

# Selection of Scheduling Criteria in Computer Systems

S. S. REDDI

W. W. Gaertner Research, Inc.  
Stamford, Connecticut 06903

---

**Abstract:** This paper considers the case in which one wishes to determine a schedule for a two-facility sequencing situation internal to computer systems. For such a problem the total time involved in both calculating an optimal sequence and performing that sequence may sometimes exceed the total time necessary to simply select and process an arbitrary sequence. Linear decision procedures are presented so as to determine whether calculation of an optimal schedule is really worthwhile. Computational aspects of the procedures are also discussed.

---

■ Sequencing problems which occur in industrial systems have been extensively investigated in the fields of operations research and industrial engineering [1]. The resources of a computer system can be considered as machines (or facilities) and most of the sequencing techniques developed for machine shops can be applied to improve the computer system's performance. As an example, a computer system can be viewed as composed of a central processor and input-output unit; when the computer system is multiprogrammed and each program (job) requires processing first on the central processor and then on the input-output unit, programs can be scheduled as in a 2-facility flowshop problem. Examples of this approach where operational research techniques have been used to improve the resource utilization of a computer system can be found in References [2] - [6].

In a computer system the overhead associated with the computation of optimal (or sub-optimal) schedules for sequencing tasks can become significant. Assume that  $n$  jobs are to be processed on  $m$  facilities according to some given job precedence constraints. Suppose that we have two algorithms OPT and HEU which generate optimal and non-

optimal schedules respectively. (An optimal schedule minimizes the time required to process all the jobs.) Let the job completion times be  $T_{OPT}$  and  $T_{HEU}$  and the overheads for for computing the schedules be  $OV_{OPT}$  and  $OV_{HEU}$  for OPT and HEU respectively. Since usually  $T_{OPT} \leq T_{HEU}$  and  $OV_{OPT} > OV_{HEU}$  and the total completion times are  $T_{OPT} + OV_{OPT}$  and  $T_{HEU} + OV_{HEU}$  when we use OPT and HEU respectively, naturally the question arises: Under what circumstances is OPT better than HEU (i.e.,  $T_{OPT} + OV_{OPT} < T_{HEU} + OV_{HEU}$ ) and how do we decide these circumstances? In this paper we first consider the 2-facility flowshop sequencing problem and develop decision procedures to resolve the above question; then we suggest extensions to the general M-facility problem. For our purposes we assume that HEU consists of picking an arbitrary permutation schedule and hence  $OV_{HEU} = 0$ .

The following notation is used for the two-facility sequencing problem. There are  $n$  independent jobs 1, 2, 3, ...,  $n$  which are to be processed first on  $F_1$  and then on  $F_2$ . Job  $i$  requires  $a_i$  and  $b_i$  units of processing on  $F_1$  and  $F_2$  respectively. A schedule  $(j_1, j_2, \dots, j_n)$  represents the situation where the processing sequence of jobs on both the facilities is  $j_1, j_2, \dots, j_n$ .  $C(S)$  is the completion time for the schedule  $S$ . Johnson [7] gives an algorithm to generate an optimal schedule which minimizes the job completion time. Assume that the overhead involved in obtaining an

---

\* A preliminary version of this paper was presented at the Second Sagamore Conference on Parallel Processing, Sagamore, New York, 1974. This research was performed while the author was at Rice University with partial support from NSF under Grant GJ 36471.

optimal schedule be  $OVJ$ . For notational convenience let  $S_{MIN} = (1, 2, 3, \dots, n)$  be the optimal schedule derived by the Johnson's algorithm and hence  $S_{MAX} = (n, n-1, \dots, 2, 1)$  maximizes the completion time. Let  $T_{MIN}$  and  $T_{ARB}$  be the completion times of an optimal and arbitrary schedule respectively. When we have

$$T_{MIN} + OVJ \geq T_{ARB} \quad (1)$$

no optimal schedule need be computed. Note

$$OVJ \geq T_{MAX} - T_{MIN} \quad (2)$$

where  $T_{MAX}$  is the completion time of a schedule which maximizes the completion time, implies (1) since  $T_{MAX} - T_{MIN} \geq T_{ARB} - T_{MIN}$ . If it is possible to determine conditions under which (2) holds we can develop decision procedures to decide whether to compute an optimal schedule or process jobs at random. Though  $T^* = T_{MAX} - T_{MIN}$  can be computed using the Johnson's algorithm this would entirely defeat our purpose of developing criteria to decide when not to compute an optimal schedule (since an optimal schedule will inevitably be generated in the course of any such computation). Instead of computing  $T^*$  we calculate upper bounds on this quantity and use these bounds to decide when not to compute an optimal schedule. Theorems 1 and 2 give upper bounds in terms of job processing times and evaluation of these upper bounds does not require generation of optimal schedules.

We need the following definitions for Theorem 1. Let

$$a_{i_1} \geq a_{i_2} \geq \dots \geq a_{i_{n-1}} \geq a_{i_n} \text{ and}$$

$$b_{j_1} \geq b_{j_2} \geq \dots \geq b_{j_{n-1}} \geq b_{j_n} \text{ where } i_p \neq i_q \text{ and } j_p \neq j_q \text{ for } p \neq q.$$

Define:

$$m = n-2/2 \text{ for even } n,$$

$$= n-1/2 \text{ for odd } n.$$

$$A_{MAX} = a_{i_1} + a_{i_2} + \dots + a_{i_m} \text{ for } n > 2,$$

$$= 0 \text{ for } n = 2.$$

$$A_{MIN} = a_{i_{m+2}} + a_{i_{m+3}} + \dots + a_{i_n} \text{ for odd } n,$$

$$= a_{i_{m+3}} + a_{i_{m+4}} + \dots + a_{i_n} \text{ for even } n \neq 2,$$

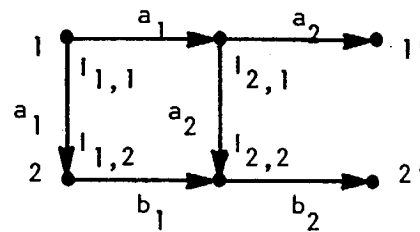
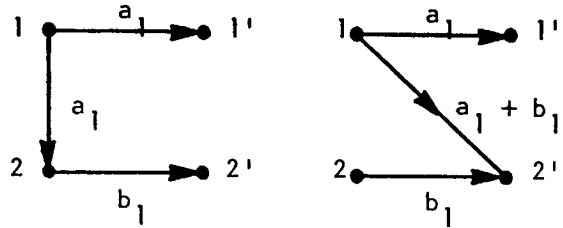
$$= 0 \text{ for } n = 2.$$

Similarly  $B_{MAX}$  and  $B_{MIN}$  are defined by replacing  $A$ ,  $a$  and  $i$  by  $B$ ,  $b$  and  $j$  respectively. Note  $A_{MAX}(B_{MAX})$  and  $A_{MIN}(B_{MIN})$  are simply the sums of the first  $m$  large and the last  $m$  small numbers taken from the set  $\{a_1, a_2, \dots, a_n\}$  ( $\{b_1, b_2, \dots, b_n\}$ ) respectively.

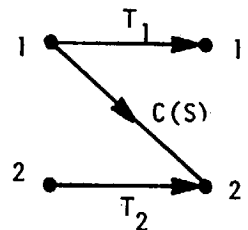
$$\begin{aligned} \Delta &= A_{MAX} - A_{MIN} + B_{MAX} - B_{MIN} \text{ for odd } n, \\ &= A_{MAX} - A_{MIN} + B_{MAX} - B_{MIN} + \text{Max} \{a_{i_{m+1}} - \\ &\quad a_{i_{m+2}}, b_{j_{m+1}} - b_{j_{m+2}}\} \text{ for even } n. \end{aligned}$$

**Theorem 1:**  $T_{MAX} - T_{MIN} \leq \Delta$ .

Before proceeding to the proof of the theorem we need some network concepts. It is possible to represent a schedule by means of a network [8]. (Such a network is essentially the same as the project network encountered in CPM or PERT). In Figs. 1a and 1b equivalent networks for a

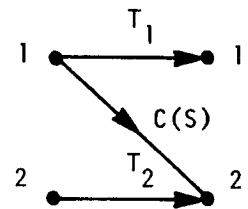


(a) Equivalent networks for a single job.



$$\begin{aligned} T_1 &= a_1 + a_2 \\ T_2 &= b_1 + b_2 \\ C(S) &= \text{Max} \{a_1 + a_2 + b_2, a_1 + b_1 + b_2\} \end{aligned}$$

(b) Equivalent networks for the sequence (1, 2)



$$\begin{aligned} T_1 &= a_{i_1} + a_{i_2} + \dots + a_{i_p} \\ T_2 &= b_{i_1} + b_{i_2} + \dots + b_{i_p} \end{aligned}$$

(c) An equivalent network for the schedule  $(i_1, i_2, \dots, i_p)$

Fig. 1.

single job and a sequence (1, 2) are shown respectively. In Fig. 1b, node  $I_{i,j}$  represents the initiation of job  $i$  on  $F_j$  and the weight of the longest path from 1 ( $I_{1,1}$ ) to  $I_{i,j}$  gives the (earliest) time at which job  $i$  can be initiated on  $F_j$ . In our equivalent networks only the longest paths between 1,2 and 1',2' are considered since they represent all the essential information we need regarding the schedule. Fig. 1c shows an equivalent network representation for the schedule  $S = (i_1, i_2, \dots, i_p)$ . The completion time of the schedule  $(1, i_1, i_2, \dots, i_p, n) = (1, S, n)$  can be given in terms of  $C(S)$  by calculating the longest path of the network shown in Fig. 2. Now we prove the theorem.

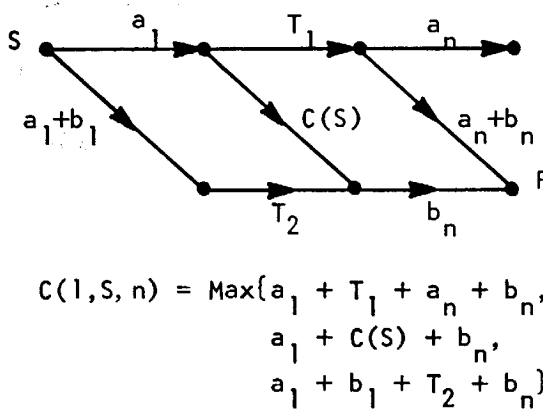


Fig. 2.

*Proof:* The theorem is proved for odd  $n$ . The proof follows the same lines for even  $n$ . Proof is by induction.

Basis:  $n = 3$ .

$$T_{\text{MIN}} = C(S_{\text{MIN}}) = C[(1,2,3)] = \text{Max}\{a_1 + b_1 + b_2 + b_3, a_1 + a_2 + b_2 + b_3, a_1 + a_2 + a_3 + b_3\}$$

$$T_{\text{MAX}} = C(S_{\text{MAX}}) = C[(3,2,1)] = \text{Max}\{a_3 + b_3 + b_2 + b_1, a_3 + a_2 + b_2 + b_1, a_3 + a_2 + a_1 + b_1\}.$$

Hence  $\text{Max}\{a_3 - a_1, a_3 - a_1 + b_1 - b_3, b_1 - b_3\} \geq T_{\text{MAX}} - T_{\text{MIN}}$ . (We are using the inequality  $\text{Max}\{X_1 - Y_1, X_2 - Y_2, \dots, X_p - Y_p\} \geq \text{Max}\{X_1, X_2, \dots, X_p\} - \text{Max}\{Y_1, Y_2, \dots, Y_p\}$ ). Let  $a_{i_1} \geq a_{i_2} \geq a_{i_3}$ , and so  $A_{\text{MAX}} = a_{i_1}$  and  $A_{\text{MIN}} = a_{i_3}$ . Similarly let  $b_{j_1} \geq b_{j_2} \geq b_{j_3}$ , and so  $B_{\text{MAX}} = b_{j_1}$  and  $B_{\text{MIN}} = b_{j_3}$ .

Since  $a_{i_1} - a_{i_3}$  ( $b_{j_1} - b_{j_3}$ ) represents the maximum possible difference between  $a_i$ 's ( $b_j$ 's) it follows that  $a_{i_1} - a_{i_3} + b_{j_1} - b_{j_3} \geq \text{Max}\{a_3 - a_1, a_3 - a_1 + b_1 - b_3, b_1 - b_3\}$ , i.e.,  $A_{\text{MAX}} - A_{\text{MIN}} + B_{\text{MAX}} - B_{\text{MIN}} \geq T_{\text{MAX}} - T_{\text{MIN}}$ .

*Induction Step:* Assume that the theorem is true for all numbers that are odd and less than  $k$ . We show the theorem is true for  $n = k$ .

We have  $S_{\text{MIN}} = (1, 2, 3, \dots, k-1, k)$  and  $S_{\text{MAX}} = (k, k-1, \dots, 3, 2, 1)$ . For the  $k-2$  jobs  $2, 3, \dots, k-1$ , it is obvious that  $S'_{\text{MIN}} = (2, 3, \dots, k-1)$  and  $S'_{\text{MAX}} = (k-1, \dots, 3, 2)$ , minimizes and maximizes the completion time respectively.

(This follows from the Johnson's algorithm.) Since  $k-2$  is odd, the theorem holds and we have

$$C(S'_{\text{MAX}}) - C(S'_{\text{MIN}}) \leq A'_{\text{MAX}} - A'_{\text{MIN}} + B'_{\text{MAX}} - B'_{\text{MIN}} \quad (3)$$

where  $A'_{\text{MAX}}, A'_{\text{MIN}}, B'_{\text{MAX}}$  and  $B'_{\text{MIN}}$  are computed for the  $k-2$  jobs  $2, 3, \dots, k-1$ .

Note  $C(S_{\text{MIN}}) = \text{Max}\{a_1 + T_1 + a_k + b_k, a_1 + C(S'_{\text{MIN}}) + b_k, a_1 + b_1 + T_2 + b_k\}$  and  $C(S_{\text{MAX}}) = \text{Max}\{a_k + T_1 + a_1 + b_1, a_k + C(S'_{\text{MAX}}) + b_1, a_k + b_k + T_2 + b_1\}$  from Fig. 3.

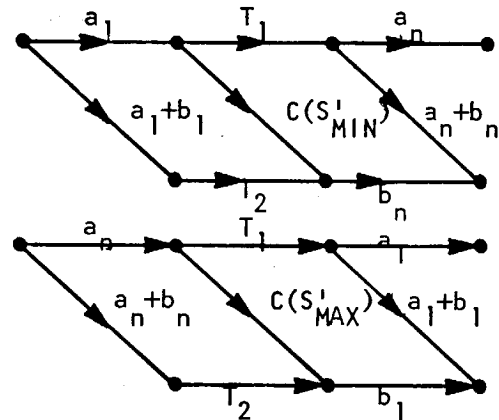


Fig. 3.

Hence  $C(S_{\text{MAX}}) - C(S_{\text{MIN}}) \leq \text{Max}\{b_1 - b_k, a_k - a_1 + C(S'_{\text{MAX}}) - C(S'_{\text{MIN}}) + b_1 - b_k, a_k - a_1\}$ . Since  $\text{Min}(a_k, b_1) - \text{Min}(a_1, b_k) \geq 0$  (because of the Johnson's algorithm) we have  $\text{Max}\{a_k - a_1, b_1 - b_k\} \geq 0$ . The above inequality reduces to

$$C(S_{\text{MAX}}) - C(S_{\text{MIN}}) \leq a_k - a_1 + C(S'_{\text{MAX}}) - C(S'_{\text{MIN}}) + b_1 - b_k \quad (4)$$

$$(3) + (4) \Rightarrow T_{\text{MAX}} - T_{\text{MIN}} \leq (A'_{\text{MAX}} + a_k) - (A'_{\text{MIN}} + a_1) +$$

$$(B'_{\text{MAX}} + b_1) - (B'_{\text{MIN}} + b_k) \leq A_{\text{MAX}} - A_{\text{MIN}} + B_{\text{MAX}} - B_{\text{MIN}}$$

When all the jobs are identical  $\Delta$  becomes 0 and the equality holds in the theorem. Only in this case the equality is reached. Consider the evaluation of  $\Delta$ . In [9] it is shown that the median of  $n$  numbers can be found within  $5.43n$  binary comparisons. Treating additions and comparisons as operations to find  $A_{\text{MAX}} - A_{\text{MIN}}$  approximately  $6.43n$  operations are needed. Hence computation of  $\Delta$  requires  $2 \cdot (6.43n)$  or approximately  $13n$  operations. However, an optimal schedule can be found by the Johnson's algorithm using the techniques of Szwarc [10] and Ford and Johnson [11] with approximately  $n \log_2 n - 0.5n$  operations. (It is assumed that  $n$  numbers can be sorted with  $n \log_2 n - 1.5n$  comparisons [12]). Hence in order that the evaluation of  $\Delta$  is meaningful we should have  $13n < n \log_2 n - 0.5n$ , i.e.,

$n > 2^{13.5} \approx 12 \times 10^3$ . Therefore it is clearly impractical to substitute  $\Delta$  for  $T_{MAX} - T_{MIN}$  in (2) and develop decision procedures to decide when not to compute an optimal schedule. However the following corollary to Theorem 1 gives a more practical method of obtaining an upper bound on  $T^*$ .

**Corollary 1:** Let  $a_{MAX}$  and  $a_{MIN}$  ( $b_{MAX}$  and  $b_{MIN}$ ) be the maximum and minimum of  $a_i$ 's ( $b_i$ 's). Let  $a^* = a_{MAX} - a_{MIN}$  and  $b^* = b_{MAX} - b_{MIN}$ . Then  $\delta = m'(a^* + b^*) \geq \Delta \geq T^*$  where  $m' = \lfloor n/2 \rfloor$  (Note  $m' = m$  if  $n$  is odd; otherwise  $m = m+1$ ).

**Proof:** The corollary follows by observing  $ma_{MAX} \geq A_{MAX}$ ,  $A_{MIN} \geq ma_{MIN}$  and hence  $ma^* \geq A_{MAX} - A_{MIN}$ .

Since  $a^*$  and  $b^*$  can be calculated with approximately  $3n$  operations [13],  $\delta$  can be evaluated with  $3.5n$  operations. (Note multiplication by  $m'$  is considered as  $m'$  addition operations.) Thus it is possible to develop practical procedures using  $\delta$  in (2). Later on we present procedures incorporating decisions of type (2) and analyze their performance. The following theorem derives upper bounds on  $T^*$  using the idle times on the facilities and does not rely on the properties of the Johnson's algorithm. It is simpler than Theorem 1 and often offers tighter bounds with less computation.

**Theorem 2:**  $\text{Min}\{D^+ + a^*, D^+ + b^*\} \geq \text{Min}\{a_{MAX}, b_{MAX}\} + \text{Min}\{D^- - a_{MIN}, D^- - b_{MIN}\} \geq T^*$  where

$$D^+ = \sum (b_i - a_i) \text{ for } i \text{ such that } b_i - a_i > 0 \text{ and}$$

$$D^- = \sum (a_i - b_i) \text{ for } i \text{ such that } a_i - b_i \geq 0.$$

**Proof:** Let  $I_{MAX}$  and  $I_{MIN}$  be the total maximum and minimum idle times on both the facilities respectively. Then it follows

$$I_{MAX} = 2T_{MAX} - \left( \sum_i a_i + \sum_i b_i \right)$$

$$I_{MIN} = 2T_{MIN} - \left( \sum_i a_i + \sum_i b_i \right)$$

and so

$$T_{MAX} - T_{MIN} = (I_{MAX} - I_{MIN}) / 2 \quad (5)$$

Since

$$T_{MIN} \geq \text{Max} \left( \sum_i a_i + b_{MIN}, \sum_i b_i + a_{MIN} \right),$$

$$I_{MIN} \geq \text{Max} \left( \sum_i (a_i - b_i) + 2b_{MIN}, \sum_i (b_i - a_i) + 2a_{MIN} \right) \quad (6)$$

For any schedule  $S = (1, 2, 3, \dots, n)$  there exists a  $j$  such that

$$C(S) = a_1 + a_2 + \dots + a_j + b_j + b_{j+1} + \dots + b_n,$$

$$1 \leq j \leq n.$$

Hence the total idle time  $I$  for  $S$  is (Fig. 4):

$$\begin{aligned} I &= (a_1 + a_2 + \dots + a_j) - (b_1 + b_2 + \dots + b_{j-1}) \\ &\quad + (b_j + b_{j+1} + \dots + b_n) - (a_{j+1} + a_{j+2} + \dots + a_n) \\ &= \sum_{i=1}^j (a_i - b_i) + \sum_{i=j+1}^n (b_i - a_i) + 2b_j \\ &= \sum_{i=1}^{j-1} (a_i - b_i) + \sum_{i=j}^n (b_i - a_i) + 2a_j \end{aligned}$$

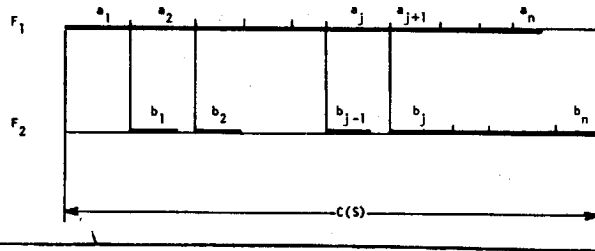


Fig. 4.

Let  $D = \sum_i |a_i - b_i|$ . It follows that for any schedule  $S$ ,

$$I \leq D + 2\text{Min}(a_{MAX}, b_{MAX}).$$

Thus  $I_{MAX} \leq D + 2\text{Min}(a_{MAX}, b_{MAX}) \quad (7)$

Subtracting (6) from (7) we get

$$I_{MAX} - I_{MIN} \leq 2\text{Min}(a_{MAX}, b_{MAX}) - \text{Max}(2b_{MIN} - 2D^+, 2a_{MIN} - 2D^-).$$

Combining the above equation with (5) we get the theorem.

The calculation of upper bounds on  $T^*$  using Theorem 2 involves approximately  $5n$  operations. Hereafter only Corollary 1 will be used for our purposes. Now we show how the upper bounds on  $T^*$  can be used in practice. As mentioned before OVJ, the overhead to compute an optimal schedule, involves approximately  $n \log_2 n - 0.5n$  binary comparisons. Each comparison between two quantities say  $a_i$  and  $a_j$  consists of the following basic operations:

1. Compute indices  $i$  and  $j$  for which  $a_i$  and  $a_j$  are to be compared.
2. Fetch  $a_i$  and  $a_j$ .
3. Compare  $a_i$  and  $a_j$
4. Store  $a_i$  and  $a_j$  in their proper locations.

Assuming that it takes two machine operations for steps 1, 2 and 4 and a single machine operation for step 3, each comparison takes (at least) 7 machine operations. If  $a_i$ 's and

$b_i$ 's are expressed in machine operations, Corollary 1 implies that if

$$7(n \log_2 n - 0.5n) \geq m'(a^* + b^*) \quad (8)$$

there is no advantage in computing an optimal schedule. For  $n = 127$ , (8) becomes  $91 \geq (a^* + b^*)$ . The inequality (8) can be interpreted as follows. When all the jobs are identical it is obvious that there is no need to schedule them and  $a^* = b^* = 0$ . It is intuitively clear that if the job processing times are almost the same and jobs are almost identical to each other, again there is no need to compute a schedule since the completion times of different schedules will be almost the same.  $a^*$  and  $b^*$  give a measure of how similar the jobs are to each other and (8) provides a quantitative rule by which we can decide when the jobs are no longer similar to each other and the schedule times are no longer close to each other (temporally). When  $n = 127$  as long as the sum of the "spreads" between the processing times on  $F_1$  and  $F_2$  is not greater than 91, there is no need to compute an optimal schedule. If we know that (8) will always hold for any set of jobs we can avoid using Johnson's algorithm altogether.

In practice the computer system processes jobsets for which (8) may hold sometimes. It may not be possible to predict whether (8) holds or not for a given jobset. In these situations  $a^*$  and  $b^*$  have to be calculated and this adds to the overall overhead. The following procedure takes into account overheads incurred in such calculations and makes a decision as to when to compute an optimal schedule in these situations. For convenience assume that the basic time unit is the time to make one binary comparison and  $a_i$ 's and  $b_i$ 's are given in terms of these time units.

**Procedure:**

1. Compute  $m'(a^* + b^*)$ .
2. Compare  $m'(a^* + b^*)$  and  $n \log_2 n - 4n$ . If  $m'(a^* + b^*) < n \log_2 n - 4n$ , go to 4.
3. Compute an optimal schedule. Go to 5.
4. Process the jobs at random (without calculating an optimal schedule). Stop.
5. Process the jobs according to the optimal schedule computed in 3. Stop.

The Procedure is shown schematically in Fig. 5. If steps 1, 2 and 4 are followed  $3.5n$  operations are involved. On the other hand we require  $n \log_2 n + 3n$  operations when steps 1, 2, 3 and 5 are followed. (There is no specific reason for choosing  $n \log_2 n - 4n$  in step 2; different choices lead to different performances of the Procedure.)

Let us consider the Procedure and decide under what circumstances it yields a performance improvement over a procedure which computes an optimal schedule all the time. Assume that the computer system processes jobsets  $S_1, S_2, \dots, S_k$  with each jobset containing  $n$  jobs. The optimal schedule time for  $S_i$  is  $C_i$ .  $R_i$  is the schedule time of a

**Procedure.**

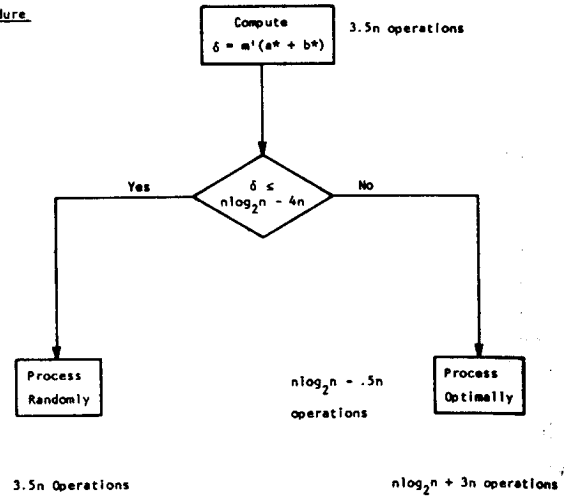


Fig. 5.

random schedule for  $S_i$ . Let  $k = k_1 + k_2$  where  $k_1$  is the number of times the Procedure computes an optimal schedule (i.e., takes the no branch in Fig. 5) and  $k_2$  the number of times the Procedure decides to process the jobs randomly (i.e., takes the yes branch in Fig. 5). Assume that the jobsets are numbered so that for  $S_1, S_2, \dots, S_{k_1}$  the optimal schedule is computed.

The total completion time  $O_1$  for all the jobsets when the Procedure is used is:

$$O_1 = k_1(n \log_2 n + 3n) + k_2(3.5n) + C_1 + C_2 + \dots + C_{k_1} + R_{k_1+1} + R_{k_1+2} + \dots + R_k.$$

The total completion time  $O_2$  when Johnson's algorithm is used is:

$$O_2 = k(n \log_2 n - 0.5n) + C_1 + C_2 + \dots + C_k.$$

The percentage improvement in the total completion time when the Procedure is used is given by  $P = (O_2 - O_1) / O_2 \times 100\%$ .

$$O_2 - O_1 = k_2(n \log_2 n - 0.5n) - k(3.5n) + (C_{k_1+1} - R_{k_1+1}) + \dots + (C_k - R_k).$$

Let  $RC_i = (R_i - C_i)$ . Note  $n \log_2 n - 4n \geq RC_i \geq 0$  for  $i = k_1 + 1, \dots, k$ . Assume that

$$RC = (RC_{k_1+1} + RC_{k_1+2} + \dots + RC_k) / k_2 \text{ and}$$

$C = (C_1 + C_2 + \dots + C_k) / k$  exist when  $k \rightarrow \infty$ . Then for sufficiently large  $k$  we have

$$P = \frac{k_2(n \log_2 n - 0.5n) - k(3.5n) - k_2 RC}{k(n \log_2 n - 0.5n) + kC} \cdot 100\% \\ = \frac{q(\log_2 n - 0.5 - RC/n) - 3.5}{(\log_2 n - 0.5) + C/n} \cdot 100\% \text{ where } q = k_2/k.$$

The implications of selecting a quantity for comparison with  $m'(a^* + b^*)$  in step 2 of the Procedure can be seen from the expression for  $P$ . (As mentioned before our choice of  $n \log_2 n - 4n$  for this quantity is arbitrary.) Denoting this quantity by  $Q$  we see that as  $Q$  increases ( $\log_2 n - 0.5 - RC/n$ ) decreases but  $q$  increases. The correct selection of  $Q$  to ensure optimum performance depends on the nature of the jobsets and may have to be determined experimentally in practice.

The dependence of  $P$  on  $q$  and  $C/n$  can be illustrated by means of an example. Let  $n = 127$ . Then  $3 \geq RC/n = 0$ . Assume  $RC/n = 1$ . Then

$$P = \frac{5.5q - 3.5}{6.5 + C/n} \times 100\%.$$

Hence  $q > 3.5/5.5 = 0.64$  in order that  $P$  is positive. This means that a performance improvement is possible only if the nature of the jobsets is so that for approximately two-thirds of them the Procedure should not compute an optimal schedule, i.e., the job processing times  $a_i$ 's and  $b_i$ 's are such that  $3n > m'(a^* + b^*)$  or  $6 > (a^* + b^*)$ . Figure 6 shows how  $P$  varies with respect to  $C^* = C/n$ , the mean flow time and  $q$ . Note  $P$  decreases as  $C^*$  increases; this is to be expected since overheads become negligible as the average schedule time increases. It may be noted that  $P$  may not appear to be high. This is partially due to the fact that the computational complexity of the Johnson's algorithm is very low. However the point to be noted is that it is possible to develop scheduling procedures which perform better than the ones which schedule optimally all the time.

The computer system can be made to keep track of parameters like  $q$ ,  $C^*$ ,  $RC$ , etc. and adapt itself to the situation by choosing appropriate scheduling procedures. So far we are concerned with the two-facility problem. For the M-facility problem Nabeshima [14] gives an ingenious

method for finding upper bounds on the completion time using the Johnson's algorithm. By using Nabeshima's techniques it is possible to evaluate upper bounds on  $T^*$  with computational complexity of  $o(Mn \log_2 n)$ . Since there are no general computationally simple optimal algorithms to solve the M-facility problem for  $M > 2$ , the upper bounds can be used to decide whether heuristics should be used to generate a schedule or process jobs at random.

#### Acknowledgment

The author would like to thank the referees for their criticisms and suggestions. He is indebted to one of the referees for suggesting the title for the paper and writing a new, concise abstract which the author reproduced verbatim (since there was no way it could be improved).

#### References

- [1] Conway, R. W., Maxwell, W. L. and Miller, L. W., *Theory of Scheduling*, Addison-Wesley, Reading, Mass. (1967).
- [2] Heller, J., "Sequencing Aspects of Multiprogramming," *Journal of Assoc. Comp. Mach.*, 8 3, 426-439 (1961).
- [3] Coffman, Jr., E. G. and Denning, P. J., *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, N.J. (1973).
- [4] Cotten, L. W. and Abd-Alla, A. M., "Processing Times for Segmented Jobs with I/O Compute Overlap," *Journal of Assoc. Comp. Mach.*, 21 1, 18-30 (1974).
- [5] Reddi, S. S. and Ramamoorthy, C. V., "On the Flowshop Sequencing Problem with No Wait in Process," *Operational Res. Qlty.*, 23 3, 323-331 (1972).
- [6] Nabeshima, I., "General Scheduling Algorithms with Applications to Parallel and Multiprogramming Scheduling," *J. Ops Res. Soc. Japan*, 14 2, 72-99 (1971).
- [7] Johnson, S. M., "Optimal Two- and Three-Stage Production Schedules with Setup Times Included," *Nav. Res. Log. Qlty.*, 1 1, 61-68 (1954).
- [8] Lomnicki, Z. A., "A Branch-and-Bound Method for the Exact Solution of the Three-Machine Scheduling Problem," *Operational Res. Qlty.*, 16, 89-100 (1965).
- [9] Blum, M. et al, "Time Bounds for Selection," *Journal of Computer and System Sciences*, 7 4, 448-461 (1973).
- [10] Szwarc, W., "On Some Sequencing Problems," *Nav. Res. Log. Qlty.*, 15 2, 127-155 (1968).
- [11] Ford, L. R. and Johnson, S. M., "A Tournament Problem," *American Math. Monthly*, 66, 387-389 (1959).
- [12] Knuth, D. E., *The Art of Computer Programming*, 3 Addison-Wesley, Reading, Mass. (1972).
- [13] Pohl, I., "A Sorting Problem and Its Complexity," *Comm. of Assoc. Comp. Mach.*, 15 6, 462-464 (1972).
- [14] Nabeshima, I., "On the Bound of Makespans and Its Application in M Machine Scheduling Problem," *Journal of Operations Res. Soc. of Japan*, 9 3 & 4, 98-136 (1967).

The results of References [9], [11] and [13] can be found in [12].

Dr. S. S. Reddi is a Project Scientist at W. W. Gaertner Research, Inc., Stamford, Connecticut. His principal interests are in Computer Architecture, Application of Parallel Processing to Image and Radar Data Processing and Microprocessors. He obtained his MS from Stanford University and his PhD from the University of Texas at Austin, both in Electrical Engineering. He is a member of ACM, IEEE and Sigma Xi.

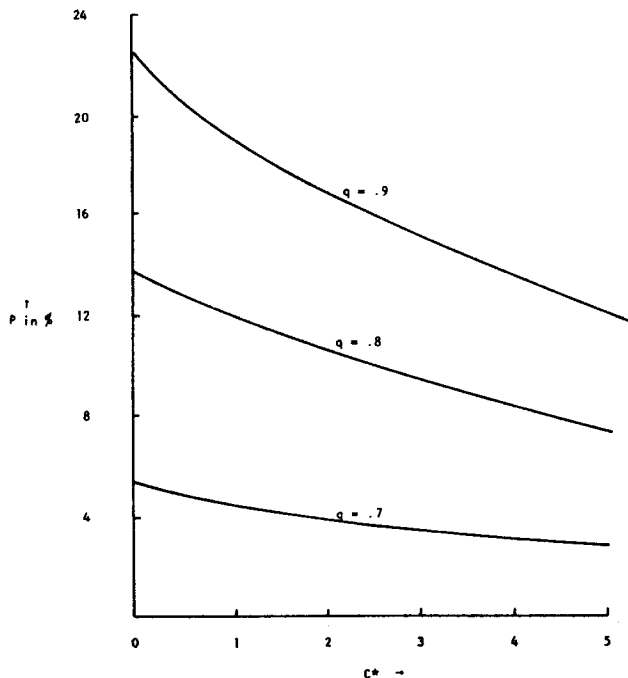


Fig. 6.