INFORMATION SCIENCES **16**, 137–150 (1978)

An Efficient Job Scheduling Algorithm for Mixed Turnaround and Deadline Applications

K. SUNDAR DAS

University of Texas at San Antonio
San Antonio, Texas 78285

and

TOBY J. TEOREY

The University of Michigan
Ann Arbor, Michigan 48109

ABSTRACT

Computer job scheduling is often performed with little understanding of the formal properties of the jobs being scheduled. One reason for this is that optimal solutions for job scheduling on computers are difficult to obtain if the job stream has mixed objectives, i.e., it consists of some jobs whose turnaround time has to be minimized and others whose deadlines must be met. A practical algorithm for scheduling mixed job streams on monoprogrammed computers, with potential application to a multiprogramming environment, is presented. The algorithm takes into account variable cost rates for each job. Experimental results illustrate the efficiency of the algorithm in terms of both its proximity to optimal solutions and its low computational complexity.

I. INTRODUCTION

Operations research literature has given considerable attention to the problem of scheduling a set of given jobs on either a single machine or multiple machines, each of which can execute at most one job at a time [1, 4, 6–10]. Such scheduling algorithms are applicable to monoprogrammed computers. Furthermore, they consider only job streams whose costs are functions of either turnaround times *or* deadline requirements. Such job streams will be called homogeneous job streams.

In contrast to this, data processing centers usually encounter mixed job streams, i.e., some jobs require the consideration of their turnaround times, while others require the consideration of their deadline times. It is the purpose of this paper to develop a job scheduling algorithm for mixed job streams. This algorithm, called the modified Wilkerson-Irwin (MWI) algorithm, is applicable

 0020-0255/78/08013713$01.75

to monoprogrammed computers operating in a batched environment. The motivation for developing the algorithm, however, grew out of the need to improve job scheduling on multiprogrammed computers. The feasibility of this algorithm for multiprogrammed computers has also been proposed [5].

Section II presents a cost function for scheduling situations in general. Section III presents a review of the literature, and Sec. IV describes the MWI algorithm and presents its computational complexity. Section V discusses the results of experiments designed to investigate the optimization solution properties and computational speed of the MWI algorithm.

Job Scheduling Terminology

Batched jobs may be broadly classified into the categories of

(1) deadline jobs, and
(2) turnaround jobs.

Deadline jobs are those which should not be delayed beyond their specified deadline times, e.g., payroll jobs. Deadline jobs may further be broken down into periodic and non-periodic deadline jobs. Periodic deadline jobs are those whose deadlines and processing requirements are known well in advance, and are usually known as production jobs. Non-periodic deadline jobs have deadlines but are submitted with little or no advance notice, e.g., test jobs for a proposed production system.

Turnaround jobs are those which are submitted on a request basis and must be processed as soon as possible for the analyst's perusal. Turnaround jobs typically have shorter turnaround time than deadline jobs.

Figure 1 illustrates the various queues of jobs in a computer system at any point in time. Q_1 represents the job mix, i.e., jobs currently in some phase of execution, and hence can contain jobs belonging to any category. The size of Q_1 is 1 in a monoprogrammed computer system. Q_2 is the set of jobs which are

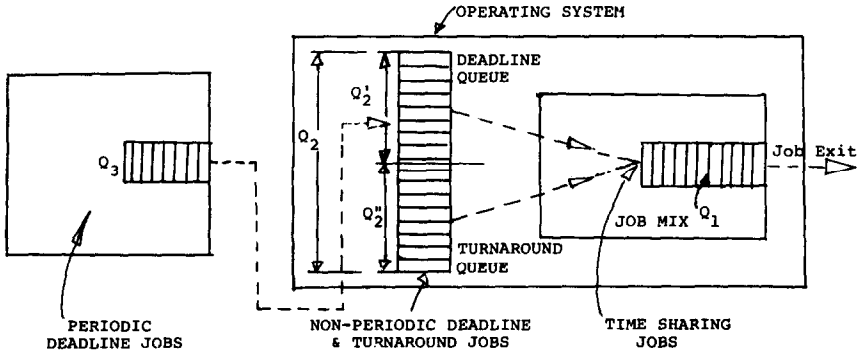

Fig. 1. Streams and queues of jobs.

to be scheduled to join Q_1 at some future time. Q_2 can further be broken down into Q_2' and Q_2'', where Q_2' is the set of deadline jobs and Q_2'' is the set of turnaround jobs. Q_3 is the queue of periodic deadline jobs.

The scheduling of jobs from Q_2 to Q_1 will be considered here. This process will be known as *job scheduling* or *initiation*. The decision to choose a job or set of jobs to join Q_1 from Q_2 occurs at either the exit of a job from Q_1 or the arrival of a job to Q_2. Thus for a given set of jobs at a decision time, the job scheduling algorithm presented here minimizes a cost function based on tardiness and turnaround times. Turnaround time, tardiness and other time relationships are defined below.

Start time of a job is the time at which it is initiated into the mix.

Completion time of a job is the time at which a job along with its output is completed.

Deadline of a job is the time at which its output is due.

Turnaround time of a job is the time between its submission and completion.

Tardiness of a job is the amount of time by which its completion time exceeds its deadline, if it does; otherwise it is zero.

Figure 2 shows the relationships between the various times for a given job i. As can be seen, the wait time and elapsed time of a job are defined in terms of the abovementioned times.

COMPUTER SYSTEM ENVIRONMENT

The following assumptions are made regarding the environment in which the computer system operates.

(1) The computer system is monoprogrammed, operating in a batched mode.

(2) The job stream is mixed in terms of individual job objectives.

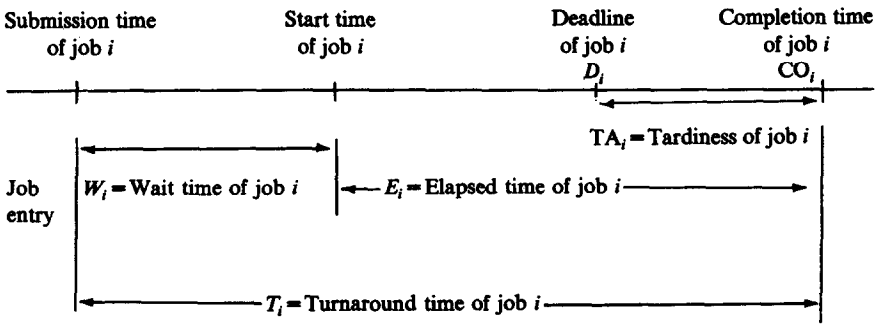

Fig. 2. Timing relationships for jobs.

(3) There are no precedence relations among jobs.

(4) The execution of a job is not suspended in order to facilitate the initiation of a new job (no pre-emption).

DESIRABLE PROPERTIES OF JOB SCHEDULING ALGORITHMS

Job scheduling algorithms considered here should have the following properties.

(1) The sequence of jobs produced by the algorithm should yield an optimal solution to the chosen cost function.

(2) The computational complexity should be low.

(3) It should be able to schedule mixed job streams.

(4) It should be able to schedule job streams in which the costs per unit tardiness and turnaround time are unequal.

(5) It should be applicable to multiprogrammed job streams with dynamic arrival patterns.

II. A COST FUNCTION FOR SCHEDULING MIXED JOB STREAMS

We now develop a unified cost function for mixed deadline and turnaround job streams.

Let

t_0 = epoch of initiation or decision,
$Q_2(t)$ = job queue at time t, i.e., the set of jobs ready to be initiated at time t,
n = number of jobs present at t_0.

The job scheduling algorithm chooses a set of jobs from $Q_2(t_0)$ at t_0 to minimize the total cost function, $Z_1(t_0, H_n)$:

$$Z_1(t_0, H_n) = \sum_{i \in Q_2'(t_0, H_n)} \mathrm{CT}_i \mathrm{TA}_i + \sum_{i \in Q_2''(t_0, H_n)} C_i T_i$$

$$+ \sum_{j=1}^{M} \int_{t_0}^{H_n} \mathrm{WH}_j [1 - u_j(x)]\, dx, \qquad (1)$$

where

TA_i = tardiness of job i,
T_i = turnaround time of job i,
CT_i = cost per unit tardiness of job i, if it is a member of the deadline category of jobs,

C_i = cost per unit turnaround time of job i, if it is a member of the turnaround category of jobs,
M = total number of resources (e.g., CPU, main storage),
$u_j(x)$ = utilization of resource j at time x,
$1-u_j(x)$ = disutilization of resource j at time x,
WH_j = weight (cost) attached to disutilization of resource j,
H_n = time required to complete n jobs,
$Q_2'(t_0, H_n)$ = the set of deadline jobs that are initiated from time t_0 through H_n,
$Q_2''(t_0, H_n)$ = the set of turnaround jobs that are initiated from time t_0 through H_n.

Since the utilization of the resources in a monoprogrammed computer is not dependent upon the sequence in which the jobs are initiated, the disutilization term in Eq. (1) is constant. Therefore, we need only minimize

$$Z_2(t_0, H_n) = \sum_{i \in Q_2'(t_0, H_n)} \mathrm{CT}_i \mathrm{TA}_i + \sum_{i \in Q_2''(t_0, H_n)} C_i T_i. \tag{2}$$

Using the relationships of Fig. 2, we rewrite $Z_2(t_0, H_n)$ as

$$Z_2(t_0, H_n) = \sum_{i \in Q_2'(t_0, H_n)} \mathrm{CT}_i \max[0, \mathrm{CO}_i - D_i] + \sum_{i \in Q_2''(t_0, H_n)} C_i[W_i(t_0) + \mathrm{CO}_i - t_0], \tag{3}$$

where $W_i(t_0)$ is the time job i has waited until time t_0.

Since $\sum_{i \in Q_2''(t_0, H_n)} C_i W_i(t_0)$ is constant over the period (t_0, H_n), it is sufficient to minimize the function

$$Z_3(t_0, H_n) = \sum_{i \in Q_2'(t_0, H_n)} \mathrm{CT}_i \max[0, \mathrm{CO}_i - D_i] + \sum_{i \in Q_2''(t_0, H_n)} C_i \max[0, \mathrm{CO}_i - t_0], \tag{4}$$

because $\mathrm{CO}_i - t_0$ is always non-negative.

If the deadline of all turnaround jobs is defined as t_0 at the decision epoch t_0, we may rewrite $Z_3(t_0, H_n)$ as the general function

$$Z_3(t_0, H_n) = \sum_{i \in Q_2(t_0, H_n)} C_i \max[0, \mathrm{CO}_i - D_i(t_0)], \tag{5}$$

where

$$C_i = \begin{cases} \mathrm{CT}_i & \text{if } i \in Q_2'(t_0, H_n), \\ C_i & \text{if } i \in Q_2''(t_0, H_n), \end{cases}$$

$$D_i(t_0) = \begin{cases} D_i & \text{if } i \in Q_2'(t_0, H_n), \\ t_0 & \text{if } i \in Q_2''(t_0, H_n). \end{cases}$$

The complicated function of Eq. (1) has been reduced to the simpler one of Eq. (5), which is simply the sum of weighted tardiness of jobs [6, 8, 9]. Note that the resource allocation problem has been specifically left out. This is because the computer system is monoprogrammed, and hence there is only one job competing for resources at the time of initiation. It is assumed that all jobs' resource requirements are less than or equal to that of the system's capacity.

III. CURRENT JOB SCHEDULING ALGORITHMS

Special cases of Eq. (5) are easily recognized in the operations research literature. The scheduling algorithm that minimizes the weighted sum of turnaround times only is a simple one: schedule jobs in ascending order of E_i/C_i, where E_i is the elapsed time of job i and C_i is the cost (weight) per unit turnaround time of job i.

An optimal rule for scheduling deadline jobs only is not so simple, however. Elmaghraby [6] and Lawler [8] developed algorithms which yielded optimal solutions for job streams with deadline jobs only and variable (unequal) costs per unit tardiness. Elmaghraby's algorithm involved branch and bound techniques, whereas Lawler's involved combinatorial methods. Both solutions have a computational complexity of the order 2^n. Emmons [7] and Srinivasan [10] have given a means of obtaining an optimal solution when the cost per unit tardiness of all jobs is equal to one. Emmon's algorithm can be modified for cases when the cost per unit tardiness is not equal to one, i.e., variable cost rates.

Wilkerson and Irwin developed a heuristic algorithm for deadline jobs only, with equal cost per unit tardiness [11]. Baker and Martin studied the properties of this algorithm with respect to the above algorithms using statistical techniques [2]. They found the Wilkerson-Irwin algorithm to be favorable in terms of near-optimality and computational complexity. Recently, Petersen devised a heuristic algorithm for job streams comprised of deadline jobs only but with variable cost rates [9]. This was demonstrated to yield optimal solutions in almost all job streams he considered. Petersen also demonstrated that a number of quick heuristics, such as ordering jobs in ascending order of deadline, yielded very poor solutions.

The heuristic algorithms discussed so far in the literature are not applicable to mixed job streams. The algorithms amenable to extensions are the Petersen and Wilkerson-Irwin algorithms. The extension of Petersen's algorithm involves the incorporation of turnaround jobs only. The extension of Wilkerson-Irwin algorithm involves the incorporation of (1) turnaround jobs and (2) unequal costs per unit tardiness and turnaround time. The extended Wilkerson-Irwin algorithm is called the modified Wilkerson-Irwin (MWI) algorithm.

The optimal and computational complexity properties of the extended algorithms are studied by means of a simulation in Sec. V.

IV. THE MODIFIED WILKERSON-IRWIN ALGORITHM

Since H_n is the time to complete n jobs in Eq. (5), the job cost objective function can be written as

$$Z = \sum_{i=1}^{n} C_i \max[0, \mathrm{CO}_i - D_i(t_0)]. \tag{6}$$

Let n_1 be the number of deadline jobs and n_2 be the number of turnaround jobs, so that $n_1 + n_2 = n$. Recall that t_0 is the decision epoch. The MWI algorithm is as follows:

1. Schedule the n_2 turnaround jobs in increasing ratio E_i/C_i, where E_i is the stand alone elapsed time of job i. Schedule the n_1 deadline jobs after the turnaround jobs in increasing order of their deadlines. Let i_j be the job in the jth position.
2. If $n_2 = n$, the algorithm terminates. If $n_2 = 0$, set $i^* = 1$ and $j^* = 2$. If $0 < n_2 < n$, set $i^* = n_2$ and $j^* = n_2 + 1$.
3. Set $t_a = t_0 + \sum_{j=1}^{i^*-1} E_{i_j}$. Note that $t_a = t_0$ if $i^* = 1$. Compare jobs in positions i^* and j^* at t_a by applying the MWI rule (see the Appendix). If it is preferable to schedule the job in j^*th position before the one in i^*th position, we say a jump has occurred. If a jump has occurred, go to step 5; otherwise go to step 4.
4. If $j^* = n$, the algorithm terminates. Otherwise, set $i^* = j^*$; $j^* = j^* + 1$ and repeat step 3.
5. If a jump has occurred, make an exchange of jobs so that the job that was formerly in position j^* is now in position i^* and vice versa. If $i^* = 1$, go to step 4; otherwise set $i^* = i^* - 1$, $j^* = j^* - 1$ and repeat step 3.

The algorithm is illustrated by means of an example. Let there be a total of 5 jobs, 2 of which are turnaround jobs. Without loss of generality, let us label the turnaround jobs in increasing order of the ratio of elapsed time to unit cost of turnaround, E_i/C_i, and the others in increasing order of their deadlines.

Thus, the initial configuration of $n=5$ and $n_2=2$ appears as in Fig. 3 with $E_i/C_i \leqslant E_2/C_2$ and $D_3 \leqslant D_4 \leqslant D_5$.

In step 3 of the algorithm, a comparison of jobs 2 and 3 is made. Let us assume that no exchange is necessary, i.e., a jump has not occurred. Applying step 4, a comparison between jobs 3 and 4 is now required. Let us say a jump has occurred. The configuration then appears as in Fig. 4. According to step 5 a comparison between jobs 2 and 4 is necessary. Let us again say a jump has occurred, in which case the configuration appears as in Fig. 5. The next comparison is between jobs 1 and 4, and let us say no jump occurs at this point. Thus, we see that each jump generates a "backward look" at jobs in the lower order positions. Finally, we compare jobs 3 and 5. Let us say no jump has occurred and therefore the sequence in Fig. 5 is the final MWI algorithm sequence. This example may also be understood by means of Table 1, where the jobs being compared, the position i^* and j^*, and the starting and ending

	Turnaround jobs		Deadline jobs		
Position	1	2	3	4	5
Job index	①	②	③	④	⑤

Fig. 3. Initial sequence for modified Wilkerson-Irwin algorithm.

Position	1	2	3	4	5
Job index	①	②	④	③	⑤

Fig. 4. Intermediate sequence of modified Wilkerson-Irwin algorithm.

Position	1	2	3	4	5
Job index	①	④	②	③	⑤

Fig. 5. Intermediate sequence of modified Wilkerson-Irwin algorithm.

TABLE 1

Sequences of the Modified Wilkerson-Irwin Algorithm

Pass No.	Comparison of jobs	i^*,j^*	Jump	Starting sequence	Ending sequences
1	2,3	2,3	no	1,2,3,4,5	1,2,3,4,5
2	3,4	3,4	yes	1,2,3,4,5	1,2,4,3,5
3	2,4	2,3	yes	1,2,4,3,5	1,4,2,3,5
4	1,4	1,2	no	1,4,2,3,5	1,4,2,3,5
5	3,5	4,5	no	1,4,2,3,5	1,4,2,3,5

sequences of jobs of each "pass" of the MWI algorithm are shown. A "pass" of the MWI algorithm is defined to be the application of steps 3 through 5 of the MWI algorithm.

COMPUTATIONAL COMPLEXITY

The lower bound for the number of job comparisons occurs when no jumps are made. This lower bound is $n-n_2$ if n_2 is not equal to zero, and $n-1$ if n_2 is equal to zero. The upper bound results when a jump occurs at every comparison. The result for this bound is given in the following theorem.

THEOREM. *The upper bound for the number of comparisons in the modified Wilkerson-Irwin algorithm is*

$$(n-n_2)(n+n_2-1)/2, \tag{7}$$

where

n = *total number of jobs, and*
n_2 = *number of turnaround jobs.*

Proof. Consider the case when $n_2=0$. In general, consider the exchange between the jobs in positions j and $j+1$. If a jump occurs, a maximum of $j-1$ "backward looks" is generated, and j takes on values of 1 through $n-1$. Thus, the upper bound for the number of computations has the summation $\sum_{j=1}^{n-1}[1+(j-1)]$, which is equal to $n(n-1)/2$. When n_2 is not equal to zero, the summation is $\sum_{j=n_2}^{n-1} j$, which is equal to $(n-n_2)(n+n_2-1)/2$.

Thus, the lower bound is of order n and the upper bound of order n^2. For $n \geqslant 5$ the upper bound is considerably better than the branch and bound algorithm, which has an upper bound of order 2^n.

V. EXPERIMENTAL INVESTIGATIONS AND CONCLUSIONS

Let $Z^*(t_0)$ be the optimal value of the objective function [Eq. (6)], and $Z(t_0,A)$ be the value of Eq. (6) for the solution yielded by algorithm A. Then the degree of optimality of algorithm A is defined to be

$$\mathrm{DO}(A)=\frac{Z^*(t_0)}{Z(t_0,A)}\times 100. \tag{8}$$

As mentioned earlier, the calculation of the optimal solution needed to obtain $Z^*(t_0)$ is very time consuming and prohibitively expensive. A comparison was

therefore made between the MWI algorithm and phase 2 of Petersen's reordering algorithm[1] [9] which yielded optimal solutions in almost all cases. Since $Z^*(t_0)$ is not explicitly obtained, the relative degree of optimality of an algorithm instead of its degree of optimality is relevant. The relative degree of optimality of an algorithm is defined as

$$\text{RDO}(A) = \frac{Z(t_0, \text{P2}) \times 100}{Z(t_0, A)} \tag{9}$$

where $Z(t_0, \text{P2})$ is the value of the objective function for the solution yielded by phase 2 of Petersen's algorithm.

A simulation experiment was designed to compare the MWI algorithm and phase 1 of Petersen's algorithm. The comparison was made for the measures of relative degree of optimality and computational speed. The input parameters to the experiment are:

(1) q, the proportion of deadline jobs.
(2) Mean elapsed times of deadline and turnaround jobs. Let these be $\bar{E}_d$ and $\bar{E}_t$ respectively.
(3) Distribution of elapsed times for deadline and turnaround jobs.
(4) Distribution and means of cost for deadline and turnaround jobs.

In the actual experiment t_0 was set to 0, the elapsed times of deadline jobs were distributed uniformly between 15 and 35, the elapsed times of turnaround jobs were distributed uniformly between 1 and 29, the costs of deadline jobs were uniformly distributed between 10 and 15, and the costs of turnaround jobs were uniformly distributed between 5 and 10. The job sets consisted of 8, 10, 12, 15 and 20 jobs, while q took on values of 0.2, 0.4, 0.6, 0.8 and 1.0.

To determine the accuracy of Petersen's algorithm, optimal solutions to selected job sets were obtained using the branch and bound technique of Elmaghraby [6]. The average computational time of the branch and bound algorithm for these was in the range of 15 seconds (IBM 360/67). In these cases, phase 2 of Petersen's algorithm yielded optimal solutions. However, the branch and bound algorithm did not converge to an optimal solution after 30 seconds of CPU time for a few job sets of 12 jobs.

Table 2 displays the relative degree of optimality and computational complexity of the MWI and Petersen's algorithms. Note that by definition, the RDO of phase 2 of Petersen's algorithm is 100%. Under the heading "Relative

[1]Petersen's algorithm consists of three phases, which are composed of certain basic reordering operations. Petersen tested 197 problems, the problem sizes of which ranged from ten to forty jobs. The completion of phase 2 yielded optimal solutions for 195 of these problems. One 20 job solution was non-optimal by 1.0% and one 40 job solution was non-optimal by 3.2% at the end of phase 2.

Degree of Optimality" there are two columns for each algorithm—one with a prime and one without a prime. The average RDO's for the columns with the prime are calculated as follows. For each job set in the row, the average is first calculated over the subset of jobs for which the RDO is less than 100%. These averages are then averaged over the number of job sets for that row and entered in Table 2. The averages for the columns without the prime are taken over all jobs, regardless of RDO. The difference between the RDO's for the columns with and without the prime for a given row yields a measure of the variability of the RDO for that row.

It is seen that in most cases phase 1 of Petersen's algorithm yields a better relative degree of optimality. The maximum difference of the RDO between MWI and phase 1 of Petersen's algorithm is 0.54%, and that between MWI and Phase 2 of Petersen's algorithm is 0.79%. The ratio of the computational time for phase 1 of Petersen's algorithm and the MWI algorithm is at least 150 for problem sizes larger than 10; the ratio between phase 2 and the MWI algorithm is even greater. In view of these considerations, the MWI algorithm is to be preferred.

In conclusion, it has been demonstrated that the modified Wilkerson-Irwin algorithm, which is applicable to monoprogrammed computers with mixed job streams and variable cost rates, provides near-optimal solutions with reasonable computational speed. A related study by Das [5] provides experimental evidence that the MWI algorithm can be effectively used for job scheduling in multiprogrammed computers.

APPENDIX—THE MODIFIED WILKERSON-IRWIN RULE

Phase 1 of Wilkerson and Irwin's original algorithm (in which $C_1 = 1$ for all i) is based on a decision rule applied to a set of jobs which have been arranged in increasing order of due dates. The decision rule is as follows: When

TABLE 2

Experimental Results of MWI and Petersen Algorithms

Prog. size	No. of Configurations	Relative degree of optimality				CPU Times (msec)		
				Petersen			Petersen	
		MWI	MWI'	Phase 1	Phase 1'	MWI	Phase 1	Phase 2
8	25	99.833	97.910	99.682	96.120	67.12	212.16	651.20
10	25	99.826	99.015	99.678	95.975	63.32	556.32	1595.28
12	25	99.513	96.042	99.812	99.267	73.32	1056.88	3631.20
15	25	99.254	97.296	99.785	98.924	92.4	2972.52	10798.12
20	9	99.210	98.578	99.606	99.291	114.89	13924.56	62338.44

comparing jobs i and j at time t, schedule the job with the earlier due date first unless $t+\max(E_i, E_j) \geqslant \max(D_i, D_j)$, in which case schedule the shorter task first.

The following rule holds in the case when not all C_i's are 1 and is a generalization of the Wilkerson-Irwin rule. A complete derivation of this rule is given in [5].

MWI RULE. If we let i and j be two jobs such that D_i is less than or equal to D_j, then at time t we schedule the job i first if either of the following conditions hold:

(1) $t+\max(E_i, E_j) \leqslant \max(D_i, D_j) = D_j$ and one of:

(a) $t+E_i+E_j \leqslant D_j$;

(b) $t+E_i > D_i$ and $t+E_i+E_j > D_j$ and
$$\frac{t+E_j-D_j}{C_i}+\frac{E_i}{C_i}-\frac{E_j}{C_j} \leqslant 0;$$

(c) $t+E_i \leqslant D_i$ and $t+E_i+E_j > D_j$
$$\frac{t+E_j-D_j}{C_i}-\frac{t+E_i+D_i}{C_j}+\frac{E_i}{C_i}-\frac{E_j}{C_j} \leqslant 0.$$

(2) $t+\max(E_i, E_j) > \max(D_i, D_j) = D_j$ and one of:

(a) $t+E_i > D_i$; $t+E_j > D_j$ and $\dfrac{E_i}{C_i}-\dfrac{E_j}{C_j} \leqslant 0$;

(b) $E_i > E_j$; $t+E_j \leqslant D_j$ and
$$\frac{t+E_j-D_j}{C_i}+\frac{E_i}{C_i}-\frac{E_j}{C_j} \leqslant 0;$$

(c) $E_i \leqslant E_j$; $t+E_i \leqslant D_i$ and
$$\frac{E_i}{C_i}-\frac{E_j}{C_j}-\frac{t+E_i-D_i}{C_j} \leqslant 0.$$

Otherwise schedule job j first. If the costs per unit tardiness of job i and j are set to 1, this reduces to Wilkerson and Irwin's original decision rule.

The authors would like to express their gratitude to Professors Kenneth R. Baker and Clifford C. Petersen for their assistance in this research effort.

REFERENCES

1. K. R. Baker, *Introduction to Sequencing and Scheduling*, Wiley, 1974.
2. K. R. Baker and J. B. Martin, Experimental comparison of solution algorithms for single machines tardiness problem, *Naval Res. Logist. Quart.* 21(1): 187–199 (Mar. 1974).
3. F. Baskett, K. M. Chandy, R. Muntz and Fernando G. Palacios, Open, closed and mixed networks of queues with different classes of customers, *J. Assoc. Comput. Mach.* 22(2): 248–260 (Apr. 1975).

4. W. Conway, L. Maxwell and W. Miller, *Theory of Scheduling*, Addison-Wesley, 1967.
5. K. Sundar Das, A scheduling methodology for computer operations, Ph.D. Dissertation, Department of Industrial and Operations Engineering, Univ. of Michigan, Ann Arbor, Mich., 1977.
6. S. E. Elmagrabhy, The one machine sequencing problem with delay costs, *J. Industrial Engineering* 19(2):105–108 (Feb. 1968).
7. H. Emmons, One machine sequencing to minimize certain functions of job tardiness, *Operations Res.* 17(4):701–715 (July 1969).
8. E. L. Lawler, On scheduling problems with deferral costs, *Management Sci.* 9(4):280–288 (July 1963).
9. C. Petersen, Solving sequencing problems through reordering operations, *AIIE Transactions* 5(1):68–73 (Mar. 1973).
10. V. Srinivasan, A hybrid algorithm for the one machine sequencing problem to minimize total tardiness, *Naval Res. Logist. Quart.* 18(3):317–327 (Sept. 1971).
11. L. J. Wilkerson and J. D. Irwin, An improved method for scheduling independent tasks, *AIIE Transactions* 3(3):239–245 (Sept. 1971).

Received May 1978