On Scheduling with Ready Times to Minimize Mean Flow Time

J. S. Deogun
Department of Computer Science, University of Nebraska, Lincoln, Nebraska 68588, USA

An algorithm for sequencing jobs on a single processor with the objective of minimizing the mean flow time, when the jobs may have unequal ready times, is developed. The procedure involves partitioning the problem into subproblems, and solving the subproblems by applying branch and bound techniques. Experimental evaluation shows that the resulting procedure when the partitioning scheme is applied is more efficient than existing algorithms.

1. INTRODUCTION

The problem of sequencing n jobs on one processor or machine has been studied extensively under different assumptions and objective functions. One of the most constraining assumptions many researchers have made in their studies of sequencing problems is the equality of the ready times of the jobs. The *ready time* of a job is the time at which the job is released to the shop by some external job generation process.[1] It is the earliest time that the job can be made available for the machine to start processing it. In the simple problem of sequencing n jobs with equal ready times and no imposed due dates with the objective of minimizing the total flow time, it has been shown[1] that the Shortest Processing Time (SPT) rule provides an optimal solution. According to this rule, jobs are sequenced from beginning to end on the basis of an ascending order of their processing times.

In general, it is conceivable that the job ready times may not be identical. The inequality of the ready times has been recognized in the literature on scheduling problems.[2–5] Dessouky and Deogun[6] studied the sequencing problem with unequal ready times to minimize mean flow time (or equivalently, to minimize mean completion time), and presented an optimal branch and bound scheduling procedure. Later, Bianco and Ricciardelli[7] presented a branch and bound algorithm for the same problem with generalized objective function to accommodate weighted completion times.

One approach to improve the efficiency of a branch and bound procedure is to employ partitioning. A procedure employing partitioning consists of a scheme for dividing the problem into subproblems, a branch and bound procedure for solving the subproblems, and a method of combining the solutions of the subproblems to form the solution of the original problem. In this paper we develop such a partitioning scheme for the problem considered by Dessouky and Deogun.[6]

2. PRELIMINARIES AND PROBLEM FORMULATION

A set N of n jobs, $N = \{i | i = 1, 2, \ldots, n\}$, is to be processed, one job at a time, on a single processor (machine). For each job i, the processing time, p_i, and the ready time, r_i, are given. The ready times r_i are assumed to be given as offsets from an origin, denoted by t_0, such that t_i, the point in time at which job i is ready, is given by $t_i = t_0 + r_i$. Therefore, ready time r_i implies that job i arrives r_i time units after t_0. Ready times r_i are independent of processing times p_i. Completion of all jobs requires establishing a sequence $S = (s_1, s_2, \ldots, s_n)$, where s_y is the index number of the job in position y. When a job parameter or variable is identified by the job's position in a given sequence rather than its index number, the position is indicated as an underlined subscript to the parameter or variable. Thus, $r_{\underline{4}} = r_{s_4}$, means the ready time of the job in position 4 in the sequence considered.

Suppose that a sequence is constructed by adding one job at a time, starting from position 1. At any point, we have a partial sequence S_K of a job set $K \subseteq N$, $S_K = (s_1, \ldots, s_k)$. The earliest start time of a job $i \in N$, $R_i(S_K)$ or simply R_i, and its earliest completion time, $C_i(S_K)$ or simply C_i, are given by:

$$R_i = \begin{cases} r_i & \text{if } i = s_1 & (1a) \\ \max(r_{\underline{y}}, C_{\underline{y-1}}) & \text{if } i = s_y \in K, y \neq 1 & (1b) \\ \max(r_i, C_{\underline{k}}) & \text{if } i \in \bar{K} = N - K & (1c) \end{cases}$$

$$C_i = R_i + p_i. \tag{2}$$

Note that, in eqn (2), if job i has been sequenced (that is, $i \in S_K$), then C_i denotes its actual completion time, otherwise C_i denotes the earliest possible completion time based on S_K.

Another concept used in this paper is that of idle machine time, or simply *idle* time. The idle time of a job i in a sequence $S(N)$, of a set of jobs, N, is the sum of all the intervals, prior to starting i and measured from t_0, during which the machine is not engaged in the processing of a job. Thus, the idle time, $I_i(S)$ or simply I_i, of job $i = s_y$ is given by:

$$I_i = I_{\underline{y}} = I_{\underline{y-1}} + \max\{r_{\underline{y}} - C_{\underline{y-1}}, 0\} \tag{3}$$

where $I_{\underline{0}} = 0$ and $C_{\underline{0}} = t_0$. The idle time preceding s_y as measured from a zero origin, will be $t_0 + I_{\underline{y}}$. The idle time gap between two consecutive jobs s_{y-1} and s_y is given by:

$$I_{\underline{y-1},\underline{y}} = I_{\underline{y}} - I_{\underline{y-1}} = \max\{r_{\underline{y}} - C_{\underline{y-1}}, 0\} \tag{4}$$

The completion time of the job s_y may thus be expressed as:

$$C_{\underline{y}} = C_{\underline{y-1}} + I_{\underline{y-1},\underline{y}} + p_{\underline{y}} \tag{5}$$

CCC–0010–4620/83/0026–0320$04.50

For any job i the flow time F_i and the waiting time W_i are defined as follows: $F_i = C_i - r_i$ and $W_i = R_i - r_i$. For a sequence S, the total completion time $C(S) = \sum_{i=1}^{n} C_i(S)$, the total flow time $F(S) = \sum_{i=1}^{n} F_i(S)$, and the total waiting time $W(S) = \sum_{i=1}^{n} W_i(S)$. Conway, Maxwell and Miller[1] show that a sequence S^* which minimizes $C(S)$ will also minimize $F(S)$ and $W(S)$. In addition, the mean values, $\bar{C}$, $\bar{F}$ and $\bar{W}$, are also minimized. The purpose of this paper is to present a procedure for determining S^* such that $C(S^*) = \min_S C(S)$. This problem, with equal or unequal ready times, is commonly called the $n/1//\bar{F}$ problem[1] or simply the $n/1/\bar{F}$ problem.[6]

3. THE OVERALL APPROACH

The proposed procedure consists of two phases: Phase I—Partitioning, and Phase II—Branch and Bound procedure. In Phase I, the problem is set up and partitioned into subproblems. The optimal subproblems are identified, and Phase I initializes Phase II for every non-optimal subproblem. Thus, Phase I also provides the overall framework for the algorithm. The overall Phase I approach is flow-charted in Fig. 1. Phase II applies implicit enumeration techniques to find the optimal solution for each individual subproblem. The implicit enumeration scheme involves a branch and bound search conducted along the branches of a tree in which a node at level k represents a partial sequence S_K of a set K of k jobs. For each node S_K, we compute a lower bound $\underline{C}(S^* \mid S_K)$ and an upper bound $\bar{C}(S^* \mid S_K)$ on $C(S^* \mid S_K)$, the minimal total completion time of any sequence starting with S_K, that is, conditional on S_K.

A node at level $k+1$ is formed by selecting a job $i \in \bar{K} = N - K$ and adding it to S_K in position $k+1$ to form (S_K, i). At each iteration, the node being expanded is called the *current node* and has the current lowest lower bound. A *closed* (fathomed) *node* is one whose corresponding partial sequence has been found *dominated*, and, hence, is eliminated from consideration. Dominance is tested between nodes generated from the same parent. A partial sequence (S_K, j) is dominated if another partial sequence (S_K, i) exists and $\bar{C}(S^* \mid (S_K, i)) \le \underline{C}(S^* \mid (S_K, j))$; it is *strictly dominated* if $\bar{C}(S^* \mid (S_K, i)) < \underline{C}(S^* \mid (S_K, j))$. We apply a number of tests (pruning rules or elimination criteria) to identify dominated nodes. The set D contains all active nodes ordered by non-decreasing lower bounds, with the current node placed at its beginning. An *active node* is one which has not been found dominated.

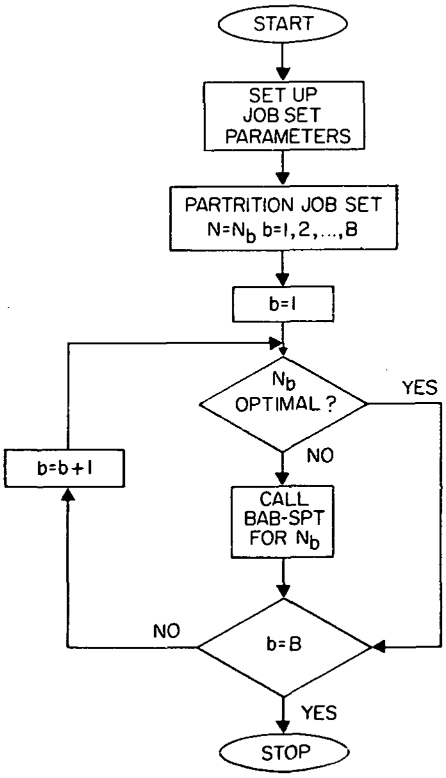

Figure 1. The overall approach.

4. PARTITIONING, OPTIMALITY AND DOMINANCE PROPERTIES

In this section, we present relevant partitioning, optimality and dominance properties of the $n/1/\bar{F}$ problem. We state the properties as theorems with proofs given in the appendix. These properties are useful in developing partitioning and branch and bound procedures. In Section 4.1 theorems relevant to partitioning procedure are presented and a partitioning scheme is described. Properties relevant to the development of the branch and bound procedure: computation of lower bound, testing a sequence for optimality, and elimination criteria, are presented in Section 4.2.

4.1 Partitioning procedure

In a sequence S, define a block $b \subseteq S$ as a set of consecutive jobs with the first job s_u having $r_{\underline{u}} > C_{\underline{u-1}}(S)$ and all other jobs $s_w \in b$ having $r_{\underline{w}} \le C_{\underline{w-1}}(S)$.

Theorem 1. Given a set N of n jobs and an SPT sequence of N, S^P, the completion time of the last job in S^P is an upper bound on the completion time of S^*, the optimal sequence of N.

Following Ref. 7, a sequencing problem can be partitioned into two problems if a partition (N_1, N_2) of the set N exists such that if S_N^*, $S_{N_1}^*$, and $S_{N_2}^*$ are optimal sequences with respect to mean flow time $\bar{F}$ on N, N_1, and N_2, respectively, then $S_N^* = S_{N_1}^* S_{N_2}^*$. The following theorem is immediate.

Theorem 2. If $S_N = (S_K, S_{\bar{K}})$ is a sequence of a job set N, where K and $\bar{K} = N - K$ are two disjoint subsets of N, and S_K and $S_{\bar{K}}$ are such that: (i) $C_{\underline{k}}(S_K) \ge C_{\underline{k}}(S_K^*)$, that is the completion time of the last job in S_K is an upper bound on the completion time of the last job in the optimal sequence S_K^*, and that (ii) $r_h \ge C_{\underline{k}}(S_K)$, where $r_h = \min_{x \in S_{\bar{K}}} r_{\underline{x}}$, then N can be partitioned into K and $\bar{K}$ such that an optimal sequence of N is obtained by optimizing K and $\bar{K}$ separately.

Corollary 2.1. In an SPT sequence $S = S_N^P$, if a block b exists such that the first job in b, s_h, has $r_{\underline{h}} = \min_{h \le x \le n} r_{\underline{x}}$, then N can be partitioned into two disjoint subsets: K, containing all jobs preceding s_h, and $\bar{K} = N - K$, such that an optimal sequence of N is obtained by optimizing K and $\bar{K}$ separately.

Theorem 3. A sufficient condition for the optimality of a block is that the first job in the block has the smallest ready time of any job in the block, and that all jobs in the block are sequenced according to SPT.

The above theorems lead to the following partitioning scheme:

Partitioning scheme. The following procedure is applied to S_N^P to partition the job set into blocks and identify optimal blocks:

(1) In a forward pass, identify each job which starts a block.
(2) In a backward pass, calculate $r_{m,y} = \min_{y \le x \le n} r_{\underline{x}}$ for each s_y, $1 \le y \le n$ and classify each block according to the status of its first job, say h, as follows:
   (a) Optimal block: when $r_h = r_{m,h}$. This follows from Theorem 3.
   (b) Non-optimal block: when $r_h > r_{m,h}$.

4.2 Development of branch and bound procedure

4.2.1 Lower bound computation. A lower bound on the value of an objective function, in a problem of minimizing an objective function under constraints, is given by the minimum value of the objective function of a modified problem in which some or all of the constraints are relaxed. It has been shown[1] that Shortest Processing Time (SPT) rule provides an optimal solution for the problem under consideration if inequality of the ready times is relaxed to make all jobs available at the same time. The approach followed in this paper defines the modified problem by constructing a relaxed job set $N' = K \cup \bar{K}'$, where $\bar{K}'$ is identical to $\bar{K}$ except that $r_j = 0$ for all $j \in \bar{K}'$. The conditional optimal sequence is obtained by ordering $\bar{K}'$ according to the SPT rule, yielding $S^*_{\bar{K}'} = S^P_{\bar{K}'}$, where the superscript P indicates that processing time is the sequencing criterion. Therefore,

$$\underline{C}(S \mid S_K) = C^*(N' \mid S_K) = C(S_K, S^P_{\bar{K}'}) \tag{6}$$

For each y satisfying $k + 1 \le y \le n$,

$$C_{\underline{y}}(S') = C_{\underline{k}}(S) + \sum_{x=k+1}^{y} p_{\underline{x}} \tag{7}$$

From eqns (6) and (7) and the definition of $C(S)$,

$$\underline{C}(S \mid S_K) = \sum_{y=1}^{k} C_{\underline{y}} + \sum_{y=k+1}^{n} \left(C_{\underline{k}} + \sum_{x=k+1}^{y} p_{\underline{x}} \right)$$

or

$$\underline{C}(S \mid S_K) = \sum_{y=1}^{k} C_{\underline{y}} + (n-k) C_{\underline{k}} + \sum_{y=k+1}^{n} \sum_{x=k+1}^{y} p_{\underline{x}} \tag{8}$$

Consider now a job $i = s_h \in \bar{K}'$, and suppose that i is pulled from its position $h > k$ and placed in position $k + 1$, with all jobs $s_y \in S^P_{\bar{K}'}$ preceding i (i.e. $k < y < h$) shifted one position later. As i is placed in the new position, it becomes part of the front sequence (S_K, i), and its original ready time r_i is restored to it. The earliest start time $R_i(S_K) = R_{\underline{h}}(S_K)$ and the earliest completion time $C_i(S_K) = C_{\underline{h}}(S_K)$ are given by eqns (1) and (2). A lower bound on the new partial sequence similar to eqn (8) may be derived as follows, where the position subscripts refer to jobs in $S' = (S_K, S^P_{\bar{K}'})$:

$$\underline{C}(S \mid S_K, i)) = \sum_{y=1}^{k} C_{\underline{y}} + C_i(S_K) + \sum_{y=k+1}^{h-1} \left(C_i(S_K) + \sum_{x=k+1}^{y} p_{\underline{x}} \right) + \sum_{y=h+1}^{n} \left(C_i(S_K) + \sum_{x=k+1}^{y} p_{\underline{x}} \right)$$

or

$$\underline{C}(S \mid (S_K, i)) = \sum_{y=1}^{k} C_{\underline{y}} + (n-k) C_i(S_K) - (n-h) p_i + \sum_{y=k+1}^{n} \sum_{x=k+1}^{y} p_{\underline{x}} - \sum_{y=k+1}^{h} p_{\underline{y}} \tag{9}$$

From eqns (8) and (9) it can be shown that

$$\underline{C}(S \mid (S_K, i)) = \underline{C}(S \mid S_K) + (n-k)(C_i(S_K) - C_{\underline{k}}) - (n-h) p_i - \sum_{y=k+1}^{h} p_{\underline{y}} \tag{10}$$

Substituting eqn (5) in eqn (10) we obtain the expression:

$$\underline{C}(S \mid (S_K, i)) = C(S \mid S_K) + (n-k) I_{\underline{k},i} + (h-k-1) p_i - \sum_{y=k+1}^{h-1} p_{\underline{y}}$$

which is a computationally efficient expression and allows us to compute a new lower bound corresponding to each job i that is eligible for placement in position $k + 1$ after S_K.

4.2.2 Optimality test. The following test is applied to the initial SPT sequence S_N^P: If in every block the first job has the smallest ready time and all jobs follow an SPT order, then the sequence is optimal.

Application of this test will obviate the need for performing the branch and bound procedure whenever the sequence passes the test. This optimality test is based on the following lemma:

Lemma 1. A sufficient condition for the optimality of a sequence is that in every block the first job has the smallest ready time of any job in the block, and that all jobs in the block are sequenced according to SPT.

4.2.3 Elimination criteria. The elimination criteria employed in the branch and bound procedure are based on the dominance properties presented below. Theorems 4–6 are stated here for completeness, their proofs can be found in Ref. 6. The proof of Theorem 7 is given in Appendix 1. We will identify R_i and C_i by the sequence in question whenever it is not clear from the context.

Theorem 4. Given a job set N and a partial sequence S_K, $K \subset N$, if a job $i \in \bar{K} = N - K$ has $p_i \le p_j$, all $j \in \bar{K}$, and a job $h \in \bar{K}$ has $R_h(S_K) \ge R_i(S_K)$, then i dominates h in position $k + 1$. If $R_h(S_K) > R_i(S_K)$, then (S_K, i) strictly dominates (S_K, h).

Corollary 4.1. If a job set N can be divided into two subsets N_1 and N_2 such that for every $i \in N_1$ and $j \in N_2$, $r_i \leq r_j$ and $p_i \leq p_j$, then an optimal solution exists in which N_1 precedes N_2.

Theorem 5. Consider a job set N of n jobs, and a partial sequence S_K of k jobs, $K \subset N$, and let job $i \in \bar{K}$ have $C_i(S_K) \leq C_j(S_K)$, all $j \in \bar{K}$. A partial sequence (S_K, j) is strictly dominated if $r_j \geq C_i(S_K)$.

Theorem 6. Given S_K, $K \subset N$, and two jobs $i, j \in \bar{K}$, if $p_j \leq p_i$ and $C_j(S_K) \geq C_i(S_K)$ then (S_K, i) dominates (S_K, j).

Theorem 7. Given $S = (S_K, S_{\bar{K}}^P)$ and a job $i \in \bar{K}$ in position $h(h > k)$ such that $C_i(S_{\underline{K}}) - C_{\underline{k}} \geq I_{\underline{h-1}} - I_{\underline{k}}$, (S_K, i) dominates (S_K, j) for any $j \in \bar{K}$ if $p_i \leq p_j$ and $R_i(S_K) \leq R_j(S_K)$.

The implication of the condition $C_i' - C_{\underline{k}} \geq I_{\underline{h-1}} - I_{\underline{k}}$ is that when i is placed in position $k+1$, no idle time will exist between i and $s_{\underline{h-1}}$. Theorem 7 stated above is similar to corollary 2 (theorem 5) of Bianco and Ricciardelli.[7] However, neither reduces to the other by simple substitution of general as opposed to equal weights.

5. THE ALGORITHM

Phase I (Partitioning)

A. Initialization

(1) Let N be the set of n jobs $\{i \mid i = 1, 2, \ldots, n\}$. Define values r_i and p_i for all $i \in N$.
(2) Arrange the job set N according to SPT and let $S = S_N^P$.

B. Partitioning and optimality test

(3) Partition job set N into B blocks, and let $N = \{N_b \mid b = 1, 2, \ldots, B\}$.
(4) Apply the optimality test of Section 4.2.2. If the sequence is optimal, go to step (9), otherwise, proceed to step (5).
(5) Set $b = 1$.
(6) If N_b is a non-optimal block go to step (7), otherwise, increment $b = b + 1$. If $b > B$, go to step (9), otherwise, repeat step (6).
(7) Initialize Phase II to optimize N_b.
(8) Increment $b = b + 1$, and go to step (6).

C. Computation of optimal values

(9) Let S^* denote the resulting optimal sequence. Compute the optimal completion time $C(S^*)$ and the optimal flow time $F(S^*)$.

Phase II (Branch and Bound)

This phase assumes that a job set N_b of n_b jobs and their ready times r_i and processing times p_i for all $i \in N_b$ have been passed to it by Phase I.

A. Initialization

(1) Set $k = 0$, $K = \emptyset$, $S_K = \emptyset$ and $\bar{K} = N_b - K$. Initialize $R_i(S_K) = r_i$ for all $i \in N_b$. Set $C(S_K)$, the completion time of the partial sequence S_K, at zero.
(2) As jobs in N_b are already sequenced according to SPT, let $S_b = (S_K, S_{\bar{K}}^P)$. Compute $C(S_b) = \sum_{i=1}^{n_b} C_i(S_b)$ and set the upper bound $\bar{C}(S_b^*) = C(S_b)$.
(3) Define a relaxed set N_b' such that $r_x' = 0$ and $p_x' = p_x$, all $x \in \bar{K}$ and set $\underline{C}(S_b^*) = C(S_{b'})$, where $S_{b'} = (S_K, S_{\bar{K}}^P)$, and set $D = \emptyset$.

B. Branching and pruning

(4) Let the eligible set $E = \bar{K}$. Compute $C_m(S_K) = \min_{x \in \bar{K}} C_x(S_K)$ and eliminate all j from E for which $r_j \geq C_m(S_K)$ (Theorem 5).
(5) Compute the lower bound $\underline{C}(S_b \mid (S_K, i))$ from eqn (12) for all $i \in E$, and eliminate i from E if its lower bound violates the upper bound constraint.
(6) If $s_{\underline{k+1}} \in E$ and $R_{\underline{k+1}}(S_K) = \min_j R_j(S_K)$, $j \in E$, then eliminate all jobs from E except $s_{\underline{k+1}}$ and go to step (8), otherwise, proceed to step (7) (Theorem 4).
(7) Eliminate all jobs j from E, if for $i, j \in E$, $p_j \leq p_i$ and $C_j(S_K) \geq C_i(S_K)$ (Theorem 6).
(8) Eliminate all j from E, if for $i, j \in \bar{K}$, $p_i < p_j$, i.e. i precedes j in $S_{\bar{K}}$, $R_i(S_K) \leq R_j(S_K)$ and $C_i(S_K) - C_{\underline{k}}(S_K) \geq I_{\underline{h-1}} - I_{\underline{k}}$, where $s_{\underline{h}} = i$ (Theorem 7).
(9) For each $i \in E$, compute the total completion time $C(S_b \mid (S_K, i))$, and update the upper bound $\bar{C}(S_b^*)$ as follows: $\bar{C}(S_b^*) = \min\{\bar{C}(S_b^*), \min_{(S_K, i)} C(S_b \mid (S_K, i)), i \in E\}$.

D. Updating set D

(10) Add node (S_K, i) for each $i \in E$ to the set D, except those which violate upper bound constraints, that is, for which $\underline{C}(S_b^* \mid (S_K, i)) > \bar{C}(S_b^*)$.
Remove the node corresponding to S_K from D.

E. Determination of current node

(11) Identify the node in D that has the lowest lower bound. Let S_Q be this node. Set $K = Q$, $\bar{K} = N_b - K$ and, if $\bar{K} = \emptyset$ or LB = UB, return to Phase I, otherwise let $S_{\bar{K}} = S_{\bar{K}}^P$.
(12) Update $R_i(S_K) = \max\{r_i, C(S_K)\}$ and $C_i(S_K) = R_i(S_K) + p_i$ for all $i \in K$. Go to step (4).

6. AN EXAMPLE

In this section the procedure is illustrated by the following example. The example is specially constructed to demonstrate most of the features of the procedure in one example. For simplicity the given set of jobs is assumed to be in SPT order.

Detailed computation up to the first iteration of phase II for block No. 3 is given below, where numbers in parentheses identify step number and phase of the SPT algorithm, e.g. (I, 3) denoted step (3) in phase I.

(I, 1) $n = 20$, and values of r_i and p_i for $i = 1, 2, \ldots, 20$ are given in Table 1.
(I, 2) The jobs are already in an SPT order.
(I, 3) The job set N is partitioned into 6 blocks as shown in Table 2. Beginnings of blocks are marked by ← in Table 1.
(I, 4) The optimality test of Section 4.2.2 fails, therefore, proceed to step (I, 5).

Table 1.

Job index i	Ready time r_i	Processing time p_i	Earliest start time R_i	Earliest completion time C_i	New block
1	0	3	0	3	←
2	2	5	3	8	
3	6	7	8	15	
4	16	8	16	24	←
5	17	11	24	35	
6	73	13	73	86	
7	60	18	86	104	
8	40	20	104	124	
9	37	29	124	153	
10	90	35	153	188	
11	190	40	190	230	←
12	107	43	230	273	
13	212	57	273	330	
14	337	60	337	397	←
15	218	68	397	465	
16	467	80	467	547	←
17	470	93	547	640	
18	550	93	640	733	
19	551	94	733	827	
20	586	98	827	925	

(I, 5) Set $b = 1$.
(I, 6) N_1 is optimal, therefore, increment $b = 2$, and repeat step (I, 6).
(I, 6) N_2 is optimal, therefore, $b = 3$, and repeat step (I, 6).
(I, 6) $N_3 = (6, 7, 8, 9, 10)$ is non-optimal, therefore initialize Phase II.
(II, 1) Set $k = 0$, $K = \emptyset$, $S_K = \emptyset$ and $\bar{K} = (6, 7, 8, 9, 10)$. Set $C(S_K) = 0$, $\underline{R}_1(S_K) = 73$, $\underline{R}_2(S_K) = 60$, $\underline{R}_3(S_K) = 40$, $\underline{R}_4(S_K) = 37$ and $\underline{R}_5(S_K) = 90$.
(II, 2) $\bar{S} = S_{\bar{K}}^{P} = (6, 7, 8, 9, 10)$, and $C(\bar{S}) = 86 + 104 + 124 + 153 + 188 = 655$.
(II, 3) $\underline{C}(S^*) = 13 + 31 + 51 + 80 + 115 = 290$.
(II, 4) $E = (6, 7, 8, 9, 10)$ and $\underline{C}_3(S_K) = 60$ is the minimum. Because $\underline{R}_1(S_K) = 73$, $\underline{R}_2(S_K) = 60$, $\underline{R}_5(S_K) = 90$ are all ≥ 60, therefore, removing jobs 6, 7, and 8 from E, $E = (8, 9)$.
(II, 5) $C(S^* \mid (S_K, 8)) = 499$ and $C(S^* \mid (S_K, 9)) = 539$.
(II, 6) Pruning rule No. 1 does not apply.
(II, 7) Pruning rule No. 2 does not apply.
(II, 8) Pruning rule No. 4 does not apply.
(II, 9) $\underline{C}(S \mid (S_K, 8)) = 499$ and $C(S \mid (S_K, 9)) = 539$.
(II, 10) $\bar{C}(S^*) = \min(655, 504, 539) = 504$.
(II, 11) Only the node corresponding to job 8 is added to the set D, because $\underline{C}(S^* \mid (S_K, 9)) = 539 > \bar{C}(S^*) = 504$.

Table 2.

Block number	Type	Job subset				
1	Optimal	1	2	3		
2	Optimal	4	5			
3	Non-optimal	6	7	8	9	10
4	Optimal	11	12	13		
5	Non-optimal	14	15			
6	Optimal	16	17	18	19	20

(II, 12) Locate the node in D which has the lowest lower bound. $k = 1$, $K = (8)$, $\bar{K} = (6, 7, 9, 10)$, LB $= 499$ and UB $= 501$. Neither $\bar{K} = \emptyset$, nor LB $=$ UB.
(II, 13) $\underline{R}_2(S_K) = 73$, $\underline{R}_3(S_K) = 60$, $\underline{R}_4(S_K) = 60$ and $\underline{R}_5(S_K) = 90$. $\underline{C}_2(S_K) = 86$, $\underline{C}_3(S_K) = 78$, $\underline{C}_4(S_K) = 80$, and $\underline{C}_5(S_K) = 115$, and go to step (II, 4).

This completes the first iteration in Phase II for block N_3. The complete tree of optimal solutions generated is shown in Fig. 2. After N_3 has been optimized, the control returns to step (8) in Phase I. Block N_4 is again optimal, so the procedure moves to block N_5, which is not optimal, and thus Phase II is again initialized. The tree of solutions generated is shown in Fig. 3. Block N_6, the last block, is again optimal, so the solution procedure is terminated. The optimal sequence is, therefore, (1, 2, 3, 4, 5, 8, 7, 6, 9, 10, 11, 12, 13, 15, 14, 16, 17, 18, 19, 20) and minimum value of total completion time is 5950.

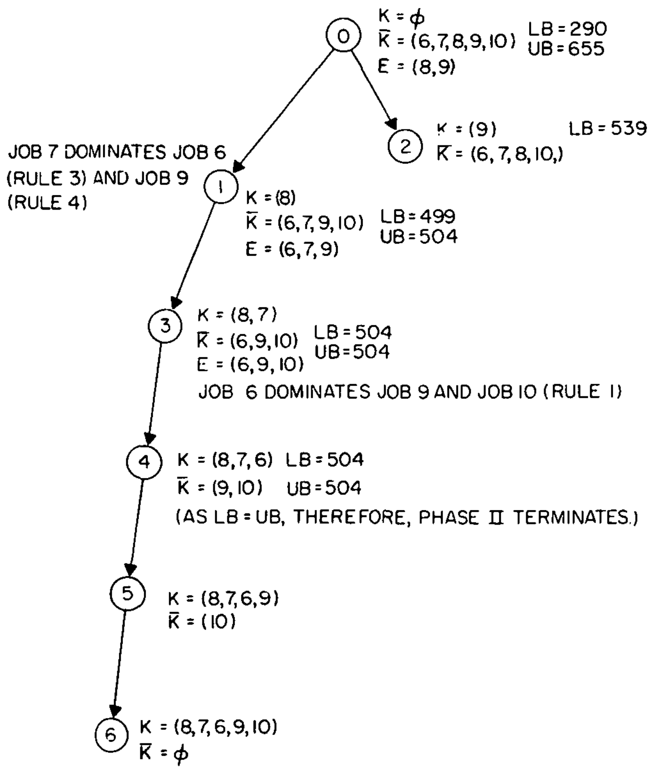

Figure 2.

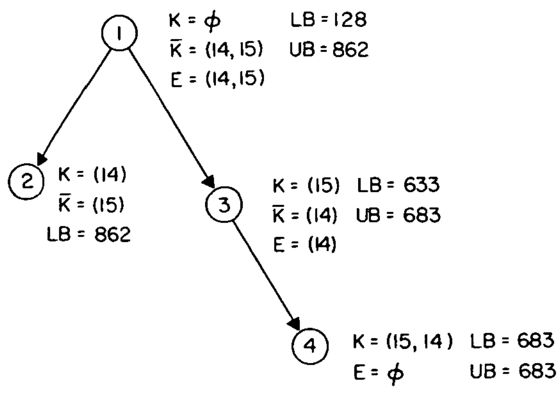

Figure 3.

7. EXPERIMENTAL EVALUATION

The algorithm was programmed and tested on a Cyber 175, using a FORTRAN G compiler. Two tests were conducted. The first test consisted of 220 job sets (problems), generated randomly from independent probability distributions of r_i and p_i. Four distributions for r_i and eleven for p_i were combined to produce 44 distribution pairs, and each distribution pair used to generate five problems. Each problem in this test consisted of 20 jobs. For the second test one job set was generated from each of the forty-four distribution pairs, except that each problem in this test consisted of 50 jobs. The probability distributions were chosen to be uniform to avoid biasing particular values within their ranges, and to eliminate the need for truncation. They are specified by the upper and lower limits of their ranges. The experimental design and the parameters used to generate problem sets for both tests conducted on the algorithm presented in this paper are exactly the same as those used by Dessouky and Deogun.[6] Thus, the two algorithms, Dessouky and Deogun's (existing) algorithm and the (new) algorithm presented in this paper, were tested on the same job sets using the same machine, which makes the comparison of the two algorithms objective.

For the new algorithm, Tables A.1 and A.2 in Appendix 2 show detailed experimental results for the first and second tests, respectively. Detailed experimental results for the existing algorithm can be found in Ref. 6. Table 3 summarizes the comparative performance of the existing and the new algorithms.

Table 3.

Test number	Limit on computation time	Performance measure	Existing algorithm	New algorithm	Percentage improvement
	2.0 s	No. of problems solved	217	217	
1		Computation time			
		Mean	0.057	0.049	14.8
		Max	1.880	1.830	2.6
	none	Computation time			
		Mean	0.112	0.083	25.8
		Max	4.610	3.470	24
2	none	Computation time			
		Mean	0.960	0.804	16.2
		Max	8.470	7.690	9

As evident from Table 3 the new algorithm presented here is more efficient as compared to the existing algorithm. Mean computation time for test one showed 14.8% improvement under a two-second limit and 25.8% improvement under no time limit. For test two, mean computation time, for all problems, showed 16.2% improvement. Though the new method is more efficient, the storage requirements, as evident from the number of nodes generated, are higher for the new method as compared to the existing method. Mean and maximum numbers of nodes generated for both existing and new algorithms are given in Table 4.

The effectiveness of the partitioning method employed is demonstrated by the fact that the maximum and average number of partitions generated by the procedure were 4 and 1.4 for the first test and 6 and 1.8 for the second test. As no partitioning scheme can be defined for the algorithm presented by Dessouky and Deogun,[6] we note the possibility that problem partitioning may not only depend on the structure of the problem but also on the structure of the procedure employed.

Table 4.

Test number		Number of nodes generated: Existing algorithm	New algorithm
1	mean	13	34
	max	514	899
2	mean	47	81
	max	540	783

8. CONCLUSIONS AND RECOMMENDATIONS

This paper presents a procedure for solving the $n/1/\bar{F}$ problem with unequal ready times. The procedure involves partitioning the problem into subproblems, and solving the subproblems by applying a branch and bound technique. Experimental results obtained are compared to those obtained by Dessouky and Deogun[6] for the same problem. The comparative performance analysis shows that the new algorithm presented here may be 16 to 25% better depending upon the size of the problem and the distributions from which problem sets are generated. However, storage requirements for the new algorithm may be higher as compared to those of the existing algorithm. Therefore, the existing method is recommended where computer storage is a critical factor. However, if computational efficiency is the consideration, the new method should be the choice.

REFERENCES

1. R. Conway, W. Maxwell and L. W. Miller, *Theory of Scheduling*, Addison-Wesley, Reading, Massachusetts (1967).
2. S. P. Bansal, Minimizing the sum of completion times of *n* jobs over *m* machines in a flowshop—a branch and bound approach. *AIIE Transactions* **9** (3), 306–311 (1977).
3. P. Bratley, M. Florian and P. Robillard, On scheduling with earliest start and due dates with application to computing bounds for the (*n*/*m*/*G*/*F*max) problem. *Naval Research Logistics Quarterly* **20**, 57–67 (1973).
4. M. I. Dessouky and C. R. Mergenthaler, The one-machine sequencing problem with early starts and due dates. *AIIE Transactions* **4** (3), 214–222 (1972).
5. J. K. Lenstra, A. H. R. Rinnooy Kan and P. Brucker, Complexity of machine scheduling problems. *Annals of Discrete Mathematics* **1** (4), 343–362 (1977).
6. M. I. Dessouky and J. S. Deogun, Sequencing jobs with unequal ready times to minimize mean flow time. *SIAM Journal on Computing* **10** (1), 192–202 (1981).
7. L. Bianco and S. Ricciardelli, Scheduling of a single machine to minimize total weighted completion time subject to release dates. *Naval Research Logistics Quarterly* **29**, 151–167 (1982).

Received November 1982

APPENDIX 1. PROOFS OF THEOREMS

Proof of Theorem 1

The theorem states that $C_n^P \geq C_n^*$. Note that for any sequence, the idle time preceding job s_k, I_k, is given by $I_k = C_k - \sum_{x=1}^{k} p_x$. Since $\sum_{x=1}^{n} p_x^P = \sum_{x=1}^{n} p_x^*$, the statement is equivalent to $I_n^P \geq I_n^*$. The theorem is proved by induction. For position 1, $I_1^* \leq I_1^P$ otherwise $r_1^* > r_1^P$ and from theorem 4, s_1^P dominates s_1^* in position 1. Assume that $I_k^* \leq I_k^P$; it will be shown that $I_{k+1}^* \leq I_{k+1}^P$.

Two possibilities exist: (i) $I_{k+1}^* = I_k^*$. This implies $I_{k+1}^* = I_k^* \leq I_k^P \leq I_{k+1}^P$. (ii) $I_{k+1}^* > I_k^*$. This implies $R_{k+1}^* = r_{k+1}^* > C_k^*$. If $R_{k+1}^* > R_{k+1}^P$ then $r_{k+1}^* = R_{k+1}^* > R_{k+1}^P \geq R_x^P \geq r_x^P$, $1 \leq x \leq k+1$, and $s_{k+1}^* \neq s_x^P$, $1 \leq x \leq k+1$. Locate within the set of jobs $J = \{j \mid j \in (S_K^P, s_{k+1}^P), j \notin (S_K^*, s_{k+1}^*)\}$, a job i such that $p_i = \min_{j \in J} p_j$. Note that since $i \in (S_K^P, s_{k+1}^P)$, $r_i > r_{k+1}^*$ and from the SPT property $p_i \leq p_x^P$, $k+1 \leq x \leq n$. Therefore, $R_i(S_K^*) = \max(r_i, C_K^*) < r_{k+1}^* \leq R_{k+1}^*$, and from Theorem 4, i strictly dominates s_{k+1}^* in position $k+1$ in S^*, contradicting the assumption of optimality of S^*. Therefore, $R_{k+1}^* \leq R_{k+1}^P$. Since for the SPT property

$$\sum_{x=1}^{k} p_x^* \geq \sum_{x=1}^{k} p_x^P, \; 1 \leq k \leq n,$$

$$I_{k+1}^* = R_{k+1}^* - \sum_{x=1}^{k} p_x^* \leq R_{k+1}^P - \sum_{x=1}^{k} p_x^P = I_{k+1}^P.$$

By induction $I_n^* \leq I_n^P$, and $C_n^* \leq C_n^P$. ▮

Proof of Corollary 2.1

Let $r_h = \min_{1 \leq x \leq y} r_x$. From eqn (1) and the definition of a block, $r_h \geq C_{h-1}(S)$. Denote the sequence of K in S by S_K and its optimal sequence by S_K^*. Theorem 1 states that $C_{h-1}(S_K^*) \leq C_{h-1}(S_K)$, thus $C_{h-1}(S_K^*) \leq r_h$. Therefore, the conditions of Theorem 2 are satisfied. ▮

Proof of Theorem 3

In any block $b \in S$, in the given sequence, the first job has the smallest r_j and p_j of any $j \in B$, hence from Theorem 4 it is optimally placed in its position within the block. From Theorem 4, the same is true for each succeeding job i in b, since i will have the smallest R_i and p_i of the remaining jobs. Therefore, jobs are optimally placed within b. From the definition of a block and the conditions on the ready times stated in the theorem, the ready time of any job in a block will be greater than or equal to the completion time of any job in a preceding block. Therefore, according to Theorem 5 a job in any block is not eligible for a position in a preceding block and no job shifts between blocks in S will reduce the total completion time of S. Since shifts within and between blocks will not improve S, S is optimal. ▮

Proof of Lemma 1

Lemma 1 follows directly from corollary 2.1 and Theorem 3. ▮

Proof of Theorem 7

From the statement of the theorem $C_j(S_K) = R_j(S_K) + p_j \geq R_i(S_K) + p_i = C_i(S_K)$, which means that the condition on idle times also applies to j. Let $j = s_x$ in S, where $x > h$. Consider the sequence S^j, where j is placed in position $k+1$ and all $s_y \in S$, $k+1 \leq y \leq x-1$ are shifted one position later. The jobs now in positions $k+2$ to x, including i, are ordered according to SPT and each one starts at the completion of the preceding one. Therefore, each one is ordered optimally, conditional on the preceding sequence. Consequently, an optimal sequence conditional on (S_K, j), say S^{j*}, will contain the first x jobs in S^j as a front sequence with i in position $h+1$. Now switch i and j to form S^i. The inequality $C_i(S_K) \leq C_j(S_K)$ implies that $C_{k+1}(S^i) \leq C_{k+1}(S^j)$. Furthermore, since all s_y, $k+2 \leq y \leq h$, are identical in both S^i and S^j, and no idle time gaps exist between s_{k+1} and s_h in S^i, $C_y(S^i) \leq C_y(S^{j*})$, all y such that $k+2 \leq y \leq h$. Since the partial sequence of jobs between positions $k+1$ and $h+1$ inclusive is the same in S^i and S^{j*}, except that the first and last jobs are switched, and since $R_{k+1}(S^i) \leq R_{k+1}(S^{j*})$ and no gaps exist within any partial sequence, $C_{h+1}(S^i) \leq C_{h+1}(S^{j*})$. From this and the fact that the set of jobs succeeding s_{h+1} is the same in both S^i and S^{j*}, we conclude that

$$\sum_{y=h+2}^{n} C_y(S^{i*}) \leq \sum_{y=h+2}^{n} C_y(S^{j*}),$$

where S^{i*} is the optimal sequence conditional on a front sequence S^i. Therefore, $C(S^{i*}) \leq (S^{j*})$, which by definition means that (S_K, i) dominates (S_K, j). ▮

APPENDIX 2

Tables A.1 and A.2 show detailed experimental results for the first and second tests, respectively.

Table A.1. Summary of results: branch and bound—SPT procedure for *n*/1/*F*

Range of ready times	Range of processing times: 1–25		26–50		1–75		51–75		1–125		76–100		1–175		101–125		1–225		126–150		1–275			
0–200	********		23	5	19	4	3	0	30	4	1	0	4	1	2	0	2	0	1	0	3	0		
	********		12	2	36	8	2	0	76	12	1	1	22	3	2	0	27	3	1	0	4	1		
	155	46	3	1	148	37	2	1	9	2	1	0	6	1	2	1	6	1	1	0	5	1		
	217	54	6	2	15	4	2	0	5	1	1	0	5	0	1	0	2	0	1	0	1	0		
	112	37	4	1	25	4	4	1	38	6	2	0	1	1	1	1	9	2	1	0	1	0		
	Min	37	Min	1	Min	4	Min	0	Min	1	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0
	Max	54	Max	5	Max	37	Max	1	Max	12	Max	1	Max	3	Max	1	Max	3	Max	0	Max	1	Max	54
	Mean	45	Mean	2	Mean	11	Mean	0	Mean	5	Mean	0	Mean	1	Mean	0	Mean	1	Mean	0	Mean	0	Mean	4
25–175	280	64	4	1	14	2	2	1	52	7	2	0	7	1	1	0	8	1	1	0	8	1		
	295	71	6	1	49	8	2	1	6	1	2	1	7	1	2	1	3	0	1	0	3	1		
	171	21	2	1	25	4	3	1	3	1	2	0	4	1	1	0	2	0	2	1	1	0		
	130	23	5	1	10	2	1	0	8	2	1	0	4	1	1	0	2	1	1	0	2	0		
	272	175	3	1	11	1	1	0	5	0	1	0	4	0	1	0	10	2	1	0	14	2		
	Min	21	Min	1	Min	1	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0
	Max	175	Max	1	Max	8	Max	1	Max	7	Max	1	Max	1	Max	1	Max	2	Max	1	Max	2	Max	175
	Mean	70	Mean	1	Mean	3	Mean	0	Mean	2	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	7
50–150	215	48	2	0	7	1	1	1	3	1	1	0	1	0	1	0	3	1	1	0	3	1		
	12	2	2	0	31	5	2	0	2	0	0	0	3	0	1	1	3	0	1	0	3	1		
	36	8	1	1	6	1	1	1	2	1	2	0	2	0	1	0	1	1	1	0	1	0		
	7	1	9	1	19	2	2	0	4	1	1	0	3	1	0	0	1	0	0	1	1	0		
	7	1	3	0	8	1	1	1	4	0	1	0	2	1	2	0	5	1	1	0	1	1		
	Min	1	Min	0	Min	1	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0
	Max	48	Max	1	Max	5	Max	1	Max	1	Max	0	Max	1	Max	1	Max	1	Max	1	Max	1	Max	48
	Mean	12	Mean	0	Mean	2	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	1
75–175	249	66	1	0	8	2	1	0	12	1	1	0	2	1	1	0	1	0	1	1	2	0		
	301	183	4	1	8	1	0	0	7	1	0	1	13	3	1	1	1	0	1	0	2	0		
	********		3	1	6	1	1	0	4	0	1	0	1	0	1	0	2	0	0	0	5	1		
	240	67	3	0	9	1	1	1	5	1	1	1	4	1	1	0	1	0	1	0	2	1		
	20	3	2	1	9	2	1	0	3	1	1	0	3	1	1	0	2	0	1	1	2	0		
	Min	3	Min	0	Min	1	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0
	Max	183	Max	1	Max	2	Max	1	Max	1	Max	1	Max	3	Max	1	Max	0	Max	1	Max	1	Max	183
	Mean	75	Mean	0	Mean	1	Mean	0	Mean	0	Mean	0	Mean	1	Mean	0	Mean	0	Mean	0	Mean	0	Mean	6
	Min	1	Min	0	Min	1	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0	Min	0		
	Max	183	Max	5	Max	37	Max	1	Max	12	Max	1	Max	3	Max	1	Max	3	Max	1	Max	2		
	Mean	51	Mean	1	Mean	4	Mean	0	Mean	1	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0	Mean	0		

Number of Jobs = 20

Minimum computation time	= 0	Minimum number of iterations	= 0
Maximum computation time	= 183	Maximum number of iterations	= 301
Mean computation time	= 4	Average number of iterations	= 15
Number exceeding ½ second	= 12	Minimum number of nodes generated	= 1
Number exceeding 1 second	= 9	Maximum number of nodes generated	= 899
Maximum number of partitions	= 4	Average number of nodes generated	= 34
Average number of partitions	= 1.4		

Note: For each box, the first column is the number of iterations and the second column is computer time in centiseconds. The *s in columns one and two imply that the corresponding problem was not solved in 2 seconds.

Table A.2. Summary of results: branch and bound—SPT procedure for $N/1/F$ (50 jobs in each set)

Range of ready times	Range of processing times: 1–25		26–50		1–75		51–75		1–125		76–100		1–175		101–125		1–125		126–150		1–275	
0–200	273	453	7	82	37	128	2	4	45	61	2	9	4	39	1	5	4	17	2	39	23	48
25–175	482	632	17	37	33	41	1	6	28	57	7	15	13	27	2	17	17	63	3	16	7	21
50–150	267	374	1	5	26	54	4	15	15	35	18	37	6	17	4	21	11	43	5	37	4	27
75–175	427	769	2	25	18	57	2	17	17	21	0	4	10	53	17	37	5	21	11	23	13	21
	Min	374		5		41		4		21		4		17		5		17		16		21
	Max	769		82		128		17		61		37		53		37		63		39		48
	Mean	557		37		70		10		43		18		34		20		36		28		29

Number of jobs = 50

Minimum computation time	= 4	Minimum number of iterations	= 0
Minimum computation time	= 769	Maximum number of iterations	= 482
Mean computation time	= 80	Average number of iterations	= 43
Number exceeding ½ second	= 12	Minimum number of nodes generated	= 1
Number exceeding 1 second	= 5	Maximum number of nodes generated	= 783
Maximum number of partitions	= 6	Average number of nodes generated	= 81
Average number of partitions	= 1.8		

Note: For each range of processing time, the first column is the number of iterations and the second column is computer time in centiseconds. The statistics, Min, Max and Mean are shown for computer time only.