AD-784 894

SCHEDULING INDEPENDENT TASKS ON NON-IDENTICAL PARALLEL MACHINES TO MINIMIZE MEAN FLOW-TIME

Douglas Clark

Carnegie-Mellon University

Prepared for:

Defense Advanced Research Projects Agency
Air Force Office of Scientific Research

June 1974

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

1

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFOSR - TR - 74 - 1352	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: AD-784 894	
4. TITLE (and Subtitle): SCHEDULING INDEPENDENT TASKS ON NON-IDENTICAL PARALLEL MACHINES TO MINIMIZE MEAN FLOW-TIME	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Douglas Clark	8. CONTRACT OR GRANT NUMBER(s): F44620-73-C-0074
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61101D A02466
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd, Arlington, VA 22209	12. REPORT DATE: June, 1974
	13. NUMBER OF PAGES: 29
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Air Force Office of Scientific Research / NM, 1400 Wilson Blvd, Arlington, VA 22209	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A collection of tasks having known processing time requirements on a set of non-identical parallel machines is to be scheduled so that the mean flow-time of the tasks is as small as possible. In this paper it is shown that a trivial extension of a simple algorithm for a restricted case performs well, and often optimally, in the general case. A principal result is that for every problem, some renumbering of the tasks will cause this algorithm to produce an optimal schedule. Upper bounds on the worst-case performance of the algorithm are given, and average performance is explored using Monte Carlo techniques.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) ii

SCHEDULING INDEPENDENT TASKS ON NON-IDENTICAL PARALLEL MACHINES TO MINIMIZE MEAN FLOW-TIME

Douglas Clark

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

June 1974

This work was supported in part by a grant from the Xerox Corporation Palo Alto Research Center and in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense, Contract Number F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research. This document has been approved for public release and sale; its distribution is unlimited.

ABSTRACT

A collection of tasks having known processing time requirements on a set of non-identical parallel machines is to be scheduled so that the mean flow-time of the tasks is as small as possible. In this paper it is shown that a trivial extension of a simple algorithm for a restricted case performs well, and often optimally, in the general case. A principal result is that for every problem, some renumbering of the tasks will cause this algorithm to produce an optimal schedule. Upper bounds on the worst-case performance of the algorithm are given, and average performance is explored using Monte Carlo techniques.

1. INTRODUCTION

The problem addressed in this paper is the sequencing of n independent tasks on m parallel and non-identical machines so that the average flow-time of the tasks is as small as possible [5]. We will assume that all n tasks or jobs are simultaneously available at time zero; that there are no feasibility or precedence constraints among the tasks; that tasks may not be preempted; that a machine can process only one job at a time; and that the processing time required by a job i on a machine j is given by a positive number p_{ij}. The inability of some machine to process some task may be represented by making the corresponding p_{ij} prohibitively large.

Fig. 1(a) is a processing time array P for an eight job, three machine problem. A schedule for this problem is shown in Fig. 1(b) in the form of a Gantt chart [4], which illustrates the parallel activity of the three machines along a horizontal time axis. The rectangular blocks in the chart have lengths equal to the processing times of the jobs with whose numbers they are labeled. The flow-time or time-in-system of a job in a particular schedule is simply the time at which that job completes its execution, where the schedule begins at time 0. Thus, in Fig. 1(b) the flow-time of job 3 is 2, of job 4 is 10, of job 6 is 12, and so on.

Borrowing some notation from [5], we will denote by j[i] the number of the job scheduled <u>ith</u> on machine j. In Fig. 1(b), 1[1] is 5, 1[2] is 4, 1[3] is 2, and so on. Let f_i be the flow-time of job i in a particular schedule, let F be the sum of the flow-times of all n jobs, and let n_j be the number of jobs scheduled on machine j. It is clear that

$$
\begin{aligned}
f_{j[1]} &= p_{j[1],j} \\
f_{j[2]} &= p_{j[1],j} + p_{j[2],j} \\
&\vdots \\
f_{j[n_j]} &= p_{j[1],j} + p_{j[2],j} + \dots + p_{j[n_j],j},
\end{aligned} \qquad (1)
$$

and in general,

$$f_{j[i]} = \sum_{k=1}^{i} p_{j[k],j}.$$

In Fig. 1(b), for example, we have

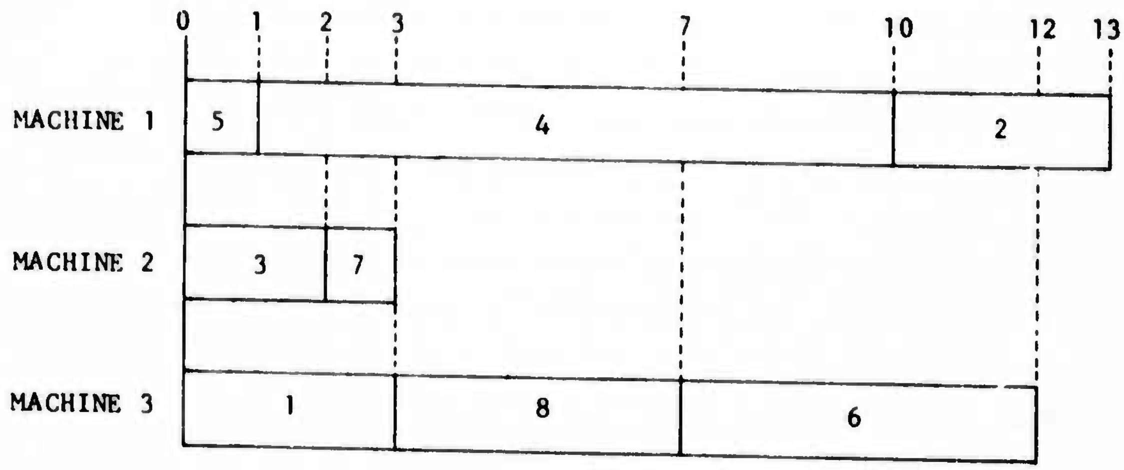
$$
\begin{aligned}
f_{1[1]} &= p_{1[1],1} &&= 1 \\
f_{1[2]} &= p_{1[1],1} + p_{1[2],1} &&= 1 + 9 \\
f_{1[3]} &= p_{1[1],1} + p_{1[2],1} + p_{1[3],1} &&= 1 + 9 + 3.
\end{aligned}
$$

(a)

P:

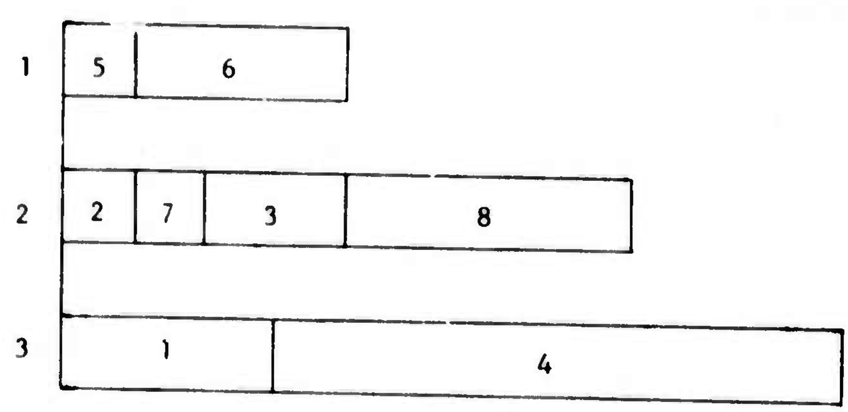
	1	2	3
1	7	4	3
2	3	1	2
3	3	2	2
4	9	9	8
5	1	1	2
6	3	6	5
7	5	1	4
8	5	4	4

(b)



F = 51

(c)



F = 34

Figure 1

The total flow-time of a schedule, F, may be expressed as a sum of contributions from each machine. Collecting terms in equations (1), we may express the contribution of machine j as

$$\sum_{i=1}^{n_j} f_{j[i]} = \sum_{i=1}^{n_j} \sum_{k=1}^{i} p_{j[k],j}$$

$$= n_j p_{j[1],j} + (n_j-1)p_{j[2],j} + \dots + 2p_{j[n_j-1],j} + p_{j[n_j],j}.$$

Summing over all machines, we get

$$F = \sum_{j=1}^{m} \sum_{i=1}^{n_j} f_{j[i]} = \sum_{j=1}^{m} \sum_{i=1}^{n_j} \sum_{k=1}^{i} p_{j[k],j}$$

$$\begin{aligned} &= n_1 p_{1[1],1} + (n_1-1)p_{1[2],1} + \dots + 2p_{1[n_1-1],1} + p_{1[n_1],1} \\ &+ n_2 p_{2[1],2} + (n_2-1)p_{2[2],2} + \dots + p_{2[n_2],2} + \\ &\vdots \\ &+ n_m p_{m[1],m} + (n_m-1)p_{m[2],m} + \dots + p_{m[n_m],m}. \end{aligned} \tag{2}$$

For example, in Fig. 1(b) we have

$$\begin{aligned} F &= 3\cdot 1 + 2\cdot 9 + 1\cdot 3 \\ &+ 2\cdot 2 + 1\cdot 1 \\ &+ 3\cdot 3 + 2\cdot 4 + 1\cdot 5 \\ &= 51. \end{aligned}$$

Our goal is to minimize mean flow-time, $\frac{1}{n}F$, but minimizing F itself is equivalent, and we will follow that approach in the rest of this paper. Fig. 1(c) gives an optimal schedule (optimal schedules are not necessarily unique) for the problem of Fig. 1(a), with F = 34.

A non-enumerative algorithm for minimizing mean flow-time in the general case was discovered by Bruno, Coffman, and Sethi [2,3]. Their algorithm is based on a reduction of the problem to a minimum-cost network flow problem, and the time required by the algorithm is $O(n^3)$ when $n \geq m$ (the case of interest) and $O(n^2m)$ when $n < m$. This paper proposes and analyzes an algorithm which finds schedules that are good, and frequently optimal, with respect to mean flow-time, and does so at very small computational cost.

Section 2 of this paper reviews an easy algorithm for an important restriction of the general problem. In Section 3 this algorithm is extended to cover the general case, and it is shown that while an optimal schedule is not always produced, the performance of the algorithm strongly depends on the ordering of the rows of the processing time array P. Section 4 examines analytically the worst-case performance of the algorithm under various ordering rules, and the algorithm's average performance under these rules is explored empirically in Section 5. Section 6 contains the conclusions of the paper.

2. MACHINE FACTOR CASE

An important and realistic restriction of the general problem arises when each p_{ij} is the product of a time associated with job i and an efficiency factor associated with machine j, that is, $p_{ij} = p_i w_j$. In this restricted case, mean flow-time is minimized by a simple procedure which follows immediately from the analysis of [5]. First, rewrite equation (2) as follows:

$$F = \sum_{j=1}^{m} \sum_{i=1}^{n_j} \sum_{k=1}^{i} p_{j[k],j} = \sum_{j=1}^{m} \sum_{i=1}^{n_j} \sum_{k=1}^{i} w_j p_{j[k]}$$

$$= n_1 w_1 p_{1[1]} + (n_1-1) w_1 p_{1[2]} + \dots + 2w_1 p_{1[n_1-1]} + w_1 p_{1[n_1]}$$

$$+ n_2 w_2 p_{2[1]} + (n_1-1) w_2 p_{2[2]} + \dots + w_2 p_{2[n_2]} +$$

$$\vdots$$

$$+ n_m w_m p_{m[1]} + (n_m-1) w_m p_{m[2]} + \dots + w_m p_{m[n_m]}.$$

Second, since F is a sum of n terms, each of which is a product of one of the n p_i's with one of the n coefficients $n_1 w_1, (n_1-1)w_1, \dots, w_1, n_2 w_2, \dots, w_2, \dots\dots, n_m w_m, \dots, w_m$, pick as coefficients the n smallest of the nm possibilities $nw_1, (n-1)w_1, \dots, w_1, nw_2, \dots, w_2, \dots\dots, nw_m, \dots, w_m$. This will determine the values of the n_j. Third, minimize F by matching the largest p_i with the smallest coefficient, the second-largest p_i with the second-smallest coefficient, and so on. If a particular p_i is matched with $(n_j-k)w_j$, then job i is scheduled (k+1)th on machine j. Picking coefficient $(n_j-k)w_j$ implies that we have already picked $(n_j-k-1)w_j, (n_j-k-2)w_j, \dots, 2w_j$, and w_j, so the resulting schedule is well-formed. (That is, it is not possible, for example, to schedule some job fourth on some machine and not schedule some other jobs first, second, and third.)

Each choice of a coefficient can be restricted to be among only m possibilities out of the total of nm. The first chosen will surely be one of $w_1, \dots, w_m$ (the smallest, in fact). Suppose it is w_i. Then the second coefficient will be the smallest of $w_1, \dots, 2w_i, \dots, w_m$. At each stage the integer multiplier of the chosen coefficient is increased by 1, and the next choice made. The schedule is thus being determined "back" to "front".

An example of this procedure is given in Fig. 2(a) for a five job, three machine problem. The jobs happen to be numbered so that $p_1 \geq p_2 \geq \dots \geq p_5$; consequently no sorting of the p_i is necessary. In each row of the table the smallest of the m potential coefficients is chosen (circled), and in the next row the circled coefficient is increased by the corresponding w_j. In row 2 there are two smallest coefficients; the choice between them is arbitrary, since different schedules with the same (optimal) F will result from different choices in the case of a tie. Fig. 2(b) shows the resulting schedule and the calculation of F.

Fig. 2(c) shows the application of an equivalent algorithm to the problem of Fig. 2(a). Here the processing-time array P is explicitly shown, and the coefficients in each row now represent the possible sequence positions on the three machines, counting from the <u>end</u> of the schedule, for the corresponding job. It is important to note that for each column j, $p_{1j} \geq p_{2j} \geq \dots \geq p_{nj}$. Let the coefficients in a particular row i be $h_1, h_2, \dots, h_m$ (in this case, m = 3). Then the minimum $h_j p_{ij}$ is chosen, p_{ij} is circled, and h_j is increased by 1 in the next row.

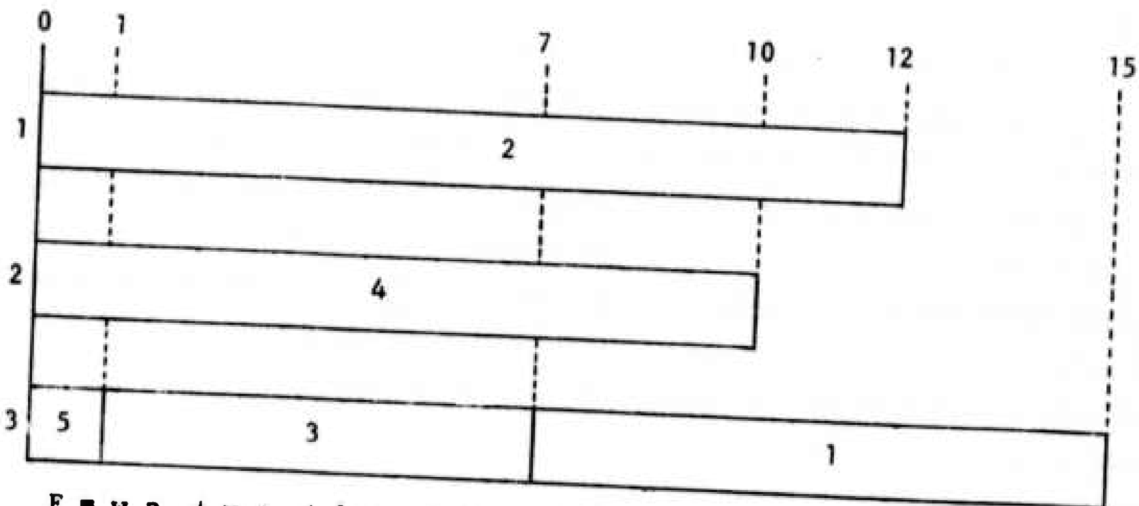
$n = 5$, $m = 3$

machine factors: $w_1 = 2$, $w_2 = 2.5$, $w_3 = 1$

(a)

i	Pi	potential coefficients 1	2	3	machine chosen
1	8	2	2.5	1	3
2	6	2	2.5	2	1
3	6	4	2.5	2	3
4	4	4	2.5	3	2
5	1	4	5	3	3

(b)



$$F = w_1p_1 + w_2p_4 + 3w_3p_5 + 2w_3p_3 + 1w_3p_1$$
$$= 2\cdot 6 + 2{,}5\cdot 4 + 3\cdot 1\cdot 1 + 2\cdot 1\cdot 6 + 1\cdot 1\cdot 8$$
$$= 45$$

(c) P:

	1	2	3	h_1	h_2	h_3
1	16	20	8	1	1	1
2	12	15	6	1	1	2
3	12	15	6	2	1	2
4	8	10	4	2	1	3
5	2	2.5	1	2	2	3

Figure 2

This is exactly equivalent to the algorithm of Fig. 2(a), and the same schedule results. The difference lies in the lack of explicit use of the w_j in the second version of the algorithm, and this difference will be exploited in Section 3.

Sorting the p_i requires computational exertion of $O(n \log n)$, and finding the smallest coefficients requires $O(nm)$, so the time complexity of the machine factor algorithm is $O(\max(n \log n, nm))$.

3. EXTENSION TO THE GENERAL CASE

The second version of the procedure of the preceding section (Fig. 2(c)), since it does not use the machine factors w_j, may be applied in the general case, though with no guarantee that optimal schedules will be found. Fig. 3(a) shows the result of applying this procedure to the problem of Fig. 1(a). Since this problem is outside the machine factor case we cannot sort the jobs according to a single processing time, so in Fig. 3(a) the job numbering of Fig. 1(a) is retained, and the schedule of Fig. 3(b) results. The value of F is not optimal, but the schedule was very quickly computed, so this algorithm will hereafter be called the Quick And Dirty (QAD) algorithm. The coefficients used at a particular stage of the schedule generation will be called the QAD coefficients and labeled $h_1, h_2, \ldots, h_m$. In each row i of P the QAD algorithm chooses the minimum of $h_1p_{i1}, h_2p_{i2}, \ldots, h_mp_{im}$ and increments the chosen h_j in the following row. As in the machine factor case, the choice of h_jp_{ij} means that job i is scheduled (n_j-h_j+1)<u>th</u> (or equivalently, h_j<u>th</u> from the <u>end</u>) on machine j, and that the term h_jp_{ij} will occur in the calculation of F. (This latter fact is very important for the rest of this paper.)

An obvious infirmity of the schedule in Fig. 3(b) is that the jobs are not scheduled in "shortest processing time first" (SPT) order on each machine. SPT order is optimal for the minimization of F on a single machine [5], so an optimal schedule for the more general problem clearly must have jobs in SPT order on each individual machine. This fact suggests that the performance of QAD may be improved by adding to it a procedure which sorts the jobs in SPT order on each machine after the schedule has been produced. Call the combined procedure QAD*. If QAD* is applied to Fig. 3(a), the result is a schedule with an improved F of 35, which, while still not optimal, is considerably better than QAD's F = 48.

Though an optimal schedule was not found in Fig. 3(a), if the rows of P are permuted as in Fig. 3(c), the optimal schedule of Fig. 1(c) will be generated by QAD (and therefore also by QAD*). This might appear at first glance to be no more than a fortunate coincidence; the following theorem, however, states otherwise.

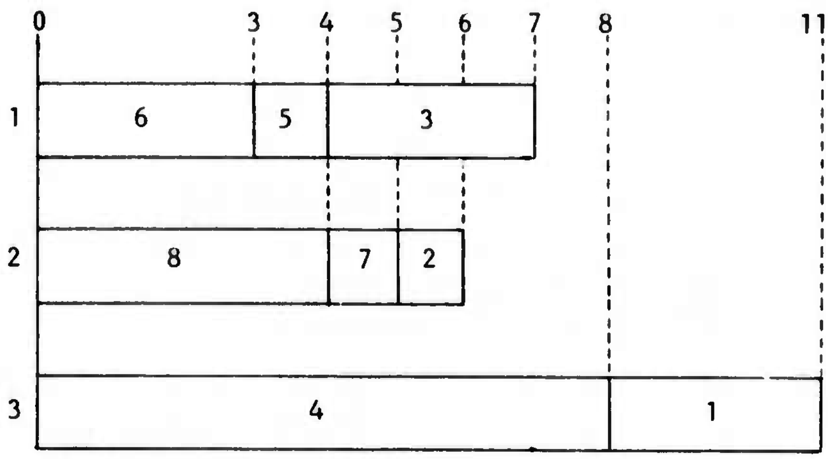
<u>Theorem 3-1</u>. Given an arbitrary processing time array P, there exists a permutation of the rows (a renumbering of the jobs) such that the QAD algorithm, operating on the permuted array, will yield a schedule with optimal (minimal) F.

<u>Proof</u>. Let P be arbitrary. The plan of the proof will be to show that if the first k jobs (jobs 1,2,...,k) of P are scheduled optimally by QAD, then one of the remaining n-k jobs can be renumbered k+1 so that it, too, will be scheduled optimally. If this can be demonstrated for k = 0,1,2,...,n-1, then the theorem will be proved. In this proof the phrase "scheduled optimally" will mean <u>scheduled</u> (by QAD) <u>in agreement with some optimal schedule</u>, both in the assignment of jobs to machines, and in the <u>order</u> in which they are assigned. Recall that QAD assigns

(a)

	1	2	3	h_1	h_2	h_3
1	7	4	3	1	1	1
2	3	1	2	1	1	2
3	3	2	2	1	2	2
4	9	9	8	2	2	2
5	1	1	2	2	2	3
6	3	6	5	3	2	3
7	5	1	4	4	2	3
8	5	4	4	4	3	3

(b)



$$F = 3p_{6,1} + 2p_{5,1} + 1p_{3,1} + 3p_{8,2} + 2p_{7,2} + 1p_{2,2} + 2p_{4,3} + 1p_{1,3}$$

$$= 3\cdot 3 + 2\cdot 1 + 1\cdot 3 + 3\cdot 4 + 2\cdot 1 + 1\cdot 1 + 2\cdot 8 + 1\cdot 3$$

$$= 48$$

(c)

	1	2	3	h_1	h_2	h_3
4	9	9	8	1	1	1
8	5	4	4	1	1	2
6	3	6	5	1	2	2
1	7	4	3	2	2	2
3	3	2	2	2	2	3
7	5	1	4	2	3	3
5	1	1	2	2	4	3
2	3	1	2	3	4	3

Figure 3

jobs from last to first on each machine.

Let S be an optimal schedule for P. Assume that the first k jobs of P are scheduled by QAD in agreement with S, where k may be any integer from 0 to n-1. (There is nothing to prove if k = n.) We will try to find a (k+1)th job such that either:

1. jobs 1 through k+1 will be scheduled by QAD in agreement with S; or
2. there is some other optimal schedule S' such that jobs 1 through k+1 will be scheduled in agreement with it.

If at least one of these two alternatives is always true, the theorem is proved.

Fig. 4 illustrates the situation with an example. The shaded jobs are jobs 1 through k, optimally scheduled by QAD. If the QAD schedule is to agree with S, the only jobs which are "candidates" for job k+1 are those marked by an asterisk in Fig. 4. There is at most one such candidate job on each machine. Let g_j $(j = 1,2,\dots,m)$ be the number of the candidate job on machine j if there is one; otherwise let $g_j = 0$. Let h_j $(j = 1,2,\dots,m)$ be the QAD coefficient for machine j in row k+1 of P.

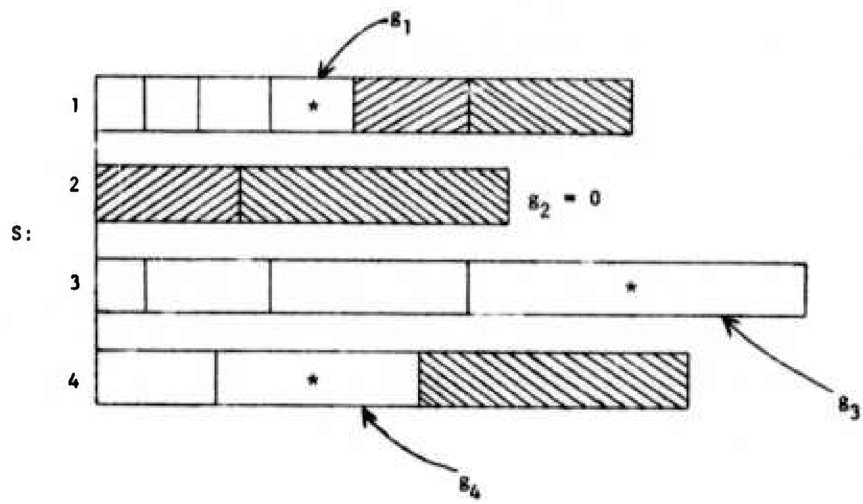


Figure 4

Construct a graph with m vertices $v_1,v_2,\dots,v_m$. For each vertex v_i, draw a directed arc (v_i,v_j) from v_i to v_j if and only if

1. $g_i \neq 0$; and
2. QAD would schedule job g_i on machine j if it became job k+1 of P.

The existence of an arc (v_i,v_j) implies that $h_j p_{g_i,j} \leq h_i p_{g_i,i}$, with equality only if the tie-breaking rule used by the QAD algorithm would choose machine j over machine i for job g_i.

If there is any arc of the form (v_i,v_i), then job g_i can become job k+1, QAD will schedule it according to S, and we are done. Assume, therefore, that there are no such arcs in the graph.

Suppose some vertex v_j has one or more arriving arcs but no departing arc, and let v_i be a predecessor of

v_j (Fig. 5(b)). Construct a new schedule S' which is identical to S except that job g_i is scheduled first on machine j (Fig. 5(a)). If we constructed the graph corresponding to S', we would find Fig. 5(c) in place of Fig. 5(b). If S' is optimal, then job g_i could become the new job k+1 in P and be optimally scheduled.

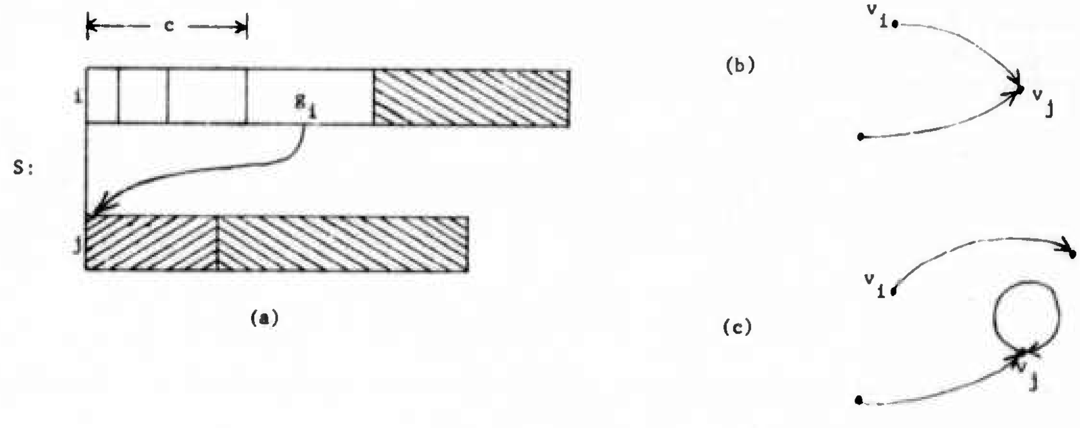


Figure 5

Denote by F^S and $F^{S'}$ the total flow-times of the two schedules. Then we have

$$F^{S'} = F^S + h_j p_{g_i,j} - h_i p_{g_i,i} - c. \tag{3}$$

The non-negative term c occurs in (3) because the removal of job g_i from machine i causes the coefficients of all preceding jobs (if there are any) to be reduced by 1 (see Fig. 5(a)). The existence of arc (v_i,v_j) implies that $h_j p_{g_i,j} \le h_i p_{g_i,i}$, and the optimality of S implies that $F^{S'} \ge F^S$. We conclude from (3), therefore, that $c = 0$, $F^{S'} = F^S$, and S' is optimal. Thus job g_i can become job k+1 of P.

Now suppose that the graph contains no vertices that have arriving arcs but no departing arc. Since there is at least one arc in the graph (because $k < n$), since the number of vertices is finite, and since there are no arcs (v_i,v_i), we conclude that there must exist a cycle in the graph. Suppose the cycle has three arcs, (v_i,v_j), (v_j,v_k), and (v_k,v_i), as shown in Fig. 6(b). (The following argument may easily be generalized to cycles of any size.) Construct a new schedule S' identical to S except that job g_i moves to job g_j's position on machine j, job g_j goes to machine k, and job g_k to machine i (Fig. 6(a)). The graph corresponding to S' would include Fig. 6(c) in place of Fig. 6(b).

With F^S and $F^{S'}$ as before we may write

$$F^{S'} = F^S + (h_j p_{g_i,j} - h_i p_{g_i,i}) + (h_k p_{g_j,k} - h_j p_{g_j,j}) + (h_i p_{g_k,i} - h_k p_{g_k,k}). \tag{4}$$

The existence of the arcs in the cycle means that each parenthesized term in (4) is non-positive, and since $F^{S'} \ge F^S$, we conclude, as before, that S' is optimal. Any one of jobs g_i, g_j, and g_k can become job k+1 in P and be optimally scheduled.

We have therefore shown what we set out to show, namely, that there must always exist, for $k = 0,1,\ldots,n-1$,

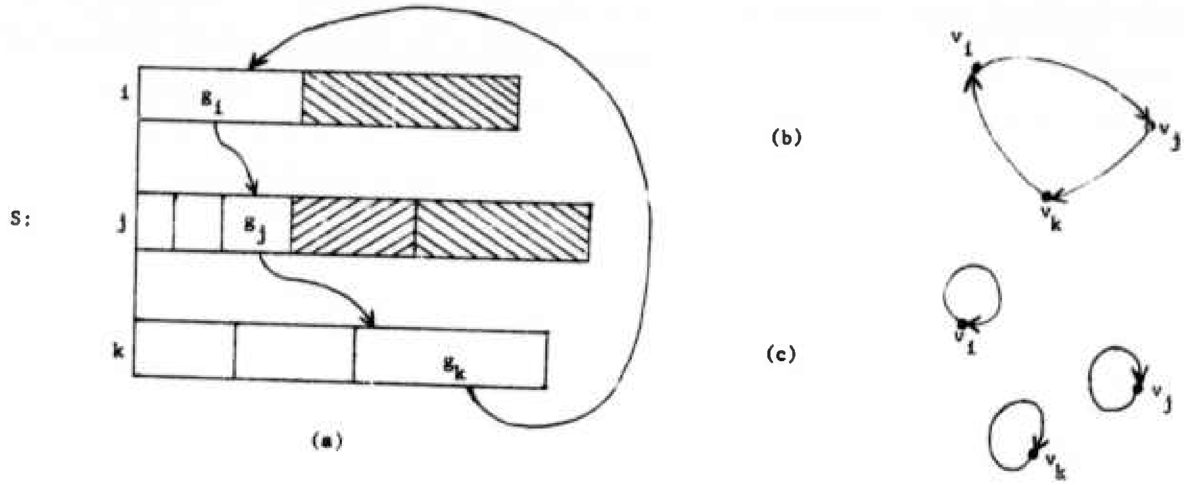


Figure 6

a (k+1)th job that will be optimally scheduled by QAD. The existence of a row-permutation of P that yields an optimal schedule follows immediately. ■

One consequence of Theorem 3-1 is that an optimal schedule may be found by applying QAD to each of the n! row-permutations of P and picking the best schedule from among the results. The existence of an $0(n^3)$ algorithm [2,3], however, precludes the use of such an approach for any but trivially small problems. Of more interest would be some computationally inexpensive rule for sorting the rows of P prior to the execution of QAD or QAD* in such a way that the discovery of some optimal schedule becomes, if not certain, at least more likely. What is needed is a scheme for finding some single number (like p_i in the machine factor case) which best represents the m processing times of a job, and which can be used as a key for sorting the rows of P. Several such sorting rules are proposed and analyzed in the next two sections of this paper.

4. WORST CASE PERFORMANCE

In this section we will examine the worst case performance of the QAD algorithm under various row-sorting rules. Since QAD* never yields a worse schedule than QAD, the performance bounds to be given for QAD will also hold for QAD*, though some bounds which are sharp for QAD are not necessarily so for QAD*.

Let F_{QAD}^{RULE} be the total flow-time of the QAD schedule for some P whose rows are sorted according to rule RULE, and let F_{OPT} be the optimal (minimal) flow-time for P. The measure of performance that we will examine is

$$\frac{F_{QAD}^{RULE}}{F_{OPT}} .$$

We will first seek upper bounds on this expression for various rules. Then, in Section 5, we will use Monte Carlo techniques to estimate the average performance of the algorithms.

Theorem 4-1. Let P^1 and P^2 be two row-permutations of an arbitrary P. Denote by F^1 and F^2 the total flow-times of their QAD scedules. Then

$$\frac{F^1}{F^2} < n \tag{5}$$

and this is a best bound.

<u>Proof</u>. Let C_i^1 and C_i^2 denote the contributions of job i to F^1 and F^2, respectively, so that

$$C_i^1 = h_j^1 p_{ij}$$
$$C_i^2 = h_k^2 p_{ik} \tag{6}$$

for some j and k, where h_j^1 and h_k^2 are the chosen QAD coefficients for job i in P^1 and P^2. We will show that for all i,

$$\frac{C_i^1}{C_i^2} \le n. \tag{7}$$

Suppose (7) is false for some i. Then, using (6), we may write

$$h_j^1 p_{ij} > n h_k^2 p_{ik}. \tag{8}$$

By the operation of the QAD algorithm, we know that

$$h_j^1 p_{ij} = \min_{1 \le g \le m} (h_g^1 p_{ig})$$

and in particular,

$$h_j^1 p_{ij} \le h_k^1 p_{ik} \tag{9}$$

where k is as in (6). Combining inequalities (8) and (9), we get

$$h_k^1 p_{ik} \ge h_j^1 p_{ij} > n h_k^2 p_{ik}$$
$$h_k^1 p_{ik} > n h_k^2 p_{ik}$$
$$h_k^1 > n h_k^2 . \tag{10}$$

Inequality (10) is plainly impossible, since h_k^1 and h_k^2 are integers between 1 and n. This contradiction proves (7).

Now write

$$\frac{F^1}{F_2} = \frac{C_1^1 + C_2^1 + \ldots + C_n^1}{C_1^2 + C_2^2 + \ldots + C_n^2} .$$

From inequality (7) and the fact that all C_i^1 and C_i^2 are positive, it follows that

$$\frac{C_1^1 + C_2^1 + \ldots + C_n^1}{C_1^2 + C_2^2 + \ldots + C_n^2} \le n. \tag{11}$$

For equality to hold in (11), it must be true that for <u>all</u> i, $C_i^1 = nC_i^2$. But clearly this cannot be, so inequality (11) becomes strict and (5) is proved.

To show that n is a best bound, let P take the following form:

$$\begin{array}{c|cccccc}
 & 1 & 2 & 3 & . & . & . & m \\
\hline
1 & \epsilon & \omega & \omega & . & . & . & \omega \\
2 & \epsilon & \omega & \omega & . & . & . & \omega \\
. & . & . & . & . & & & . \\
. & . & . & . & & . & & . \\
. & . & . & . & & & . & . \\
n-1 & \epsilon & \omega & \omega & . & . & . & \omega \\
n & X & \omega & \omega & . & . & . & \omega
\end{array} \tag{12}$$

where $\epsilon << X << \omega$. Let P^1 be as given and construct P^2 by interchanging rows 1 and n. The QAD algorithm will schedule all jobs on machine 1 for both P^1 and P^2; only the job order will be different. We will have

$$\frac{F^1}{F^2} = \frac{\epsilon + 2\epsilon + 3\epsilon + \ldots + (n-1)\epsilon + nX}{X + 2\epsilon + 3\epsilon + \ldots + (n-1)\epsilon + n\epsilon}$$

and

$$\lim_{\epsilon \to 0} \frac{F^1}{F^2} = \frac{nX}{X} = n.$$

Thus we may, for $m \geq 1$ and $n \geq 2$, construct an array P with two row-permutations P^1 and P^2 such that the ratio F^1/F^2 is arbitrarily close to n. Therefore n is a best bound. ∎

<u>Corollary 4-1</u>. Let F_{QAD} be the total flow-time of the QAD schedule for an arbitrary P under an arbitrary row-permutation. Then

$$\frac{F_{QAD}}{F_{OPT}} < n \tag{13}$$

and this is a best bound.

<u>Proof</u>. That n is an upper bound follows directly from Theorem 3-1 and Theorem 4-1. That it is a best bound may be seen by examining (12) in the proof of Theorem 4-1. In that example $F^2 = F_{OPT}$. ∎

The argument for the sharpness of the bounds in Theorem 4-1 and Corollary 4-1 depends on the peculiar structure of the processing-time array (12). If QAD* were used instead of QAD, the optimal schedule for (12) — SPT on machine 1 — would be found. Thus the bound n in (5) and (13) is not necessarily a best bound for QAD*.

The sensible use of both algorithms depends on some scheme for arranging the rows of P prior to their execution. The approach we take will be to sort the rows so that the values of some single-valued function of each row are in non-increasing order. Perhaps the most obvious choice for such a function is the <u>average</u> of the m processing times of a job. Two other possibilities are the <u>maximum</u> and <u>minimum</u> of the m times. Notice that any of these three rules will cause QAD to perform optimally in the machine factor case, and it seems reasonable to require that this be true of whatever sorting rule we adopt.

Denote by AVE, MAX, and MIN, the rules of sorting in order of non-increasing row average, maximum, and minimum, respectively. We can easily show, unfortunately, that AVE and MAX do not improve the worst case performance of QAD. For consider the following array P:

$$\begin{array}{c|cccccc}
 & 1 & 2 & . & . & . & m \\
\hline
1 & \epsilon & \omega+X & . & . & . & \omega+X \\
2 & \epsilon & \omega+X & . & . & . & \omega+X \\
. & . & . & . & & & . \\
. & . & . & & . & & . \\
. & . & . & & & . & . \\
n-1 & \epsilon & \omega+X & . & . & . & \omega+X \\
n & X & \omega & . & . & . & \omega
\end{array} \tag{14}$$

where $\epsilon << X << \omega$. The rows are already arranged according to both AVE and MAX. The optimal schedule for (14) is SPT on machine 1, but QAD schedules job n first instead of last on machine 1. Therefore, using the argument from the proof of Theorem 4-1, the bounds

$$\frac{F_{QAD}^{AVE}}{F_{OPT}} < n$$

$$\frac{F_{QAD}^{MAX}}{F_{OPT}} < n$$

are best bounds. We have already observed that QAD* cannot be tricked in this way, so these may not be best bounds for QAD*.

The worst case behavior of QAD under the MIN rule is less than about half as bad as under the AVE and MAX rules. To establish this we will first need two lemmas.

Lemma 4-1. Let r_i denote the minimum value in row i of an arbitrary P whose rows have been arranged according to the MIN rule. Then

$$F_{QAD}^{MIN} \le r_1 + 2r_2 + \ldots + nr_n \tag{15}$$

with equality only if the r_i occur in a single column of P.

Proof. Let C_i denote the contribution of job i to F_{QAD}^{MIN}, so that

$$F_{QAD}^{MIN} = C_1 + C_2 + \ldots + C_n.$$

If we can show that for all i, $C_i \le ir_i$, then (15) will follow directly. The value of a particular C_i will be h_jp_{ij} for some j. Suppose that r_i occurs in column k, so $r_i = p_{ik}$. Clearly, $h_k \le i$ and $r_i \le p_{ij}$. By the operation of QAD we know that $h_jp_{ij} \le h_kp_{ik}$. Then we may write

$$C_i = h_jp_{ij} \le h_kp_{ik} = h_kr_i \le ir_i$$

and (15) is proved. The necessary condition for equality in (15) is obvious. ∎

Lemma 4-2. If $r_1 \ge r_2 \ge \ldots \ge r_n > 0$ then

$$\frac{r_1 + 2r_2 + \ldots + nr_n}{r_1 + r_2 + \ldots + r_n} \le \frac{n+1}{2} \tag{16}$$

with equality if and only if $r_1 = r_2 = \ldots = r_n$.

Proof.[†] Inequality (16) is equivalent to

$$\frac{n+1}{2} - \frac{r_1 + 2r_2 + \ldots + nr_n}{r_1 + r_2 + \ldots + r_n} \geq 0$$

which, since all r_i are positive, is itself equivalent to

$$(n+1)(r_1 + r_2 + \ldots + r_n) - 2(r_1 + 2r_2 + \ldots + nr_n) \geq 0. \tag{17}$$

Collect terms in (17) to get

$$(n-1)r_1 + (n-3)r_2 + (n-5)r_3 + \ldots + (5-n)r_{n-2} + (3-n)r_{n-1} + (1-n)r_n \geq 0.$$

Now match r_1 with r_n, r_2 with r_{n-1}, and so on, to get

$$(n-1)(r_1-r_n) + (n-3)(r_2-r_{n-1}) + \ldots + (n-n)r_{\lceil\frac{n}{2}\rceil} \geq 0 \tag{18}$$

for odd n; and

$$(n-1)(r_1-r_n) + (n-3)(r_2-r_{n-1}) + \ldots + 1(r_{\frac{n}{2}}-r_{\frac{n}{2}+1}) \geq 0 \tag{18'}$$

for even n. Due to the constraint that $r_1 \geq \ldots \geq r_n > 0$, each term in (18) and (18') is non-negative. This establishes (17) and therefore (16). Equality holds in (18) and (18') if all terms are 0, or equivalently, $r_1 = r_2 = \ldots = r_n$; otherwise, some term will be positive and (16) will be strict. ■

Theorem 4-2.

$$\frac{F_{QAD}^{MIN}}{F_{OPT}} < \frac{n+1}{2}. \tag{19}$$

Proof. First notice that if the r_i are row minima of P,

$$F_{OPT} \geq r_1 + r_2 + \ldots + r_n \tag{20}$$

with equality only if the r_i lie in different columns of P. (This implies $m \geq n$.) Inequality (20) and Lemma 4-1 allow us to write

$$\frac{F_{QAD}^{MIN}}{F_{OPT}} \leq \frac{r_1 + 2r_2 + \ldots + nr_n}{r_1 + r_2 + \ldots + r_n}. \tag{21}$$

The necessary conditions for equality in (20) and in (15) of Lemma 4-1 are contradictory. The inequality in (21) is therefore strict. Using this fact together with Lemma 4-2, we get (19) directly. ■

The bound in (19) has not been shown to be a best bound for either QAD or QAD*. In fact, the author has been unable to construct an array P which violates the surprising bound in the following conjecture.

[†] This lemma was proved by Professor G. W. Stewart.

Conjecture:

$$\frac{F_{QAD}^{MIN}}{F_{OPT}} < 2.$$

We now turn briefly to the time complexity of QAD and QAD* with row-sorting rules. Each of the three sorting rules examined takes time proportional to m for each of the n rows of P, or O(nm), to calculate the sort keys. Sorting the rows takes an additional O(n log n); the guts of QAD require O(nm); and the final SPT sort of QAD* takes at most O(n log n). Both algorithms with sorting rules, therefore, have time complexity O(max(mn, n log n)). Not surprisingly, this is the same order of complexity displayed by the machine factor algorithm. For purposes of comparison, constructing an SPT schedule for a single machine - perhaps the simplest scheduling problem of all - itself requires time of O(n log n).

5. AVERAGE PERFORMANCE

Most algorithms behave much better most of the time than they do in the worst case, and the QAD algorithm is no exception. In this section we will use Monte Carlo methods to examine the average performance of QAD and QAD* under various row-sorting rules for two models of parallel machine systems. Model I is extremely simple and correspondingly remote from reality; Model II, a computer system model, is considerably more down-to-earth.

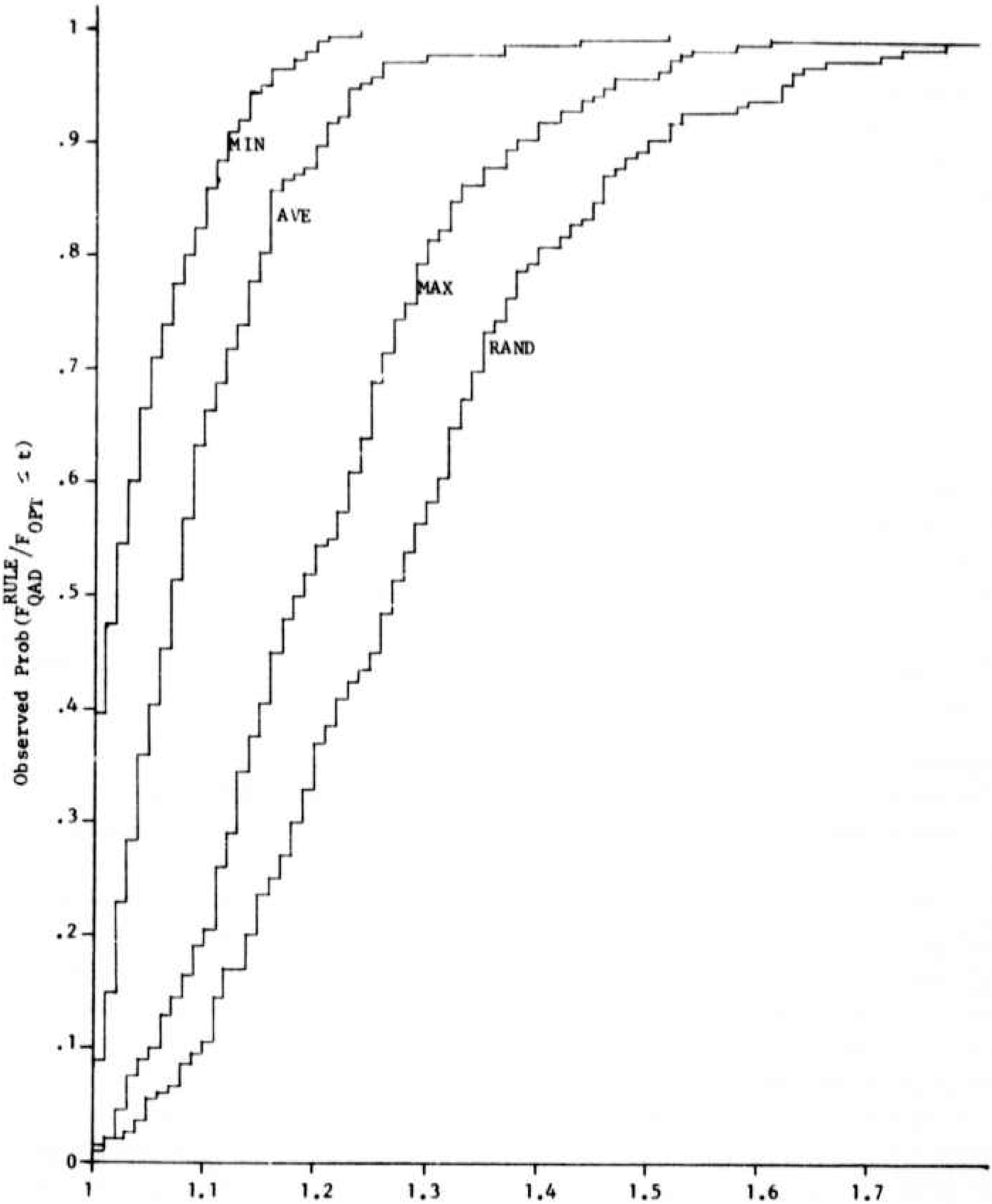
The measure of performance used, as in Section 4, is F_{QAD}^{RULE}/F_{OPT} (and F_{QAD*}^{RULE}/F_{OPT}). In a typical experiment, a large number of arrays P were randomly generated and the values of this performance measure calculated for each of four sorting rules: MIN, AVE, MAX, and a new rule, RAND, which arranges the rows of P in random order. The algorithm of Bruno, Coffman, and Sethi [2,3] was used to calculate F_{OPT}. The rules MIN, AVE, etc. used with QAD* will be denoted by MIN*, AVE*, etc.

5.1 Model I

In this model, the p_{ij} are integers independently drawn from a uniform distribution over a specified range. Fig. 7 shows the results of an experiment in which were generated 200 arrays with n = 8 and m = 3, where the range of the uniform distribution was from 1 to 100. Each of the four rules was applied with QAD to each of the 200 problems, and the four curves of Fig. 7 are the sample cumulative distribution functions for the values of F_{QAD}^{RULE}/F_{OPT}. The point (1.2, 0.90) on the AVE curve, for example, means that 90 percent (180) of the 200 values of F_{QAD}^{AVE}/F_{OPT} were less than or equal to 1.2. We may immediately observe that MIN is substantially better than the other rules for this experiment, and that the performance of QAD under the MIN rule is quite good: 39.5 percent of the sample problems were scheduled optimally, and 95 percent of the solutions were at most 15 percent worse than the optimal schedule.

The performance of MIN is little improved by the final SPT sort of the QAD* algorithm, though the other rules are improved to varying degrees. Fig. 8 shows the results of applying QAD* to the same 200 problems. The curve for MIN* is omitted because its closeness to AVE* would have obscured both curves, and because it is nearly identical to MIN in Fig. 7.

Table 1 shows the extent to which each rule outperformed each of the others in this experiment. For instance, AVE* yielded a better schedule than MAX in 85 percent (170) of the 200 sample problems. Table 2 gives



1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
Observed Prob($F_{QAD}^{RULE}/F_{OPT} \le t$)
MIN
AVE
MAX
RAND
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
t

Figure 7

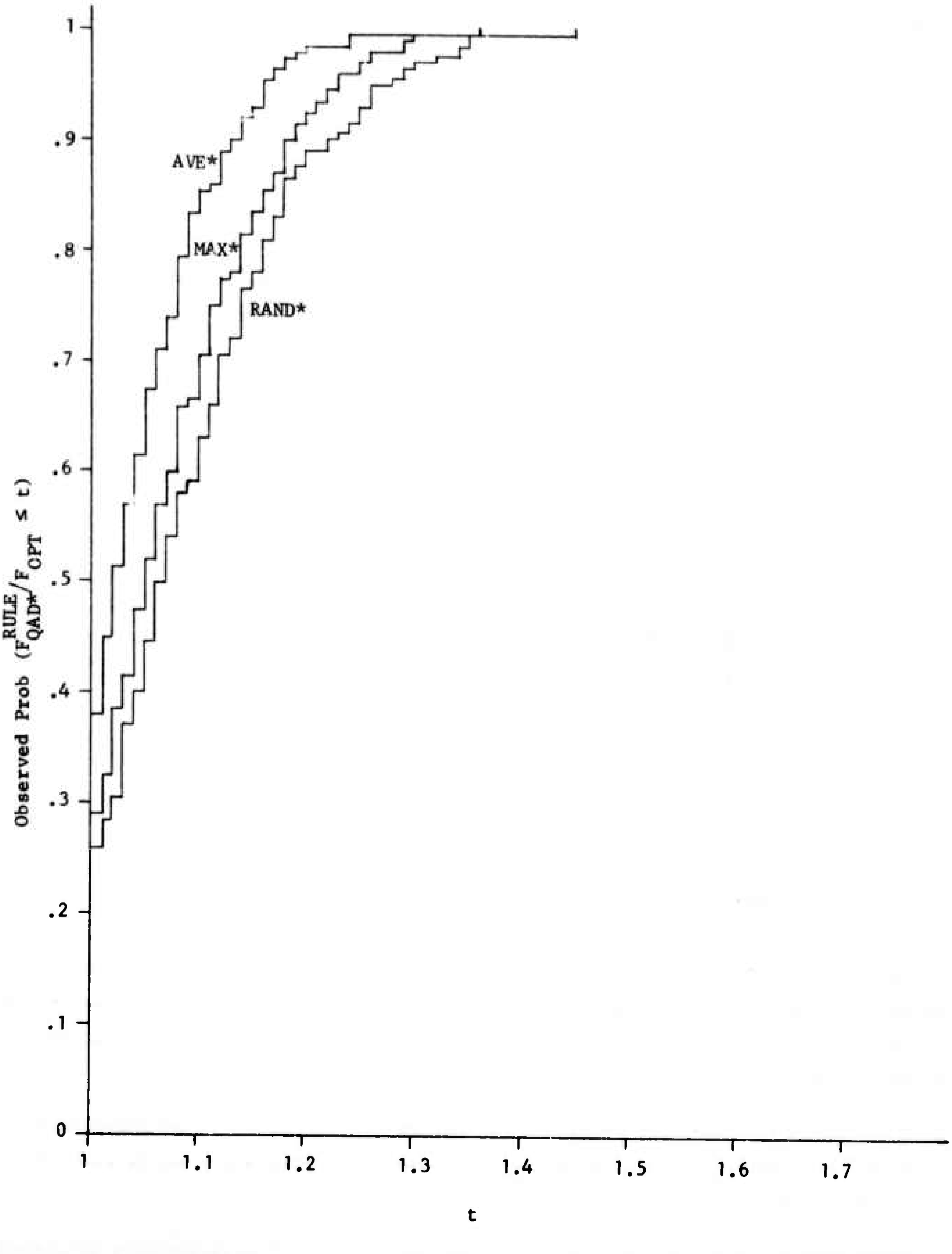

1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
AVE*
MAX*
RAND*
Observed Prob $(F^{RULE}_{QAD*}/F_{OPT} \le t)$
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
t

Figure 8

four useful characteristics of each sample distribution: how often an optimal schedule was found; the mean of the distribution; the value at and below which 95 percent of the samples lie; and the maximum value observed in the experiment.

TABLE 1

Rule A better than Rule B (percentage)

A \ B	MIN	MIN*	AVE	AVE*	MAX	MAX*	RAND	RAND*
MIN	0	0	70	30.5	90.5	51.5	95.5	55.5
MIN*	1	0	70	30.5	90.5	51.5	95.5	55.5
AVE	19.5	19.5	0	0	82	39	86	41.5
AVE*	27.5	27	73	0	93.5	49	94.5	56.5
MAX	7	7	9	5	0	0	64.5	21.5
MAX*	23.5	23.5	54	24.5	93.5	0	87.5	43.5
RAND	3.5	3.5	12.5	4.5	33	10.5	0	0
RAND*	21	21	52	22.5	76.5	33	96.5	0

TABLE 2

Distribution Characteristics

	PERCENTAGE OPTIMAL	MEAN	95% LEVEL	OBSERVED MAXIMUM
MIN	39.5	1.038315	1.141304	1.234177
MIN*	40	1.038233	1.141304	1.234177
AVE	9	1.087132	1.228572	1.516129
AVE*	38	1.043284	1.158228	1.447581
MAX	1	1.204095	1.45946	2.02963
MAX*	29	1.069569	1.220109	1.357934
RAND	1.5	1.280514	1.614815	1.917808
RAND*	26	1.086278	1.259434	1.449367

The evidence thus far presented suggests that for Model I, MIN is the best rule. The fact that MIN* is marginally better would seem to be outweighed by the additional computational effort it requires. Two characteristics of the results of this experiment turn out to be true for every experiment, both in Model I and Model II, that will be described:

1. The rules listed in order of goodness of performance are MIN, AVE, MAX, RAND for both QAD and QAD*.
2. MIN* outperformed MIN virtually never; RAND* outperformed RAND virtually always; and the other rules were in between.

Several parameters of the experiment were varied to test the sensitivity of the performance of QAD with MIN to changes in the model. Two experiments of 100 samples were run, one with m = 6 and all other parameters as in the first experiment, and the other with the range of the uniform distribution changed from 1-100 to 1-1000 and other parameters as before. Both experiments yielded MIN curves and distribution characteristics not substantially different from those of Fig. 7 and Table 2. Taking into account the limited number and nature of the experiments (restrictions imposed mainly by the computational requirements of the optimal algorithm), we may tentatively conclude that performance in Model I is relatively insensitive to variations in these parameters.

Other experiments demonstrated that these performance results <u>are</u> sensitive to changes in n. Three experiments were performed with the uniform distribution as before: one with 100 sample P's of size n = 16 by m = 4; one with 20 samples of size 32 by 5; and one with only five samples of size 64 by 6. The results of the first two of these (MIN curves only) are compared with the MIN curve of Fig. 7 on an expanded horizontal scale in Fig. 9. It appears that as n increases (with m = log n), the sample mean of the distributions remains relatively stationary, while the sample variance decreases. The three means are: 8 by 3, 1.038315; 16 by 4, 1.039334; and 32 by 5, 1.0371205. Further support is lent to this tentative conclusion by the five values of F_{QAD}^{MIN}/F_{OPT} for the 64 by 6 experiment: 1.02552, 1.029267, 1.03074, 1.043362, and 1.056319, with a mean of 1.0370414.

It is difficult to conceive of a realistic situation in which the p_{ij} are independent and uniformly distributed. A job which is very fast on some machine is likely to be fast on some other machines as well; a machine which runs one job faster than the other machines is likely to run other jobs quickly too. We turn now to a more realistic model.

5.2 Model II

Model II is a model of a multiprocessor computer system not unlike Carnegie-Mellon University's C.mmp [7]. The machines are different models of the Digital Equipment Corporation PDP-11 computer [6], and the jobs are computing jobs from a small number of distinct job classes. Each job class has its own vector of m machine factors. A job's m processing times are calculated by drawing a single time from an exponential distribution and multiplying by the machine factors associated with its job class. (Times are rounded to the nearest integer.) The machine factors used here were calculated from PDP-11 performance figures given in [1].

The first experiment to be described considered 200 problems with n = 8 and m = 3, where the computers were a PDP-11 Model 20, a Model 40, and a Model 45. There were three job classes: Class 1, "average" jobs, of which there were 4; Class 2, floating-point jobs, of which there were 3; and Class 3, a single job which could run only on the Model 40 because (for example) it required a particular peripheral device connected only to that machine. Below are the machine factors for these job classes:

	PDP-11/20	PDP-11/40	PDP-11/45
Class 1	1	.556	.556
Class 2	1	.291	.134
Class 3	∞	.556	∞

Note that floating-point jobs exploit the specialized hardware of the 11/40 and especially the 11/45 to a much greater extent than "average" jobs. The exponential distribution used in this experiment had mean 1000

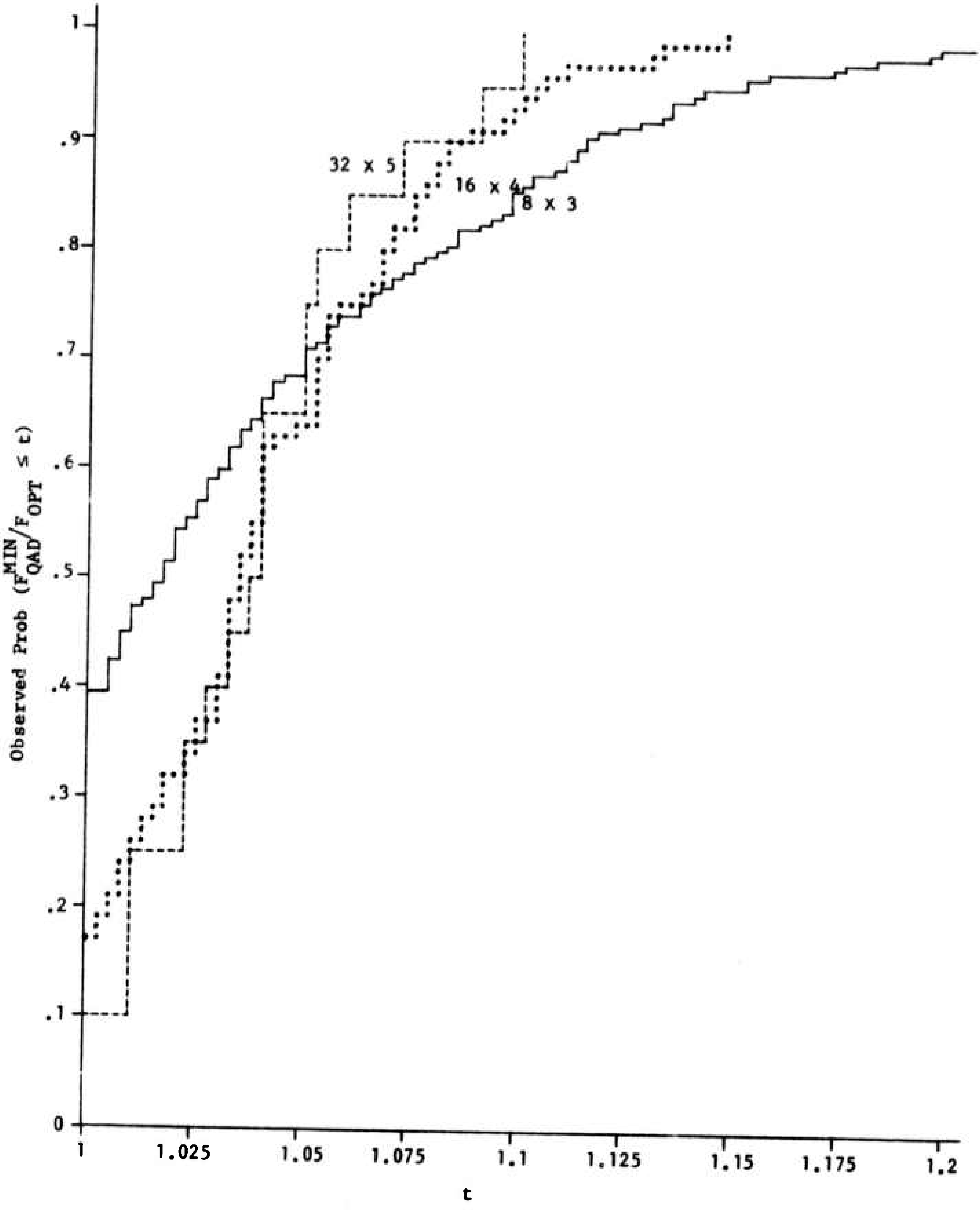

1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
Observed Prob $(F_{QAD}^{MIN}/F_{OPT} \le t)$
32 x 5
16 x 4
8 x 3
1
1.025
1.05
1.075
1.1
1.125
1.15
1.175
1.2
t

Figure 9

(milliseconds). For purposes of the row-sorting rules, the average and maximum processing times of a Class 3 job were considered to be its processing time on the 11/40.

The results of this experiment are shown in Figures 10 (QAD) and 11 (QAD*) and in Tables 3 and 4. Comparing Fig. 10 with Fig. 7 we observe — perhaps with some surprise — that QAD's performance is better in Model II than in Model I for all rules except RAND. The MIN rule, in particular, found optimal schedules in over 60 percent of the cases, and did no more than about 8 percent worse than optimal in 95 percent of the cases.

TABLE 3

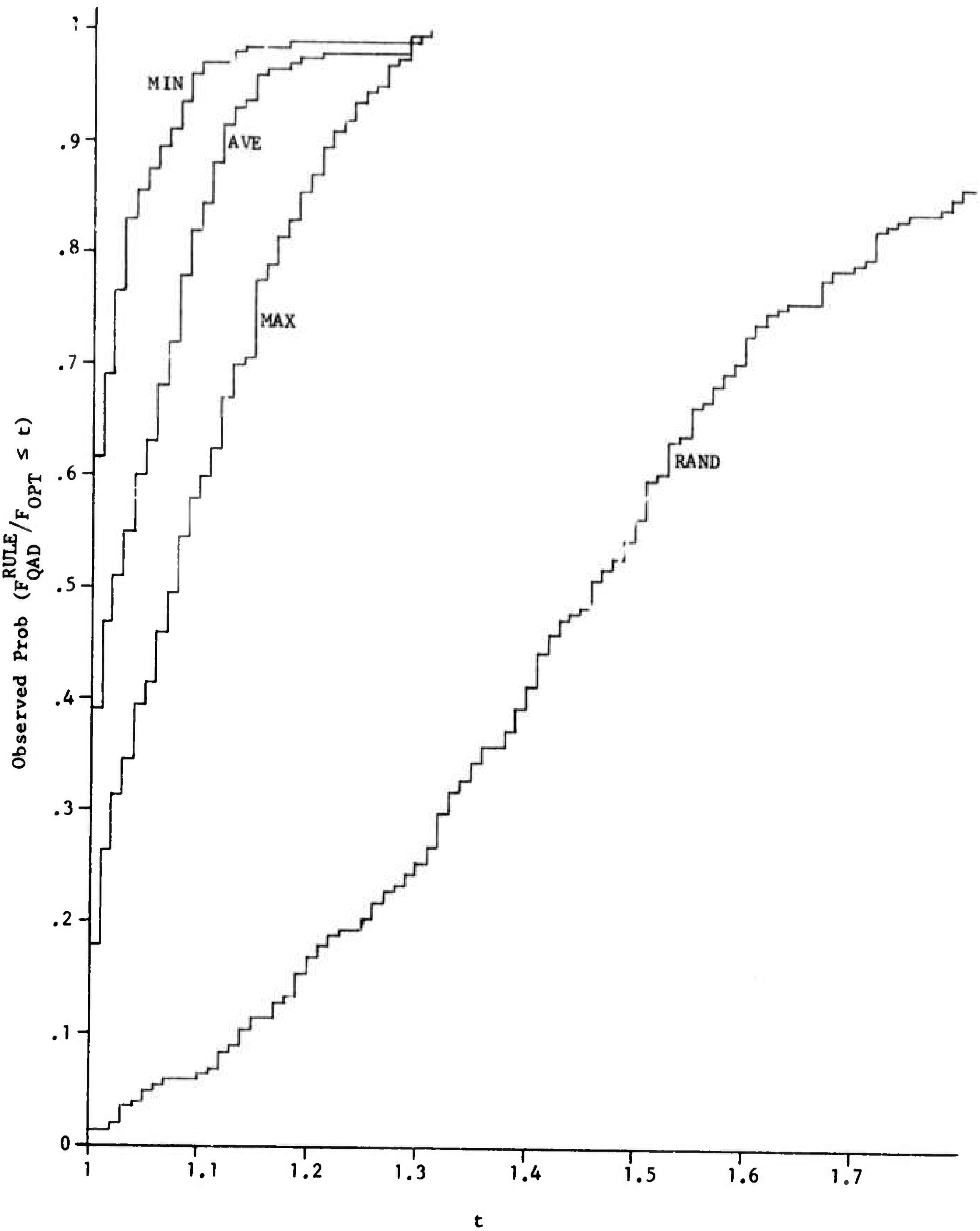
Rule A better than Rule B (percentage)

A \ B	MIN	MIN*	AVE	AVE*	MAX	MAX*	RAND	RAND*
MIN	0	0	48	47.5	71.5	66.5	98.5	89
MIN*	0	0	48	47.5	71.5	66.5	98.5	89
AVE	13.5	13.5	0	0	45.5	37	95.5	79.5
AVE*	13.5	13.5	18.5	0	56	37.5	96.5	81.5
MAX	10	10	0	0	0	0	93.5	63
MAX*	11	11	11	0	47.5	0	94.5	70.5
RAND	0	0	3	2.5	5	4	0	0
RAND*	5	5	15.5	13	32.5	22.5	97.5	0

TABLE 4

Distribution Characteristics

	PERCENTAGE OPTIMAL	MEAN	95% LEVEL	OBSERVED MAXIMUM
MIN	61.5	1.01767	1.081088	1.303683
MIN*	61.5	1.01767	1.081088	1.303683
AVE	39	1.044154	1.143041	1.303683
AVE*	40.5	1.039521	1.139162	1.303683
MAX	18	1.088859	1.257068	1.303683
MAX*	24.5	1.06943	1.188495	1.303683
RAND	1.5	1.476827	1.952063	2.249682
RAND*	4.5	1.153185	1.442467	1.811287



MIN
AVE
MAX
RAND
Observed Prob $(F_{QAD}^{RULE}/F_{OPT} \le t)$
0
.1
.2
.3
.4
.5
.6
.7
.8
.9
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
t

Figure 10

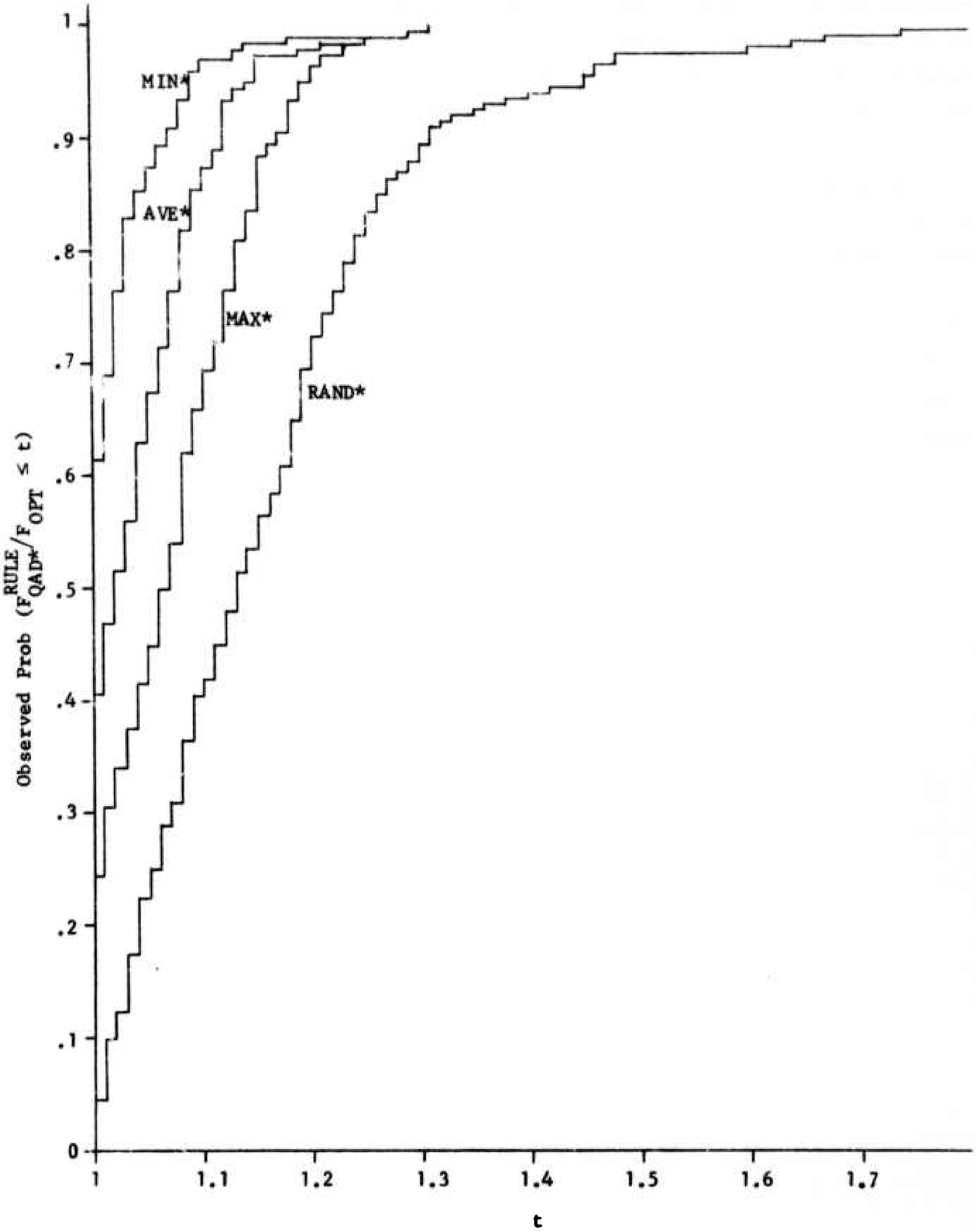
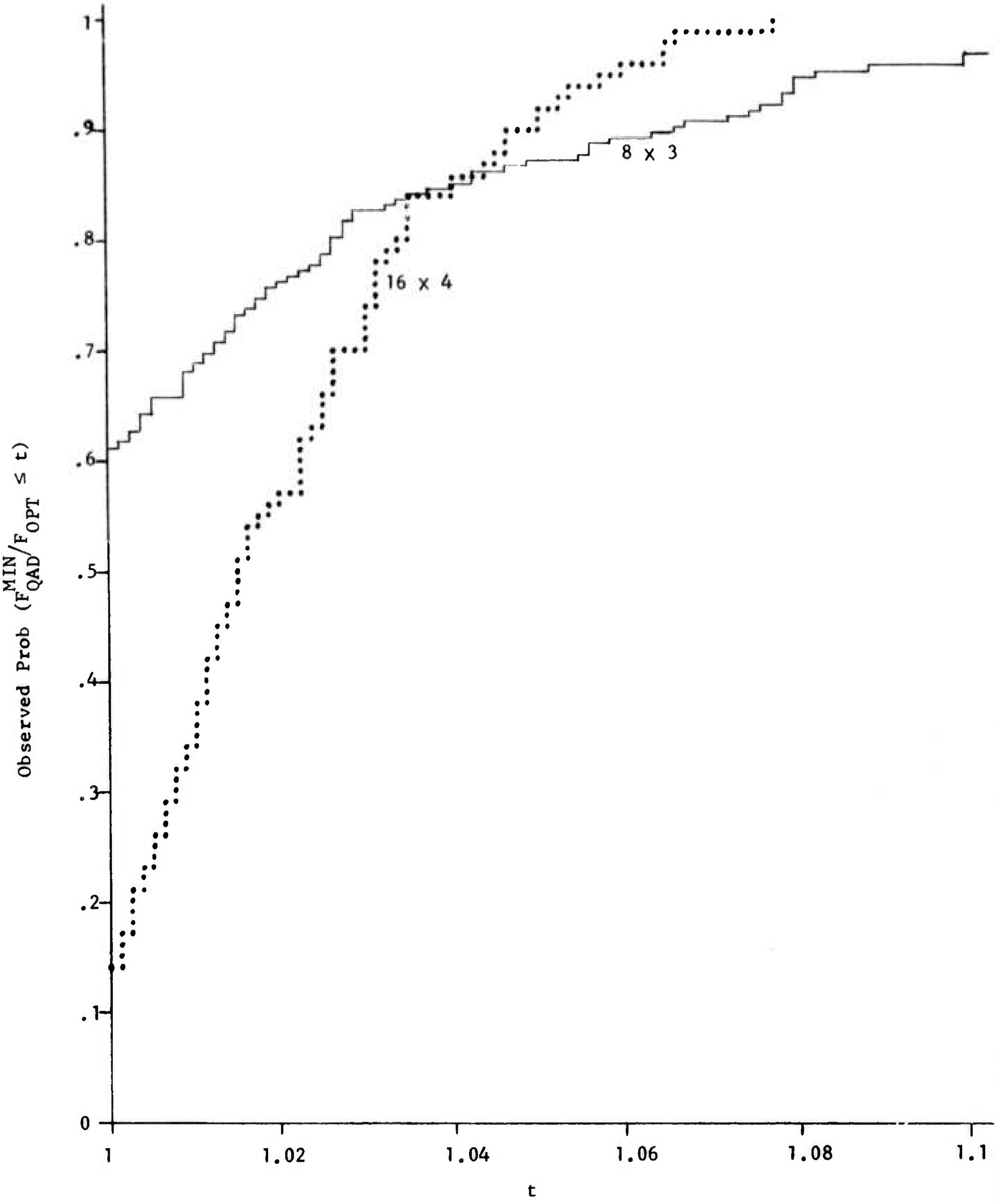

MIN*
AVE*
MAX*
RAND*
1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
Observed Prob $(F^{RULE}_{QAD*}/F_{OPT} \le t)$
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
t

Figure 11

The sensitivity of these results to changes in the model's parameters was examined by running several additional experiments. As in Model I, changing the range of values of the p_{ij} had little effect. An experiment of 100 sample arrays was run with all parameters as before, except that the mean of the exponential distribution was changed from 1000 to 60000 (milliseconds). The performance of QAD under the MIN rule was not significantly different from that of the first experiment. Again, strong conclusions are ruled out by the limited extent of the testing, but performance in this model would tentatively appear not to depend on the exponential distribution.

In further agreement with Model I, performance was found to be sensitive to changes in n. An experiment was performed in which the number of jobs in each class was doubled, bringing the total to 16. A fourth machine was added to the model, a PDP-11 Model 05, whose machine factors for the three job classes are: Class 1, 1.25; Class 2, 1.25; and Class 3, ∞. One hundred sample arrays were tested. The results appear in comparison with the 8 by 3 results in Fig. 12 on a greatly expanded horizontal scale. The means of the two experiments are: 8 by 3, 1.01767; 16 by 4, 1.019837. Approximately the same behavior found in Model I is demonstrated here: as n increases (with $m = \log n$), the sample mean appears to remain relatively stationary, while the sample variance decreases.



1
.9
.8
.7
.6
.5
.4
.3
.2
.1
0
Observed Prob $(F_{QAD}^{MIN}/F_{OPT} \le t)$
8 x 3
16 x 4
1
1.02
1.04
1.06
1.08
1.1
t

Figure 12

6. CONCLUSION

For the scheduling problem studied in this paper, the QAD algorithm seems to be a reasonable alternative to the optimal algorithm of Bruno, Coffman, and Sethi [2,3]. The QAD algorithm (with the MIN rule) takes computational time of $O(\max(mn, n \log n))$, while the optimal algorithm requires $O(\max(mn^2, n^3))$. Furthermore, QAD is an extremely simple algorithm, and easy to work by hand (as might be required in an industrial shop, for instance). By contrast, the optimal algorithm is quite difficult. While QAD does not always find optimal schedules, it frequently does (and it always can); and its performance, bounded in the worst case, appears from limited experimental results to be very good most of the time.

There is, of course, much room for further work. The most interesting unanswered question is whether there exists a simple row-sorting rule which will guarantee QAD's production of optimal schedules. It might be the case, however, that sorting according to any function of a row is by itself insufficient; more information about the processing time array might be required to discover the optimal row-permutation promised by Theorem 3-1. If an optimal rule cannot be found, perhaps row-sorting rules more fruitful than the simple ones considered here could be discovered. A proof of the conjecture in Section 4 would be extremely interesting. Finally, the computational complexity of this problem is unknown, and the work reported here only begins to suggest that the complexity is less than $O(n^3)$.

Acknowledgments. I am grateful to Pete Stewart for the proof of Lemma 4-2; to Jack McCredie and J. G. Ramage for advice on Section 5; and to Sam Fuller, Thérèse Flaherty, and David Stevenson for comments and criticism throughout.

REFERENCES

1. Bell, C. G., and Kaman, C. The Effect of Semiconductor Memory Technology on the Design of the PDP-11 Series Minicomputers. Digital Equipment Corporation, October, 1973.

2. Bruno, J. A Scheduling Algorithm for Minimizing Mean Flow Time. TR 141, Computer Science Department, Penn. State University.

3. Bruno, J., Coffman, E. G., and Sethi, R. Algorithms for Minimizing Mean Flow Time. IFIP Congress 74, Stockholm, Sweden.

4. Clark, W. The Gantt Chart (3rd edition). Pitman and Sons, London, 1952.

5. Conway, R. W., Maxwell, W. L., and Miller, L. W. Theory of Scheduling. Addison-Wesley, 1967.

6. PDP-11 Processor Handbook. Digital Equipment Corporation, 1972.

7. Wulf, W. A., and Bell, C. G. C.mmp — A Multi-Mini-Processor. Proc. FJCC, 1972, 765-778.