SELF-ORGANIZING SEQUENTIAL SEARCH PROCEDURES

by

NANCY KAY SUNDHEIM

B.S., Kansas State University, 1980
B.S., Kansas State University, 1982

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Statistics

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

James J. Higgins

Major Professor

TABLE OF CONTENTS

		Page
Chapter 1.	Introduction and Literature Review	1
	Introduction	1
	Permutation Rules	4
	Measures of Comparison	6
	Counter Rules	11
	Change of Assumptions	22
Chapter 2.	The Mathematical Model and Asymptotic Results .	25
	Notation	25
	Mathematical Model	26
	Theorem 1	28
	Theorem 2	38
Chapter 3.	Small Sample Case	40
	Measures of Closeness	40
	The Simulation	42
References		47
Appendix A.	A Listing of the Simulation Program	49
Appendix B.	Tests of the Random Number Generator	55
Appendix C.	Graphs of the Average Cost and the 5-point Moving Average for Cost Versus Request Number .	59

CHAPTER 1. INTRODUCTION AND LITERATURE REVIEW

A file is a set of records arranged sequentially. Suppose a particular record is requested. If each record in a file, starting with the first one, is searched until the requested record is reached, the file is called a sequential search file. There are many applications in computer software which deal with this type of file. For example, SAS consists of many different procedures such as ANOVA, GLM, FUNCAT, PRINT, etc. When a program calling ANOVA is submitted, the computer must first find this procedure. It does so by searching each procedure, starting with the first, until it finds ANOVA. Another example is the PASCAL language. The only type of file available in PASCAL is the sequential file. Some people, who view this as a shortcoming of the language, have devised nonstandard extensions to the compiler which add direct access files. Since sequential search files occur in so many contexts, researchers in computer science have shown strong interest in them. This chapter first describes the problem in more detail, including assumptions, and then reviews the literature.

For convenience it shall be assumed that it takes one unit of time to search one record. Thus, it takes i units of time to locate the record in the ith position. The probability that a record R_i is requested from a file containing N records shall be denoted p_i. It will be assumed that the requests are independent and that the probabilities p_i, $i=1,2,...,N$ do not change over time. It can be assumed without loss of generality that $p_1 \geqslant p_2 \geqslant ... \geqslant p_N > 0$ and $p_1 + p_2 + ... + p_N = 1$.

The expected search time of an arrangement is defined as $\sum_{k=1}^{N} P_{i(k)}k$, where $P_{i(k)}$ is the probability that the ith record, which is in the kth position, will be requested. Different arrangements of the records will have different expected search times. The cost of a certain arrangement is defined as the expected search time of that arrangement. It measures the average cost of searching, assuming that the cost of searching a record is proportional to the search time. Ideally, the file will be arranged so that the most requested record is first; next most requested is second, etc. In this way, the cost of searching the file will be minimum. However, the probabilities p_i are rarely, if ever, known beforehand. Fifteen methods are described below that do not make use of a priori knowledge of the values of p_i, i = 1,2,...,N.

The first method, one which easily comes to mind, is called the frequency counter scheme. Just keep track of the number of times each record is requested. Then, after many requests have been made, arrange the file so that the record most requested is in front, the second most requested record is next and so on. By the Law of Large Numbers, this method will eventually give the optimal ordering. However, there are two major problems with this scheme. The first is that it may take too much time to accumulate the data. The second problem is that it may require too much storage when implemented on the computer. Since computers store numbers in binary code, even a relatively low number requires many bits of storage. For example, 100 is represented in a computer by 1100100. So the 3-digit number 100 requires 8 bits of storage, not 3 (7 to identify the number and 1 for the sign.) If very many requests are to be made of a system then even a large amount of storage is quickly over-flowed.

To deal with the problems above, several self-organizing, sequential search schemes have been developed. A self-organizing scheme reorders the file after each request according to a specified algorithm, or rule. The sequential search schemes can be grouped into two categories: Permutation Rules and Counter Rules. Counter Rules perform better than Permutation Rules. The appeal of the Permutation Rules is that they require the least storage capacity.

Permutation Rules have been studied by Bentley and McGeoch [3], Bitner [4], Burville and Kingman [5], Hendricks [8],[9],[10], Knuth [11], Letac [15], McCabe [16], Nelson [17], Rivest [18], Savchenko [19], Takanami and Fujii [20] and Tenenbaum [21]. The two most popular Permutation Rules are Move to Front (MTF) and Transposition (TR). Both of these rules begin with a random ordering of the file. In the MTF rule, when a record is requested it is searched for and then moved to the front of the file. All records previously in front of this record are moved one position back. (If the requested record is in the first position then nothing is done.) For example, suppose the current arrangement of a file of 5 records is $\{R_1,R_2,R_3,R_4,R_5\}$. If R_4 is requested then the arrangement becomes $\{R_4,R_1,R_2,R_3,R_5\}$.

Intuitively this should improve the cost of the file because records that are requested most often tend to stay up front while those requested less often tend to drift towards the back.

The TR rule simply transposes the requested record with the record immediately in front of it. All other records are not moved. (Again, nothing need be done if the requested record is in the first position.) For example, suppose a file with a current arrangement of $\{R_1,R_2,R_3,R_4,R_5\}$. If R_4 is requested the arrangement becomes $\{R_1,R_2,R_4,R_3,R_5\}$. Thus, the most requested records drift toward the front of the file. Rivest [18] noted that the TR scheme was a better approximation to the frequency counter rule than any other permutation rule. A record is moved up more than one position in the frequency counter rule only when two or more of the preceding records have counters that are equal. This is very unlikely to occur often if all the p_i's are distinct. When this does not happen then the frequency counter rule either does a simple transposition or nothing. So one should expect the TR rule to perform, at least in the long run, similar to the counter rule.

There is also an intermediate scheme called the Move Up k rule. This rule moves a requested record up k places. (If $k>i$ then the record is simply moved to the front of the file.) When $k=1$ then this is just the TR rule. If $k=N$ then this becomes the MTF rule.

There are two important characteristics of these rules which are used to compare them. The first measure is the asymptotic cost. McCabe [16] showed that (for the MTF and TR rules) the probabilities of occurrence of the N! different arrangements will become stable after the scheme has been invoked for a long time. (Details can be found in the next section of this paper.) These steady state probabilities will be denoted by π_i, $i=1,2,...,N!$ and the costs of the arrangments by c_i, $i=1,2,...,N!$. Thus, the expected cost at steady state is $c_1\pi_1 + c_2\pi_2 + ... + c_{N!}\pi_{N!}$. This cost is called the asymptotic cost of the scheme. It is generally compared to the cost of the optimal ordering of the file (called the optimal cost.) The optimal cost is defined as $\sum_{i=1}^{N} ip_i$.

The second characteristic is how quickly the scheme attains steady state. This is called its rate of convergence.

First, consider the asymptotic costs of the MTF and the TR rules. Rivest [18] has shown that the asymptotic costs of the MTF and the TR rules are both lower than the expected cost of a random ordering of the file. Rivest [18] has also shown that the asymptotic cost of the TR rule is lower than that of the MTF for any probability distribution. Furthermore, he showed that the TR scheme has lower asymptotic cost than the intermediate scheme for any k. However, Bitner [4] demonstrated, by considering two different probability distributions, that it is possible for the MTF to converge to its asymptotic cost much quicker than the TR. These two distributions are as follows:

(1) $p_1=1$ and $p_i=0$, $2 \leqslant i \leqslant N$

(2) $p_1=0$ and $p_i=1/(N-1)$, $2 \leqslant i \leqslant N$

Bitner [4] defined a measure of convergence called the overwork. It is denoted OV and is defined to be the area between the cost curve and its asymptote (see Figure 1.) Note that the "steeper" the cost curve is, the smaller the overwork will be. Bitner [4] derived the general form of the overwork for the MTF scheme, but not for the TR scheme. However, he showed that OV_{MTF} is lower than OV_{TR} for the two distributions above. He also derived OV_{MTF} for the following distribution:

$$p_i = 1/(iH_N), \quad 1 \leqslant i \leqslant N, \quad H_N = \sum_{i=1}^{N} (1/i) \quad \text{(Zipf's Law)}$$

A simulation run using the formula derived for OV_{MTF} and an approximation of OV_{TR} showed, again, that OV_{MTF} is much smaller than OV_{TR}. From this Bitner [4] concluded that the MTF rule converges much more quickly than the TR rule.

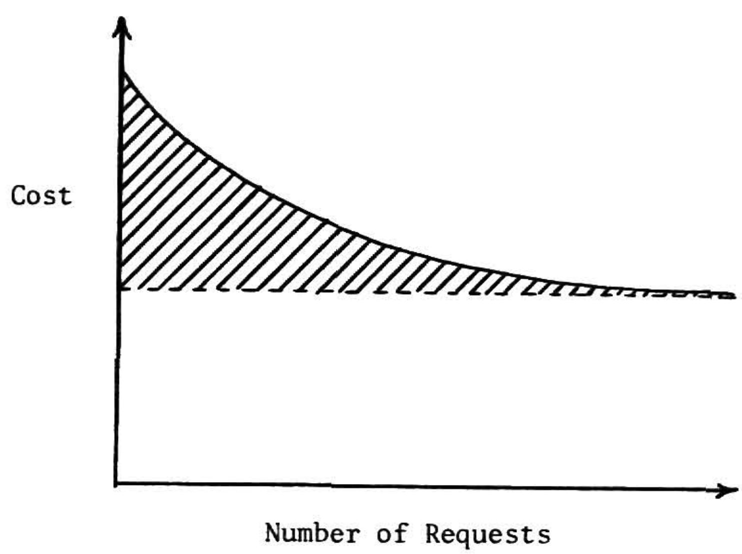


FIG. 1. The overwork is the area between the cost curve and its asymptote (shaded above.)

Although the above work was not complete, Bitner's conclusion is generally accepted as fact. Intuitively it seems reasonable. In the initial random ordering, many high probability records may be far down in the file. When one of these records is requested it is immediately moved to the front. The TR scheme, by comparison, will move these records only one position at a time. It will take much longer for these high probability records to make it to the front. So the MTF is a better scheme if there are few requests that will be made.

Since both of these are desirable properties it seems natural to consider a hybrid method. Use the MTF rule for the first so many requests and the TR rule from then on. This will combine the properties of both rules. The difficulty of this hybrid method is deciding just when the switch should be made. It has not been considered very seriously because of this difficulty.

All the methods and results described above assume an initial random ordering of the files. Rivest [18] suggested that the methods would converge quicker if the initial ordering were obtained by the following procedure: start with an empty file and add to the end of the file those records which are searched for (requested), but not found. This is equivalent to what Bitner [4] calls the First Request

(FR) rule. In this rule, the first time a record is requested it is moved up the file until it comes to the top or to a previously requested record. After that it is not moved. For example, suppose the initial ordering of a file is $\{R_4, R_2, R_3, R_5, R_1\}$. Suppose the first request is for R_2. The file becomes $\{R_2, R_4, R_3, R_5, R_1\}$. Suppose the second request is for R_1. The file is now $\{R_2, R_1, R_4, R_3, R_5\}$. Now suppose that the third request is also for R_1. This record has previously been requested so it is not moved. Thus, the file remains $\{R_2, R_1, R_4, R_3, R_5\}$. After each record has been requested once, then subsequent requests will not change the file.

Rivest [18] investigated the use of this procedure to initialize the file before using the MTF or TR rule. He showed that after the procedure is finished (after all the records have been requested at least once), the system is at steady state relative to the MTF rule, so further use of the MTF rule will not improve the expected cost (see chapter 2, theorem 2.) However, the TR rule will improve the expected cost from this point because it is not yet at steady state. Rivest [18] claims that using the TR rule after the file has been initialized with this procedure is the most efficient method. It converges quickly but still has the low asymptotic cost of the TR scheme.

Bitner [4] proposed a slight modification. Use the initializing procedure (FR rule) the first time a record is requested. Then use the TR rule on all subsequent requests (rather than waiting until all records have been requested at least once.) Since this starts improving the cost before the initializing procedure is finished, Bitner [4] claims this method is most efficient.

It should be noted, though, that Rivest [18] and Bitner's [4] hybrid methods require a slightly more complex algorithm. These methods should be used only when the overhead of the more complex algorithm is offset by the more desirable characteristics.

Counter rules have been studied by Bitner [4], Gonnet, Munro and Suwanda [7] and Lam, Siu and Yu [14]. Baer [2] was the first to look at the idea of counting requests for the closely related problem of binary searches. (Only sequential searches are considered in this paper. Allen and Munro [1] have some discussion of the similarities and differences between binary and sequential searches.) Bitner [4] then adapted the idea to sequential searches.

The Permutation rules presented above attempt to reorder the file without counting the number of requests for each record. (For this reason they are sometimes referred to as memoryless or memory-free rules.) The counter rules, as the name implies, keep track of (count) the number of times each record is requested. To do this requires "extra" storage. This extra information is stored in what are called fields. A field is simply a storage location.

The frequency counter rule discussed above is, of course, the most basic counter rule. Each record has an associated field which contains the number of times that record has been requested. However, as the number of requests for a record increases, the storage space, or field, is overflowed. The problems with this rule motivated the development of all other methods discussed in this paper.

Bitner [4] suggested two modifications to the Frequency Counter Rule which attempt to overcome the problem of storage. The first modification reduces the counts by dividing each field by a constant (generally 2) or subtracting a constant when one of the fields becomes full. For example, suppose the size of the field is 6 bits. This would be 1 bit for the sign and 5 bits to identify the number.

This would be 1 bit for the sign and 5 bits to identify the number. The largest binary number that could be stored in this field would be +11,111. This converts to 31 in decimal form. When the first record reaches 31 requests then the number of requests in every field is divided by 2. If the fields are small then it does not take very long to fill the fields. In this case, the reduction must be done fairly frequently and the cost becomes prohibitive.

The second modification is referred to as the Difference Rule. This scheme stores with the ith record (in the ith field) the difference between the counts of the (i-1)th record and the ith record. For example, suppose the current number of requests for the (i-2)th to the (i+1)th records are those shown below in Table 1a. The numbers contained in their respective fields are also shown in Table 1a. These are the differences in the number of requests. For instance, 21-13=8 is kept in the (i-1)th field.

Table 1a. Counts and differences for certain records.
(Before the 9th request for the ith record.)

record	i-2	i-1	i	i+1
# of requests	21	13	8	7
# stored in field	-	8	5	1

Now suppose the ith record is requested. This changes the number of requests and the differences as shown in Table 1b, below.

Table 1b. Counts and differences for certain records. (After the 9th request for the ith record.)

record	i-2	i-1	i	i+1
# of requests	21	13	9	7
# stored in field	-	8	4	2

The cost is relatively cheap because only two fields are altered with each request. The disadvantage is that even though the rate of growth of the fields is smaller than with the Frequency Counter Rule, it still requires an unbounded amount of storage.

The Limited Difference Rule is nearly the same as this second modification. It simply imposes an upper bound. Subsequent requests will not increase a field after this upper bound has been reached. The Limited Difference Rule is not optimal. That is, the asymptotic cost of the rule is not the optimum cost. However, Bitner [4] showed that the asymptotic cost of this rule does approach the optimal cost as the upper bound on the difference is increased. Bitner [4] also showed that the rule converges quite rapidly. Until this maximum difference is reached, the rule behaves exactly like the second modification of the Frequency Counter Rule. Thus, for this initial period, the Limited Difference Rule converges quite quickly to its asymptotic cost. This is, therefore, a nearly optimal rule.

There is a class of schemes called the Wait C, Move and Clear rules. All fields are initially set to zero. When a record is requested, its field is incremented. When a record has been requested c+1 times then that record is moved according to some permutation rule. Then all the fields are reset to zero and the procedure is repeated. The cost of resetting all the fields to zero can be quite significant. However, if all the counters are in one area, rather than with the records they are associated with, then this cost becomes very reasonable. This scheme has only been analyzed with the MTF and the TR permutation rules. Bitner [4] showed that the asymptotic cost of the Wait C, MTF and Clear scheme is less than that of the MTF. He also showed that for any c and for any probability distribution, the asymptotic cost of Wait C, TR and Clear is less than the asymptotic cost of Wait C, MTF and Clear. As with the Limited Difference Rule, the asymptotic cost of Wait C, Move and Clear is not optimum (for any permutation rule). However, Bitner [4] demonstrated that as c approaches infinity this asymptotic cost approaches optimum. The drawback of this scheme is its convergence. The best case occurs when the same record is requested c+1 times in a row and a move will be made every c+1 requests. So the convergence is decreased by at least a factor of c+1. Thus, the Wait C, Move and Clear schemes are outperformed by the Limited Difference Rule.

A similar class of schemes is the Wait C and Move Rules. All fields are initially set to zero. When a record is requested, its field is incremented. When a field reaches c + 1, then that record is moved according to a permutation rule. Rather than reset all the fields to zero, this class of schemes only resets the field of that record to zero and then repeats. Bitner [4] showed that the Wait C and MTF has a lower asymptotic cost than the MTF. However, he also showed that not only is the asymptotic cost of Wait C and MTF not the optimum cost, but the asymptotic cost does not even approach optimum as c approaches infinity. Bitner [4] ran a simulation of the Wait C and TR rule but the rule has not as yet been completely analyzed. The Wait C and Move rules make a move, on the average, every c+1 requests. This means they converge faster than the Wait C, move and Clear rules but slower than the Limited Difference rule. Once again, then, the Limited Difference rule is superior in asymptotic cost and convergence.

Lam, Siu, and Yu [14] suggest another approach. First use the TR rule. After reaching steady state, further reordering of the file will not improve the performance of the TR rule. Optimum cost will never be achieved by the TR rule. So Lam, Siu and Yu [14] suggest switching at this point to a modification of the frequency counter rule. In other words, begin counting the number of requests for each record once steady state has been achieved. The frequency counter rule reorders the records in decreasing order of the f_i's (f_i is the number of requests for record i.) This ignores the fact that the file is at steady state when counting begins. Lam, Siu, and Yu [14] therefore propose to reorder the records in decreasing order of (f_i+N-i). This treats the record at position i as if it has received an additional count of (N-i). They show that this modification, when started at steady state, is optimal for any finite number of requests. This means that even a small amount of "extra" storage can give the optimal cost. They claim this is better than any other counter method, and they show that it is better in terms of asymptotic cost. However, the obvious drawback is its rate of convergence. The scheme depends on the file first reaching steady state under the TR rule. The TR rule is known to have relatively slow convergence. The other problem with this scheme is determining just when steady state has been achieved.

Gonnet, Munro and Suwanda [7] proposed a class of schemes called the Simple k-in-a-row schemes (Simple k). As the name implies, a record is not moved until it is requested k times in a row. It is then moved according to the MTF, TR or some other permutation rule. This requires some extra storage; more than for the permutation rules, but less than for the counter rules described above. Gonnet, Munro and Suwanda [7] showed that for fixed k this scheme requires only (logN+logk) extra bits of storage. For example, suppose N=50 and k=4. A record will be moved only if it is requested 4 times in a row. The scheme requires log50+log4 = 2.301 or 3 extra bits of storage. The counter rules require much more. Each of the 50 records have an associated field to store the number of requests. Suppose the size of a field is 6 bits (a very small field); then the amount of extra storage is 50x6 = 300 bits. This is very large compared to 3 bits for the k-in-a-row scheme.

Gonnet, Munro and Suwanda [7] looked at the Simple k, MTF rule and the Simple k, TR rule. They showed that the Simple k, MTF has lower asymptotic cost than the MTF and the Simple k, TR has lower asymptotic cost than the TR. In both cases they showed that the asymptotic cost is not optimal. The asymptotic costs of both schemes do, however, decrease as k increases. They also showed that the Simple k, TR has lower asymptotic cost than Simple k, MTF.

Gonnet, Munro and Suwanda [7] considered a modification to the Simple k-in-a-row schemes called the Batched k-in-a-row (Batched k). This views the requests as being batched into groups of k consecutive requests. A reordering of the file occurs only if all k requests in a batch are for the same record. At first this appears to be equivalent to the Simple k. However, it is not. Consider the following example. Let k=3 and N=20. Suppose the following requests were made of a file:

18, 7, 1, 1, 1, 3, 2, 7, 7, 7, 2, 1, 4, 4, 4, 4, 3, 12, 8, 9, 19

In the Simple k, when a record is requested 3 times in a row it is moved. So, in the above example, First record 1 is moved, then record 7 and then record 4. In the Batched k the requests are viewed as batches of 3 requests. The requests above would be grouped as follows:

batch 1 : [18,7,1]	batch 5 : [4,4,4]
batch 2 : [1,1,3]	batch 6 : [4,3,12]
batch 3 : [2,7,7]	batch 7 : [8,9,19]
batch 4 : [7,2,1]	

Only batch 5 has all the requests for the same record. So record 4 is the first (and only) record to be moved.

Gonnet, Munro and Suwanda [7] showed that the asymptotic cost of the Batched k, MTF and the Batched k, TR are not optimal. However, they decrease as k increases. The Batched k, MTF has not been compared to the Batched k, TR. Gonnet, Munro and Suwanda [7] state that intuitively the Batched approach should perform better than the Simple k. Roughly, the effect of the Batched k is to raise the probability of request of each record to the power k. This makes (after normalization) the large probabilities larger and the small probabilities even smaller. They show that for the MTF rule the Batched approach does give lower asymptotic cost than the Simple approach.

The two k-in-a-row schemes are better than the permutation rules alone because they give lower asymptotic costs. They are better than the counter rules above in the sense that they require much less storage. However, the asymptotic costs of the k-in-a-row schemes have not been compared to the asymptotic costs of the counter rules above. Furthermore, the rates of convergence of the k-in-a-row schemes have not been considered at all. Thus, it is difficult to effectively compare these rules with the ones described above.

Many researchers feel that if extra storage is going to be used then a non-sequential search procedure should be employed. Therefore it appears that none of the counter rules are very practical. The frequency counter rule should never be used unless the counts are already needed for other purposes. Thus, researchers mostly consider this a dead end and are directing their efforts towards the permutation rules.

Many feel that non-sequential searching is also generally more efficient than the permutation rules, but that there are some instances in which the permutation rules are useful. These instances are when a low overhead is desired. Although the TR has a lower asymptotic cost, the MTF has other advantages such as quick convergence. Also, the additional overhead necessary for the hybrid method does not usually justify its use over the use of the MTF.

It should be noted that there has been some work on the problem when the assumptions have been changed. Rivest [18] indicates that if there is correlation between successive requests, then it can be shown that the MTF scheme will perform more efficiently. Konneker and Varol [12] expanding on this idea, presented some modified search schemes and concluded that the improvement in the search time may not be worth the extra work involved. Nelson [17] proposed a scheme in which the searching is started where the record is thought to be. He derived a prediction interval for the position of the requested record.

Bentley and McGeoch [3] studied the frequency counter rule, MTF and TR schemes from a different viewpoint - that of their worst-case performance rather than their expected performance. They concluded that even though the previous probabilistic analyses showed that the TR rule is superior to the MTF rule, their worst case analysis showed the opposite to be true. They also felt that the MTF would be better than the frequency counter rule.

Bentley and McGeoch [3] ran several simulations on both Pascal files and text files. Their results indicated that the MTF was always better than the TR and usually better than the frequency request rule. Their explanation for this in the Pascal files is the presence of what they call the high locality. In other words, infrequently used words such as INTEGER, VAR and END appear in groups rather than being uniformly distributed throughout the file. This also occurred in the text files, although not to as great a degree as in the Pascal files. The idea of locality is the same as assuming there is a dependency among the requests.

The small amount of work done in this area seems to indicate that whenever any dependency exists between the requests then the MTF is a much better scheme. Since it seems to be very realistic for these correlations to exist, much more work is needed in this area. The idea of a worst case analysis could also be greatly expanded. A question overlooked by researchers is the small sample behavior of sequential search schemes. That is, there have been no studies of the case in which records have been requested a small number of times. Investigations of this question would shed light on the "start-up" behavior of search schemes.

CHAPTER 2. THE MATHEMATICAL MODEL AND ASYMPTOTIC RESULTS

In this chapter the mathematical model is developed. Then the proofs are shown of two of the asymptotic results presented in chapter 1. These particular results were chosen for three reasons. First, they are the most interesting; second, they are important results; and third, they give a good overall example of the type of mathematics used in the study of sequential search files.

First, however, some quantities which are useful in developing the model and in proving the three theorems are defined below. For easy reference and for the sake of completeness the notation previously introduced is repeated here.

N = the number of records in a file

R_i = the ith record, $i=1,2,\ldots,N$

$R_{i(k)}$ = the ith record, R_i, is in the kth position

p_i = the probability that R_i will be requested, $i=1,2,\ldots,N$
where $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_N$

$p_{i(k)}$ =the probability of the ith record, which is in the kth position

f_i = the number of times record i is requested, $i=1,2,\ldots,N$

$F = \sum_{i=1}^{N} f_i$ = the total number of requests made to a file

OC = optimal cost of a file = $\sum_{i=1}^{N} ip_i$

N! = the number of possible orderings of the file

π_j = the steady state probability that arrangement j will occur, j=1,2,...,N! $\left(\sum_{j=1}^{N!} \pi_j = 1\right)$

c_j = the cost of arrangement j, j=1,2,...,N!

AC_S = the asymptotic cost of scheme S = $\sum_{j=1}^{N!} \pi_j c_j$

q(m,n) = the probability of occurrence of the nth ordering after the next request if the present ordering is known to be the mth (a transition probability)

Q =the N!*N! transition matrix consisting of elements q(m,n)

b(i,j) = the asymptotic probability that R_i is before R_j in the file (for $1 \leqslant i < j \leqslant N$)

Now the mathematical model will be developed. The present ordering of the file is dependent solely on the last request, the previous ordering and the scheme employed. If the present ordering is known then the next ordering will be completely determined by the next request. So the probability of occurrence of any ordering after the next request can be predicted if the present ordering and the scheme are known. For a file of N records there are N! possible orderings. Suppose these N! orderings are listed and indexed by 1,2,...,N!. The probability of occurrence of the nth ordering after the next request if the present ordering is known to be the mth is called a transition probability and is denoted by q(m,n). Naturally this transition matrix depends on p_i, i=1,2,...,N and the scheme used.

This situation can be described by a Markov chain with N! states. The transition matrix of the chain will be denoted Q. Obviously this chain is finite and, since it is possible with the right set of requests to obtain any other ordering from the present ordering, it is irreducible. For the MTF and the TR schemes it is easily seen that their Markov chains are also aperiodic. The literature considers only schemes which have aperiodic chains. If a Markov chain is finite, irreducible and aperiodic then there exists a limiting distribution. That is, $\lim_{F \to \infty} Q^F = \Pi$ where Q^F is the Fth power of the matrix Q. Since the limiting probabilities are independent of starting values, the rows of Π are identical. The elements of any row are denoted Π_1, $\Pi_2, \ldots, \Pi_{N!}$. These stationary probabilities, Π_j, $j=1,2,\ldots,N!$, are given by the following set of equations: $\underline{\Pi} = \underline{\Pi}\,\underline{Q}$. (Proofs of these last two statements can be found in Feller [5].)

Next are the two theorems and their proofs.

<u>THEOREM 1</u>: The asymptotic cost of the TR rule is lower than that of the MTF for any probability distribution. They are equal only when $N=2$ or when all the p_i $(i=1,2,\ldots,N)$ are equal.

<u>Proof</u>: Several intermediate results are necessary before this theorem can be proved. (The proof of this theorem is attributed to Rivest [18]. However, the intermediate steps below were shown by Hendricks in [8] and [10]. The following version of this proof can be found in Lam [13].)

<u>Theorem 1a</u>. Let $b_1(j,i)$ and $b_2(j,i)$ denote the asymptotic probability that R_j is before R_i in the file for schemes 1 and 2, respectively. Let AC_1 and AC_2 denote the asymptotic costs of schemes 1 and 2. If $b_1(j,i) \leqslant b_2(j,i)$ whenever $i<j$, then $AC_1 \leqslant AC_2$.

Proof

The asymptotic cost, AC, is defined as $AC=\sum_{j=1}^{N!} \pi_j c_j$,

It is desirable now to express AC in another form.

First define M_i to denote the expected position of R_i.

$$\text{So } AC=\sum_{i=1}^{N} p_i M_i .$$

(It can be shown that these two definitions are equal.)

Define $X^{(i)}$ = position of R_i and

$$X_j^{(i)} = \begin{cases} 1 \text{ if } R_j \text{ is before } R_i \text{ in the file} \\ 0 \text{ otherwise} \end{cases}$$

$$\text{Now } M_i = E[X^{(i)}] = E\left[1+\sum_{j=1}^{N} X_j^{(i)}\right]$$

$$=1+\sum_{j=1}^{N} E[X_j^{(i)}]$$

$$=1+\sum_{j=1}^{N} P[X_j^{(i)}=1]$$

$$=1+\sum_{j=1}^{N} b(j,i)$$

Now AC can be expressed by

$$AC=\sum_{i=1}^{N} p_i M_i$$

$$= \sum_{i=1}^{N} p_i \, [1+\sum_{j=1}^{N} b(j,i)]$$

$$= \sum_{i=1}^{N} p_i + \sum_{i=1}^{N} \sum_{j=1}^{N} p_i \, b(j,i)$$

$$=1 + \sum_{i=1}^{N} \sum_{j=1}^{N} p_i \, b(j,i)$$

$$=1+ \sum_{i=1}^{N} \{ \sum_{j=1}^{i-1} p_i \, (1-b(i,j)) + 0 + \sum_{j=i+1}^{N} p_i \, b(j,i)\}$$

$$= \sum_{i=1}^{N} p_i + \sum_{i=1}^{N} (i-1)p_i - \sum_{i=2}^{N} \sum_{j=1}^{i-1} p_i \, b(i,j)$$

$$+ \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} p_i \, b(j,i)$$

$$= \sum_{i=1}^{N} ip_i + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} p_i \, b(j,i)$$

$$- \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} p_j \, b(j,i)$$

$$= \sum_{i=1}^{N} ip_i + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (p_i - p_j) b(j,i)$$

This is the asymptotic cost for any scheme.

Thus the difference between the costs of two schemes is

$$AC_2 - AC_1 = \sum_{i=1}^{N} ip_i + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (p_i - p_j) b_2(j,i) -$$

$$\sum_{i=1}^{N} ip_i - \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (p_i - p_j) b_1(j,i)$$

$$= \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (p_i - p_j)[b_2(j,i) - b_1(j,i)]$$

Since $p_i \geq p_j$, if $b_1(j,i) \leq b_2(j,i)$ then $AC_1 \leq AC_2$

.

__Theorem 1b__. $b(j,i)$ for the MTF scheme is equal to $\dfrac{p_j}{p_j + p_i}$.

__Proof__

Define $b^{(F)}(j,i)$ to be the probability that R_j is before R_i $(i \neq j)$ in the file after F requests have been made. This event will occur if either

(i) R_j has been requested more recently than R_i. This is actually a set of mutually exclusive events. These are illustrated below.

$p_j(1-p_j-p_i)^{F-1}$ = probability that R_j was the first request and none of the following requests were for for R_i or R_j.

$p_j(1-p_j-p_i)^{F-2}$ = probability that R_j was last requested on the second request. (It could also have been the first request.) None of the remaining (F-2) requests were for R_i or R_j.

[Note: This is also a set of mutually exclusive events. Any of the N records could have been requested first. This probability then becomes

$$p_1 p_j (1-p_j-p_i)^{F-2} + p_2 p_j (1-p_j-p_i)^{F-2} + \ldots + p_N p_j (1-p_j-p_i)^{F-2}$$

$$=(p_1+p_2+\ldots+p_N) p_j (1-p_j-p_i)^{F-2} = p_j (1-p_j-p_i)^{F-2} \quad]$$

.

.

.

$p_j (1-p_j-p_i)^0$ = probability that R_j was last requested on the Fth request.

(ii) both R_j and R_i are not requested throughout, in which case the probability that R_j is placed before R_i is 1/2.

Hence

$$b^{(F)}(j,i) = p_j \sum_{k=0}^{F-1} (1-p_j-p_i)^k + (1/2)(1-p_j-p_i)^F$$

As F gets very large $b^{(F)}(j,i)$ approaches $b(j,i)$, the second term approaches zero and the summation in the first term becomes the geometric series. Thus,

$$b_{MTF}(j,i)=\frac{p_j}{p_j+p_i}$$

<u>Theorem</u> <u>1c</u>. The stationary distribution of the TR rule is given by:

$$P_j=\{\prod_{k=1}^{N} p_{i(k)j}^{N-k}\}/L, \quad L=\sum_{o}\{\prod_{j=1}^{N} p_{o(i,k,j)}^{N-j}] \quad \text{for } j=1,\ldots,N!$$

where the summation ranges over all possible orderings and $p_{i(k)j}$ denotes the probability of the ith record which is in the kth position in the jth arrangement. O(i,k,j) is the set of all possible orderings so $p_{o(i,j,k)}$ is $p_{i(k)j}$ in the oth ordering.

Note: L is a normalizing factor. Dividing the values by L produces a set of N! probabilities that sum to 1.

<u>Proof</u>

For ease, consider the case when the file is arranged as $(R_1,R_2,R_3,\ldots,R_N)$. By the TR rule, this state can only be reached from:

$(R_1, R_2, R_3, \dots, R_N)$ when R_1 is requested

$(R_2, R_1, R_3, \dots, R_N)$ when R_1 is requested

$(R_1, R_3, R_2, \dots, R_N)$ when R_2 is requested

.

.

.

$(R_1, R_2, \dots, R_N, R_{N-1})$ when R_{N-1} is requested

The stationary probabilities of these states will be denoted $\pi_1, \pi_2, \pi_3, \dots, \pi_N$ and they, of course, must satisfy the set of equations $\underline{\pi} = \underline{\pi}\ \underline{Q}$. It is sufficient to show that the following equation holds

$$\pi_1 = p_1 \pi_1 + p_1 \pi_2 + p_2 \pi_3 + \dots + p_{N-1} \pi_N$$

π_1 is assumed to be the value given in the theorem.

So,

$$\begin{aligned}
(p_1^{N-1} p_2^{N-2} \cdots p_{N-1})/L = {} & p_1 (p_1^{N-1} p_2^{N-2} \cdots p_{N-1})/L \\
& + p_1 (p_2^{N-1} p_1^{N-2} \cdots p_{N-1})/L \\
& + p_2 (p_1^{N-1} p_3^{N-2} p_2^{N-3} \cdots p_{N-1})/L \\
& + \dots + p_{N-1} (p_1^{N-1} p_2^{N-2} \cdots p_N)/L \\
& \\
= {} & p_1 (p_1^{N-1} p_2^{N-2} p_3^{N-3} \cdots p_{N-1})/L
\end{aligned}$$

$$+p_2(p_1^{N-1}p_2^{N-2}p_3^{N-3}\dots p_{N-1})/L$$

$$+p_3(p_1^{N-1}p_2^{N-2}p_3^{N-3}\dots p_{N-1})/L$$

$$+\dots+p_N(p_1^{N-1}p_2^{N-2}\dots p_{N-1})/L$$

$$=(p_1+p_2+\dots+p_N)(p_1^{N-1}\dots p_{N-1})/L$$

$$=(p_1^{N-1}p_2^{N-2}p_3^{N-3}\dots p_{N-1})/L$$

<u>Theorem</u> <u>1d</u>. Let G be any arrangement of records in which R_i precedes R_j with k records between them. Let G′ be the new arrangement of records obtained by interchanging the positions of R_i and R_j in G. Then

$$\frac{P(G)}{P(G')} = \frac{p_i^{k+1}}{p_j^{k+1}}$$

where P(G), P(G′) are the probabilities of occurrence of G, G′ respectively under the stationary distribution of the TR scheme.

<u>Proof</u>

Let G be the arrangement $(R_{i(1)} \ldots R_i \ldots R_j \ldots R_{i(N)})$ where $R_{i(x)}$ is the ith record that is in the xth position, R_i is in the ath position and R_j is in the (a+k+1)th position. So G′ is the arrangement $(R_{i(1)} \ldots R_j \ldots R_i \ldots R_{i(N)})$. Using theorem 1c

$$\frac{P(G)}{P(G')} = \frac{(p_{i(1)}^{N-1} \ldots p_i^{N-a} \ldots p_j^{N-a-k-1} \ldots p_{i(N-1)})/L}{(p_{i(1)}^{N-1} \ldots p_j^{N-a} \ldots p_i^{N-a-k-1} \ldots p_{i(N-1)})/L}$$

$$= \frac{p_i^{k+1}}{p_j^{k+1}}$$

Now the main theorem can be proved.

Let G be the event that R_j precedes R_i in the file so $P(G)=b_{TR}(j,i)$. If $A_1,\ldots,A_S$ are all those arrangements for which R_j precedes R_i, then $G=\{A_1,\ldots,A_S\}$. Hence, $b_{TR}(j,i)=P(A_1)+\ldots+P(A_S)$ where $P(A_i)$ is the probability of occurrence of the arrangement A_i in the stationary distribution of the TR scheme. Let A'_k denote the new arrangement obtained by interchanging the positions of R_i and R_j in A_k. So G′ is the set of all arrangements in which R_i precedes R_j in the file. Obviously P(G)=1−P(G′). By Theorem 1d,

$$\frac{P(A_k)}{P(A_k')} = \frac{p_j^t}{p_i^t} \quad \text{for some } t \geqslant 1 \quad (k=1,2,\ldots,S)$$

$$\frac{P(A_k)}{P(A_k')} \leqslant \frac{p_j}{p_i} \qquad \text{since } p_i \geqslant p_j$$

This is true for all arrangements A_k so this can be expanded to

$$\frac{P(G)}{P(G')} = \frac{P(A_1)+\ldots+P(A_S)}{P(A_1')+\ldots+P(A_S')} \leqslant \frac{p_j}{p_i} \qquad \text{or}$$

$$p_i P(G) < p_j [1-P(G)]$$

Solving for $P(G)=b_{TR}(j,i)$ gives

$$b_{TR}(j,i) \leqslant \frac{p_j}{(p_j+p_i)}$$

So by Theorem 1b,

$$b_{TR}(j,i) \leqslant b_{MTF}(j,i)$$

and by Theorem 1a,

$$AC_{TR} \leqslant AC_{MTF}$$

The cases of equality when N=2 or when all the p_i are equal are quite easily done and are left to the reader.

THEOREM 2: After the FR (First Request rule) initializing procedure (after each record has been requested at least once) the system is at steady state relative to the MTF scheme.

Proof: Rivest [18] proved the following theorem. From this the main assertion above follows directly.

Theorem 2a. For any probability distribution the probability of obtaining a given final ordering of the file after any number of requests is the same for the MTF and the FR rules.

Proof

For any sequence of requests $f_1,...,f_k$ to the MTF rule, the sequence of requests $f_k,...,f_1$ to the FR rule produces the same final ordering of the file. Let $S=\{f_1,...,f_k\}$ be the sequence of requests to the MTF rule, and $S'=\{f_k,\ldots,f_1\}$ the sequence of requests to the FR rule. P(S) and P(S') are their respective probabilities.

$$P(S)=p_1 p_2 \ldots p_k \text{ and } P(S')=p_k p_{k-1} \ldots p_1 \text{ so } P(S)=P(S')$$

Since the probability of obtaining a given final ordering of the file is the same for both schemes then their stationary probability distributions are the same. After every record has been requested at least once the FR rule is at steady state. (No records are moved again once this point is reached.) Thus, the MTF rule is also at steady state.

CHAPTER 3. SMALL SAMPLE CASE

There are no analytical results for the small sample case. Before looking at the small sample case, there is a need for a measure of how close the current ordering of the file is to the optimal ordering. There are three ways of approaching this.

The first approach is to maximize $W = \sum_{i=1}^{N} d_i^2 p_i = \sum_{i=1}^{N} (i-R_i)^2 p_i$

Then a measure of closeness could be defined as

$$D = 1 - \left[\frac{\sum_{i=1}^{N} d_i^2 p_i}{\max\{\sum_{i=1}^{N} d_i^2 p_i\}}\right]$$

When the file is in the worst ordering possible then W is at its maximum and D=0. When the file is in the best ordering possible, then $W = \sum_{i=1}^{N} (i-i)^2 p_i = 0$ and D=1. So this is a normalized measure between 0 and 1. This would seem to be a good measure, however, maximizing W is not so easy. This depends on the probabilities, p_i.

In general, finding the maximum requires looking at all possible orderings to find the one which gives the largest value of W. Even with a file of 10 records this requires looking at

10!=362,880 orderings.

The second approach is to look at the correlation between the ordering of the file and the optimal ordering. The probability that record R_i is in the kth position is not known. So the distribution P_i is used instead. In this way, records with the greater chance of being selected will contribute more to the value of the correlation coefficient. This produces the empirical correlation below.

$$EC = \frac{\sum_{i=1}^{N} iR_i p_i - (\sum_{i=1}^{N} ip_i)(\sum_{i=1}^{N} R_i p_i)}{[\sum_{i=1}^{N} i^2 p_i - (\sum_{i=1}^{N} ip_i)^2][\sum_{i=1}^{N} R_i^2 p_i - (\sum_{i=1}^{N} R_i p_i)^2]}$$

It is easy to see that when the file is in the optimal ordering, then $i=R_i$ for all i and the empirical correlation is equal to 1. For the case when the file is in the reverse ordering it is easy to show that the empirical correlation is equal to -1.

The third approach is to use Spearman's Rank correlation since the values of i and R_i are, indeed, ranks. Spearman's Rank correlation is

$$SRC = 1 - [\frac{6 \sum_{i=1}^{N} (i-R_i)^2}{N(N^2 - 1)}]$$

A simulation was run to investigate some of the properties of the MTF and the TR rules for small samples. (A listing of the program is in Appendix A.) The simulation was run for N=20 and N=100 with a linear decrease in the probabilities. These probabilies were:

$$N/S, (N-1)/S, (N-2)/S, \ldots, 1/S \qquad \text{where} \quad S = \sum_{i=1}^{N} i$$

Two cases were considered. The worst case is the one that starts with all the records in reverse order. The other case is one that starts with the correlations at about zero, hereafter referred to as the zero case. The records were initially ordered as $R_{1(1)}, R_{4(2)}, R_{5(3)}, \ldots, R_{6(N-2)}, R_{3(N-1)}, R_{2(N)}$, where $R_{i(k)}$ denotes R_i is in the kth position.

Three measures were calculated:

(1) Empirical Correlation, EC

(2) Spearman's Rank Correlation, SRC

(3) Cost, Cost=the position of the requested record.

Table 2 shows the number of requests made to the file and the number of replications for the four simulation runs. The zero case had the 3 measures calculated for the first 25 requests and then for every 25th request thereafter. The different number of replications for all the cases is because of computer difficulties. These programs took a lot of computer time, particularly when N=100. For example, 15 replications for the worst case when N=100 required 3 minutes of execution time. Since funds were limited, the number of replications was kept to a minimum.

Table 2. Number of requests and number of replications for the simulation runs.

N	Case	No. of Req.	No. of Rep.
20	Worst	200	20
100	Worst	200	15
20	Zero	600	50
100	Zero	600	27

In all cases the conclusions are clear. The TR scheme very steadily but very, very slowly moves toward the optimum. This result was expected. When N=100, worst case, after 200 requests both correlations for the TR are only about -0.996. The MTF in this case has the correlations at about 0.42. Even when N=20, zero case, the MTF reaches 0.35 after about 15 requests. The TR takes about 75 requests for the correlations to reach 0.35.

The MTF, however, jumps around a lot more than the TR. The EC for the MTF when N=20 seems to oscillate about 0.35. The SRC seems to oscillate around 0.50. For N=100 these values are the same, it just takes longer to reach these points. The EC and the SRC for the

TR when N=20 and when N=100 seems to be still steadily increasing. When N=20, zero case, after 600 requests the EC=0.796 and the SRC=0.854. The standard deviation in the correlations for the MTF is larger than the standard deviation in for the TR by roughly a factor of 10. So, as expected, the TR is better in the long run even though the MTF is initially the best scheme.

Appendix C contains graphs of the average cost (averaged over replications) versus request number for the four simulation runs. It also contains graphs of the 5-point moving average for cost versus request number for the four runs.

The asymptotic cost, AC, can be calculated for the MTF. Table 3 shows these values for the simulation cases. Four other quantities are given in Table 3. These are defined below:

1) OS = The approximate request number at which the cost of the MTF begins to oscillate around its AC. (So the MTF is at steady state after this number of requests.)
2) L(C) = The approximate number of requests required before the cost of the TR rule is consistently lower than that of the MTF rule. Since the cost jumps around so much this is based on a 5 point moving average.

3) L(EC) = The approximate number of requests required before EC of the TR is consistently higher than that of the MTF.

4) L(SRC) = The approximate number of requests required before the SRC of the TR is consistently higher than that of the MTF.

Table 3. Results of the simulation.

N	Case	AC(MTF)	OS	L(C)	L(EC)	L(SRC)
20	Worst	8.876	30	*	*	*
100	Worst	41.616	*	*	*	*
20	Zero	8.876	21	250	75	200
100	Zero	41.616	125	500	**	**

* Number of requests is greater than 200.
** Number of requests is greater than 600.

The results shown in Table 3 indicate that when the MTF is at steady state, the TR is not as good as the MTF. Furthermore, the MTF continues to perform better than the TR for quite some time. All three measures in every case show that the number of requests until the TR performs better than the MTF is quite a bit larger than the number of requests required for the MTF to achieve steady state. This indicates that the MTF should always be considered when the number of requests to the file are relatively low. Even when the file consists of only 20 records the MTF performs better than the TR for a relatively large amount of time. In the zero case the TR takes more than triple the number of requests for its EC to become larger than the EC of the MTF. The number of requests required for the other measures to indicate that the TR is performing better is even larger. These results also indicate that the hybrid method which starts with the MTF and then switchs to the TR is a potentially good method.

In summary, the TR is better than the MTF in the long run. However, the MTF is much better for an initial period. Based on this simulation, this initial period is relatively long. The MTF continues to perform better than the TR for quite some time after the MTF has achieved steady state. This enforces the suggestion of several authors that the MTF should be very seriously considered when the file is small. When the file is large, then a non-sequential search technique should probably be employed.

REFERENCES :

1. ALLEN,B. and MUNRO,I. Self-organizing binary search trees. J. ACM,Vol.25,No.4,Oct.1978,pp.526-535

2. BAER,J.L. Weight-balanced trees. Proc. AFPIS 1975 NCC, pp.467-472

3. BENTLEY,J.L. and McGEOCH,C.C. Amortized Analyses of Self-Organizing Sequential Search Heuristics. J. ACM, to appear Fall 1984

4. BITNER,J.R. Heuristics that dynamically organize data structures. SIAM J. COMPUT.,Vol.8,No.1,Feb.1979,pp.82-110

5. BURVILLE,P.J. and KINGMAN,J.F.C. On a model for storage and search. J. Appl. Prob.,Vol.10,1973,pp.697-701

6. FELLER,W. An Introduction to Probability Theory and Its Applications, Vol.1 John Wiley and Sons, New York 1957

7. GONNET,G.H.,MUNRO,J.I. and SUWANDA,H. Exegesis on self-organizing linear search. SIAM J. COMPUT., Vol.10, No.3, Aug.1981, pp.613-637

8. HENDRICKS,W.J. The stationary distribution of an interesting Markov chain. J. Appl. Prob.,Vol.9,1972,pp.231-233

9. HENDRICKS,W.J. An extension of a theorem concerning an interesting Markov chain. J. Appl. Prob., Vol.10,1973, pp.886-890

10. HENDRICKS,W.J. An account of self-organizing files. SIAM J. COMPUT., Vol.5,No.4,1976,pp.715-723

11. KNUTH,D.E. The Art of Computer Programming, Vol.3 : Sorting and Searching Addison-Wesley, Reading Mass. 1973,pp.389-402

12. KONNEKER,L.K. and VAROL,Y.L. A note on heuristics for dynamic organization of data structures. INFO. PROC. LETTERS Vol.12, No.5, 1981 pp.213-216

13. LAM,K. Self-Organizing Files. UMAP,Vol.4,NO.1,1983,pp.53-84

14. LAM,K.,SIU,M.K. and YU,C.T. A generalized counter scheme. Theo. Comp. Sci., Vol.16,1981,pp.271-278

15. LETAC,G. Transience and recurrence of an interesting Markov chain. J. Appl. Prob., Vol.11,1974,pp.818-824

16. McCABE,J. On serial files with relocatable records. Operations Res., Vol.12,1965,pp.609-618

17. NELSON,P.R. A prediction interval search scheme for the move-to-the-front replacement algorithm. J. Inst. Maths Appl. Vol.24,No.2,Sept.1979,pp.231-236

18. RIVEST,R. On self-organizing sequential search heuristics. Comm. ACM, Vol.19,No.2,Feb.1976,pp.63-67

19. SAVCHENKO,L.A. General self-organizing sequential search procedure. Prog. Comp. Software, Vol.7,No.5, 1981,pp.282-285

20. TAKANAMI,I. and FUJII,M. On heuristic construction of self-organizing files for sequential searches. TRANS. IECE of Japan, Vol.E60,No.2,1977,p.112

21. TENENBAUM, A. Simulations of dynamic sequential search algorithms. Comm. ACM, Vol.21,No.9,1978,pp.790-791

APPENDIX A. A LISTING OF THE SIMULATION PROGRAM

```
//*++ PRINT SERVICE UNATTEND LINES 9 TIME 3,0 VMMSG
/*REGION        700K
// EXEC PASCLG
//SYSIN DD *
PROGRAM  SIM(INPUT,OUTPUT);
(*                                                                    *)
(*             THIS PROGRAM SIMULATES THE MOVE TO FRONT              *)
(*               AND THE TRANSPOSITION SCHEMES FOR A                 *)
(*                    SEQUENTIAL SEARCH FILE.                         *)
(*                                                                    *)
CONST
  N=20;
  NUMBER_REQ=600;
  REP=50;
  OPTION='P';
  DECREASE='L';
  ORDER='Z';
TYPE
  KEY = ARRAY(.1..N.) OF INTEGER;
  PROBABILITY = ARRAY(.1..N.) OF REAL;
  DESCRIPTIVE = ARRAY(.0..NUMBER_REQ.) OF REAL;
VAR
  RMTF,RTR                    : KEY;
  P,CUMMP                     : PROBABILITY;
  CORRMTF,CORRTR, RNUM        : REAL;
  SPEARMTF,SPEARTR,DDD        : REAL;
  COSTMTF,COSTTR              : INTEGER;
  I,J,L,M,C,D,H,X,RANDOMSEED,REQ,Z,Y,U,V,T,MULT : INTEGER;
  CORRSMTF,CORRSQMTF,CORRSTR,CORRSQTR      : DESCRIPTIVE;
  SPEARSMTF,SPEARSQMTF,SPEARSTR,SPEARSQTR  : DESCRIPTIVE;
  COSTSMTF,COSTSQMTF,COSTSTR,COSTSQTR      : DESCRIPTIVE;
  AVE_CR_MTF,AVE_CR_TR,AVE_S_MTF,AVE_S_TR  : DESCRIPTIVE;
  AVE_CS_MTF,AVE_CS_TR,SD_CR_MTF,SD_CR_TR  : DESCRIPTIVE;
  SD_S_MTF,SD_S_TR,SD_CS_MTF,SD_CS_TR      : DESCRIPTIVE;
(*                                                                    *)
(*             THIS FUNCTION GENERATES RANDOM NUMBERS                *)
(*                                                                    *)
FUNCTION RANDOM(VAR SEED:INTEGER):REAL;
BEGIN
  RANDOM:=SEED/65535;
  SEED:=(25173*SEED+13849) MOD 65536
END;
(*                                                                    *)
(*               THIS FUNCTION CONVERTS THE RANDOM                   *)
(*                     NUMBER INTO A REQUEST                          *)
(*                                                                    *)
FUNCTION REQUEST(CUMMPROB:PROBABILITY;RN:REAL):INTEGER;
VAR
  K : INTEGER;
BEGIN
  IF RN<=CUMMPROB(.1.) THEN
    REQUEST:=1
  ELSE
    FOR K:=2 TO N DO
      IF (RN>CUMMPROB(.K-1.)) AND (RN<=CUMMPROB(.K.)) THEN
        REQUEST:=K
END;
```

```
(*                                                                          *)
(*          THIS FUNCTION CALCULATES THE MODIFIED CORRELATION               *)
(*                                                                          *)
FUNCTION CORRELATION(REC:KEY;PROB:PROBABILITY;N:INTEGER):REAL;
VAR
  K : INTEGER;
  MEANX,MEANY,MEANXY,SQX,SQY,STDX,STDY : REAL;
BEGIN
  MEANX:=0.0; MEANY:=0.0; MEANXY:=0.0;
  SQX:=0.0; SQY:=0.0;
  FOR K:=1 TO N DO
    BEGIN
      MEANX:= MEANX +K*PROB(.K.);
      MEANY:= MEANY + REC(.K.)*PROB(.K.);
      MEANXY:= MEANXY + K*REC(.K.)*PROB(.K.);
      SQX:= SQX + K*K*PROB(.K.);
      SQY:= SQY + REC(.K.)*REC(.K.)*PROB(.K.);
    END;
  STDX:= SQX - MEANX*MEANX;
  STDY:= SQY - MEANY*MEANY;
  CORRELATION:= (MEANXY -MEANX*MEANY)/SQRT(STDX*STDY)
END;
(*                                                                          *)
(*          THIS FUNCTION CALCULATES SPEARMANS RANK CORRELATION             *)
(*                                                                          *)
FUNCTION SPEARMAN(REC:KEY;N:INTEGER):REAL;
VAR
  K : INTEGER;
  DIFF : REAL;
BEGIN
  DIFF:=0.0;
  FOR K:=1 TO N DO
    DIFF:=DIFF + SQR(K-REC(.K.));
  SPEARMAN:= 1.0 - (6*DIFF)/(N*(N*N-1))
END;
(*                                                                          *)
(*                 THIS PROCEDURE SETS THE PROBABILITIES                    *)
(*                    (BOTH LINEAR AND EXPONENTIAL)                         *)
(*                                                                          *)
PROCEDURE SETPROB(DEC:CHAR;VAR PROB:PROBABILITY);
VAR
  E,F,G,K : INTEGER;
  SUMP    : REAL;
BEGIN
  IF DEC='L' THEN
    BEGIN
      SUMP:=0;
      FOR K:=1 TO N DO
        SUMP:=SUMP+K;
      PROB(.N.):=(1/SUMP);
      FOR G:=(N-1) DOWNTO 1 DO
        PROB(.G.):=(N-G+1)*PROB(.N.);
    END;
  IF DEC='E' THEN
    BEGIN
      SUMP:=1;
      FOR E:=1 TO N DO
```

```
          SUMP:= SUMP+EXP(-E);
        PROB(.1.):=(1/SUMP);
        FOR F:=2 TO N DO
          PROB(.F.):=EXP(-(F-1))*PROB(.1.);
      END;
END;
(*                                                                  *)
(*              THIS PROCEDURE REORDERS THE FILE ACCORDING          *)
(*                      TO THE MOVE TO FRONT RULE                   *)
(*                                                                  *)
PROCEDURE MOVETOFRONT (RECREQ,N:INTEGER;VAR REC:KEY;VAR COST:INTEGER);
VAR
  K,B : INTEGER;
BEGIN
  FOR K:=1 TO N DO
    IF REC(.K.)=RECREQ THEN
      BEGIN
        COST:=K;
        IF K<>1 THEN
          BEGIN
            FOR B:=K DOWNTO 2 DO
              REC(.B.):=REC(.B-1.);
            REC(.1.):=RECREQ
          END
      END
END;
(*                                                                  *)
(*              THIS PROCEDURE REORDERS THE FILE ACCORDING          *)
(*                      TO THE TRANSPOSITION RULE                   *)
(*                                                                  *)
PROCEDURE TRANSPOSITION(RECREQ,N:INTEGER;VAR REC:KEY;VAR COST:INTEGER);
VAR
  K : INTEGER;
BEGIN
  FOR K:=1 TO N DO
    IF REC(.K.)=RECREQ THEN
      BEGIN
        COST:=K;
        IF K<>1 THEN
          BEGIN
            REC(.K.):=REC(.K-1.);
            REC(.K-1.):=RECREQ
          END
      END
END;
(*                                                                  *)
(*                                                                  *)
(*                      THE MAIN PROGRAM BEGINS HERE                *)
(*                                                                  *)
(*                                                                  *)
BEGIN
  RANDOMSEED:=8191;
(*                                                                  *)
(*                          INITIALIZE ARRAYS                       *)
(*                                                                  *)
  FOR D:=0 TO NUMBER_REQ DO
    BEGIN
```

```
        CORRSMTF(.D.):=0.0;
        CORRSQMTF(.D.):=0.0;
        CORRSTR(.D.):=0.0;
        CORRSQTR(.D.):=0.0;
        SPEARSMTF(.D.):=0.0;
        SPEARSQMTF(.D.):=0.0;
        SPEARSTR(.D.):=0.0;
        SPEARSQTR(.D.):=0.0;
        COSTSMTF(.D.):=0.0;
        COSTSQMTF(.D.):=0.0;
        COSTSTR(.D.):=0.0;
        COSTSQTR(.D.):=0.0;
      END;
(*                                                                  *)
(*                  SET INITIAL PROBABILITIES                       *)
(*               COMPUTE CUMMULATIVE PROBABILITIES                  *)
(*                                                                  *)
  SETPROB(DECREASE,P);
  CUMMP(.1.):=P(.1.);
  FOR L:=2 TO N DO
    CUMMP(.L.):=CUMMP(.L-1.) + P(.L.);
(*                                                                  *)
(*           SET INITIAL ORDERING, CALCULATE INITIAL                *)
(*                  CORRELATIONS AND COST                           *)
(*                                                                  *)
  FOR H:=1 TO REP DO
    BEGIN
      CASE ORDER OF
        'W': FOR V:=1 TO N DO
               RMTF(.V.):=(N-V+1);
        'B': FOR V:=1 TO N DO
               RMTF(.V.):=V;
        'Z': BEGIN
               RMTF(.1.):=1;
               RMTF(.2.):=N;
               RMTF(.3.):=N-1;
               FOR T:=1 TO (N DIV 4 - 1) DO
                 BEGIN
                   MULT:=4*T;
                   RMTF(.MULT.):=MULT DIV 2;
                   RMTF(.MULT+1.):=RMTF(.MULT.)+1;
                   RMTF(.MULT+2.):=N+MULT DIV 2 - MULT;
                   RMTF(.MULT+3.):=RMTF(.MULT+2.)-1
                 END;
               RMTF(.N.):=N DIV 2
             END
      END;
      FOR U:=1 TO N DO
        RTR(.U.):=RMTF(.U.);
      CORRTR:=CORRELATION(RTR,P,N);
      SPEARTR:=SPEARMAN(RTR,N);
      CORRSTR(.0.):=CORRSTR(.0.)+CORRTR;
      CORRSMTF(.0.):=CORRSTR(.0.);
      CORRSQTR(.0.):=CORRSQTR(.0.)+CORRTR*CORRTR;
      CORRSQMTF(.0.):=CORRSQTR(.0.);
      SPEARSTR(.0.):=SPEARSTR(.0.)+SPEARTR;
      SPEARSMTF(.0.):=SPEARSTR(.0.);
```

```
      SPEARSQTR(.0.):=SPEARSQTR(.0.)+SPEARTR*SPEARTR;
      SPEARSQMTF(..0.):=SPEARSQTR(.0.);
(*                                                                    *)
(*                        WRITE HEADINGS                              *)
(*                                                                    *)
      IF OPTION='P' THEN
        BEGIN
          WRITE ('1');
          WRITELN (' ':41,'REPLICATION',H:4);
          WRITELN;
      WRITE (' ':5,'REQUEST   REQUEST   CORRELATION   CORRELATION');
      WRITELN ('     SPEARMANS    SPEARMANS     COST      COST');
      WRITE (' ':7,'NO.',' ':7,'FOR',' ':9,'MTF',' ':11,'TR',' ':12);
      WRITELN ('MTF',' ':10,'TR',' ':9,'MTF',' ':7,'TR');
      WRITE (' ':5,'-------   ------   ----------   -----------');
      WRITELN ('.    ---------    ---------     ----      ----');
          WRITELN;
          WRITELN;
          WRITE (' ':8,'0',' ':17,CORRTR:8:5,' ':6,CORRTR:8:5);
          WRITELN (' ':6,SPEARTR:8:5,' ':5,SPEARTR:8:5);
        END;
(*                                                                    *)
(*                        MAIN SIMULATION                             *)
(*                                                                    *)
      FOR M:=1 TO NUMBER_REQ DO
        BEGIN
          RNUM:=RANDOM(RANDOMSEED);
          REQ:=REQUEST(CUMMP,RNUM);
          MOVETOFRONT(REQ,N,RMTF,COSTMTF);
          TRANSPOSITION(REQ,N,RTR,COSTTR);
          IF (M<=25) OR (M MOD 25 =0) THEN
           BEGIN
            CORRMTF:=CORRELATION(RMTF,P,N);
            SPEARMTF:=SPEARMAN(RMTF,N);
            CORRTR:=CORRELATION(RTR,P,N);
            SPEARTR:=SPEARMAN(RTR,N);
(*                                                                    *)
            CORRSMTF(.M.):=CORRSMTF(.M.)+CORRMTF;
            CORRSQMTF(.M.):=CORRSQMTF(.M.)+CORRMTF*CORRMTF;
            SPEARSMTF(.M.):=SPEARSMTF(.M.)+SPEARMTF;
            SPEARSQMTF(.M.):=SPEARSQMTF(.M.)+SPEARMTF*SPEARMTF;
            COSTSMTF(.M.):=COSTSMTF(.M.)+COSTMTF;
            COSTSQMTF(.M.):=COSTSQMTF(.M.)+COSTMTF*COSTMTF;
            CORRSTR(.M.):=CORRSTR(.M.)+CORRTR;
            CORRSQTR(.M.):=CORRSQTR(.M.)+CORRTR*CORRTR;
            SPEARSTR(.M.):=SPEARSTR(.M.)+SPEARTR;
            SPEARSQTR(.M.):=SPEARSQTR(.M.)+SPEARTR*SPEARTR;
            COSTSTR(.M.):=COSTSTR(.M.)+COSTTR;
            COSTSQTR(.M.):=COSTSQTR(.M.)+COSTTR*COSTTR;
(*                                                                    *)
            IF OPTION='P' THEN
              BEGIN
                WRITE (' ':6,M:3,' ':7,REQ:4,' ':6,CORRMTF:8:5,' ':6);
                WRITE (CORRTR:8:5,' ':6,SPEARMTF:8:5,' ':5);
                WRITELN (SPEARTR:8:5,' ':6,COSTMTF:4,' ':5,COSTTR:4);
              END
         END
```

```
         END
      END;
(*                                                              *)
(*        WRITE HEADINGS FOR MEANS AND STANDARD DEVIATIONS      *)
(*                                                              *)
    WRITELN ('1');
    WRITE (' ':19,'MEANS AND STANDARD DEVIATIONS (AVERAGED OVER');
    WRITELN (' REPLICATIONS)');
    WRITELN;
    WRITELN (' ':33,'NO. OF RECORDS IN A FILE =',N:5);
    WRITELN (' ':36,'NO. OF REPLICATIONS =',REP:4);
    WRITELN;
    WRITELN;
    WRITE (' ':5,'REQUEST    CORRELATION     CORRELATION       ');
    WRITELN ('SPEARMANS        SPEARMANS',' ':9,'COST',' ':12,'COST');
    WRITE (' ':7,'NO.',' ':10,'MTF',' ':13,'TR',' ':14);
    WRITELN ('MTF',' ':13,'TR',' ':14,'MTF',' ':14,'TR');
    WRITE (' ':5,'-------    ------------    ------------     ');
    WRITE ('------------    ------------    ------------');
    WRITELN ('    ------------');
    WRITELN;
    WRITELN;
(*                                                              *)
(*            CALCULATE MEANS AND STANDARD DEVIATIONS           *)
(*                                                              *)
    DDD:=REP*(REP-1);
    FOR X:=0 TO NUMBER_REQ DO
     IF (X<=25) OR (X MOD 25 =0) THEN
      BEGIN
        AVE_CR_MTF(.X.):=CORRSMTF(.X.)/REP;
        AVE_CR_TR(.X.):=CORRSTR(.X.)/REP;
        AVE_S_MTF(.X.):=SPEARSMTF(.X.)/REP;
        AVE_S_TR(.X.):=SPEARSTR(.X.)/REP;
        AVE_CS_MTF(.X.):=COSTSMTF(.X.)/REP;
        AVE_CS_TR(.X.):=COSTSTR(.X.)/REP;
        SD_CR_MTF(.X.):=(REP*CORRSQMTF(.X.)- SQR(CORRSMTF(.X.)))/DDD;
        SD_CR_TR(.X.):=(REP*CORRSQTR(.X.) - SQR(CORRSTR(.X.)))/DDD;
        SD_S_MTF(.X.):=(REP*SPEARSQMTF(.X.) - SQR(SPEARSMTF(.X.)))/DDD;
        SD_S_TR(.X.):=(REP*SPEARSQTR(.X.) - SQR(SPEARSTR(.X.)))/DDD;
        SD_CS_MTF(.X.):=(REP*COSTSQMTF(.X.) - SQR(COSTSMTF(.X.)))/DDD;
        SD_CS_TR(.X.):=(REP*COSTSQTR(.X.) - SQR(COSTSTR(.X.)))/DDD;
(*                                                              *)
(*               WRITE MEANS AND STANDARD DEVIATIONS            *)
(*
        WRITE (' ':6,X:3,' ':6,AVE_CR_MTF(.X.):8:5,' ':8);
        WRITE (AVE_CR_TR(.X.):8:5,' ':8,AVE_S_MTF(.X.):8:5,' ':8);
        WRITE (AVE_S_TR(.X.):8:5);
        WRITELN (' ':8,AVE_CS_MTF(.X.):8:4,' ':8,AVE_CS_TR(.X.):8:4);
        WRITE (' ':20,SD_CR_MTF(.X.):8:4,' ':8);
        WRITE (SD_CR_TR(.X.):8:4,' ':8);
        WRITE (SD_S_MTF(.X.):8:4,' ':8,SD_S_TR(.X.):8:4,' ':8);
        WRITELN (SD_CS_MTF(.X.):8:4,' ':8,SD_CS_TR(.X.):8:4)
      END
END.
//GO.SYSIN DD *
/*
```

APPENDIX B. TESTS OF THE RANDOM NUMBER GENERATOR

A subroutine in Pascal was used to generate random numbers for the simulation. Three tests were used to examine the 7500 values generated. These were:

I. Runs Test - tests for randomness

II. Kolgomorov-Smirnov Test - tests for uniformity between 0 and 1

III. Autocorrelations - tests for independence

The tests were replicated with three different seed values: 8191, 37249 and 65521.

I. Runs Test

A run is a group of consecutive numbers that are either all above or all below 0.5. This test is based on the statistic

$$Z_c = \frac{R - \mu_R}{\sigma_R} \qquad \text{where}$$

R = the number of runs

$$\mu_R = \frac{2n_1 n_2}{n_1 + n_2} + 1$$

$$\sigma_R = \frac{2n_1 n_2 (2n_1 n_2 - n_1 - n_2)}{(n_1 + n_2)(n_1 + n_2 - 1)}$$

and Z_c is normally distributed.

A Pascal program was written to perform the test. The results were:

seed value	Z
8191	-1.6374
37249	-0.4798
65521	1.6209

For a significance level of 0.10 we find that $Z_{\alpha/2} = 1.645$, so in all cases there is not enough evidence to conclude there is a pattern in the data.

II. Kolmogorov-Smirnov Test

This is a test for goodness of fit. This test is performed by the WHITETEST option in PROC SPECTRA (SAS). The results were:

seed	d
8191	0.0113
37249	0.0121
65521	0.0124

For a significance level of 0.10 and N=7500, $d_{\alpha/2} = \frac{1.36}{N} = 0.0157$.

So in all cases there is not enough evidence to conclude the data is not uniformly distributed between 0 and 1.

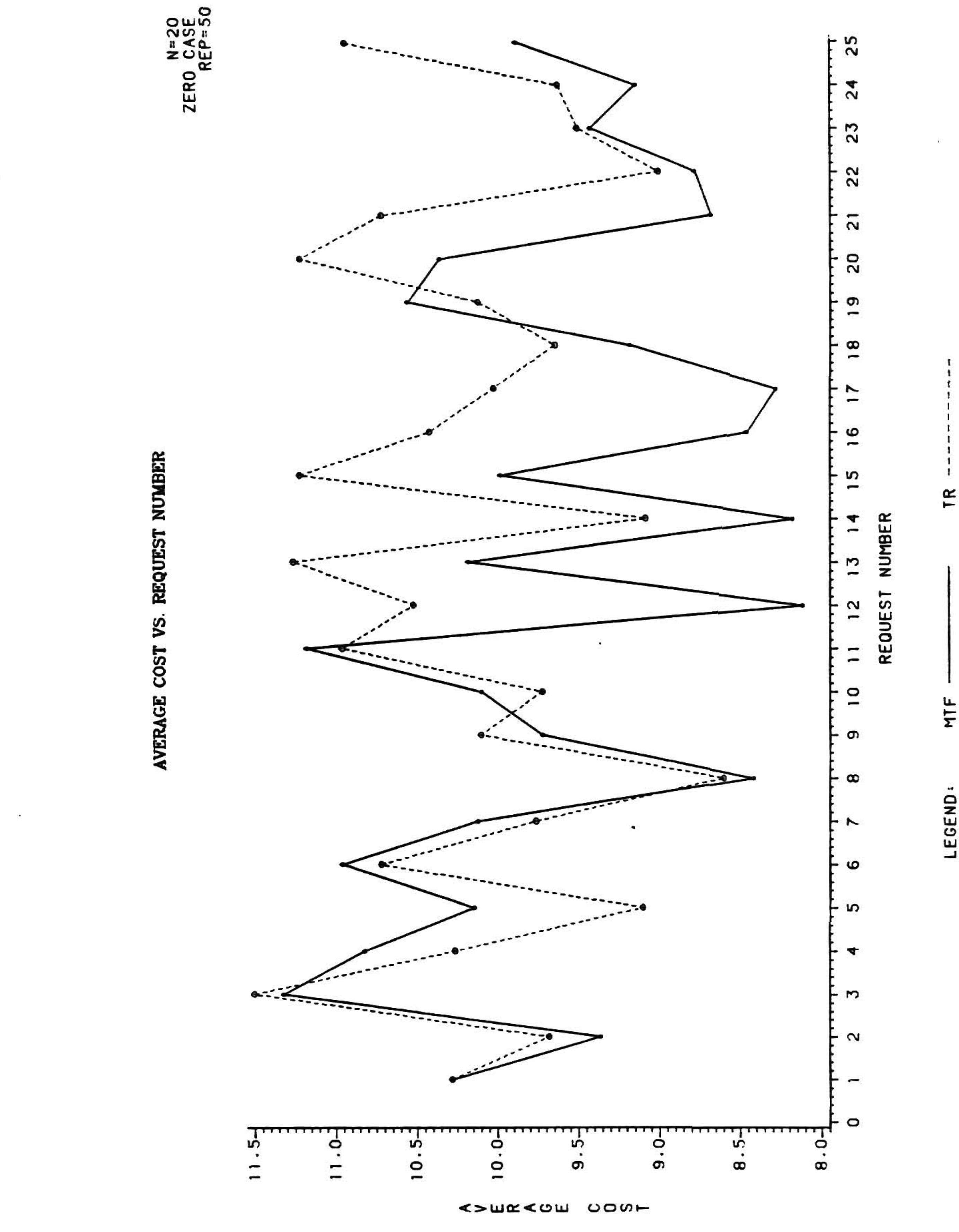
III. Autocorrelations

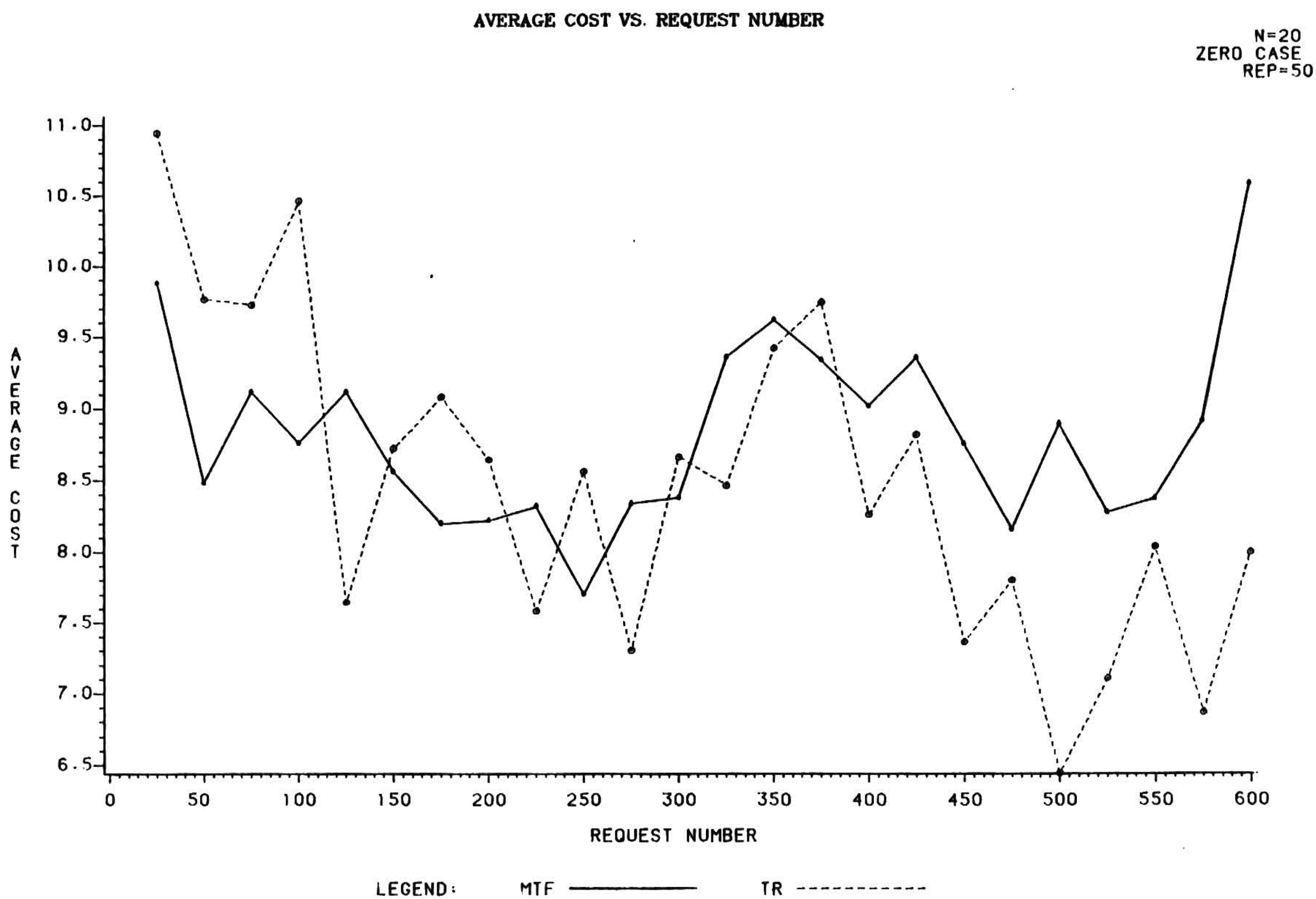
Autocorrelations for a lag of 100 were obtained by PROC AUTOREG (SAS). These autocorrelations were quite small. A t-ratio for the first 99 autoregressive parameters were also calculated . Using the same significance level of 0.10, the t-ratio is compared to $t_{(\alpha/z)}=1.645$ (d.f.=N-1). The results were:

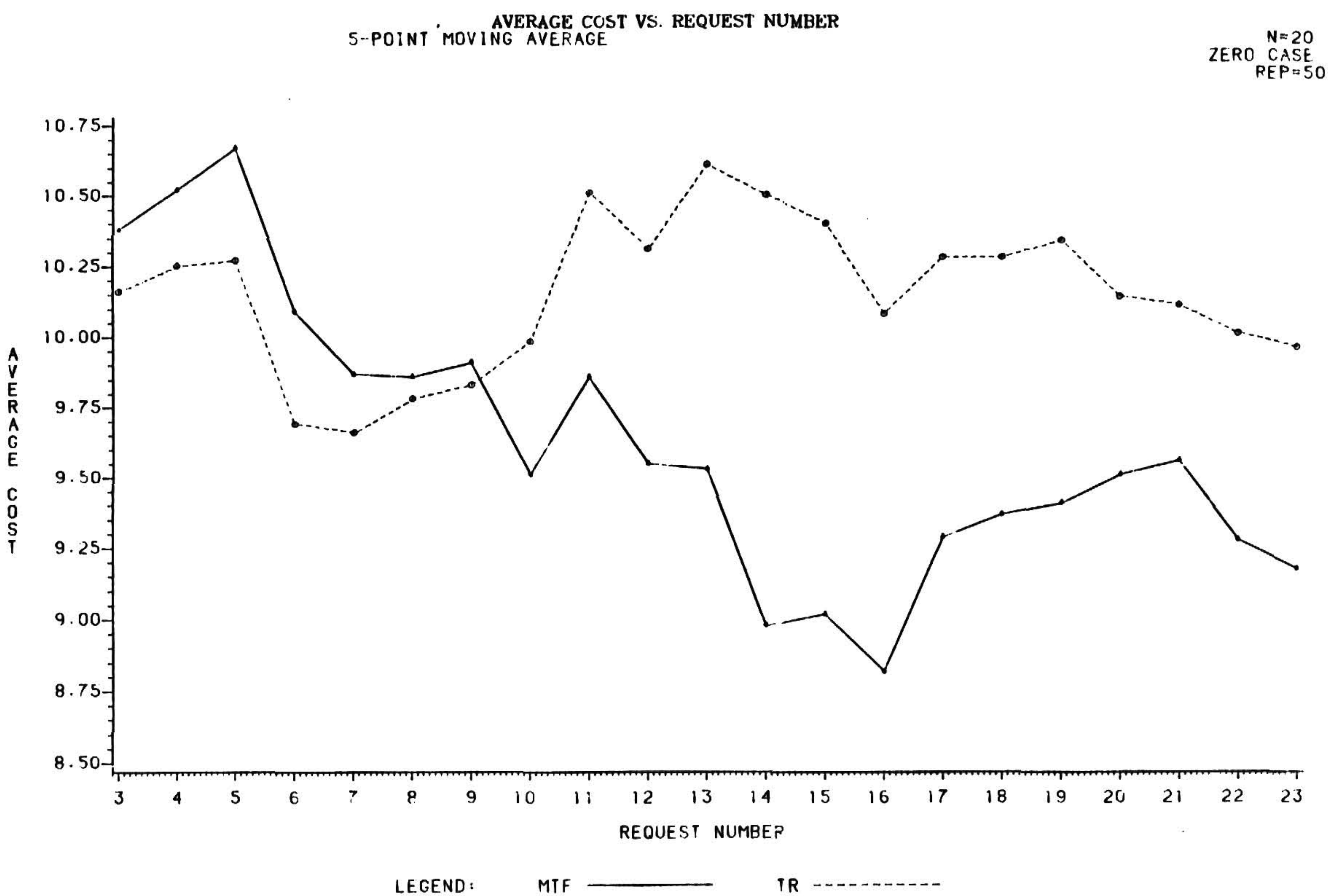
seed	No. of significant ratios
8191	6
37249	8
65521	4

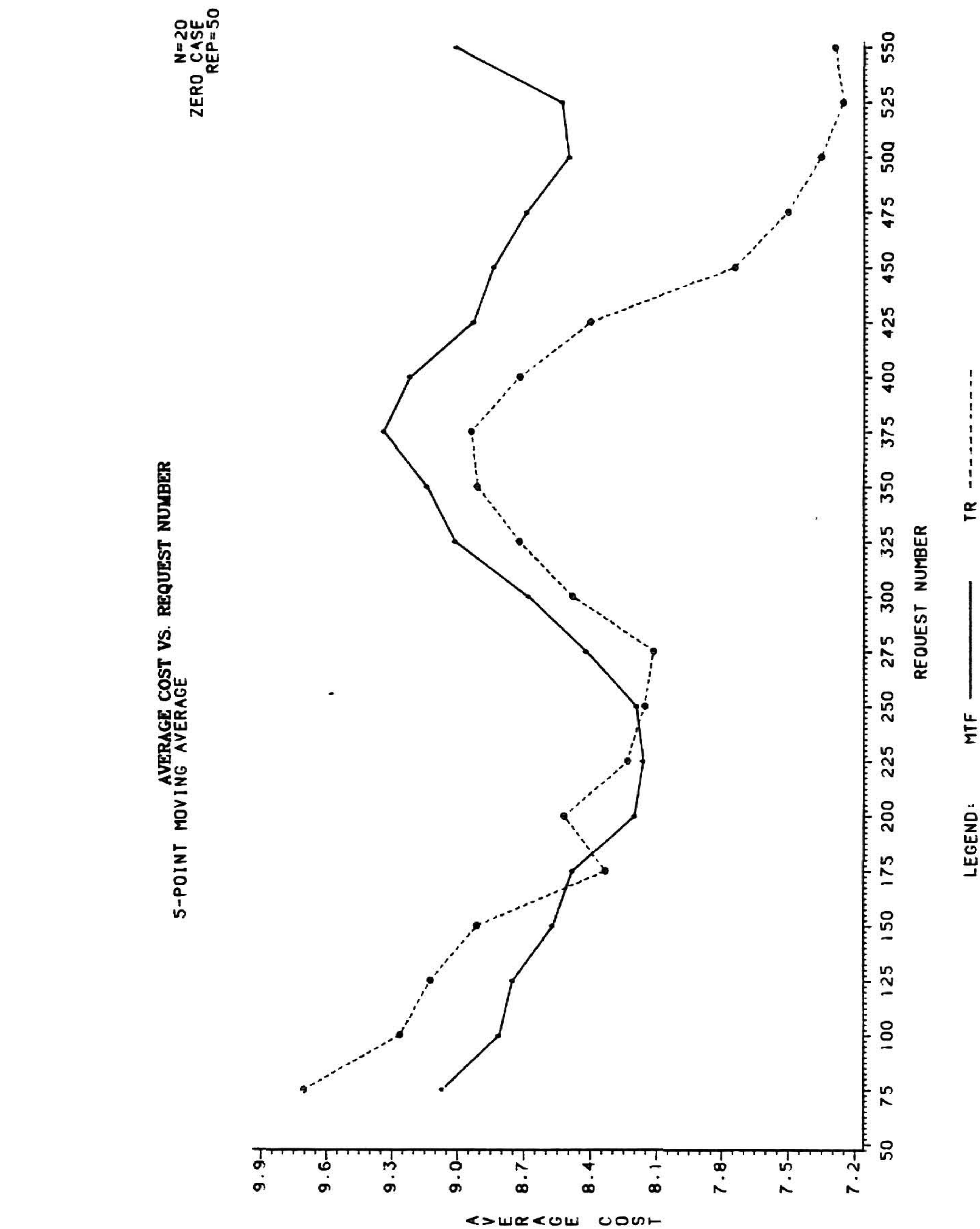
With a lag of 100 a few significant correlations are expected. Thus, the random numbers appear to be independent.

APPENDIX C. GRAPHS OF THE AVERAGE COST AND THE 5-POINT MOVING AVERAGE FOR COST VERSUS REQUEST NUMBER

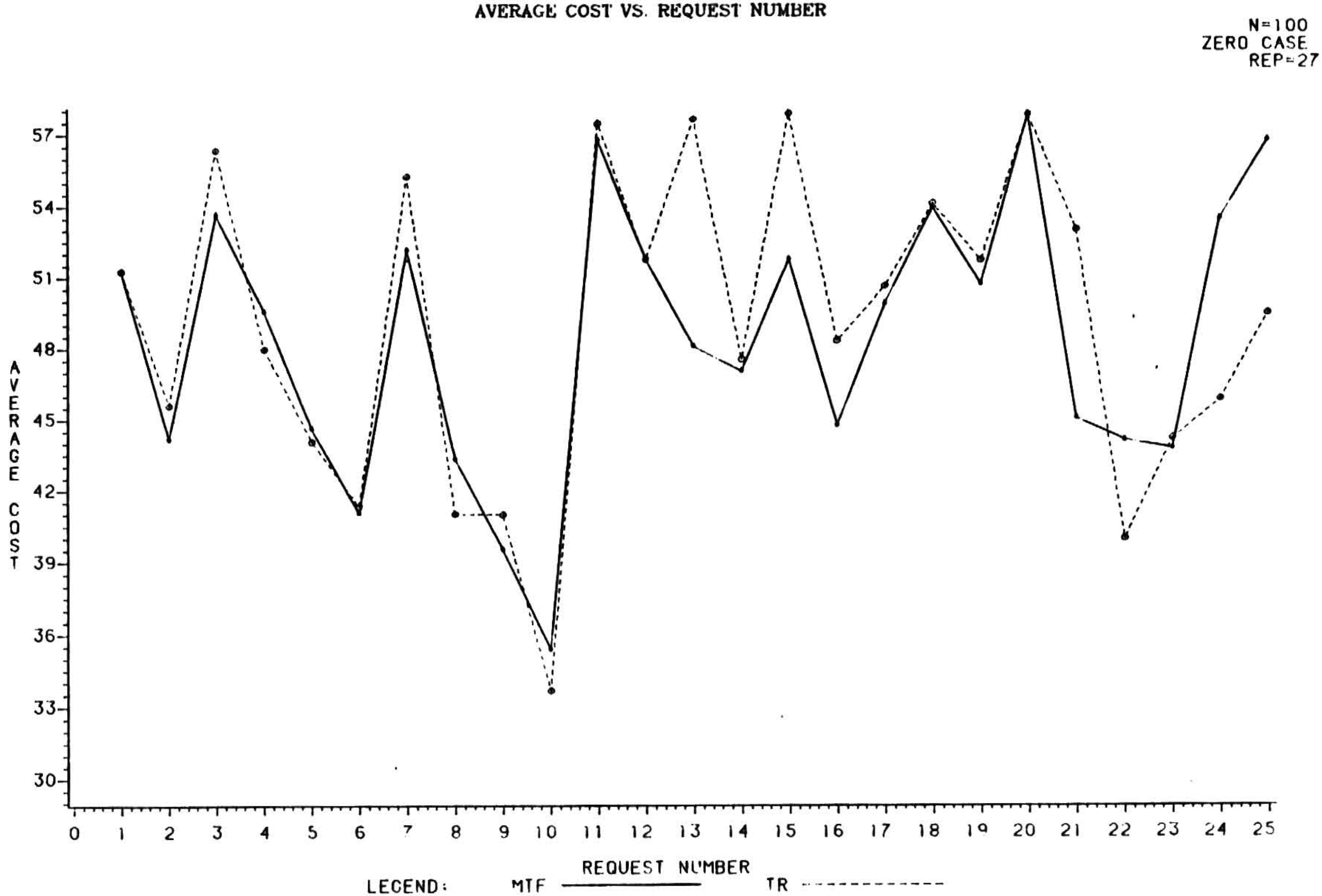

AVERAGE COST VS. REQUEST NUMBER
N=20
ZERO CASE
REP=50
AVERAGE COST
11.5
11.0
10.5
10.0
9.5
9.0
8.5
8.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
REQUEST NUMBER
LEGEND: MTF
TR


AVERAGE COST VS. REQUEST NUMBER
N=20
ZERO CASE
REP=50
AVERAGE COST
11.0
10.5
10.0
9.5
9.0
8.5
8.0
7.5
7.0
6.5
0
50
100
150
200
250
300
350
400
450
500
550
600
REQUEST NUMBER
LEGEND: MTF
TR

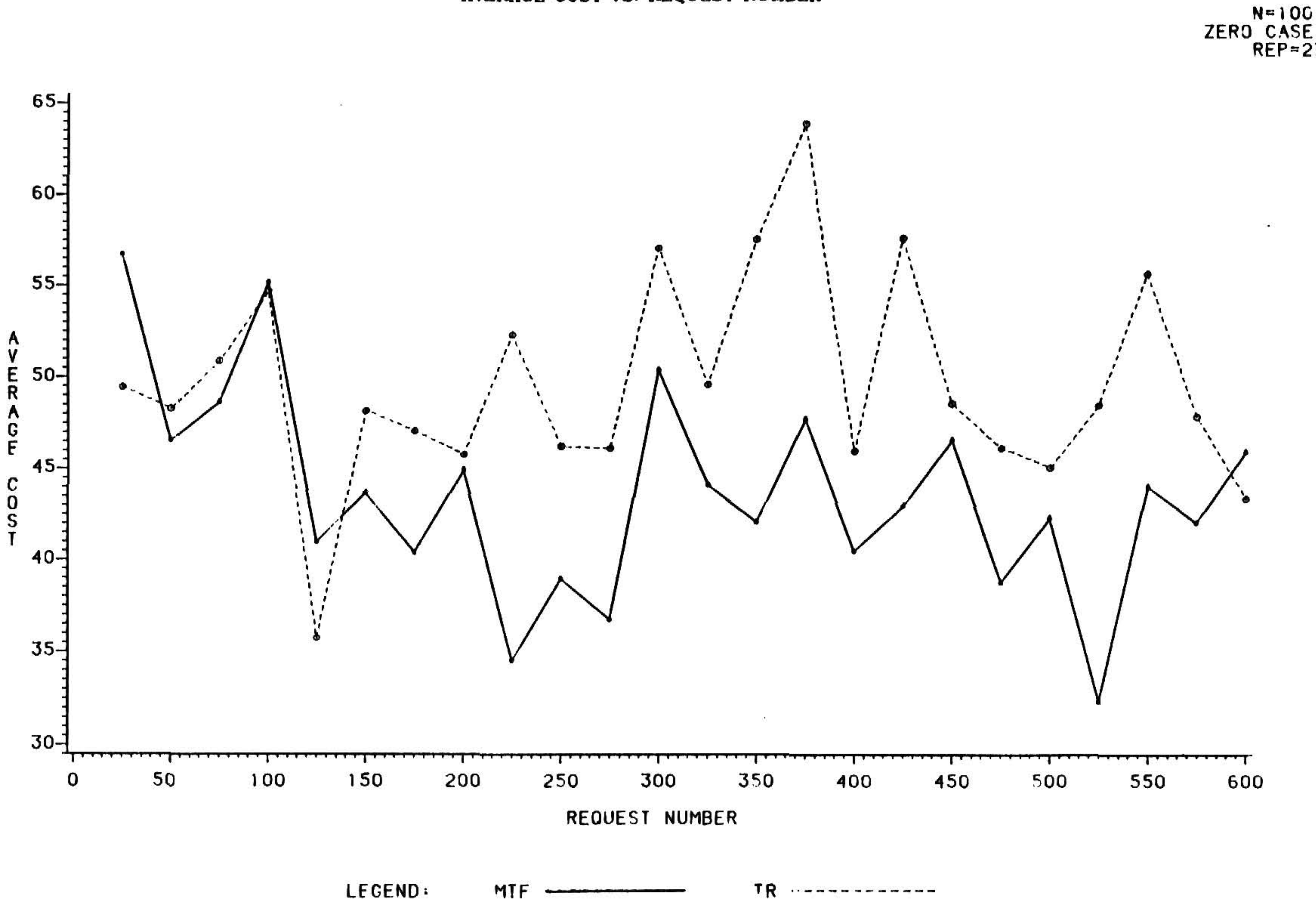

AVERAGE COST VS. REQUEST NUMBER
5-POINT MOVING AVERAGE
N=20
ZERO CASE
REP=50
AVERAGE COST
10.75
10.50
10.25
10.00
9.75
9.50
9.25
9.00
8.75
8.50
3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
REQUEST NUMBER
LEGEND: MTF TR


AVERAGE COST VS. REQUEST NUMBER
5-POINT MOVING AVERAGE
N=20
ZERO CASE
REP=50
AVERAGE COST
9.9
9.6
9.3
9.0
8.7
8.4
8.1
7.8
7.5
7.2
50
75
100
125
150
175
200
225
250
275
300
325
350
375
400
425
450
475
500
525
550
REQUEST NUMBER
LEGEND: MTF
TR

AVERAGE COST VS. REQUEST NUMBER

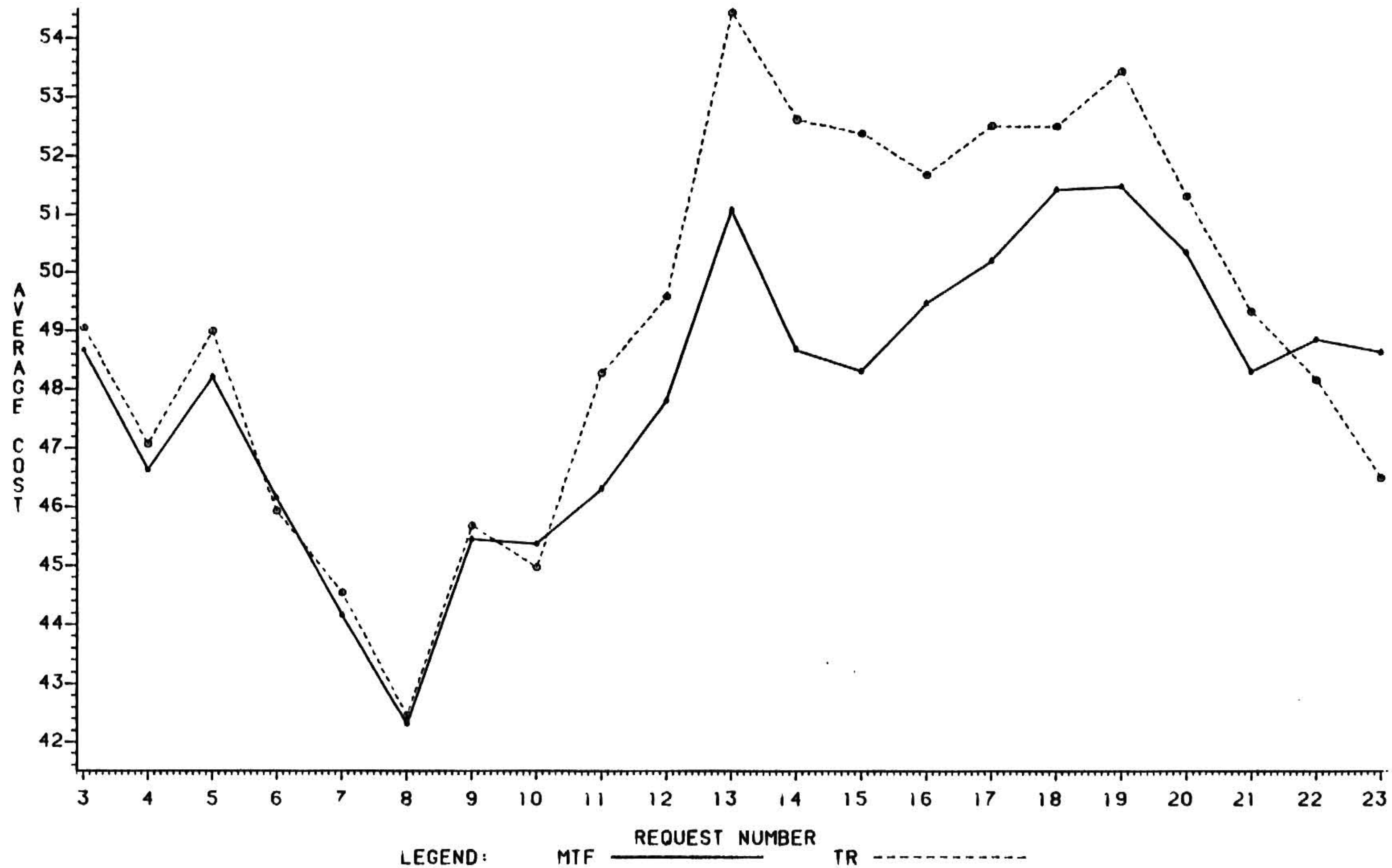


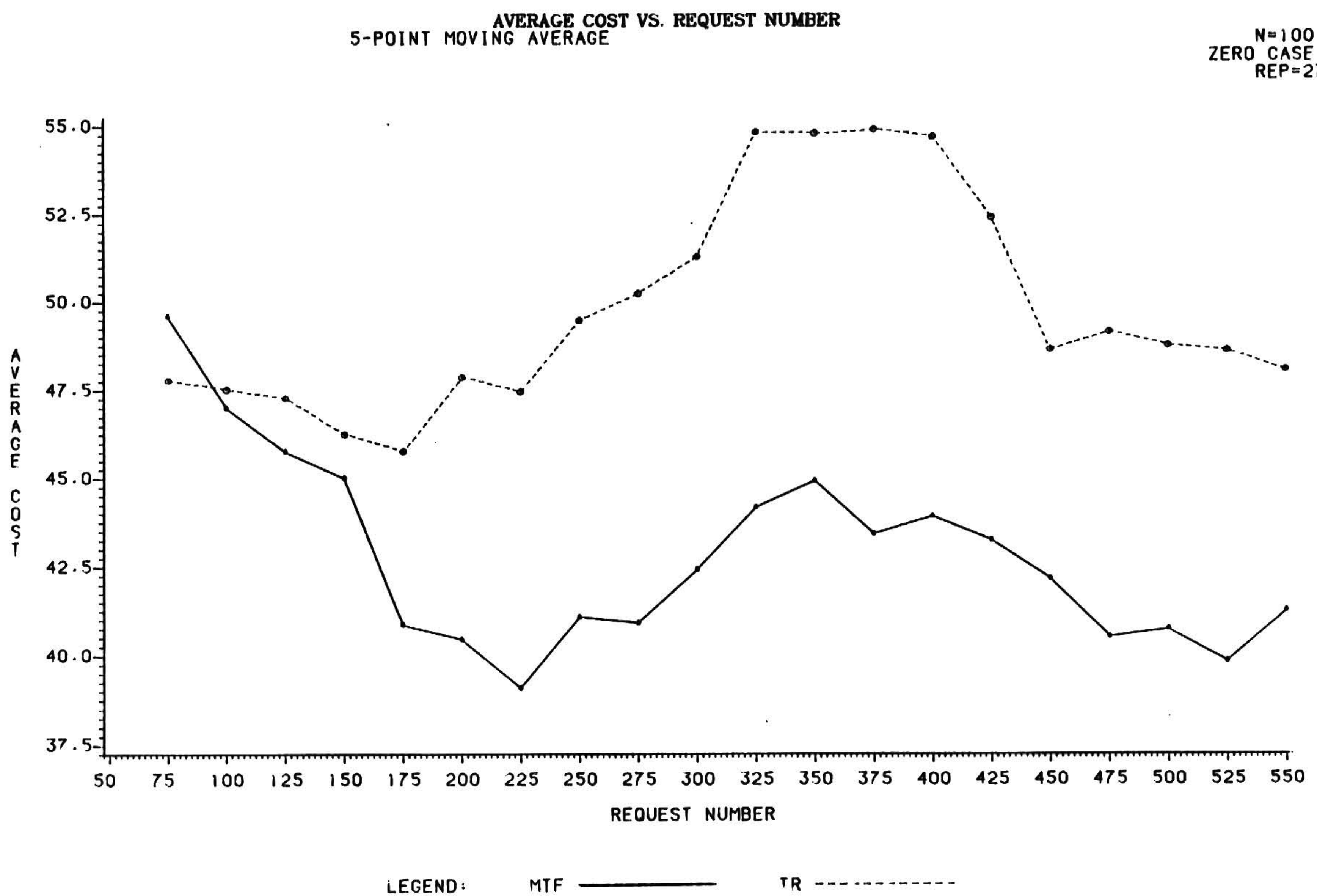
AVERAGE COST VS. REQUEST NUMBER

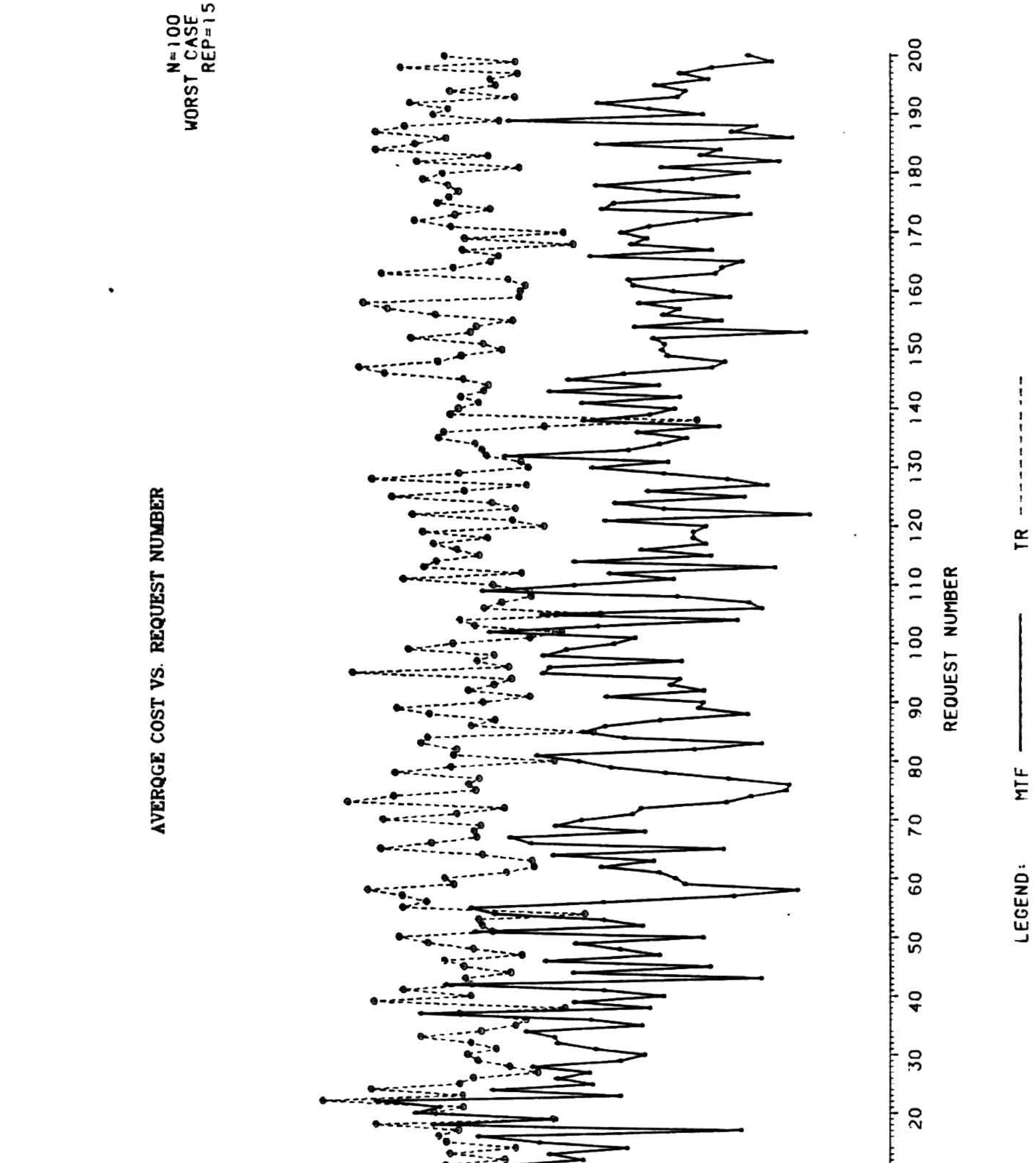


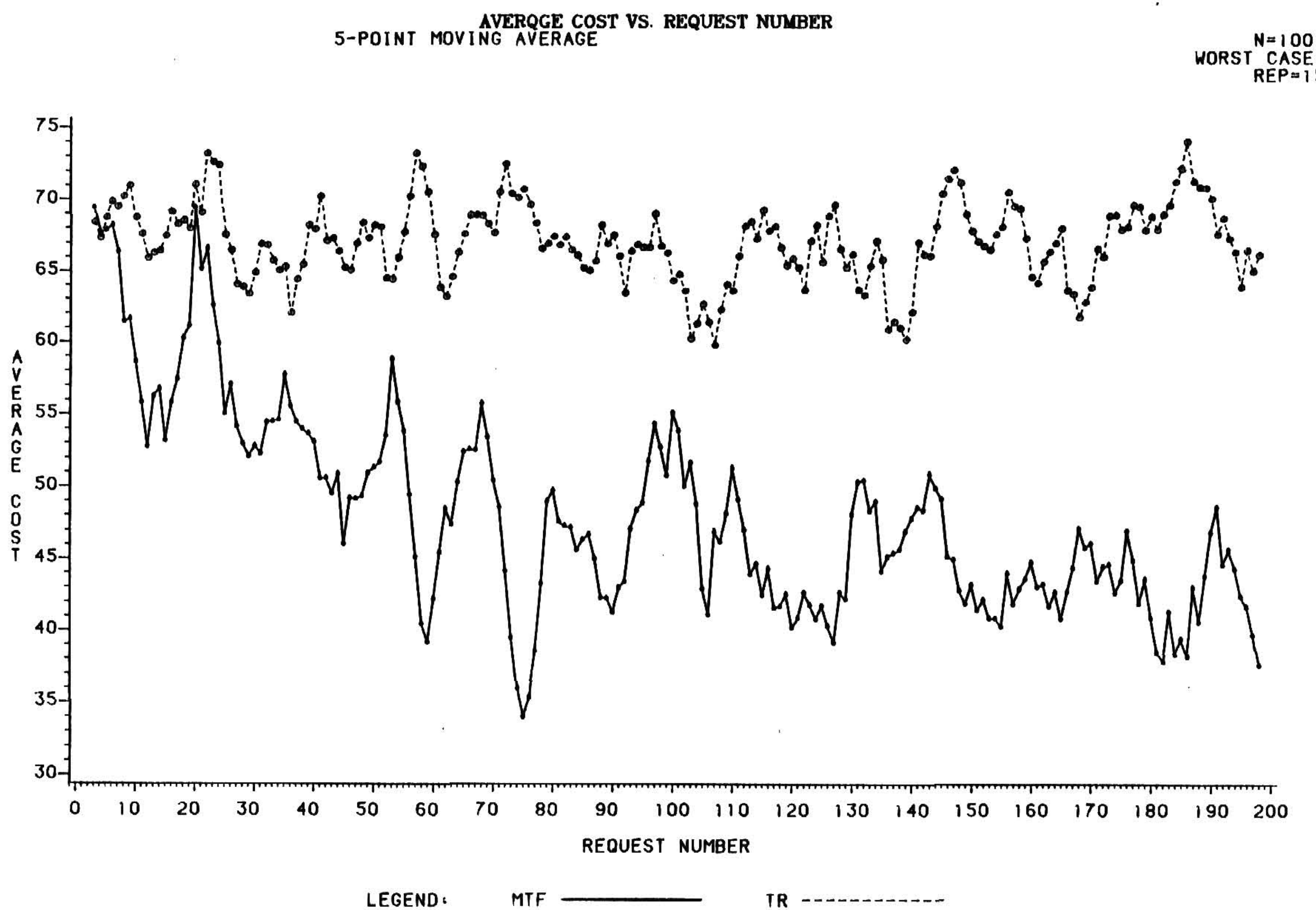
AVERAGE COST VS. REQUEST NUMBER
5-POINT MOVING AVERAGE

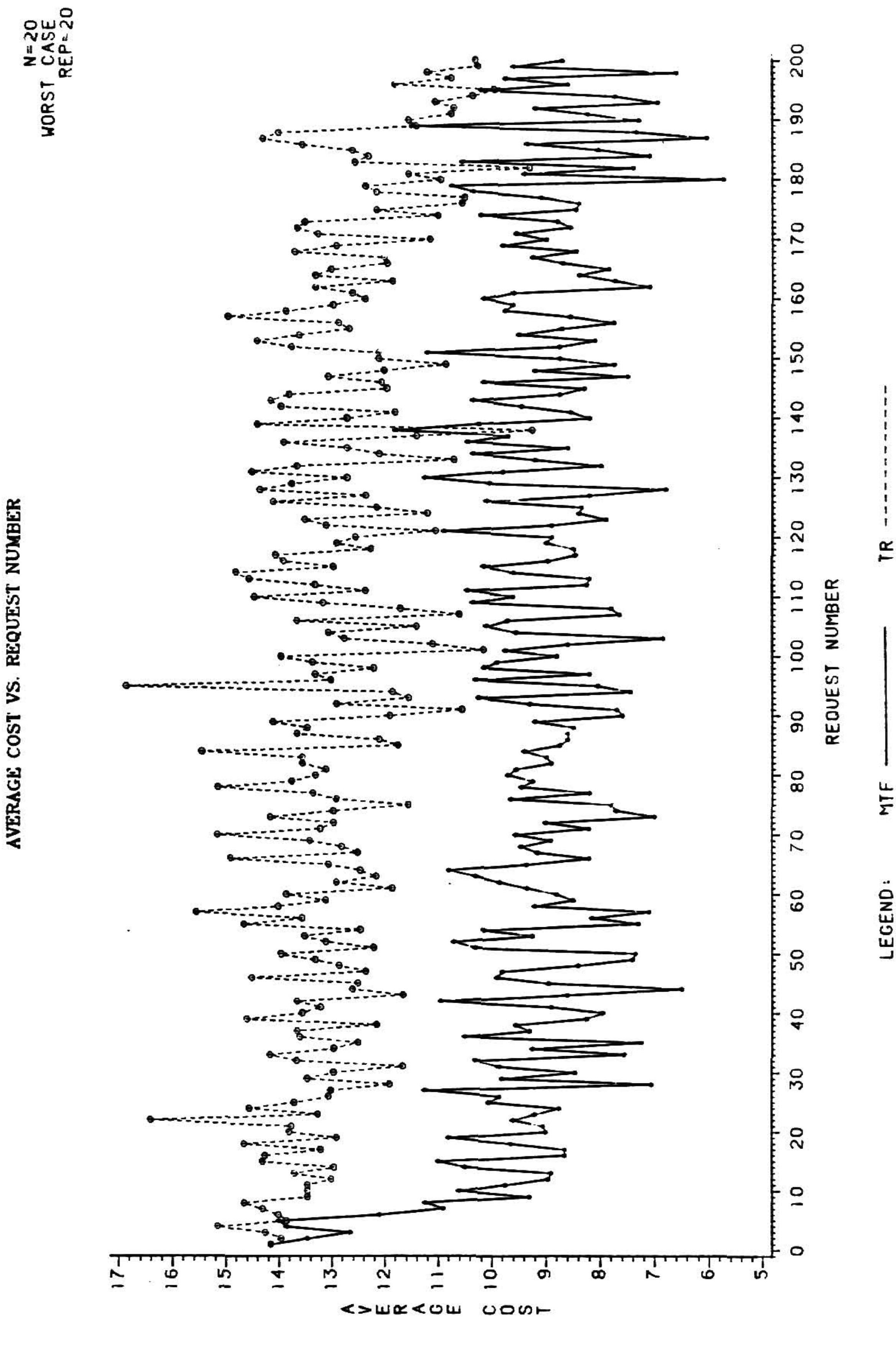
N=100
ZERO CASE
REP=27

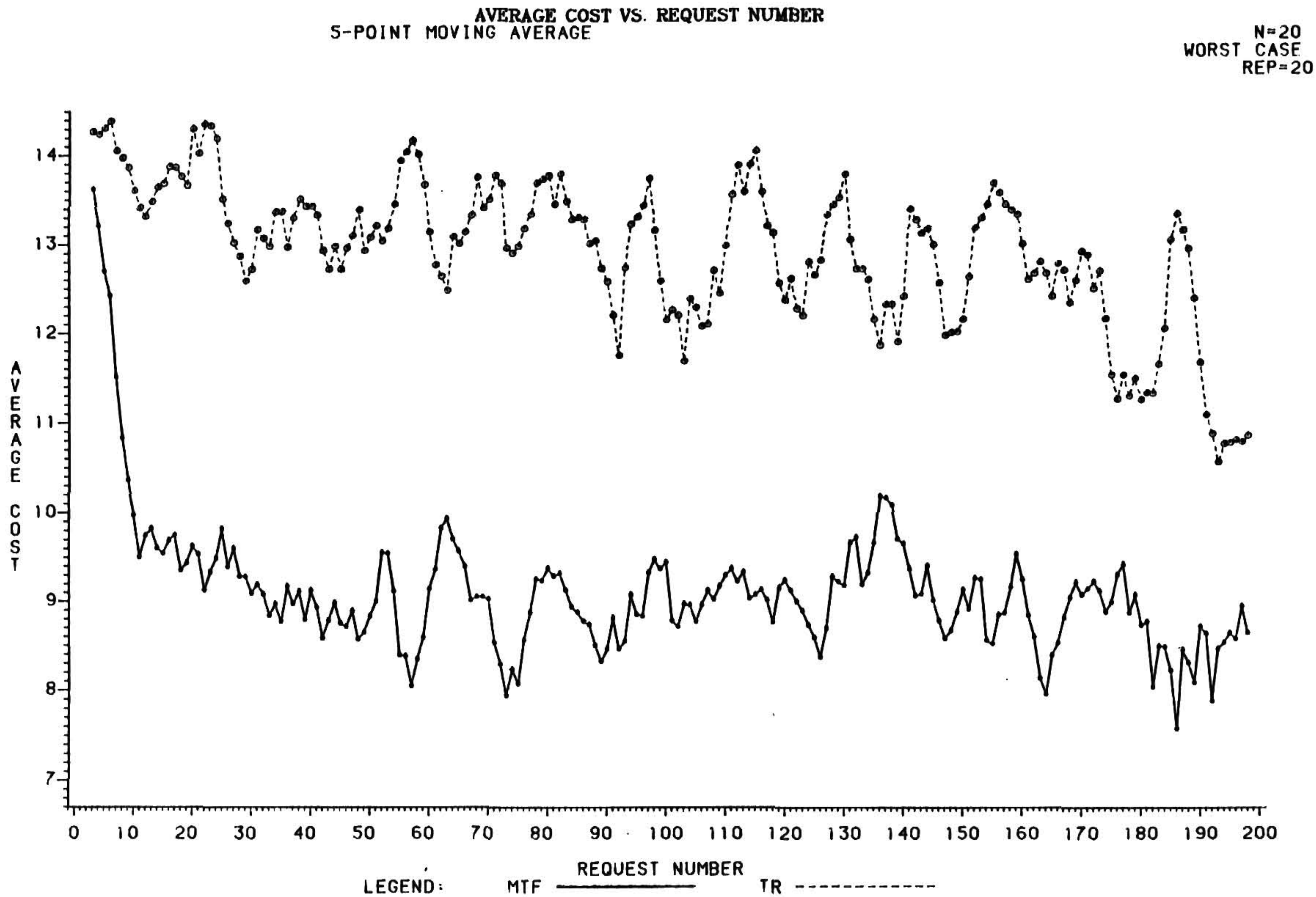



AVERAGE COST VS. REQUEST NUMBER
5-POINT MOVING AVERAGE
N=100
ZERO CASE
REP=27
AVERAGE COST
55.0
52.5
50.0
47.5
45.0
42.5
40.0
37.5
50 75 100 125 150 175 200 225 250 275 300 325 350 375 400 425 450 475 500 525 550
REQUEST NUMBER
LEGEND: MTF
TR


AVERQGE COST VS. REQUEST NUMBER
N=100
WORST CASE
REP=15
AVERAGE COST
REQUEST NUMBER
LEGEND: MTF
TR
20
30
40
50
60
70
80
90
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200


AVERQGE COST VS. REQUEST NUMBER
5-POINT MOVING AVERAGE
N=100
WORST CASE
REP=15
AVERAGE COST
75
70
65
60
55
50
45
40
35
30
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
REQUEST NUMBER
LEGEND:
MTF
TR


AVERAGE COST VS. REQUEST NUMBER
N=20
WORST CASE
REP=20
AVERAGE COST
REQUEST NUMBER
LEGEND: MTF ———— TR - - - - - - -


AVERAGE COST VS. REQUEST NUMBER
5-POINT MOVING AVERAGE
N=20
WORST CASE
REP=20
AVERAGE COST
14
13
12
11
10
9
8
7
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200
REQUEST NUMBER
LEGEND:
MTF
TR

SELF-ORGANIZING SEQUENTIAL SEARCH PROCEDURES

by

NANCY KAY SUNDHEIM

B.S., Kansas State University, 1980
B.S., Kansas State University, 1982

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Statistics

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

A file is a set of records arranged sequentially. If each record in a file, starting with the first one, is searched until the requested record is reached, the file is called a sequential search file.

This paper reviews the literature on self-organizing sequential search procedures. First the permutation schemes and their results are described. Permutation schemes are often desirable because they require no extra storage space. The most popular ones are the Move to Front (MTF) and the Transposition (TR). Analytical results indicate that the TR has a lower asymptotic searching cost, but that the MTF converges to its asymptotic searching cost much quicker. The counter schemes and their results are then described. Many researchers feel that if any extra storage is used then a non-sequential searching technique should be employed. So the counter schemes suggested in the literature are considered by many to be impractical. All analytical results deal with the asymptotic case. The mathematical model and the proofs of two of the major results are presented.

There are no analytical results for the small sample case. Before looking at this case there is a need for a measure of how close the ordering of the records in a file is to the optimal ordering. Three different approaches to measuring this closeness are discussed. A simulation was run using the MTF and the TR schemes. The results indicated that the TR performs better in the long run. However, the MTF is initially a much better scheme. The results also suggest that this initial period is relatively long. This enforces the belief of several researchers that the MTF should be very seriously considered when a sequential search procedure is desired.