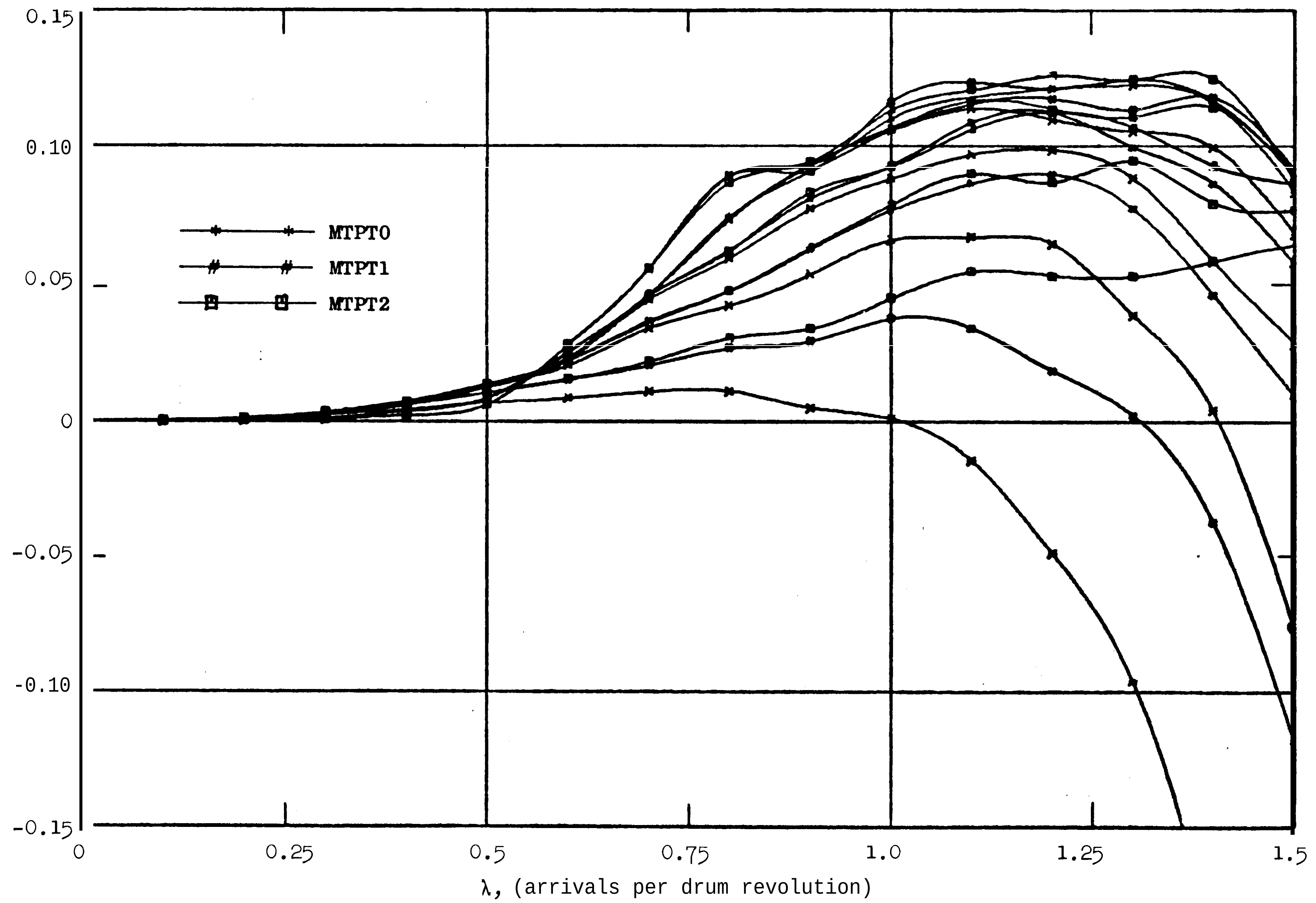


Figure 5.3. The difference between the expected waiting time of moving-head disks when using the SLTF discipline and a MTPT discipline, divided by the $\overline{W}$ for the SLTF discipline.

cylinders and is slightly more than 10 per cent for heavily loaded situations.

Figure 5.4 shows the standard deviation of the cases shown in Fig. 5.1. The remarks made concerning the mean I/O waiting time apply unchanged to the standard deviation of the I/O waiting times. The only additional observation that can be made is that the coefficient of variation, i.e. the standard deviation divided by the mean, is decreasing as the number of cylinders increases, and this is independent of the scheduling discipline used. This would be expected since the I/O waiting time is made up of intervals of processing time at other cylinders that are independent, random variables, and from the property that the mean and variance of a sum of independent random variables is the sum of the individual means and variances, respectively, we know the coefficient of variation of the waiting time should decrease as the square root of the number of cylinders.

6. Conclusions

The graphs of Secs. 4 and 5 are the real conclusions of the simulation study reported on in this article.

The purpose of this article is to empirically examine what application MTPT disciplines will have in situations with random arrivals. Section 3 shows that in situations where: (i) the coefficient of variation of record sizes is less than one, (ii) it is important to minimize the variance in waiting times, or (iii) it is important to minimize the mean duration of busy intervals; MTPT disciplines offer modest gains. It is important, however, to implement

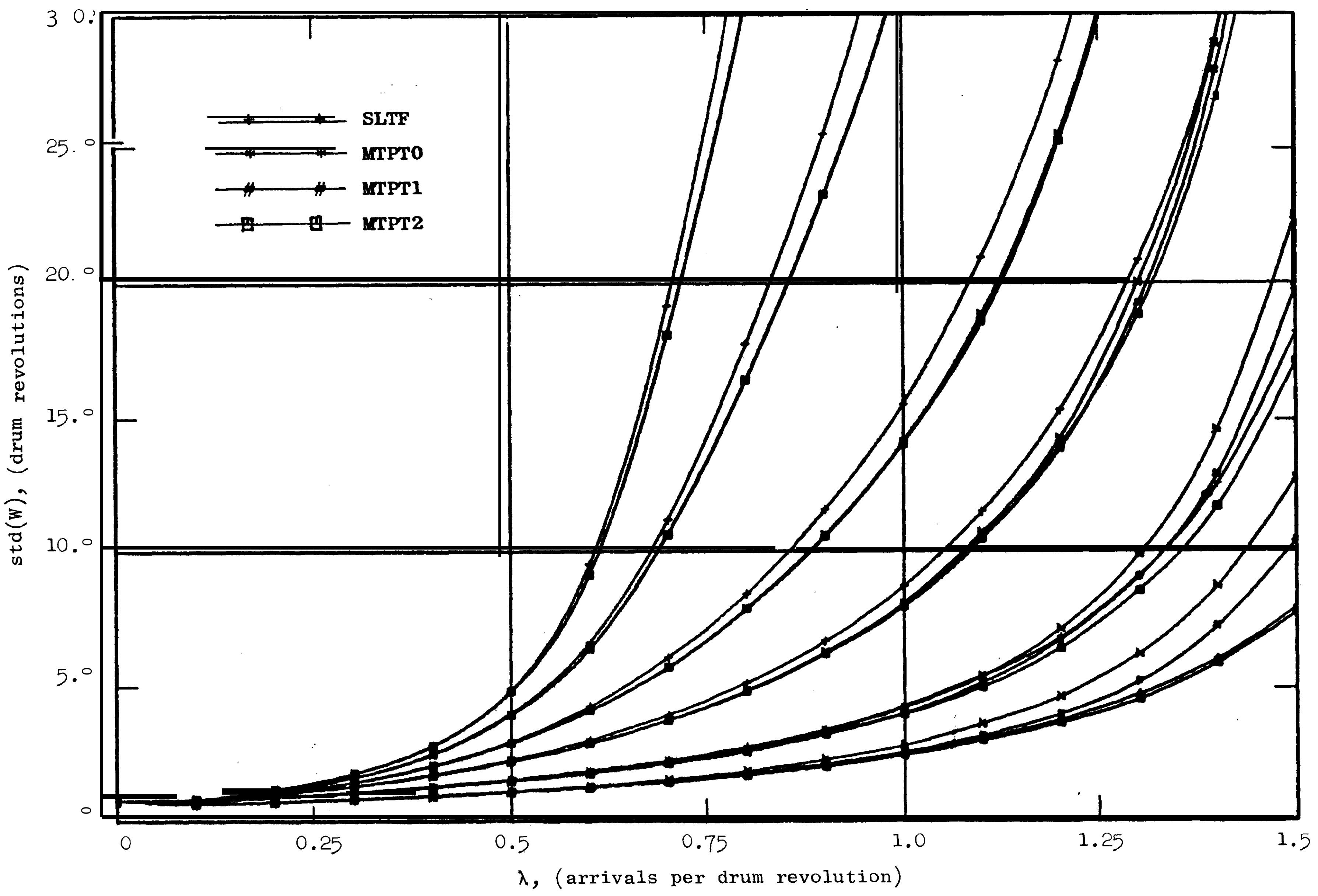


Figure 5.4. The standard deviation of the waiting time for moving-head disks.

as good a MTPT discipline as possible, and unfortunately only the MTPT0 algorithm has been shown to enjoy an efficient computation time, on the order of 100N + 50 microseconds for the naive implementation presented here. More work will be required in order to find efficient algorithms for the MTPT1 and MTPT2 scheduling disciplines.

Although the relative performance of the SLTF and MTPT scheduling disciplines have been considered here, little insight has been gained into what the optimal drum scheduling algorithm is when random arrivals are allowed, or even how close the disciplines studied in this article are to an optimal scheduling discipline. An intriguing topic of further research in this area will be to investigate optimal scheduling disciplines for random arrivals, and even if algorithms to implement the discipline are too complex to allow practical application, they will still provide an excellent measure of the sub-optimality of more practical scheduling disciplines.

The results of applying the MTPT discipline to a moving-head disk is encouraging. For heavy loads improvements of over 10 per cent are consistently achieved and just as importantly, it is relatively unimportant which of the MTPT disciplines is used. In other words, MTPT0, which has an efficient implementation, offers very nearly as much of an improvement over SLTF as does MTPT1 and MTPT2 when 5 or more cylinders are actively in use.

In the course of this study, the performance of MTPT2 was traced several times. It was observed that as the queue of outstanding requests grew, the probability that the MTPT2 discipline could use the shortest-latency record also grew. This observation leads to the reasonable, but as yet unproved, property of MTPT schedules that as the queue of I/O requests grows, with probability approaching unity there exists a MTPT sequence that begins with a SLTF sub-sequence. If this conjecture is true, then an obvious implementation feature of MTPT disciplines appears; when the depth of the I/O queue exceeds some threshold, suspend the MTPT algorithm in favor of a SLTF algorithm until the queue size drops below the threshold.

Acknowledgements

The author gratefully acknowledges many halpful suggestions from F. Baskett, E. J. McCluskey, and H. S. Stone.

Appendix
IMPLEMENTATION OF THE MTPT DRUM SCHEDULING ALGORITHM

This appendix lists the implementation, in FORTRAN IV, of the MTPT drum scheduling algorithm developed in Fuller [1971]. The results discussed in this report are based upon the subroutines listed here, and the formal parameters of the subroutines are compatible with the conventions of the simulator described in Fuller [1972]. Three versions of the MTPT scheduling algorithm are included here: MTPTO, an implementation of the original MTPT algorithm of Chapter 4; MTPT1, an obvious modification to MTPTO; and MTPT2, a shortest-latency-time-first version of MTPTO. Both MTPT1 and MTPT2 are described in detail in Chapter 6. Also included in this appendix is a restatement, in English, of the original MTPT drum scheduling algorithm.

1. A Statement of the Original MTPT Drum Scheduling Algorithm

Listed here is an informal, English statement of the original minimal-total-processing-time (MTPT) drum scheduling algorithm developed in Puller [1971].

The Minimal-Total-Processing-Time Scheduling Algorithm

1. Based on the unique value associated with each node, sort f_0, f_i, and s_i, $1 \le i \le N$, into one circular list. If $f_i = s_j$ for any i and j then f_i must precede s_j.
2. Set the list pointer to an arbitrary element in the list.
3. Scan in the direction of nodes with increasing value for the next (first) f_i in the list.
4. Place this f_i on a pushdown stack.
5. Move the pointer to the next element and if it is an f_i go to Step 4, else continue on to Step 6. In the latter case, the element must be an s_i.
6. Pop the top f_i from the stack and move the pointer to the next element in the circular list.
7. If the circular list has been completely scanned go to Step 8, else if the stack'is empty go to Step 3, else go to Step 5.
8. Let the bottom f_i on the pushdown stack be identified as f_{δ}. Change the circular list to an ordinary list where the bottom element is f_{δ}.
9. Match f_{δ} to s_0, and starting from the top of the list match the kth s_i to the kth f_i. (This constructs the permutation ψ'.)
10. Determine the membership of the cycles of ψ'.

11. Moving from the top to the bottom of the list, if adjacent arcs define a zero cost interchange and if they are in disjoint cycles perform the interchange. (This step transforms ψ' to ψ^0.)

12. Moving from the bottom to the top of the list, perform the positive cost, type 2a, interchange on the current arc if it is not in the same cycle as the arc containing f_δ. (The permutation defined at the end of this step is ψ_{mtpt}.)

2. Implementation of the Original MTPT Algorithm, MTPTO

```
      SUBROUTINE MTPTO(QSIZE,QUEUE,START,FINISH,HDPOS,SDOREC,RESCHD)
         INTEGER*2 QSIZE,QUEUE(1),SDOREC
         REAL*8 START(1),FINISH(1),HDPOS
         LOGICAL*1 RESCHD
C
C           This subroutine is an implementation of the drum
C        scheduling algorithm described in 'An Optimal Drum
C        Scheduling Algorithm', (Fuller, 1971).  This procedure
C        finds a schedule for the outstanding I/O requests
C        such that the total processing time is minimized.
C
C           The formal parameters have the following inter-
C        pretation:
C
C           QSIZE   ::= the number of requests to be scheduled.
C
C           QUEUE   ::= a vector of length QSIZE that contains
C                       the integer identifiers of the I/O requests
C                       to be scheduled.  The current implementation
C                       restricts the identifiers to be positive
C                       integers less than 1001.
C
C           START   ::= START(I) is the starting address of I/O
C                       request i.
C
C           FINISH  ::= FINISH(i) is the finishing address of
C                       I/O request i.
C
C           HDPOS   ::= the present position of the read-write heads.
C
C           SDOREC  ::= the identifier of the pseudo record.
C
C           RESCHD  ::= a boolean variable to signal when rescheduling
C                       is required.
C
C
      INTEGER I,J,N,COMPNS,QPLUS1,QMIN1,QMIN2,TEMP,K,L,M
      INTEGER FINDCY
      INTEGER*2 STACK,FPOINT,SPOINT,DELTA,DELTAS
      INTEGER*2 FNODES(1001),F(1001),SNODES(1001),S(1001)
      REAL*8 PERIOD/1.00/
      REAL*8 LATEND,LATSTR,SJ,FJ,ADJUST
C
      COMMON /OPTIM/ PERM,CYCLE,LEVEL
      INTEGER*2 PERM(1001),CYCLE(1001),LEVEL(1001)
C
      RESCHD =.FALSE.
      IF(QSIZE.LE.1) RETURN
C
C        Initialize data structures and constants
C
      QPLUS1 = QSIZE + 1
      QMIN1 = QSIZE - 1
      QMIN2 = QSIZE - 2
      DO 100 I=1,QSIZE
         FNODES(I) = QUEUE(I)
         SNODES(I) = QUEUE(I)
  100 CONTINUE
```

```
C         Enter current position of read-write heads.
             FNODES(QPLUS1)= SDOREC
             FINISH(SDOREC)= HDPOS
C
C         Sort list of F and S nodes.
C
      CALL SORT(FNODES,FINISH,QPLUS1)
      CALL SORT(SNODES,START,QSIZE)
C
C         Find F(DELTA).
C
      STACK = 0
      FPOINT = 1
      SPOINT = 1
      N = 2*QSIZE + 1
      DO 300 I=1,N
         IF(FINISH(FNODES(FPOINT)).LE.START(SNODES(SPOINT))) GO TO 310
            IF(STACK.GT.0) STACK = STACK - 1
            SPOINT = SPOINT + 1
            IF(SPOINT.LE.QSIZE) GO TO 300
               IF(STACK.GT.0) GO TO 335
                  DELTA = FPOINT
                  DELTAS = 1
                  GO TO 335
  310       IF(STACK.GT.0) GO TO 330
               DELTA = FPOINT
               DELTAS =MOD(SPOINT-1,QSIZE)+ 1
  330       STACK = STACK + 1
            FPOINT = FPOINT + 1
            IF(FPOINT.GT.QPLUS1) GO TO 335
  300 CONTINUE
C
C         redefine S and F nodes relative to F(DELTA).
C
  335 DO 340 I=1,QSIZE
         F(I) = FNODES(MOD(DELTA+I-2,QPLUS1)+1)
         S(I+1) = SNODES(MOD(DELTAS+I-2,QSIZE)+1)
  340 CONTINUE
      F(QPLUS1) = FNODES(MOD(DELTA+QPLUS1-2,QPLUS1)+1)
      DELTA = I
      ADJUST = PERIOD -FINISH(F(DELTA))
C
C         Construct the permutatlon Psi'.
C
      PERM(F(1))= SOOREC
      DO 400 I=2,QPLUS1
         PERM(F(I)) = S   W
  400 CONTINUE
C
C         Determine the membershfp of the cycles of Psi'.
C
      DO 500 I=1,QPLUS1
         CYCLE(F(I)) = F(I)
  500 CONTINUE
      COMPNS = 0
      DO 501 K=1,QPLUS1
         I = F(K)
         IF(CYCLE(I).NE.I) GO TO 501
            COMPNS = COMPNS + 1
            LEVEL(I) = 1
            J = I
  502       J = PERM(J)
            IF(J.EQ.I) GO TO 501
               LEVEL(I) = 2
               CYCLE(J) = I
               GO TO 502
  501 CONTINUE
      IF(COMPNS.EQ.I) GO TO 800
C
C        Transform Psi' to PsI(0).
C
     'DO 600 I=1,QMIN1
         J = QPLUS1 - I
         IF(DMOD(ADJUST+START(S(J)),PERIOD).LT.
     1      DMOD(ADJUST+FINISH(F(J+1)),PERIOD).OR.
     2      (FINDCY(F(J)).EQ.FINDCY(F(J+1)))) GO TO 600
               CALL MERGE(F(J),F(J+1))
               COMPNS = COMPNS - 1
               IFWOMPNS.EQ.I) GO TO 800
  600 CONTINUE
C
C        Transform PsI(0) to PhI(0).
C
      DO 700 I=2,QPLUS1
         IF(FINDCY(F(DELTA)).EQ.FINDCY(F(I))) GO TO 700
            CALL MERGE(F(DELTA),F(I))
            DELTA = I
            COMPNS = COMPNS - 1
            IF(COMPNS.EQ.I) GO TO 800
  700 CONTINUE
C
C        Construct schedule from Phi (0).
C
  800 J = SDOREC
      DO 810 I-1,QSIZE
         J = PERM(J)
         QUEUE(I) = J
  810 CONTINUE
      RETURN
      END




      INTEGER FUNCTION FINDCY(NODE)
         INTEGER*2 NODE
         COMMON /OPTIM/ PERM,CYCLE,LEVEL
         INTEGER*2 PERM(1001),CYCLE(1001),LEVEL(1001)
C
C           This is a function subroutine whose value is an
C        integer identifying the cycle of the permutation in
C        which NODE is a member.  CYCLE is a tree structure
C        defining the cycles of the permutation.
C
         FINDCY = NODE
   10    IF(FINDCY.EQ.CYCLE(FINDCY)) RETURN
            FINDCY = CYCLE(FINDCY)
            GO TO 10
      END
      SUBROUTINE MERGE(NODE1,NODE2)
         INTEGER*2 NODE1,NODE2
C
C           MERGE connects the tree representation of CYCLE1
C        and CYCLE2.  The integer vectors CYCLE and LEVEL
C        define the membership of the cycles of the permutation.
C           MERGE also executes the interchange of the successors
C        of NODE1 and NODE2.
C
         INTEGER*2 C1,C2,TEMP
         INTEGER FINDCY
         COMMON /OPTIM/ PERM,CYCLE,LEVEL
         INTEGER*2 PERM(1001),CYCLE(1001),LEVEL(1001)
         C1 = FINDCY(NODE1)
         C2 = FINDCY(NODE2)
C
C           Merge the two cycle structures.
C
         IF(LEVEL(C1).GE.LEVEL(C2)) GO TO 100
            CYCLE(C1) = C2
            GO TO 200
  100    IF(LEVEL(C1).EQ.LEVEL(C2)) LEVEL(C1) = LEVEL(C1) + 1
         CYCLE(C2) = C1
C
C           Perform the Interchange on the permutation.
C
  200    TEMP = PERM(NODE1)
         PERM(NODE1) = PERM(NODE2)
         PERM(NODE2) = TEMP
      RETURN
      END


      SUBROUTINE SORT(NODES,VALUE,N)
         INTEGER*2 NODES(1),N
         REAL*8 VALUE(1)
C
C              Shellsort.  For further discussion of Shellsort
C           see Shell(1959), Hibbard(1963), and Knuth(1971).
C
         INTEGER*4 I,J,D,Y
         REAL+8 VALUEY
         D = 1
  200    D = D + D
         IF(D - N) 200,208,207
  207       D = D/2
  208       D = D - 1
  201    IF(D.LE.0) RETURN
         I = 1
  202    J = I
         Y = NODES(I+D)
         VALUEY = VALUE(NODES(I+D))
  203    IF(VALUEY.LT.VALUE(NODES(J))) GO TO 204
  205       NODES(J+D) = Y
            I = I + 1
            IF((I+D).LE.N) GO TO 202
               D = (D-1)/2
               GO TO 201
  204       NODES(J+D) = NODES(J)
            J = J - D
            IF(J.GT.0) GO TO 203
               GO TO 205
      END
```

3. An Obvious Modification to MTPTO, MTPT1

```
      SUBROUTINE MTPT1(QSIZE,QUEUE,START,FINISH,HDPOS,SDOREC,RESCHD)
         INTEGER*2 QSIZE,QUEUE(1),SDOREC
         REAL*8 START(1),FINISH(1),HDPOS
         LOGICAL*1 RESCHD
C
         INTEGER I,J,N,COMPNS,QPLUS1,QMIN1,QMIN2,TEMP,K,L,M
         INTEGER FINDCY
         INTEGER*2 STACK,FPOINT,SPOINT,DELTA,DELTAS
         INTEGER*2 FNODES(1001),F(1001),SNODES(1001),S(1001)
         REAL*8 PERIOD/1.00/
         REAL*8 LATEND,LATSTR,SJ,FJ,ADJUST
C
         CALL MTPT0(QSIZE,QUEUE,START,FINISH,HDPOS,SDOREC,RESCHD)
         QMIN2 = QSIZE - 2
         IF(QMIN2.LE.2) RETURN
         LATSTR = PERIOD - DMOD(HDPOS,PERIOD)
         DO 900 I=1,QMIN2
            J = I+ 1
            LATEND = DMOD(LATSTR+START(QUEUE(I)),PERIOD)
            DO 920 J=J,QMIN1
               SJ = DMOD(LATSTR+START(QUEUE(J)),PERIOD)
               IF(SJ.GT.LATEND) GO TO 920
               FJ = DMOD(LATSTR+FINISH(QUEUE(J)),PERIOD)
               IF((FJ.LT.SJ).OR.(FJ.GT.LATEND)) GO TO 920
                  TEMP = QUEUE(J)
                  K = J - I
                  DO 930 L=1,K
                     M = J-L
                     QUEUE(M+1) = QUEUE(M)
  930             CONTINUE
                  QUEUE(I ) = TEMP
               LATEND= DMOD(LATSTR+START(TEMP), PERIOD)
  920       CONTINUE
            LATSTR = PERIOD - DMOD(FINISH(QUEUE(I)),PERIOD)
  900    CONTINUE
         RETURN
      END
```

4. The Shortest-Latency-Time-First MTPT Algorithm, MTPT2

```
      SUBROUTINE MTPT2(QS,Q,START,FINISH,HD,SDOREC,RESCHD)
         INTEGER*2 QS,Q(1),SDOREC
         REAL*8 START(1),FINISH(1),HD
         LOGICAL*1 RESCHD
C
         INTEGER I,J,K,REND,RBEGIN
         INTEGER*2 QSM1
C
         IF(QS.LE.1) RETURN
         CALL MTPT0(QS,Q,START,FINISH,HD,SDOREC,RESCHD)
         IF(QS.LE.2) RETURN
         RBEGIN = Q(1)
         REND = Q(QS)
         QSM1 = QS - 1
C
         DO 100 I=1,QS
            CALL SLTF(QS,Q,START,FINISH,HD,SDOREC,RESCHD,I)
            IF(Q(1).EQ.RBEGIN) RETURN
            DO 200 J=2,QS
  200          QT(J-1) = Q(J)
            CALL MTPT0(QSM1,QT,FINISH(Q(1)))
            RESCHD = .TRUE.
            IF(QT(QSM1).EQ.REND) RETURN
  100    CONTINUE
         WRITE(6,101) QS
  101    FORMAT(10X,'ERROR IN MTPT2; QS = ',I4,';')
         CALL MTPT0(QS,Q,START,FINISH,HD,SDOREC,RESCHD)
         RETURN
      END
```

REFERENCES

Abate, J. and Dubner, H. (1969) Optimizing the performance of a drum-like storage. IEEE Trans. on Computers C-18, 11 (Nov., 1969), 992-996.

Abate, J., Dubner, H., and Weinberg, S. B. (1968) Queueing analysis of the IBM 2314 disk storage facility. J. ACM 15, 4 (Oct., 1968), 577-589.

Adel'son-Vel'skiy, G. M. and Landis, E. M. (1962) An algorithm for the organization of information. Doklady, Akademiia Nauk SSR, TOM 146, 236-266. Also available in translation as: Soviet Mathematics 3, 4 (July, 1962), 1259-1263.

Bottomly, J. S. (1970) SUPERMON statistics. SFSCC Memo. (Aug. 13, 1970), Stanford Linear Accelerator Center, Stanford, Calif.

Coffman, E. G. (1969) Analysis of a drum input/output queue under scheduling operation in a paged computer system. J. ACM 16, 1 (Jan., 1969), 73-90.

Denning, P. J. (1967) Effects of scheduling on file memory operations. Proc. AFIPS SJCC 30 (1967), 9-21.
Keywords: drum scheduling, queueing models.

Frank, H. (1969) Analysis and aptisimation of disk storage devices for time sharing systems. J. ACM 16, 4 (Oct. 1969), 602-620.

Fuller, S. H. (1971) An optimal drum scheduling algorithm. Technical Report No. 12, Digital Systems Laboratory, Stanford University, Stanford, Calif. (April, 1971).

Fuller, S. H. (1972A) The expected difference between the SLTF and MTPT drum scheduling disciplines. Technical Report No. 28, Digital Systems Laboratory, Stanford University, Stanford, Calif. (Aug., 1972).

Fuller, S. H. (1972B) A simulator of computer systems with storage units having rotational delays. Technical Note 17, Digital Systems Laboratory, Stanford University, Stanford, Calif. (Aug., 1972).

Fuller, S. H. (1972C) Performance of an I/O Channel with multiple paging drums. Technical Report No. 27, Digital Systems Laboratory, Stanford University, Stanford, Calif. (Aug., 1972).

Fuller, S. H. and Baskett, F. (1972) An analysis of drum storage units. Technical Report No. 26, Digital Systems Laboratory, Stanford University, Stanford, Calif. (Aug., 1972).

Hibbard, T. N. (1963) An empirical study of minimal storage sorting. C. ACM 6, 5 (May, 1963), 206-213.

Hoare, C.A.R. (1961) Algorithm 64, quicksort. C. ACM 4, 7 (July, 1961), 321.

IBM. (1971) IBM system/360 and system/370 FORTRAN IV language. File No. S360-25, Order No. GC28-6515-8.

IBM. (1965) IBM system/360 component descriptions -- 2314 direct access storage facility and 2844 auxiliary storage control. File No. S360-07, Form A26-3599-2.

IBM. (1971B) IBM 3830 storage control and 3330 disk storage. Order No. GA26-1592-1.

Johnson, R. and Johnston, T. (1971) PROGLOOK user's guide SCC-007, Stanford Computation Center, Stanford University, Stanford, Calif. (Oct., 1971).

Seaman, P.H., Lind, R.A., and Wilson, T.L. (1969) An analysis of auxiliary-storage activity. IBM Systems Journal 5, 3 (1969) 158-170.

Shell, D. L. (1959) A high-speed sorting procedure. C. ACM 2, 7 (July, 1959), 30-32.

Skinner, C. E. (1967) Priority queueing systems with server-walking time. Operations Research 15, 2 (1967), 278-285.

Stone, H. S. and Fuller, S. H. (1971) On the near-optimality of the shortest-access-time-first drum scheduling discipline. Technical Note 12, Digital Systems Laboratory, Stanford University, Stanford, Calif., (Oct., 1971).

Teorey, T. J. and Pinkerton, T. B. (1972) A comparative analysis of disk scheduling policies. C. ACM 15, 3 (March, 1972), 177-184.

van Emden, M. H. (1970) Algorithm 402, increasing the efficiency of quicksort. C. ACM 13, 11 (Nov., 1970), 693-694