Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors

OSCAR H. IBARRA AND CHUL E. KIM

University of Minnesota, Minneapolis, Minnesota

ABSTRACT. The finishing time properties of several heuristic algorithms for scheduling n independent tasks on m nonidentical processors are studied In particular, for $m = 2$ an $n \log n$ time-bounded algorithm is given which generates a schedule having a finishing time of at most $(\sqrt{5} + 1)/2$ of the optimal finishing time A simplified scheduling problem involving identical processors and restricted task sets is shown to be P-complete However, the LPT algorithm applied to this problem yields schedules which are near optimal for large n

KEY WORDS AND PHRASES finishing time, heuristic algorithm, scheduling independent tasks, nonidentical processors, identical processors, time-bounded algorithm, LPT schedule, SPT schedule, P-complete problem, time complexity

CR CATEGORIES. 4 32, 5 39

1. *Introduction*

We are given $m \geq 2$ processors $P_1, \ldots, P_m$ and a set T of $n \geq 2$ independent tasks $J_1, \ldots, J_n$. The processors are nonidentical in the sense that processing time functions $\mu_1, \ldots, \mu_m$ are defined on T so that a task J_i requires $\mu_j(J_i)$ time to process on processor P_j, $1 \leq j \leq m$.[1] This model of a multiprocessor system was introduced in [1, 2], where it was shown that an $O(\max(mn^2, n^3))$ time-bounded algorithm exists which obtains a schedule (of T on the m processors) having the least mean flow time. In this paper we are concerned with the problem of finding a schedule whose finishing time is as small as possible (such a schedule will be called *optimal*). This problem is known to be P-complete [7–9].[2] Hence it seems unlikely that a polynomial time-bounded algorithm exists for this problem. In [7] a dynamic programming type algorithm of exponential time complexity was given to find an optimal schedule. It was also shown in [7] that a polynomial time-bounded algorithm exists which obtains a schedule with a finishing time arbitrarily close to the optimal finishing time. However, the complexity of the algorithm was $O((1/\epsilon) \cdot n^{2m})$, where ϵ is the relative error. A special case of our problem when $\mu_j/\mu_1 = s_j$ (i.e. the processors have uniform speeds) was studied in [5, 7].

In Section 2 we look at several simple heuristic algorithms and analyze their worst-case behavior as measured by the ratio $\hat{f}/f^*$, where $\hat{f}$ is the finishing time of the schedule obtained by the algorithm and f^* is the optimal finishing time. In Section 3 we consider the case of there being only two processors. An $n \log n$ algorithm that has a better worst-case behavior than any of the algorithms developed in Section 2 is presented. Finally, in

This research was supported by the National Science Foundation under Grant DCR72-03728-A01 and by University of Minnesota Research Grant 435-0806-8520-02.

Authors' present addresses. O H. Ibarra, Department of Computer Science, University of Minnesota, Minneapolis, MN 55455; C E. Kim, Department of Computer Science, University of Maryland, College Park, MD 20742

[1] For convenience, we shall usually write a task J_i as $J_i(\mu_1(J_i), \ldots, \mu_m(J_i))$

[2] The terminology "P-complete" is defined in [9] Karp's notion of "completeness" is slightly different from this

Section 4, we consider the special case of a multiprocessor system in which all the processors are identical, i.e. $\mu_i = \mu_j = \mu$ for all $i \neq j$, and the processing time of each task does not vary "too much," i.e. $\max_i \mu(J_i)/\min_i \mu(J_i) = \rho$ for some $\rho > 1$.[3] We show that this simplified problem remains P-complete, but the LPT scheduling strategy [6] applied to this problem produces schedules with $\hat{f}/f^* \to 1$ as $n \to \infty$.

2. *Nonidentical Multiprocessor Scheduling*

In this section we consider five different heuristic algorithms and study their worst-case behaviors. Some employ very simple heuristics and may seem inferior to others. However, it can be shown that for any pair of these algorithms, examples exist for which one gives a better result than the other Several such examples are discussed.

In every algorithm that we shall consider, the input/output are as follows:

Input. T = set of n independent tasks $J_i(\mu_1(J_i), \ldots, \mu_m(J_i))$, $1 \leq i \leq n$.

Output. $\hat{L} = \{L_j \mid 1 \leq j \leq m\}$ and $\hat{f}$. L_j is the set of tasks scheduled on processor P_j, and $\hat{f}$ is the finishing time of the schedule, $\hat{f} = \max_j\{\sum_{J \in L_j} \mu_j(J)\}$.

A-schedule. The ith task in the list T is scheduled on the processor that minimizes its finishing time.

ALGORITHM A(T, $\hat{L}_A$, $\hat{f}_A$)

Step 1 (Initialize L_j and t_j) $L_j \leftarrow \varnothing$ and $t_j \leftarrow 0$ for $1 \leq j \leq m$

Step 2 (Schedule the tasks and find $\hat{f}_A$)

For $i \leftarrow 1$ **to** n **step** 1 **do**

find the smallest j such that $t_j + \mu_j(J_i) \leq t_l + \mu_l(J_i)$ for all $1 \leq l \leq m$, $L_j \leftarrow L_j \cup \{J_i\}$, $t_j \leftarrow t_j + \mu_j(J_i)$ **end**,

$\hat{f}_A \leftarrow \max_j t_j$, **return**

B-schedule. For each task J, let $\mu_{\min}(J) = \min_j \mu_j(J)$. Algorithm B($T$, $\hat{L}_B$, $\hat{f}_B$) first orders the tasks in T according to nonincreasing $\mu_{\min}(J)$, and then calls Algorithm A to schedule T.

C-schedule. For each task J in T, let $\mu_{\max}(J) = \max_j \mu_j(J)$. Algorithm C($T$, $\hat{L}_C$, $\hat{f}_C$) orders the tasks in T according to nonincreasing $\mu_{\max}(J)$ and calls Algorithm A.

D-schedule. After having scheduled i tasks, the algorithm schedules a task (from among the remaining $(n - i)$ tasks) which gives the least finishing time.

ALGORITHM D(T, $\hat{L}_D$, $\hat{f}_D$)

Step 1 (Initialize L_j and t_j) $L_j \leftarrow \varnothing$ and $t_j \leftarrow 0$ for $1 \leq j \leq m$

Step 2 (Completed?) **If** $T = \varnothing$ **then** [$\hat{f}_D \leftarrow \max_j t_j$, **return**].

Step 3 (Schedule a task). Find a task J in T such that $\min_j\{t_j + \mu_j(J)\} \leq \min_j\{t_j + \mu_j(J')\}$ for all $J' \in T$, let j be such that $t_j + \mu_j(J)$ is minimum; $L_j \leftarrow L_j \cup \{J\}$, $t_j \leftarrow t_j + \mu_j(J)$, $T \leftarrow T - \{J\}$, **go to** step 2

E-schedule. Algorithm E(T, $\hat{L}_E$, $\hat{f}_E$) is the same as Algorithm D except that step 3 is replaced with

Step 3 Find a task J in T such that $\min_j\{t_j + \mu_j(J)\} \leq \min_j\{t_j + \mu_j(J')\}$ for all $J' \in T$, let j be such that $t_j + \mu_j(J)$ is minimum, $L_j \leftarrow L_j \cup \{J\}$, $t_j \leftarrow t_j + \mu_j(J)$, $T \leftarrow T - \{J\}$, **go to** step 2

THEOREM 1. *The table below summarizes the behavior of the algorithms presented above.*

Algorithm	*Run time*	*Worst-case bound for* $\hat{f}/f^*$	*Is the bound the best possible?*
A	$O(n)$	m	*yes*
B	$O(n \log n)$	m	*yes*
C	$O(n \log n)$	m	*yes*
D	$O(n^2)$	m	? [4]
E	$O(n^2)$	m	*yes*

[3] $\max_i \mu(J_i)$ is an abbreviation for $\max_i\{\mu(J_i)\}$

[4] The bound can be approached for $m = 2$ (see Remark (4)).

PROOF.

(1) *Run time.* It is obvious that Algorithm A runs in time $O(n)$. Evaluating $\mu_{\min}(J)$ or $\mu_{\max}(J)$ for all J in T takes $O(n)$ time, and sorting T takes $O(n \log n)$ time. It follows that Algorithms B and C have time complexity $O(n \log n)$. For Algorithms D and E, scheduling a task after i tasks have been scheduled takes $O(n - i)$ time. So the run time of the algorithms is $O(n^2)$.

(2) *Worst-case bound for $\hat{f}/f^*$.* Let $g(n) = \sum_{i=1}^{n}(\min_{1\le j\le m} \mu_j(J_i))$. Clearly $f^*(n) \ge (1/m)g(n)$. For Algorithm A, it is an easy induction on n to show that $\hat{f}_A(n) \le g(n)$. Therefore $\hat{f}_A/f^* \le m$. Since Algorithms B and C employ Algorithm A it follows that $\hat{f}_B/f^* \le m$ and $\hat{f}_C/f^* \le m$. Now we show that $\hat{f}_D \le g(n)$ by induction on n. Trivially, $\hat{f}_D(1) \le g(1)$. Assume that $f_D(k) \le g(k)$, and consider a set T_{k+1} of $k + 1$ tasks. Suppose that J_{k+1} is the task that is scheduled last by Algorithm D. For the set of k tasks $T_k = T_{k+1} - \{J_{k+1}\}$, $\hat{f}_D(k) \le g(k)$ by the induction hypothesis. Thus for T_{k+1} the algorithm gives $\hat{f}_D(k + 1) \le \hat{f}_D(k) + \min_{1\le j\le m} \mu_j(J_{k+1}) \le g(k + 1)$. Thus $\hat{f}_D \le g(n)$ and so $\hat{f}_D/f^* \le m$. A similar argument shows that $\hat{f}_E/f^* \le m$.

(3) *The bounds are the best possible for all but Algorithm D.* The following example shows that the bounds for Algorithms A and B can be approached. Consider the set of m tasks,

$$\begin{aligned}
&J_1(u, u, \ldots, u),\\
&J_2(u, 2u - 1, \ldots, 2u - 1),\\
&J_3(2u, u, 3u - 2, \ldots, 3u - 2),\\
&\vdots\\
&J_l((l - 1)u, (l - 2)u, \ldots, 2u, u, lu - (l - 1), \ldots, lu - (l - 1)),\\
&\vdots\\
&J_m((m - 1)u, (m - 2)u, \ldots, 2u, u, mu - (m - 1)).
\end{aligned}$$

(See Figure 1.) So $\hat{f}_A = \hat{f}_B = mu - (m - 1)$, $f^* = u$, and $\hat{f}_A/f^* = \hat{f}_B/f^* = m - (m - 1)/u \to m$ as $u \to \infty$. Now we show that the bound for Algorithms C and E can be approached. Consider the set of $(m - 1)m + 1$ tasks, where $u \gg m$:

$$\begin{aligned}
&J_1(u, u, \ldots, u), J_2(1, u, \ldots, u), \ldots, J_m(1, \ldots, 1, u),\\
&J_{m+1}(u - 1, u - 1, \ldots, u - 1), J_{m+2}(1, u - 1, \ldots, u - 1), \ldots, J_{2m}(1, \ldots, 1, u - 1),\\
&\vdots\\
&J_{(m-2)m+1}(u - (m - 2), u - (m - 2), \ldots, u - (m - 2)), J_{(m-2)m+2}(1, u - (m - 2), \ldots,\\
&\quad u - (m - 2)), \ldots, J_{(m-1)m}(1, 1, \ldots, 1, u - (m - 2)),\\
&J_{(m-1)m+1}(u - (m - 1), u - (m - 1), \ldots, u - (m - 1)).
\end{aligned}$$

(See Figure 2.) Thus

$$\begin{aligned}
\hat{f}_C/f^* = \hat{f}_E/f^* &= [mu - m(m - 1)/2]/[u + (m - 2)m + 1]\\
&= [m - m(m - 1)/2u]/\{1 + [(m - 2)m + 1]/u\} \to m
\end{aligned}$$

as $u \to \infty$. □

Remarks.

(1) It is interesting to note that $\hat{f}_A \ge \hat{f}_B$, $\hat{f}_A \ge \hat{f}_C$, $\hat{f}_A \ge \hat{f}_D$, and $\hat{f}_A \ge \hat{f}_E$ are not

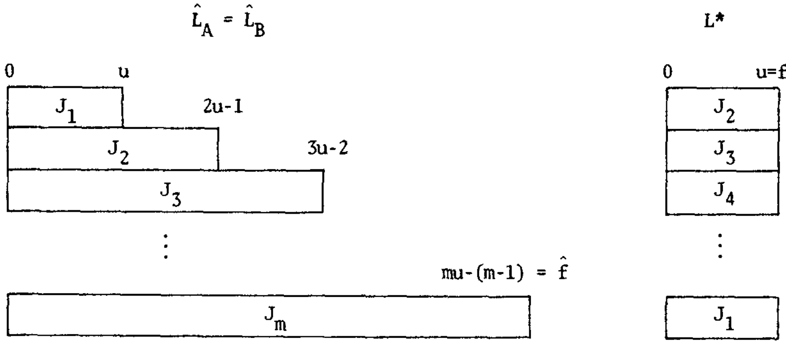


FIG 1. A worst-case example for Algorithms A and B

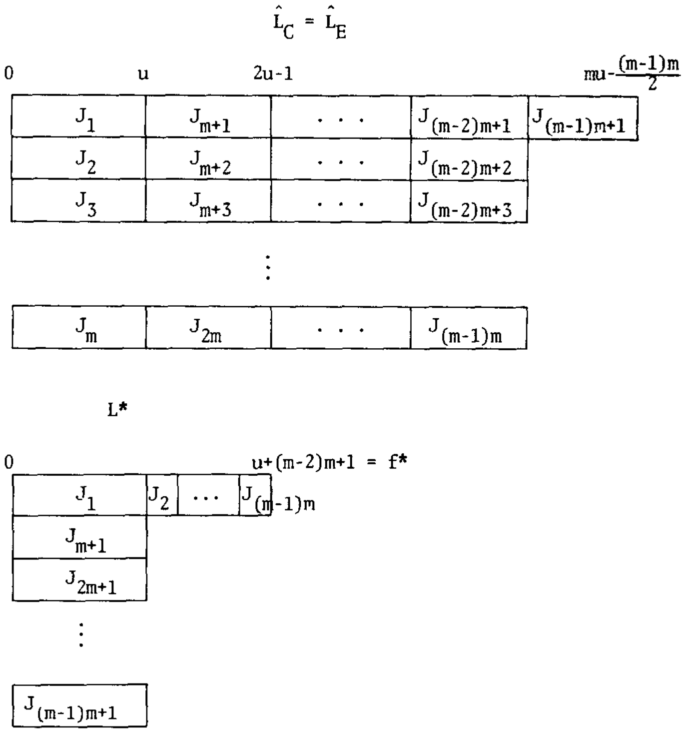


FIG 2 A worst-case example for Algorithms C and E

necessarily true, as the following examples show. Consider the set $T_1 = \{J_1(u - 1, 2u), J_2(u, u)\}$. Then $\hat{f}_A = f^* = u$ and $\hat{f}_B = 2u - 1$. For the set $T_2 = \{J_1(u, 2u, 2u), J_2(u - 1, u, 3u)\}$, $\hat{f}_A = f^* = u$ and $\hat{f}_C = 2u - 1$. For $T_3 = \{J_1(u + 1, 2u), J_2(u, u)\}$, $\hat{f}_A = f^* = u + 1$ and $\hat{f}_D = 2u$. Finally, for $T_4 = \{J_1(u, 2u), J_2(u + 1, u + 1)\}$, we have $\hat{f}_A = f^* = u + 1$ and $\hat{f}_E = 2u$.

(2) The following shows that $\hat{f}_B \geq \hat{f}_D$ and $\hat{f}_B \geq \hat{f}_E$ are not necessarily true: For T_3 in (1), $\hat{f}_B = u + 1$ and $\hat{f}_D = 2u$, and in the last example of Theorem 1, $\hat{f}_B = f^* = u + (m - 2)m + 1$ while $\hat{f}_E = mu - (m - 1)m/2$.

(3) It is not necessary that $\hat{f}_C \geq \hat{f}_D$ and $\hat{f}_C \geq \hat{f}_E$. For example, $\hat{f}_C = u + 1$ and $\hat{f}_D = 2u$ for the set T_3 in (1), and $\hat{f}_C = u + 1$ and $\hat{f}_E = 2u$ for T_4 in (1).

(4) We are unable to show that the bound m can be approached for Algorithm D when $m \geq 3$. In fact, we have no example that yields $\hat{f}_D/f^* > 2$. However, the example involving the set T_3 in (1) shows that $\hat{f}_D/f^*$ can approach 2. Hence the bound for Algorithm D is the best possible for $m = 2$. A slight modification of the example will show that $\hat{f}_D/f^*$ can approach 2 for m-processor scheduling for any $m \geq 2$.

(5) Now suppose that the processors are identical, i e. $\mu_i(J) = \mu_j(J)$ for all $1 \leq i, j \leq m$. Then Algorithms A and D become arbitrary list scheduling and SPT scheduling,[5] respectively [4, 6]. These schedules yield $\hat{f}/f^* \leq 2 - 1/m$ [6]. Algorithms B, C, and E, on the other hand, reduce to the LPT scheduling algorithm which has a bound $\hat{f}/f^* \leq 4/3 - 1/3m$ [6].

3. *Scheduling on Two Nonidentical Processors*

Every algorithm in Section 2 was shown to have a worst-case bound of m for the ratio $\hat{f}/f^*$. Moreover, this bound is the best possible for all but Algorithm D. For this algorithm, we were only able to show that the ratio m is approachable for $m = 2$. We now

[5] An SPT (LPT) schedule is a schedule obtained as a result of an algorithm which, whenever a processor becomes free, assigns the task whose processing time is the shortest (largest) of those tasks not yet assigned.

present a simple heuristic algorithm for the two-processor system that has a better worst-case behavior than any of the algorithms of Section 2. We have not been able to extend the idea used in the algorithm to the case $m \geq 3$.

We first describe the algorithm informally. The algorithm begins by temporarily scheduling the tasks in such a way that each task is assigned to the processor for which the task has a smaller processing time. If the finishing times of both processors are the same, the schedule is obviously optimal. However, if one processor is idle while the other is not, then the schedule may not be optimal. Without loss of generality, assume that P_1 has a longer finishing time. Then it may be possible to reduce the finishing time by reassigning some of the tasks on P_1 onto P_2. The idea is to decrease the finishing time of P_1 as much as possible while minimizing the increase in the finishing time of P_2. Thus if two tasks J_1 and J_2 on P_1 are such that $\mu_1(J_1)/\mu_2(J_1) \geq \mu_2(J_2)/\mu_2(J_2)$, then J_1 may be the better candidate than J_2 for the reassignment. The algorithm lists the tasks on P_1 in this order, and then reassigns them to P_2 as long as the finishing time can be reduced. The formal description of the algorithm follows.

ALGORITHM F(T, $\hat{L}$, $\hat{f}$)

Step 1. (Initialize) $L_1 \leftarrow \varnothing$; $L_2 \leftarrow \varnothing$, $t_1 \leftarrow 0$, $t_2 \leftarrow 0$

Step 2 (Schedule each task on the processor with the shorter processing time and check which processor finishes last).

```
For i ← 1 to n step 1 do
  if μ1(Ji) ≤ μ2(Ji)
    then [L1 ← L1 ∪ {Ji}; t1 ← t1 + μ1(Ji)]
    else [L2 ← L2 ∪ {Ji}; t2 ← t2 + μ2(Ji)] end
If t1 = t2 then [f̂ ← t1, return]
If t1 > t2 then [α ← 1; β ← 2, go to step 3]
         else [α ← 2; β ← 1]
```

Step 3 ($t_\alpha > t_\beta$, sort L_α and reschedule some tasks in L_α onto P_β)

3.1 Sort L_α so that, by changing subscripts if necessary, $L_\alpha = \{J_1, \quad , J_k \mid k \leq n\}$ and $\mu_\alpha(J_i)/\mu_\beta(J_i) \leq \mu_\alpha(J_{i+1})/\mu_\beta(J_{i+1})$ for $1 \leq i \leq k-1$

```
3 2  For i ← k to 1 step − 1 do
       if tα > tβ + μβ(Ji) then [Lα ← Lα − {Ji}; tα ← tα − μα(Ji); Lβ ← Lβ ∪ {Ji}; tβ ← tβ + μβ(Ji)] end
3.3  If tα ≥ tβ then [f̂ ← tα; return]
            else [f̂ ← tβ, return]
```

LEMMA 1. *Algorithm F has time complexity $O(n \log n)$.*

PROOF. We look at the time needed for each step of the algorithm. Step 1 takes a constant amount of time while step 2 takes $O(n)$ time. For step 3 let $|L_\alpha| = m \leq n$. Then steps 3.1 and 3.2 take $O(m \log m)$ and $O(m)$ time, respectively. Step 3.3 takes a constant amount of time. It follows that the algorithm runs in $O(n) + O(m \log m) + O(m) \leq O(n \log n)$ time. □

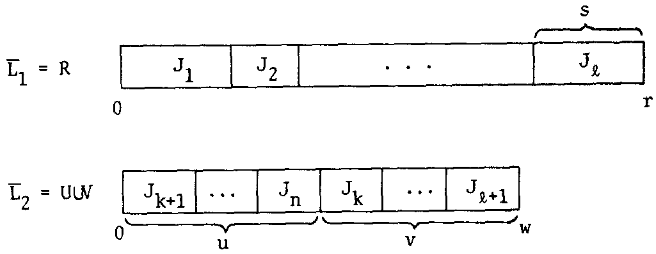
Consider the case $t_1 > t_2$ after step 2. (The case $t_2 > t_1$ is treated similarly.) Suppose that at this time $L_1 = \{J_1, \ldots, J_k\}$ and $L_2 = \{J_{k+1}, \ldots, J_n\}$, where $\mu_1(J_i)/\mu_2(J_i) \leq \mu_1(J_{i+1})/\mu_2(J_{i+1})$ for $1 \leq i \leq k-1$. Let l be the first (largest) integer such that $t_1 \leq t_2 + \mu_2(j_l)$ during the execution of 3.2 of step 3. We consider this intermediate schedule $\bar{L}$, $\bar{L}_1 = \{J_1, \ldots, J_l\}$ and $\bar{L}_2 = \{J_{l+1}, \ldots, J_n\}$. We let $R = \bar{L}_1$, $U = \{J_{k+1}, \ldots, J_n\}$, and $V = \{J_{l+1}, \ldots, J_k\}$ (see Figure 3). An optimal schedule L^* will have a configuration shown on Figure 4. Note that $\bar{f} = \max\{\sum_{j\in\bar{L}_1}\mu_1(J), \sum_{j\in\bar{L}_2}\mu_2(J)\}$ and $f^* = \max\{\sum_{J\in L_1^*}\mu_1(J), \sum_{j\in L_2^*}\mu_2(J)\}$. Let $r = \sum_{J\in R}\mu_1(J)$, $u = \sum_{j\in U}\mu_2(J)$, $v = \sum_{j\in V}\mu_2(J)$, and $w = u + v$. For $j = 1, 2$ let $r_j = \sum_{J\in R_j}\mu_j(J)$, $u_j = \sum_{J\in U_j}\mu_j(J)$, and $v_j = \sum_{J\in V_j}\mu_j(J)$.

We need two lemmas before we can prove the result of this section.

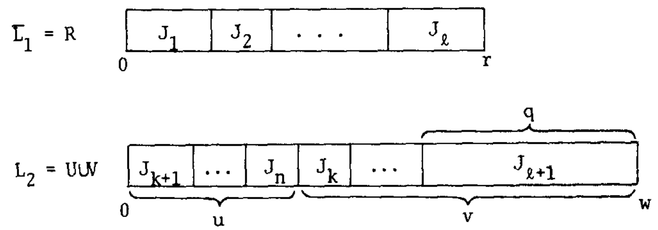
LEMMA 2. *Suppose $\bar{f} = r > w$ (Figure 3 (a)). Then $\bar{f}/f^* \leq (\sqrt{5} + 1)/2$.*

PROOF. Let $s = \mu_1(J_l)$ and $q = \mu_2(J_l)$. Obviously $f^* \geq s$. If $r < [(\sqrt{5} + 1)/2]s$, $\hat{f}/f^* \leq r/s < (\sqrt{5} + 1)/2$. So assume that $r \geq [(\sqrt{5} + 1)/2]s$.

Let $d = r - w$. Clearly $q \geq d$ and $d \leq r$. We claim that a reduction of the finishing time of P_1 by z increases the finishing time of P_2 by at least $(q/s)\cdot z$, where $q/s \geq 1$. Also if

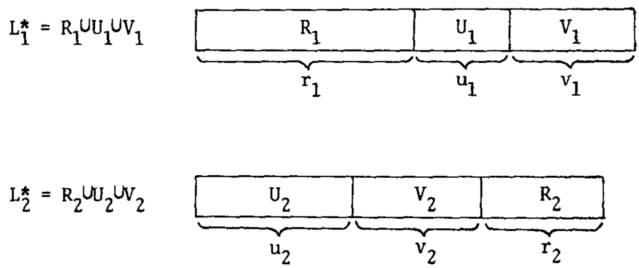


(a) $\bar{f} = t_1 = r$



(b) $\bar{f} = t_2 = w$

FIG 3 Schedule $\bar{L}$



$R = R_1 \cup R_2$, $U = U_1 \cup U_2$ and $V = V_1 \cup V_2$.

FIG 4 An optimal schedule L^*

$d - z$ becomes smaller than $(q/s)\cdot z$, then the finishing time of the schedule starts increasing. Let y be the largest z satisfying $d - z \geq (q/s)\cdot z$, i.e. $d - y = (q/s)\cdot y$ or $y = (s\cdot d)/(s + q)$. Then in fact our claim is that $f^* \geq r - y$. Now we prove this. If $r_1 + u_1 + v_1 \geq r - y$, then obviously $f^* \geq r - y$. So assume that $r_1 + u_1 + v_1 < r - y$. Then

$$f^* \geq r_2 + u_2 + v_2 \geq u_2 + v_2 + (r - r_1)\cdot q/s > u_2 + v_2 + (u_1 + v_1 + y)\cdot q/s = u_2 + u_1\cdot q/s + v_2 + v_1\cdot q/s + y\cdot q/s \geq u + v + y\cdot q/s = w + y\cdot q/s = w + d - y = r - y.$$

Hence

$$\frac{\bar{f}}{f^*} \leq \frac{r}{r - y} = \frac{r}{r - (s\cdot d)/(s + q)} \leq \frac{r}{r - (s\cdot d)/(s + d)} \leq \frac{r}{r - (s\cdot r)/(s + r)} = \frac{s + r}{r} \leq \frac{\sqrt{5} + 1}{2},$$

since $r \geq [(\sqrt{5} + 1)/2]s$. □

LEMMA 3. *Suppose* $\bar{f} = w \geq r$ *(Figure 3(b)). Then* $\bar{f}/f^* \leq (\sqrt{5} + 1)/2$.

PROOF. Let $s = \mu_1(J_{l+1})$ and $q = \mu_2(J_{l+1})$. Clearly $f^* \geq s$. Then

$$\frac{\bar{f}}{f^*} \leq \frac{w}{s} \leq \frac{r+s}{s} = 1 + \frac{r}{s}. \tag{1}$$

Now consider the schedule which results from the one in Figure 3(b) by moving the task J_{l+1} back to P_1. As in the proof of Lemma 2, we claim that the finishing time of an optimal schedule cannot be reduced from $r + s$ by more than z, where z must satisfy $(w - q) + (q/s)\cdot z \leq r + s - z$. Let y be the maximum possible z. Then y satisfies $(w - q) + (q/s)\cdot y = r + s - y$. Thus we are claiming that $f^* \geq r + s - y = w - q + y\cdot(q/s)$. So assume that $r_1 + u_1 + v_1 < r + s - y$. Then

$$\begin{aligned} f^* &\geq r_2 + u_2 + v_2 \geq u_2 + v_2 + (r - r_1)\cdot(q/s) > u_2 + v_2 + (u_1 + v_1 + y - s)\cdot(q/s) \\ &= u_2 + u_1\cdot(q/s) + v_2 + v_1\cdot(q/s) + y\cdot(q/s) - q \geq u + v + y\cdot(q/s) - q = w - q \\ &\quad + y\cdot(q/s). \end{aligned}$$

Now $f^* \geq w - q + y\cdot(q/s)$ implies that $y\cdot(q/s) \leq q$. It follows that

$$\frac{\bar{f}}{f^*} \leq \frac{w}{w - q + y\cdot(q/s)} = \frac{w - q + q}{w - q + y\cdot(q/s)} \leq \frac{q}{y\cdot(q/s)} = \frac{s}{y}. \tag{2}$$

Also, since $w - q + y\cdot(q/s) = r + s - y$, we have

$$\frac{\bar{f}}{f^*} \leq \frac{w}{w - q + y\cdot(q/s)} = \frac{w}{r + s - y} \leq \frac{r+s}{r+s-y} = 1 + \frac{y}{r+s-y}. \tag{3}$$

Now (1) becomes $\bar{f}/f^* \leq 1 + r/s \leq 1 + y/s$ if $y \geq r$, and (3) reduces to $\bar{f}/f^* \leq 1 + y/s$ if $r \geq y$. In either case we have

$$\frac{\bar{f}}{f^*} \leq 1 + \frac{y}{s}. \tag{4}$$

Combining (2) and (4), we have $\bar{f}/f^* \leq \min\{s/y,\ 1 + y/s\}$. The maximum of this minimum occurs when $s/y = 1 + y/s$ or $s = [(\sqrt{5} + 1)/2]y$. Hence $\bar{f}/f^* \leq (\sqrt{5} + 1)/2$. □

THEOREM 2. *Algorithm F has time complexity $O(n \log n)$ and produces a schedule with the finishing time $\hat{f}$ satisfying $\hat{f}/f^* \leq (\sqrt{5} + 1)/2$. Moreover, the bound is the best possible.*

PROOF. The time is given by Lemma 1. Since P_1 and P_2 are symmetric, by Lemmas 2 and 3, $\bar{f}/f^* \leq (\sqrt{5} + 1)/2$. ($\bar{f}$ is interpreted in the same way for the case $t_1 < t_2$ after step 2 of the algorithm.) It is obvious that $\hat{f} \leq \bar{f}$. Thus $\hat{f}/f^* \leq (\sqrt{5} + 1)/2$. Now we give an example which shows that the bound can be approached. Let $T = \{J_1(u, [(\sqrt{5} + 1)/2]u), J_2([(\sqrt{5} + 1)/2]u, [(\sqrt{5} + 3)/2]u - 1)\}$. Algorithm F yields $L_1 = \{J_1\}$ and $L_2 = \{J_2\}$. Hence $\hat{f} = [(\sqrt{5} + 3)/2]u - 1$. The optimal schedule has $L_1^* = \{J_2\}$ and $L_2^* = \{J_1\}$, giving $f^* = [(\sqrt{5} + 1)/2]u$. Thus

$$\frac{\hat{f}}{f^*} = \left(\frac{\sqrt{5}+3}{2}u - 1\right)\Big/\frac{\sqrt{5}+1}{2}u = \frac{\sqrt{5}+1}{2} - \frac{\sqrt{5}-1}{2u} \to \frac{\sqrt{5}+1}{2}$$

as $u \to \infty$. □

4. *Identical Multiprocessor Scheduling*

In this section we look at the case in which all processors are identical; i.e. the processing time of a task is the same on every processor.[6] Even for this case there is no known nonenumerative algorithm for finding a schedule with the optimal (least) finishing time. Finding an optimal schedule is computationally as difficult as solving such problems as the traveling salesman, knapsack, partition, and maximum clique. These problems are

[6] For this system, a task is denoted by $J(\mu(J))$, J being the identity of the task and $\mu(J)$ its processing time

called P-complete problems [8, 9]. It is well known, however, that the LPT schedule produces an $\hat{f}$ such that $\hat{f}/f^* \le \frac{4}{3} - 1/3m$ [6]. The following example shows that this bound is the best possible for any number of jobs, $n \ge 5$: Consider the set of n tasks $T = \{J_1(3), J_2(3), J_3(2), J_4(2), J_5(2), J_6(x), \dots, J_n(x)\}$ to be scheduled on two identical processors, where $x \ll 1/n$. Then the LPT schedule yields $\hat{f} = 7$ while $f^* = 6 + [(n-5)/2]x$. Hence $\hat{f}/f^* = 7/\{6 + [(n-5)/2]x\} \to \frac{7}{6}$ as $x \to 0$. Similar examples show that the bound can be approached for any $m \le 2$.

We shall show that the above problem remains P-complete even when a reasonable restriction is imposed on the processing time. However, the LPT schedule applied to this problem yields an $\hat{f}$ which is near optimal for large n. The simplified problem is the following:

ρ-PT (processing time) problem ($\rho > 1$). Given a set of tasks, $T = \{J_1(\mu(J_1)), \dots, J_n(\mu(J_n))\}$ satisfying $\max_i \mu(J_i)/\min_i \mu(J_i) = \rho$, find a schedule with the minimum finishing time.

In order to prove that the ρ-PT problem is P-complete, we make use of the partition problem, a known member of the P-complete class [8].

Partition problem. Given a multiset $S = \{s_1, \dots, s_n\}$ of positive integers, find partition S_1 and S_2 of S such that $| \sum_{s_i \in S_1} s_i - \sum_{s_j \in S_2} s_j |$ is minimum; i.e. minimize the size of the larger partition.

Suppose a partition problem K has a multiset $S = \{s_1, \dots, s_n\}$. We construct a ρ-PT problem H that is equivalent to K; i.e. a solution to K gives a solution to H and vice versa. Let $\delta = \min\{\delta_1, \delta_2\}$, where $\delta_1 = 2\rho - \lfloor 2\rho \rfloor$ and $\delta_2 = \lceil 2\rho \rceil - 2\rho$. If $\delta \ne 0$, define $W = (3/\delta) \cdot \sum_1^n s_j$, and if $\delta = 0$, define $W = 2 \sum_1^n s_j$. We construct the problem H as follows: Let T be the set of $2n + 2$ tasks defined by

$$T = T_1 \cup T_2 \cup T_3 = \{J_1, \dots, J_n\} \cup \{J_{n+1}, \dots, J_2\} \cup \{J_{2n+1}, J_{2n+2}\},$$

where $\mu(J_i) = 1 + q_i = 1 + s_i/W$ for $J_i \in T_1$, $\mu(J_i) = 1$ for $J_i \in T_2$, $\mu(J_i) = \rho$ for $J_i \in T_3$.[7] The set of tasks in T is to be scheduled on two identical processors in such a way that the finishing time is minimal.

We note that $\sum_1^n q_i = \delta/3$ if $\delta \ne 0$ and $\sum_1^n q_i = \frac{1}{2}$ if $\delta = 0$. If $\delta = 0$, then $\rho \ge 1.5$, and if $\delta \ne 0$, then $\rho \ge 1 + \delta/2$. Thus $\mu(J_i) \le \rho$ for all $1 \le i \le 2n + 2$. Hence H is a ρ-PT problem. It is obvious that K can be transformed in H in polynomial time.

LEMMA 4. *In the case $\delta \ne 0$, any optimal schedule of T has J_{2n+1} and J_{2n+2} scheduled on different processors.*

PROOF. Suppose both J_{2n+1} and J_{2n+2} are scheduled on the same processor, say P_1. Then the fractional part of the finishing time of P_1, ν_1, is given by $\delta_1 \le \nu_1 \le 1 - \delta_2 + \delta/3$. For if no task in T_1 is scheduled on P_1, $\nu_1 = \delta_1$, and if every task in T_1 is scheduled on P_1, $\nu_1 = \delta_1 + \delta/3 = 1 - \delta_2 + \delta/3$. So ν_1 lies between δ_1 and $1 - \delta_2 + \delta/3$. Similarly the fractional part of the finishing time of P_2, ν_2, is given by $0 \le \nu_2 \le \delta/3$. Thus the difference between the finishing times of the two processors d is at least $\min\{\delta_1 - \delta/3, \delta_2 - \delta/3\}$. Since $\delta = \min\{\delta_1, \delta_2\}$, $d \ge \frac{2}{3}\delta$. This schedule, however, cannot be optimal. For consider the schedule in which $T_1 \cup \{J_{2n+1}\}$ is scheduled on P_1 and $T_2 \cup \{J_{2n+2}\}$ is scheduled on P_2. The difference between the finishing times of P_1 and P_2 is $\delta/3$. So this schedule is strictly better than any schedule with J_{2n+1} and J_{2n+2} scheduled on the same processor. □

LEMMA 5. *Suppose $\delta = 0$. Then there is an optimal schedule of T with J_{2n+1} and J_{2n+2} scheduled on different processors.*

PROOF. Consider the case in which ρ is not an integer. Since δ is zero, $\delta_1 = \delta_2 = 0$. Then $\rho = m + \frac{1}{2}$ for some integer $m \ge 1$. Let us consider a schedule L' in which both tasks J_{2n+1} and J_{2n+2} are scheduled on the same processor, say P_1. Since $\sum_{J \in T} \mu(J) = 2(n + \rho) + \frac{1}{2}$, the finishing time of L' is greater than or equal to $n + \rho + \frac{1}{4}$. We compare L' with a schedule L in which P_1 processes J_{2n+1}, J_{n+1}, and all tasks in T except J_1, and

[7] The fractional parts are represented by ordered pairs of integers, i e n/m by (n, m).

P_2 processes J_{2n+2}, J_1, and all tasks in T_2 except J_{n+1}. The finishing time of this schedule is $\max\{\rho + n + \sum_2^n q_i, \rho + n + q_1\} < n + \rho + \frac{1}{2}$ since $0 < q_i < \frac{1}{2}$ and $\sum_1^n q_i = \frac{1}{2}$.

Case 1. P_1 finishes last in L'. Then P_1 must process at least $n - \lceil\rho\rceil$ tasks from $T_1 \cup T_2$. Otherwise the finishing time of P_1 is less than $2\rho + (n - \lfloor\rho\rfloor + \frac{1}{2}) - 1 = n + \rho$. This contradicts the assumption that P_1 finishes last. So P_1 is greater than or equal to $n + \rho + \frac{1}{2}$. However, this schedule cannot be optimal since L has a smaller finishing time.

Case 2. P_2 finishes last in L'. Then P_2 should process at least $n + \lfloor\rho\rfloor + 1$ tasks from $T_1 \cup T_2$. Otherwise, its finishing time is less than or equal to $n + \lfloor\rho\rfloor + \frac{1}{2} = n + \rho$, which is a contradiction. So the finishing time of P_2 is greater than or equal to $n + \rho + \frac{1}{2}$. Again, this cannot be optimal.

Cases 1 and 2 show that an optimal schedule must have J_{2n+1} and J_{2n+2} on different processors.

Consider now the case in which ρ is an integer greater than or equal to n. Suppose that in an optimal schedule J_{2n+1} and J_{2n+2} are on the same processor. Then the finishing time would be greater than or equal to $\rho + n + \frac{1}{2}$ for $\rho = n$ and greater than or equal to $\rho + n + 1$ for $\rho > n$. But this is a contradiction since the schedule L described above has a finishing time less than $\rho + n + \frac{1}{2}$.

Finally, if ρ is an integer less than n and an optimal schedule has both J_{2n+1} and J_{2n+2} on the same processor, say P_1, then we can get another optimal schedule by interchanging J_{2n+2} on P_1 with ρ tasks on P_2 which are in T_2. (At least ρ tasks from T_2 are on P_2 since otherwise the finishing time would be greater than or equal to $2\rho + n - \rho + 1 = \rho + n + 1$.) □

COROLLARY 1. *There is an optimal schedule of T which assigns J_{2n+1} on P_1 and J_{2n+2} on P_2, and P_1 and P_2 process the same number of tasks.*

PROOF. That there is an optimal schedule of T in which J_{2n+1} is on P_1 and J_{2n+2} is on P_2 follows from Lemmas 4 and 5. If P_1 and P_2 do not process the same number of tasks, then one of these processors will have a finishing time of at least $\rho + n + 1$. But then the schedule cannot be optimal, since a schedule with $J_1, \ldots, J_n, J_{2n+1}$ on P_1 and $J_{n+1}, \ldots, J_{2n}, J_{2n+2}$ on P_2 yields a finishing time equal to $\rho + n + \sum_1^n q_i < \rho + n + 1$. □

THEOREM 3. *The ρ-PT problem is P-complete.*

PROOF. We prove that the problems K and H are equivalent by showing that a solution to one gives a solution to the other.

Suppose, by rearranging elements in S if necessary, that a solution to K is the partition $S_1 = \{s_1, \ldots, s_l\}$ and $S_2 = \{s_{l+1}, \ldots, s_n\}$, and assume that S_1 is the larger partition. Then the schedule with $Q_1 = \{J_1, \ldots, J_l, J_{n+1}, \ldots, J_{2n-l}, J_{2n+1}\}$ on P_1 and $Q_2 = \{J_{l+1}, \ldots, J_n, J_{2n-l+1}, \ldots, J_{2n}, J_{2n+2}\}$ on P_2 is optimal, with Q_1 determining the finishing time. Suppose not. Without loss of generality we may assume that P_1 finishes last in an optimal schedule. By Corollary 1, H has an optimal schedule with $Q_1' = \{J_{h_1}, \ldots, J_{h_v}\} \cup \{J_{n+1}, \ldots, J_{2n-v}\} \cup \{J_{2n+1}\}$ on P_1 and the remaining tasks on P_2, where $\{J_{h_1}, \ldots, J_{h_v}\} \subseteq T_1$. By assumption

$$(1 + q_1) + \cdots + (1 + q_l) + \underbrace{1 + \cdots + 1}_{n-l} + \rho > (1 + q_{h_1}) + \cdots + (1 + q_{h_v}) + \underbrace{1 + \cdots + 1}_{n-v} + \rho.$$

Then $q_1 + \cdots + q_l > q_{h_1} + \cdots + q_{h_v}$, and so $s_1 + \cdots + s_l > s_{h_1} + \cdots + s_{h_v}$. This contradicts the assumption that the partition $S = S_1 \cup S_2$ is a solution to K. Hence the schedule Q_1 on P_1 and Q_2 on P_2 is an optimal schedule of problem H. Conversely by Corollary 1 and by rearranging T_1 if necessary, a solution to H is a schedule with $Q_1 = \{J_1, \ldots, J_k\} \cup \{J_{n+1}, \ldots, J_{2n-k}\} \cup \{J_{2n+1}\}$ on P_1 and $Q_2 = \{J_{k+1}, \ldots, J_n\} \cup \{J_{2n-k+1}, \ldots, J_{2n}\} \cup \{J_{2n+2}\}$ on P_2. We assume that P_1 finishes last, i.e. $\sum_{J_i \in Q_1} \mu(J_i) \geq \sum_{J_i \in Q_2} \mu(J_i)$. We claim that $S_1 = \{s_1, \ldots, s_k\}$ and $S_2 = \{s_{k+1}, \ldots, s_n\}$ is a solution to K. Suppose not and assume that K has a solution having the partition $S = S_1' \cup S_2'$ such that $S_1' = \{s_{h_1}, \ldots, s_{h_u}\}$, and it is the larger partition. This implies that $\sum_{s_j \in S_1'} s_j < \sum_{s_j \in S_1} s_j$. But then $(1/W)(s_{h_1} + \cdots + s_{h_u}) < (1/W)(s_1 + \cdots + s_k)$,

$$(1 + q_{h_1}) + \cdots + (1 + q_{h_u}) + \underbrace{1 + \cdots + 1}_{n-u} + \rho < (1 + q_1) + \cdots + (1 + q_k) + \underbrace{1 + \cdots + 1}_{n-k} + \rho,$$

and so $\sum_{J_j \in Q_1} \mu(J_j) < \sum_{J_j \in Q_1'} \mu(J_j)$, where $Q_1' = \{J_{h_1}, \ldots, J_{h_u}\} \cup \{J_{n+1}, \ldots, J_{2n-u}\} \cup \{J_{2n+1}\}$. ($W$ is as defined previously.) This is a contradiction.

Thus the partition problem is reducible to the ρ-PT problem. Since the reduction can be carried out in polynomial time, it follows that a polynomial time algorithm for the ρ-PT problem implies a polynomial time algorithm for the partition problem. From [9] we conclude that the ρ-PT problem is P-complete. □

We now show that the LPT-schedule of the ρ-PT problem is near optimal for large n.

THEOREM 4. *For the ρ-PT problem, the LPT-schedule yields an $\hat{f}$ such that $\hat{f}/f^* \le 1 + 2(m - 1)/n$ when $n \ge 2(m - 1)\rho$. Thus the LPT-schedule is near optimal for large n.*[8]

PROOF. Let $T = \{J_1, \ldots, J_n\}$ be ordered in nonincreasing processing time. Suppose that in the LPT-schedule, J_l is the task with the latest completion time, $l \le n$. We assume, with no loss of generality, that J_l is scheduled on P_1. Then the completion time of P_i, $2 \le i \le m$, is no earlier than the start time of J_l. Hence $f^* \ge \hat{f} - [(m - 1)/m]\mu(J_l)$ or $\hat{f} \le f^* + [(m - 1)/m]\mu(J_l)$. Since $\mu(J_i) \ge \mu(J_l)$ for all $1 \le i \le l$, $f^* \ge (l/m)\mu(J_l)$. Now $(n - l)$ tasks $J_{l+1}, J_{l+2}, \ldots, J_n$ are scheduled on $m - 1$ processors during the processing time of J_l, and $\mu(J_i) \ge \mu(J_n)$ for $l + 1 \le i \le n$. It follows that $\mu(J_l) \ge (n - l)\mu(J_n)/(m - 1)$. Clearly $\mu(J_l) \le \rho \cdot \mu(J_i)$ for all $l \le i \le n$. Then $n - l \le (m - 1)\ \mu(J_l)/\mu(J_n) \le (m - 1)\rho$, and so $l \ge n - (m - 1)\rho$. Assuming that $n \ge 2(m - 1)\rho$, we have

$$\frac{\hat{f}}{f^*} \le \frac{f^* + [(m - 1)/m]\mu(J_l)}{f^*} \le 1 + \frac{[(m - 1)/m]\mu(J_l)}{(l/m)\ \mu(j_l)} = 1 + \frac{m - 1}{n - (m - 1)\rho} \le 1 + \frac{2(m - 1)}{n}. \quad \square$$

REFERENCES

1 BRUNO, J, COFFMAN, E G. JR, AND SETHI, R. Scheduling independent tasks to reduce mean finishing time. *Comm ACM 17*, 7 (July 1974), 382-387

2 BRUNO, J, COFFMAN, E G. JR, AND SETHI, R Algorithm for minimizing mean flow time Information Processing 74, North-Holland, Amsterdam, pp 504-510

3. COFFMAN, E.G. JR, AND SETHI, R A generalized bound on LPT sequencing Proc Int. Symp. Comptr Performance Modeling, Measurement and Evaluation, March 1976, pp 306-317. (available from ACM).

4 CONWAY, R W., MAXWELL, W L, AND MILLER, L W *Theory of Scheduling*. Addison-Wesley Publishing Co, Reading, Mass, 1967

5 GONZALEZ, T, IBARRA, O H, AND SAHNI, S Bounds for LPT schedules on uniform processors. Comptr Sci Tech Rep No 75-1, U of Minnesota, Minneapolis, Minn, 1974

6 GRAHAM, R L Bounds on multiprocessing timing anomalies. *SIAM J Applied Math 17*, 2 (March 1969), 416-429

7 HOROWITZ, E, AND SAHNI, S Exact and approximate algorithms for scheduling nonidentical processors *J ACM 23*, 2 (April 1976), 317-327

8 KARP, R M Reducibility among combinatorial problems In *Complexity of Computer Computations*, R E Miller and J W. Thatcher, Eds, Plenum Press, New York, 1972, pp. 85-103

9 SAHNI, S Computationally related problems *SIAM J Comptg 3*, 4 (Dec 1974), 262-279

RECEIVED APRIL 1975; REVISED JULY 1976

[8] It was pointed out to us by the referee that a result similar to our Theorem 4 had appeared [3]