values of $t_1, t_2, \ldots, t_{m-1}$ in terms of t using (4.2.4). ■

<u>Corollary 4.2.2</u> There exist uniform processor systems and job sets S for which $\hat{f}/f^* \simeq 1.5$.

<u>Proof</u> From Theorem 4.2.1 we know that there are jobs sets, S , for which $\hat{f}/f^* = c$ where c is a positive root of (4.2.8). Let s be a root. Rearranging terms, we get:

$$2s^m - 1 = \sum_{0 \le i < m} s^i$$

$$= \frac{s^m - 1}{s-1}$$

or $2s^{m+1} - 3s^m - s + 2 = 0$.

Since $s > 1$, for $m \to \infty$ we have $s \to 3/2$ as a root. ■

<u>Example 4.2.1</u>

(a) $m = 2$:

Then we have $2s_2^2 - s_2 - 2 = 0$, where we find $s_2 = \frac{1+\sqrt{17}}{4}$. Of course, $s_1 = 1$. Let $t_2 = t_3 = 1$. From equation (4.2.4), we find $t_1 = \frac{2t}{s_2} \cdot s_1 = \frac{8}{1+\sqrt{17}}$.

One easily verifies that $\hat{f}/f^* = \frac{1+\sqrt{17}}{4}$.

(b) $m = 3$:

The equation to use is $2s_3^3 - s_3^2 - s_3 - 2 = 0$. $s_3 = 1.384$ is an approximate root of this equation.

Using equation (4.2.7), we find $s_2 = \frac{s_3(2s_3-1)}{2} = 1.223$ and $s_1 = 1$. Let $t_3 = t_4 = t = 1$. Using equation (4.2.4), find $t_2 = \frac{2t}{s_3} \cdot s_2 = 1.767$ and $t_1 = \frac{2t}{s_3} \cdot s_1 = 1.445$. Again we can check that f/f^* is approximately 1.384.

(c) Some other roots of (4.2.8) are 1.493 for $m = 10$ and 1.499 for $m = 20$. ■

4.3 Special Case (1, 1,...,1 , s)

In this section we study the special case in which all but one of the $m \geq 1$ processors has a speed of 1. The mth processor P_m has a speed $s \geq 1$. The main result of this section is stated below as Theorem 4.3.1.

Theorem 4.3.1 For $m \geq 2$ the ratio $\hat{f}/f^*$ has the following bounds:

(i) $\hat{f}/f^* \leq (1 + \sqrt{17})/4$ for $m = 2$

(ii) $\hat{f}/f^* \leq 3/2 - 1/(2m)$ for $m > 2$.

Proof (i) is proved in Lemma 4.3.2. (ii) follows from Lemmas 4.3.1 - 4.3.6 and the fact that the bound is a monotone increasing function in m. ■

Before proving the theorem we derive a general bound for $\hat{f}/f^*$ in terms of m and s.

Lemma 4.3.1 For an m-processor system with $s_i = 1$ for $1 \leq i < m$ and $s_m = s$, $\hat{f}/f^* \leq \frac{2(m-1+s)}{m-1+2s}$.

Proof If $m = 1$, the lemma is obviously true since $\hat{f}/f^* = 1$. Now assume that the lemma holds for $1,2,\ldots,$ $m-1$ processors but fail for m $(m \geq 2)$. For this m, Let $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ be the smallest set of jobs for which $\hat{f}/f^* > \frac{2(m-1+s)}{m-1+2s}$. Suppose a processor is idle in either the LPT or optimal schedule of S.

Then $\hat{f}^S_m/f^{*S}_m \leq \hat{f}^S_{m-1}/f^{*S}_{m-1} \leq \frac{2(m-2+s)}{m-2+2s} < \frac{2(m-1+s)}{m-1+2s}$ by Lemma 4.2.1 .

So we may assume that no processor is idle in either the LPT or optimal schedule of S. We consider two cases, both leading to a contradiction.

Case 1 The LPT schedule is as shown in Fig. 4.3.1 , where each T_i represents the sum of execution times of jobs scheduled on P_i prior to the assignment of t_n, $T_1 + T_2 + \ldots + T_m = t_1 + t_2 + \ldots + t_{n-1}$. By assumption, no processor is idle. Hence $T_i > 0$ for $2 \leq i \leq m$. Since the first $m-1$ processors have speed 1, we may assume that $T_i \geq T_1$ and $1 < i \leq m-1$. Now if $T_1 = 0$, then $\hat{f} = t_n$. But $f^* \geq t_n$ since by assumption no processor is idle in the optimal schedule. Then $\hat{f}/f^* = 1$. So we may also assume that $T_1 \geq t_n$. Then

$$\hat{f}/f^* \leq \frac{T_1 + t_n}{(\Sigma T_i + t_n)/(m-1+s)} \leq \frac{(m-1+s)(T_1 + t_n)}{(m-1)T_1 + T_m + t_n}$$

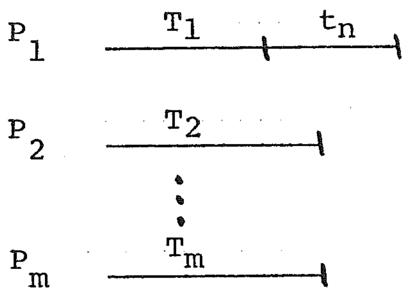


Fig. 4.3.1

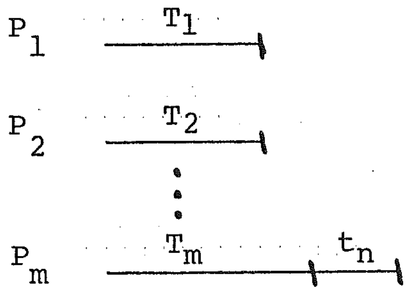


Fig. 4.3.2

Now, since t_n determines the finish time,

$\frac{T_m + t_n}{s} \geq T_1 + t_n$ or $T_m \geq sT_1 + (s-1)t_n$. Then

$$\hat{f}/f^* \leq \frac{(m-1+s)(T_1+t_n)}{(m-1)T_1+sT_1+(s-1)t_n+t_n}$$

$$= \frac{(m-1+s)(T_1+t_n)}{(m-1)T_1 + s(T_1+t_n)}$$

$$= \frac{m-1+s}{s + \frac{(m-1)T_1}{T_1+t_n}}$$

$$\leq \frac{m-1+s}{s + \frac{m-1}{2}}$$

since the minimum $\frac{T_1}{T_1+t_n}$ with the constrain $T_1 \geq t_n$ occurs when $T_1 = t_n$.

Hence, $\hat{f}/f^* \leq \frac{2(m-1+s)}{m-1+2s}$, a contradiction.

<u>Case 2</u> Suppose the LPT schedule is as shown in Fig. 4.3.2, where we again assume that $T_i \geq T_1 \geq t_n$. We may also assume that $T_m > 0$; otherwise $\hat{f}/f^* = 1$ since $\hat{f} = t_n/s$.

Then $\hat{f}/f^* \le \dfrac{\dfrac{T_m + t_n}{s}}{(\Sigma T_i + t_n)/(m-1+s)}$

$$\le \frac{(m-1+s)\left(\frac{T_m + t_n}{s}\right)}{(m-1)T_1 + T_m + t_n}$$

Since t_n is scheduled on P_m, $T_1 + t_n \ge \dfrac{T_m + t_n}{s}$.

We have two subcase:

(a) We can find a T such that $t_n \le T \le T_1$ and $T + t_n = \dfrac{T_m + t_n}{s}$. Then

$$\hat{f}/f^* \le \frac{(m-1+s)(T+t_n)}{(m-1)T + T_m + t_n}$$
$$= \frac{(m-1+s)(T+t_n)}{(m-1)T + s(T_1 + t_n)}$$
$$= \frac{(m-1+s)}{s + \frac{(m-1)T}{T + t_n}}$$
$$\le \frac{(m-1+s)}{s + \frac{m-1}{s}} = \frac{2(m-1+s)}{m-1+2s}$$

Again, we get a contradiction.

(b) If (a) is not possible, we let $T = t_n$. Then $T + t_n = 2t_n > \dfrac{T_m + t_n}{s}$ or $T_m < (2s-1)t_n$. Then

$$\hat{f}/f^* \le \frac{(m-1+s)\left(\frac{T_m + t_n}{s}\right)}{(m-1)T_1 + T_m + t_n}$$
$$\le \left(\frac{m-1+s}{s}\right)\frac{T_m + t_n}{(m-1)t_n + T_m + t_n}$$
$$= \left(\frac{m-1+s}{s}\right)\frac{1}{1 + \frac{(m-1)t_n}{T_m + t_n}}$$

$$\leq \left(\frac{m-1+s}{s}\right) \frac{1}{1 + \dfrac{(m-1)t_n}{(2s-1)t_n + t_n}}$$

$$= \frac{2(m-1+s)}{m-1+2s} \quad , \text{ a contradiction. } \blacksquare$$

The bound for $m = 2$ follows from the following lemma.

Lemma 4.3.2 For an m processor system with $s_i = 1$, $1 \leq i < m$ and $s_m = s$, $\hat{f}/f^* \leq \frac{(3-m) + \sqrt{(3-m)^2 + 16(m-1)}}{4}$ Moreover, for $m = 2$, the bound is tight.

Proof Let $k > 1$ be the desired bound for $\hat{f}/f^*$. Let $Q = \Sigma s_i = m-1+s$. First we show that if $s \leq \frac{2Q(k-1)}{m-1}$, then $\hat{f}/f^* \leq k$. Suppose not. Let $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ be the smallest set of jobs for which $\hat{f}/f^* > k$. Then t_n determines the finish time and by Lemma 4.2.2, $\hat{f}/f^* \leq 1 + \frac{(m-1)t_n}{Qf^*}$. Hence $f^* < \frac{(m-1)t_n}{Q(k-1)}$. It follows that the number of jobs on each processor in the optimal schedule of S is less than $\frac{(m-1)s}{Q(k-1)} \leq 2$. But then this case, $\hat{f}/f^* = 1$. This contradicts the assumption that S produces a bound $> k$. Thus if $s \leq \frac{2Q(k-1)}{m-1}$, then $\hat{f}/f^* \leq k$. This, in turn, implies that if $Q \leq (m-1) + \frac{2Q(k-1)}{m-1}$, then $\hat{f}/f^* \leq k$ or that

(4.3.1) if $Q \leq \frac{(m-1)^2}{m-2k+1}$ then $\hat{f}/f^* \leq k$. Now by Lemma 4.3.1, we have $\hat{f}/f^* \leq \frac{2(m-1+s)}{m-1+2s} = \frac{2(m-1+s)}{2(m-1+s)-(m-1)}$

$= \frac{2Q}{2Q-(m-1)}$. It follows that if $\frac{2Q}{2Q-(m-1)} \leq k$, then $\hat{f}/f^* \leq k$ or

(4.3.2) if $Q \geq \frac{(m-1)k}{2(k-1)}$ then $\hat{f}/f^* \leq k$.

To satisfy (4.3.1) and (4.3.2) simultaneously, we must have $\frac{(m-1)^2}{m-2k+1} = \frac{(m-1)k}{2(k-1)}$, from which we get

$k = \frac{(3-m) + \sqrt{(3-m)^2 + 16(m-1)}}{4}$. In this case $\hat{f}/f^* \leq k$ for all Q .

For the case $m = 2$, we have $k = \frac{1 + \sqrt{17}}{4}$ which is tight since we have seen an example for which the bound is achieved. ■

In arriving at the proof of the theorem for $m > 2$, it is necessary to prove four lemmas. To begin with, we show that if for any set of jobs, S , an optimal schedule has more than one job on any of the processors $P_1, P_2, \ldots, P_{m-1}$ then $\hat{f}^S/f^{*S} \leq 3/2 - 1/(2m)$.

Lemma 4.3.3 For any set of jobs, S , either

(i) processors P_1-P_{m-1} have at most one job scheduled on each in every optimal schedule

or (ii) $\hat{f}^S_m/f^{*S}_m \leq 3/2 - 1/(2m)$.

Proof Suppose (ii) is not true for some set of jobs. Let $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ be the smallest set of jobs for which $\hat{f}^S_m/f^{*S}_m > 3/2 - 1/(2m)$. From Lemma 4.2.2 we get

$$\hat{f}_m^S/f_m^{*S} \le 1 + \frac{(m-1)\,t_n}{(m-1+s)\,f_m^*} > 3/2 - 1/(2m)$$

or

$$\frac{(m-1)\,t_n}{(m-1+s)\,f_m^*} > \frac{m-1}{2m}$$

or

$$t_n > \frac{m-1+s}{2m}\, f_m^*$$

$$\ge (1/2)\ f_m^*$$

I.e., $f_m^* < 2t_n$ which, in turn, means that none of the processors P_1-P_{m-1} can have more than one job scheduled on them in an optimal schedule. ■

Next, we prove that if $s \ge m-1$ then $\hat{f}/f^* \le 4/3$.

Lemma 4.3.4 If $s \ge m-1$ then $\hat{f}/f^* \le 4/3 \le \frac{3}{2} - \frac{1}{2m}$ for $m > 2$.

Proof Lemma 4.3.1 gives

$$\hat{f}/f^* \le \frac{2(m-1+s)}{m-1+2s}$$

The right hand side of the above inequality is a decreasing function of s . Hence, for $s \ge m-1$ we obtain

$$\begin{aligned}\hat{f}_m/f_m^* &\le \frac{4m-4}{3(m-1)}\\ &= 4/3\\ &\le 3/2 - 1/(2m) \qquad m > 2 .\end{aligned}$$ ■

As a result of Lemmas 4.3.3 and 4.3.4 the only counter examples to Theorem 4.3.1 are sets of jobs, S, for which the optimal schedules have at most one job on each of $P_1 - P_{m-1}$ and the speed, s , of P_m is $<m-1$.

The next two lemmas show that for this kind of an optimal and $s < m-1$ the bound of theorem 4.3.1 cannot be violated.

Lemma 4.3.5 Let $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ be the smallest set of jobs for which $\hat{f}/f^* > 3/2 - 1/(2m)$. If in the LPT schedule, t_i is the only job scheduled on one of the processors, $P_1, P_2, \ldots, P_{m-1}$ and if in an optimal schedule t_j is the only job scheduled on one of the processors, $P_1, P_2, \ldots, P_{m-1}$ then, either

$$\text{(i)} \quad \hat{f}^S_m / f^{*S}_m \leq f_{m-1}/f^*_{m-1}$$

or

$$\text{(ii)} \quad t_i < t_j$$

Proof From Lemma 4.1.1 it follows that t_n determines the finish time $\hat{f}^S$. If anyone of the processors $P_1, P_2, \ldots, P_m$ is idle in an optimal solution (i.e. no jobs have been scheduled on it) then $f^*_m = f^*_{m-1}$. But, $\hat{f}^S_m \leq \hat{f}^S_{m-1}$ and so $\hat{f}^S_m/f^{*S}_m \leq \hat{f}^S_{m-1}/f^{*S}_{m-1}$. We may therefore assume that no processor is idle in any optimal solution. Hence, $f^{*S}_m \geq t_n$. If $i = n$ then $\hat{f}^S_m = t_n$ (as t_i is the only job on some processor $P_1, P_2, \ldots, P_{m-1}$) and $\hat{f}^S_m/f^{*S}_m \leq 1$. Therefore $i \neq n$. Now, we have

$$f^{*S}_m = \max\{t_j, f^{*S-\{t_j\}}_{m-1}\}$$

$$\geq f^{*S-\{t_j\}}_{m-1}$$

$$\geq f^{*S-\{t_i\}}_{m-1} \qquad \ldots \text{ as } \quad t_i \geq t_j$$

but, $\hat{f}^S_m = \hat{f}^{S-\{t_i\}}_{m-1} \qquad \ldots$ as $i \neq n$

$$\therefore \quad \hat{f}^S_m / f^{*S}_m \leq \hat{f}^{S-\{t_i\}}_{m-1} / f^{*S-\{t_i\}}_{m-1}$$
$$\leq \hat{f}_{m-1}/f^*_{m-1} \quad ■$$

Lemma 4.3.6 When $s < m-1$ and an optimal schedule for any set of jobs S has at most one job on each of processors $P_1 - P_{m-1}$ then $\hat{f}_m/f^*_m \leq 3/2 - 1/(2m)$.

Proof Let $S = (t_1 \geq t_2 \geq \ldots \geq t_n)$ be the smallest set of jobs and m the least $m > 2$ for which the lemma is not true. From Lemma 4.3.1 we obtain

$\hat{f}/f^* \leq 1 + \dfrac{m-1}{m-1+s} \dfrac{t_n}{f^*}$. By assumption $\hat{f}/f^* > 3/2 - 1/(2m)$. Therefore,

$$1 + \frac{(m-1)}{m-1+s} \frac{t_n}{f^*} > 3/2 - 1/(2m)$$

or

$$f^* < \frac{2m}{m-1+s} t_n \qquad \ldots \quad (4.3.3)$$

If $\#_m$ is the number of jobs on P_m in an optimal schedule then, $f^* \geq \#_m t_n / s$. Substituting this inequality into (4.3.3) yields:

$$\#_m < \frac{2sm}{m-1+s} \qquad \ldots \quad (4.3.4)$$

The right hand side of the inequality (4.3.4) is an increasing function of s . Since $s < m-1$ (4.3.4) yields the following bound on $\#_m$:

$$\#_m < \frac{2(m-1)m}{2(m-1)} = m .$$

The optimal schedule has at most one job on each of $P_1 - P_{m-1}$. Hence, $n \leq 2m-2$.

The remainder of the proof shows that if $n \leq 2m-2$ then Lemma 4.3.5 can be used to show that $\hat{f}^S_m / f^{*S}_m \leq \hat{f}^S_{m-1} / f^{*S}_{m-1}$ thus contradicting the assumption that this was the least m for which the lemma was false . (The contradiction comes about as $3/2 - 1/(2m)$ is monotone increasing in m and the fact that when $m = 3$ this bound is $4/3$ which is greater than the known bound for $m = 2$.) Clearly, we may assume that each processor has at least one job scheduled on it in every optimal schedule.

Let k be the smallest index (i.e. largest job) on any of the processors $P_1 - P_{m-1}$ in an optimal schedule. Then, the schedule obtained by assigning job t_{k+i-1} to processor P_i , $1 \leq i < m$ and the remain - ing jobs to processor P_m has a finish time no greater than the optimal finish time f^{*S}_m . Such a schedule shall be denoted by OPT_k . Clearly, $1 \leq k \leq n-m+2$. Since, $n \leq 2m - 2$ at least one of the processors $P_1 - P_{m-1}$ has exactly one job scheduled on it (every processor must have at least one job on it as otherwise , by the definition of LPT $\hat{f} \leq t_n$ but $f^* \geq t_n$). Let the index of this job be i . Then, t_i must be the largest job amongst jobs scheduled on P_1-P_{m-1} in the LPT schedule (this again follows from the definition of LPT). But, $s < m-1$ implies $t_i \geq t_{m-1}$ as LPT cannot schedule all of the first $m-1$ jobs on P_m when $s < m-1$. For all $k \geq 1$, OPT_k has a job with index $j = k + m - 2 \geq m - 1$ on P_{m-1} and this is the only job on P_{m-1}. By the ordering on the jobs, $t_j \leq t_{m-1}$. So, $t_i \geq t_j$. Lemma 4.3.5 now implies that $\hat{f}_m^S / f_m^{*S} \leq \hat{f}_{m-1} / f_{m-1}^*$; a contradiction. ■

Having shown that $\hat{f}/f^*$ is indeed bounded as in Theorem 4.3.1, the next question is: How good is the bound. From the previous section we know that the bound for $m = 2$ is tight. Lemma 4.3.7 shows that the bound is also tight for $m = 3$ and that for all $m > 3$ it is possible to have an $\hat{f}/f^*$ arbitrarily close to 4/3. Lemma 4.3.8 shows that for $m = 4$ and 5 there is no set of jobs S for which $\hat{f}/f^* > 4/3$. This shows that the bound of $3/2 - 1/(2m)$ is not a tight bound for all values of m and leads us to conjecture that for $m \geq 3$ the bound is in fact 4/3. Note the closeness of this bound of 4/3 to the bound $4/3-1/(3m)$ obtained by Graham [13] for the case of $s = 1$ (i.e. m identical processors).

<u>Lemma 4.3.7</u> For $m \geq 3$ and any $\varepsilon > 0$, there is a set of jobs, S, and a speed $s > 1$ for which

$\hat{f}/f^* > 4/3 - \varepsilon$.

Proof For any $m \geq 3$ consider the set of jobs $t_1 = 1.5$ $t_2 = 1.5$, $t_j = 1$, $3 \leq j \leq m+2$ and $s = 2 + \varepsilon'$ with ε' very close to zero . The LPT schedule has jobs t_1, t_2 and t_{m+2} on P_m with $\hat{f} = 4/(2+\varepsilon')$. One optimal schedule is shown in figure 3.3 . $f^* = 1.5$. Hence, $\hat{f}/f^* = \frac{8}{6+3\varepsilon'} \to 4/3$ as $\varepsilon' \to 0$. ■

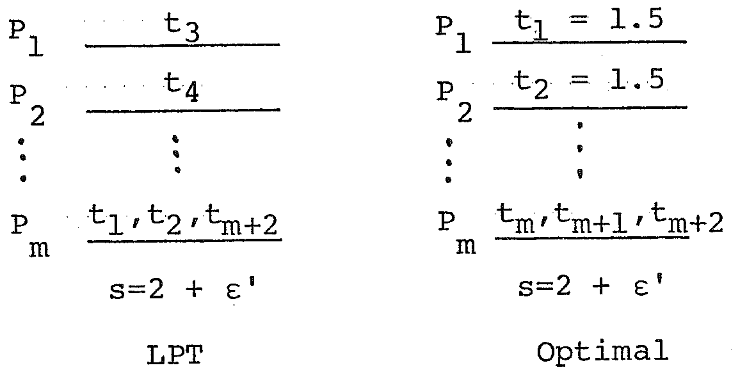


Figure 4.3.3 LPT and Optimal Schedules for Lemma 4.3.7

Lemma 4.3.8 For $m = 4$ and 5 , $\hat{f}/f^* \leq 4/3$

Proof We prove only the case $m = 4$. The proof for $m = 5$ is very similar and does not use any new tech - niques. The proof for $m = 4$ is by cases on the possible values of n and s . In what follows, we assume that the smallest set of jobs for which $\hat{f}/f^* \geq 4/3$ is of size n and then arrive at a contradic - tion for all values of n .

<u>case a</u> $s \geq 3$. Substituting in Lemma 4.3.1 we obtain, for $s \geq 3$ and $m = 4$, $\hat{f}/f^* \leq \frac{3+s}{1.5+s} \leq \frac{6}{4.5} = 4/3$.

<u>case b</u> $n \leq 4$ there is either only one job on each of the four processors are idle in the optimal schedule . In the first case $\hat{f}/f^* = 1$, in the second $\hat{f}_4/f^*_4 \leq \hat{f}_3/f^*_3 \leq 4/3$.

<u>case c</u> $n = 5$ In both the LPT and optimal schedule there is job t_i schedule alone on one of P_1-P_3 . Lemma 4.3.5 applies and $\hat{f}_4/f^*_4 \leq \hat{f}_3/f^*_3 \leq 4/3$.

<u>case d</u> $n = 6$ $1.5 \leq s < 3$ Lemma 4.2.2 yields $\hat{f}/f^*$ $\leq 1 + \frac{3t_n}{(3+s)f^*}$. By the assumption on the set of jobs $\hat{f}/f^* > 4/3$. So, $\frac{3t_n}{(3+s)f^*} > 1/3$ or $f^* < \frac{9}{3+s} t_n$ $\leq 2t_n$. The number of jobs on P_1 -P_3 is thus restricted to 1 and the number on P_4 is restricted to ≤ 4 . The total number of jobs, n , must be ≤ 7 . When $n = 2m - 2 = 6$, the proof of Lemma 4.3.6 applies as the optimal has at most one job on each of P_1-P_3 and $s \leq m-1$. Hence, $\hat{f}_4/f^*_4 \leq \hat{f}_3/f^*_3 \leq 4/3$. $s < 1.5$: (i) $t_1 \notin P_4$ in optimal. There must be at least 2 jobs on P_4 as otherwise we may interchange the job on P_4 with t_1 without increasing the finish time. So, at least two processors in the optimal schedule have only one job each. We may assume these jobs to be t_1 and t_2 . Since $s < 2$, t_2 is in P_1 and alone in the LPT

schedule. Lemma 4.3.5 now applies and $\hat{f}/f^* \leq 4/3$.

(ii) $t_1 \in P_4$ and $t_2 \notin P_4$ in optimal. Lemma 4.3.5 again applies.

(iii) $t_1 \in P_4$ and $t_2 \in P_4$ in optimal. Now. either Lemma 4.3.5 applies or $\hat{f} = (t_1+t_6)/s \leq f^*$.

<u>case e</u> $n = 7$: $1.5 \leq s < 3$ From case d we know that in the optimal each of P_1-P_3 has exactly one job scheduled on it while there are 4 jobs scheduled on P_4 . We examine all the possibilities.

(i) If $t_1 \notin P_4$ in the optimal then the optimal may be assumed to be:

P_1	t_1
P_2	t_2
P_3	t_3
P_4	t_4, t_5, t_6, t_7

In the LPT schedule jobs t_2 and t_3 cannot be alone on P_1 , P_2 or P_3 as, then Lemma 4.3.5 would apply and $\hat{f}_4/f_4^* \leq \hat{f}_3/f_3^*$. Also, if t_1 is the only job on P_4 in the LPT schedule, then $\hat{f} \leq (t_1 + t_7)/s$ while $f^* \geq t_1$. So, $\hat{f}/f^* \leq (t_1 + t_7)/(st_1) \leq 2/s \leq 4/3$. This takes care of all possible LPT schedules with 7 jobs.

(ii) $t_1 \in P_4$ in the optimal and $t_2 \notin P_4$. This is very similar to (i). Unless in the LPT schedule t_1 is the only job scheduled on P_4, Lemma 4.3.5 applies and $\hat{f}/f^* \leq 4/3$. If t_1 is the only job on P_4 then $\hat{f} \leq (t_1 + t_7)/s$ while $f^* \geq (t_1 + t_5 + t_6 + t_7)/s \geq \hat{f}$.

The only remaining possibility is :

(iii) $t_1 \in P_4$ and $t_2 \in P_4$ in optimal. Now, $\hat{f} \leq (t_1 + t_2 + t_6 + t_7)/s$ for all possible LPT schedules, while $f^* \geq (t_1 + t_2 + t_6 + t_7)/s$.

<u>$s < 1.5$</u>: (i) if $t_1 \notin P_4$ and $t_2 \notin P_4$ in optimal then $f^* \geq t_2 + t_7$ case d $f^* < \frac{9}{3+s} t_n$ => no more than two jobs on each of $P_1 - P_3$. But, $\hat{f} \leq t_2 + t_7$ as there are only 7 jobs.

(ii) $t_1 \notin P_4$ and $t_2 \in P_4$ in optimal => $f^* \geq \max\{t_1, \frac{t_2 + t_7}{s}\}$ since, $\hat{f} \leq t_2 + t_7$ $\hat{f}/f^* \leq s$ and so s must be $> 4/3$ if $\hat{f}/f^*$ is to be $> 4/3$.

<u>$4/3 < s < 1.5$</u>:

If t_1 is alone or with t_7 only on P_4 in the LPT schedule then $\hat{f} \leq (t_1 + t_7)/s$ $\leq 3f^*/(2s) \leq 9f^*/8$

So, t_1 must be paired with a job other than t_7. Hence, $\hat{f} \leq t_3 + t_7$

if $t_3 \notin P_4$ in optimal then $f^* \geq t_3 + t_7$

if $t_3 \in P_4$ in optimal then $f^* \geq (t_2 + t_3)/s$

$\Rightarrow t_3 \leq s/2f^*$ $\Rightarrow \hat{f} \leq (\frac{s}{2} + \frac{1}{2}) f^* \leq 1.25f^*$ ($t_7 \leq f^*/2$ as with $s \leq 1.5$ the number of jobs on P_4 must be less than 3)

(iii) $t_1 \in P_4$ in the optimal

If t_1 is alone in the optimal then $\hat{f}/f^*$ is no worse than for identical processors. So, $\hat{f}/f^* \leq 4/3$ (see [13]) .

If t_1 is alone in the LPT schedule or coupled only with t_7 then $\hat{f} \leq (t_1 + t_7)/s \leq f^*$. So, $\hat{f}$ must be $\leq t_3 + t_7$.

If $t_2 \in P_4$ in optimal then $f^* \geq (t_1 + t_2)/s \geq 2t_2/s$ but $\hat{f} \leq t_3 + t_7 \leq (\frac{s}{2} + \frac{1}{2}) f^* \leq 1.25f^*$.

If $t_2 \notin P_4$ in optimal then t_2 is alone on P_1 . In the LPT, since , t_1 is not alone on P_4 , t_2 must be alone on P_1 . So, Lemma 4.3.5 applies and $\hat{f}/f^* \leq 4/3$.

<u>case f</u> $n = 8$

If $s \geq 1.5$ then case d requires at most 1 job on each of P_1-P_3 in optimal and at most 4 jobs on P_4. So, $n \leq 7$.

<u>$s \leq 1.5$</u>

(i) If $t_1 \notin P_4$ in optimal then since there can be at most 2 jobs on each processor, $f^* \geq t_1 + t_8$. Using the technique of case d we get for $\hat{f}/f^* > 4/3$, $f^* < \frac{9}{3+s} t_8 \leq (9/4)t_8$ or $t_8 > (4/9)f^*$. Hence, $t_1 < (5/9)f^*$. If t_1 is the only job on P_4 in the LPT schedule or t_1 and t_8 are the only jobs on P_4 then $\hat{f} \leq (t_1 + t_8)/s \leq f^*$. Hence, $\hat{f} \leq t_2 + t_8 \leq 2t_1 \leq (10/9)f^*$.

(ii) $t_1 \varepsilon P_4$ in optimal.
$f^* \geq (t_1 + t_8)/s$ and so for $\hat{f}/f^* > 4/3$ there must be a job other than t_1 and t_8 on P_4 in the LPT schedule. This implies $\hat{f} \leq t_2 + t_8 \leq f^*$ if $t_2 \notin P_4$ in optimal. Assume now that both t_1 and t_2 are on P_4 in the optimal. Then $f^* \geq (t_1 + t_2)s \geq 2t_2/s \Rightarrow t_2 \leq s/2f^*$. This, together with the knowledge that $t_8 \leq f^*/2$ and $\hat{f} \leq t_2 + t_8$ results in $\hat{f} \leq (\frac{s}{2} + \frac{1}{2})f^* \leq 1.25f^*$.

<u>case g</u> $n > 8$ Substitttuting this into Lemma 4.2.2 yields

$$\hat{f}/f^* \leq 1 + \frac{m-1}{n} \leq 1 + (3/9) = 4/3 \ .$$

This takes care of all the possibilities and so for $\hat{f}_4/f_4^* \leq 4/3$. ■

<u>Conjecture</u> $\hat{f}/f^* \leq 4/3$ for $m \geq 3$ and $s_i = 1$, $1 \leq i < m$ and $s_m \geq 1$.

CHAPTER V

OPEN SHOP

5.1 Introduction

A shop consists of $m \geq 1$ processors (or machines). Each of these processors performs a different task. There are $n \geq 1$ jobs. Each job i has m tasks. The processing time for task j of job i is $t_{j,i}$. Task j of job i is to be processed on processor j, $1 \leq j \leq m$. A schedule for a processor j is a sequence of tuples $(\ell_i, s_{\ell_i}, f_{\ell_i})$, $1 \leq i \leq r$. The ℓ_i are job indexes, s_{ℓ_i} is the start time of job ℓ_i and f_{ℓ_i} is the finish time. Job ℓ_i is processed continuously on processor j from s_{ℓ_i} to f_{ℓ_i}. The tuples in the schedule are ordered such that $s_{\ell_i} < f_{\ell_i} \leq s_{\ell_{i+1}}$, $1 \leq i < r$. There may be more than one tuple per job and it is assumed that $\ell_i \neq \ell_{i+1}$, $1 \leq i < r$. It is also required that each job i spends exactly $t_{j,i}$ total time on processor j. A schedule for a m-shop is a set of m processor schedules. One for each processor in the shop. In addition, these m processor schedules must be such that no job is to be processed simultaneously on two or more processors. A shop schedule will be abbreviated to schedule in future references. The finish time of a schedule is the latest completion time of the individual processor schedules and represents the time at which all tasks have been completed. An optimal finish time (OFT) schedule is one which has the least finish time amongst all schedules. A non-preemptive schedule is one in which the individual processor schedules has at most one tuple (i, s_i, f_i) for each job i to be scheduled. For any processor, j, this allows for $t_{j,i} = 0$ and also requires that $f_i - s_i = t_{j,i}$. A schedule in which no restriction is placed on the number of tuples per job per processor is preemptive. Note that all non-preemptive schedules are also preemptive while the reverse is not true.

Open shop schedules differ from flow shop and job shop [5,7] schedules in that in an open shop, no restrictions are placed on the order in which the tasks for any job are to be processed. In this Chapter we shall investigate OFT schedules for the open shop. It is clear that when $m = 1$, OFT schedules can be trivially obtained. We shall therefore restrict ourselves to the case $m > 1$. First, in section 5.2 we show that preemptive and nonpreemptive OFT schedules can be obtained in linear time when $m = 2$. This contrasts with Johnson's O(n log n) algorithm [7, p89] for the 2 processor flow shop. When $m > 2$ OFT preemptive schedules can still be obtained in polynomial time (section 5.3).

For nonpreemptive scheduling, however, finding OFT schedules when $m > 2$ is NP-Complete. These results may be compared to similar results obtained for flow shop and job shop OFT scheduling. In [11] and in Chapter VI it is shown that finding nonpreemptive OFT schedules for the flow shop when $m > 2$ and the job shop when $m > 1$ are NP-Complete. In Chapter VI it is also shown that finding preemptive OFT schedules for the 3 processor flow shop and 2 processor job shop are NP-Complete. Thus, as far as the complexity of finish time scheduling is concerned, open shops are easier to schedule when a preemptive schedule is desired.

5.2 OFT Scheduling for m = 2

In this section, a linear time algorithm to obtain a nonpreemptive and preemptive OFT schedule for the case of two processors is presented. For notational simplicity, we denote $t_{1,i}$, the task time on processor 1, by a_i and $t_{2,i}$ by b_i, $1 \leq i \leq n$. Informally, the algorithm proceeds by dividing the jobs into two groups A and B. The jobs in A have $a_i \geq b_i$ while those in B have $a_i < b_i$. The schedule is build from the "middle" with jobs from A being added on at the right while those from B are added on at the left. The schedule from the jobs in A is such that there is no idle time on processor 1 (except at the end) and for each job in A, it is possible to start its execution on processor 2 immediately following its completion on processor 1. The part of the schedule made up with jobs in B is such that the only idle time on processor 2 is at the beginning. In addition, the processing of a job on processor 1 can be started such that its processing on processor 2 can be carried out immediately after completion on processor 1. Finally, some finishing touches involving only the first and last jobs in the schedule are made. This guarantees an optimal schedule.

<u>line no.</u>

```
 1   Algorithm OPENSHOP
     // This algorithm finds a minimum finish time non-
        preemptive schedule for the open shop problem
        with task times ($a_i$ ; $b_i$), $1 \le i \le n$
        Initialize variables: $a_0$;$b_0$ represent a dummy
                                        job
                                     $T_i$ = sum of task times
                                        assigned to processor
                                        i,  $1 \le i \le 2$.
```

```
                         ℓ = index of leftmost job in
                             the schedule
                         r = index of rightmost job in
                             the schedule.
                         S_i = sequence for processor i
                               1 ≤ i ≤ 2 //
2   T_1 ← T_2 ← a_0 ← b_0 ← ℓ ← r ← 0 ; S ← null
    // schedule the n jobs //
3   for i ← 1 to n do
4      T_1 ← T_1 + a_i ; T_2 ← T_2 + b_i
5      if a_i ≥ b_i then [ if a_i ≥ b_r then
                         [ // put r on right, || means
                              string concatenation //
6                        S ← S || r ; r ← i ]
                              else  [// put i on right//
7                        S ← S || i ]]
8               else [ if b_i ≥ a_ℓ then
                         [ // put ℓ on left //
9                        S ← ℓ || S ; ℓ ← i ]
                              else
                         [ // put i on left //
10                       S ← i || S ]]
11  end   //now start finishing touch  //
12  delete all occurrances of job 0 from S
13  if T_1 - a_ℓ < T_2 - b_r then [ S_1 ← S || r || ℓ;
                                     S_2 ← ℓ || S || r ]
```

```
14                          else [ $S_1 \leftarrow \ell \,||\, S \,||\, r$ ;
                                   $S_2 \leftarrow r \,||\, \ell \,||\, S$ ]
       // an optimal schedule is obtained by processing
          jobs on processor i in the order specified by
          $S_i$, $1 \le i \le 2$.  The exact schedule may be det-
          ermined using Theorem 5.2.1 //
15     return
16     end of OPENSHOP  ■
```

Example 5.2.1 Consider the open shop problem with 6 jobs having task times as below:

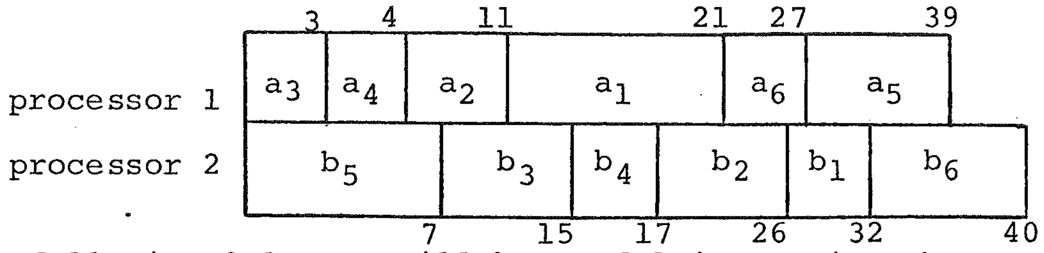
Processor \ Job	1	2	3	4	5	6
1	10	7	3	1	12	6
2	6	9	8	2	7	8

Initially, $\ell = r = 0$ and $S = \emptyset$. The following table gives the values of S, r, ℓ at the end of each iteration of the for loop 3-12.

End of iteration	S	r	ℓ
1	0	1	0
2	00	1	2
3	200	1	3
4	4200	1	3
5	42001	5	3
6	420016	5	3

After deleting the 0's from S we have S = 4216, r = 5,

$\ell = 3$, $T_1 = 39$ and $T_2 = 40$. Since $T_1 - a_3 > T_2 - b_5$ we get $S_1 = 342165$ and $S_2 = 534216$. Processing by these permutations gives the Gantt chart:



The following 2 lemmas will be useful in proving the correctness of algorithm OPENSHOP.

<u>Lemma 5.2.1</u> Let the set of jobs being scheduled be such that $a_i \geq b_i$, $1 \leq i \leq n$ and let D be the permutation obtained after deleting the 0's from S in line 12 of algorithm OPENSHOP and concatenating r to the right. The jobs 1-n may be scheduled in the order D such that:

(i) there is no idle time on processor 1 except following the completion of the last task on this processor

(ii) For every job i, its processor 1 task is completed before the start of its processor 2 task

(iii) for last job r, the difference, Δ, between the completion time of task 1 and the start time of task 2 is zero.

<u>Proof</u> The proof is by induction on n. The lemma is clearly true for $n = 1$. Assume that the lemma is true

for $1 \leq n < k$. We shall show that it is also true for $n = k$. Let the k jobs be $J_1, J_2, \ldots, J_k$ and let r' be the value of r at the beginning of the iteration of the "for" loop of lines 3-11 when $i = n$. From the algorithm it is clear that the per - mutation, D' , obtained at line 12 when the k-1 jobs $J_1, J_2, \ldots, J_{k-1}$ are to be scheduled is of the form D"r'. Moreover, D = D"r'k or D = D"kr'. From the induction hypothesis, it follows that tha jobs $J_1, J_2, \ldots, J_{k-1}$ can be scheduled according to the permutation D"r' so as to satisfy (i) - (iii) of the lemma. I.e., these k-1 jobs may be scheduled as in figure 5.2.1 . Let i be the job immediately preceding r' in D'. In case k = 2, let i = 0 with $a_0 = b_0 = 0$.

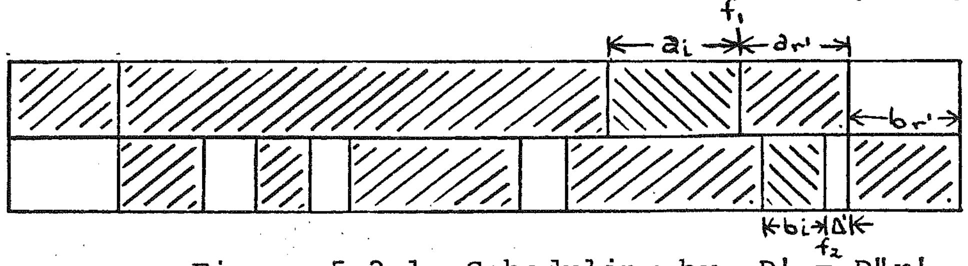


Figure 5.2.1 Scheduling by D' = D"r'

indicate task processing. Last job is r'. $\Delta' \geq 0$.

If $A_k \geq b_{r'}$ then D = D'k and it is clear the job k can be added on to the schedule of figure 5.2.1 at the right end, so that (i) - (iii) of the lemma hold.

If $A_k < b_{r'}$ then D = D"kr'. Now, job r' is moved a_k units to the right so that a_k can be accomodated between i and r' satisfying (i). Let f_1 be the finish time of a_i and $f_2 \geq f_1$ be the finish time of b_i. The finish time of a_k is then $f_1 + a_k < f_1 + a_{r'}$ as $a_{r'} \geq b_{r'}$. By (iii) the start time of $b_{r'}$ has to be $f_1 + a_k + a_{r'}$. Also, we know, from the induction hypothesis, that $f_1 + a_{r'} - f_2 = \Delta' \geq 0$. I.e. $f_1 + a_{r'} \geq f_2$. The earliest that b_k may be scheduled is $\max\{f_1 + a_k, f_2\} < f_1 + a_{r'}$. This implies that there is enough time between the start time of $b_{r'}$ and the earliest start time of b_k to complete the processing of b_k. ■

Lemma 5.2.2 Let the set of jobs being scheduled be such that $a_i < b_i$, $1 \leq i \leq n$ and let C be the permutation obtained after deleting the 0's from S in line 12 of algorithm OPENSHOP and concatenating ℓ to the left. The jobs may be scheduled in the order C such that:

(i) there is no idle time on processor 2 except at the beginning

(ii) for every task i, its processor 1 task is completed before the start of its processor 2 task.

(iii) for the first job, ℓ, the difference, Δ between the completion time of task 1 and the start time of task 2 is zero.

Proof The proof is similar to that of Lemma 5.2.1. ■

Lemma 5.2.3 Let (a_i, b_i) be the processing times for job i on processors 1 and 2 respectively, $1 \leq i \leq n$. Let f^* be the finish time of an optimal finish time preemptive schedule. Then, $f^* \leq \max\{\max_i\{a_i + b_i\}, T_1, T_2\}$ where $T_1 = \sum_1^n a_i$ and $T_2 = \sum_1^n b_i$.

Proof Obvious. ■

We are now ready to prove the correctness of algorithm OPENSHOP.

Theorem 5.2.1 Algorithm OPENSHOP generates optimal finish time schedules.

Proof Let $J_1, J_2, \ldots, J_n$ be the set of jobs being scheduled. Let A be the subset with $a_i \geq b_i$ and B be the remaining jobs. It is easy to verify that the theorem is true when either A or B is empty. So, assume A and B to be nonempty sets. Let E be the permutation obtained after deleting the 0's from $\ell||S||r$ in line 12. Then $E = CD$ where C consists solely of jobs in B and D consists solely of jobs in A . From Lemmas 5.1.1 and 5.1.2, it follows that the jobs in A and B may be scheduled in the orders D and C to obtain schedules as in figure 5.2.2. In the schedules of figure 5.2.2, the processor 1 tasks for C and the processor 2 tasks for D have to be scheduled such that all the idle time appears either at the end or at the beginning.

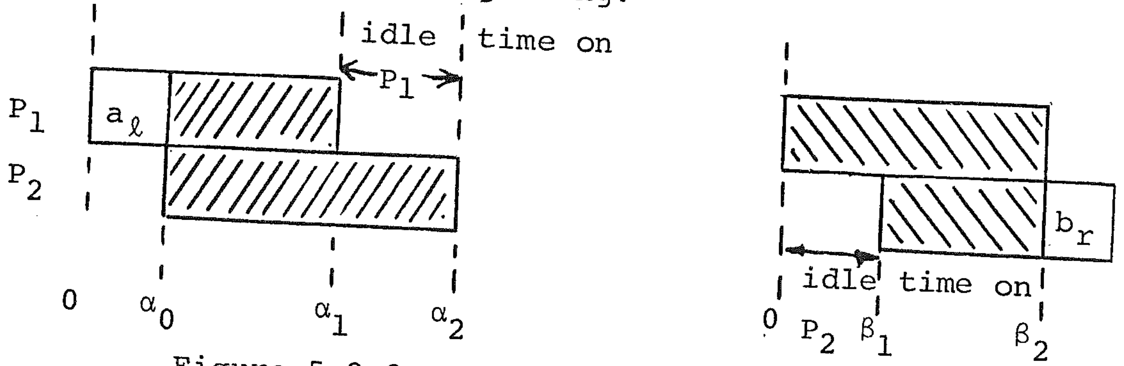


Figure 5.2.2 Partial schedules obtained for sets B and A respectively.

Let $T_1 = \sum_1^n a_i$ and $T_2 = \sum_1^n b_i$. The schedule for the entire set of jobs is obtained by merging the two schedules of figure 5.2.2 together so that either

(a) The blocks on P1 meet first. This happens when $(\alpha_2 - \alpha_1) \leq \beta_1$.

or (b) The blocks on P2 meet first. This happens when $(\alpha_2 - \alpha_1) > \beta_1$.

Let us consider these two cases separately.

<u>case a</u> $\alpha_2 - \alpha_1 \leq \beta_1$

This happens when $T_1 - a_\ell \geq T_2 - b_r$. In this case, line 14 of the algorithm results in the tasks on P1 being processed in the order CD while those on

P2 are processed in the order rCD' where D' is D with r deleted. The section $\alpha_0 - \alpha_2$ of figure 5.2.2 (a) is now shifted right until it meets with $\beta_1 - \beta_2$ of figure 5.2.2 (b). Task b_r is moved to the leftmost point. The finish time of the schedule obtained becomes $\max\{a_r + b_r, T_1, T_2\}$ which by Lemma 5.2.3 is optimal.

case b $\alpha_2 - \alpha_1 > \beta_1$

This happens when $T_1 - a_\ell < T_2 - b_r$. In this case, line 13 of the algorithm results in the tasks on P1 being processed in the order $C'D\ell$ where C' is C with ℓ deleted. Tasks on P2 are processed in the order CD. The schedule is obtained by processing tasks on P2 with no idle time starting at time 0. Tasks on P1 are processed with no idle time (except at the end) in the order $C'D$. Task a_ℓ is started as early as possible following $C'D$. The finish time is seen to be $\max\{a_\ell + b_\ell, T_1, T_2\}$ which by Lemma 5.2.3 is optimal.

This completes the proof. ■

Corollary 5.2.1 Algorithm OPENSHOP generates optimal preemptive schedules for $m = 2$.

Proof Follows immediately from the proof of Theorem 5.2.1 . ■

<u>Lemma 5.2.4</u> The time complexity of algorithm OPENSHOP is $O(n)$.

<u>Proof</u> The "for" loop of lines 3-11, is iterated n times. Each iteration takes a fixed amount of time. The remainder of the algorithm takes a constant amount of time. Hence the complexity is $O(n)$. ■

5.3 Preemptive OFT Scheduling $m > 2$

We now show that optimal preemptive schedules can be found in polynomial time when $m > 2$. To begin with, we present a fairly simple algorithm to do this. This algorithm reduces the problem to that of finding maximal matchings in bipartite graphs [14]. Refinements of this basic procedure then lead to a more efficient algorithm.

Given a set of n jobs with task times $t_{j,i}$, $1 \leq i \leq n$ and $1 \leq j \leq m$ for a m processor open shop , we define the following quantities:

$T_j = \sum_{1 \leq i \leq n} t_{j,i}$ ... total time needed on processor j, $1 \leq j \leq m$

$L_i = \sum_{1 \leq j \leq m} t_{j,i}$ ... length of job i, $1 \leq i \leq n$

From a simple extension of Lemma 5.2.3 to m processors, we know that every preemptive schedule must have a finish time that is at least

$$\alpha = \max_{i,j} \{T_j, L_i\} \qquad (5.3.1)$$

We will in fact show that the optimal preemptive schedule always has a finish time of α. From the given open shop problem we construct a bipartite graph with $2(n+m)$ vertices. $n + m$ of these are labeled $J_1, J_2, \ldots, J_{n+m}$ to represent the n jobs together with m fictitious jobs that we shall introduce. The remaining vertices are labeled $M_1, M_2, \ldots, M_{n+m}$ to represent the m processors together with n fictitious processors. The bipartite graph, G, will contain undirected weighted edges between J and M type vertices. The weight, $w(J_i, M_j)$, of an edge (J_i, M_j) will represent the amount of processing time job i requires on processor j. The weight of a node , $p(J_i) = L_i$ or $p(M_j) = T_j$, is the sum of the weights of the edges incident to this node. To begin with, the following edges with nonzero weight are included in G:

$$E_1(G) = \{(J_i, M_j) \text{ and } w(J_i, M_j) = t_{j,i} \mid t_{j,i} \neq 0, \; 1 \leq i \leq n, \; 1 \leq j \leq m\} \qquad (5.3.2)$$

Now, a set of edges, $E_2(G)$, connecting $J_1, J_2, \ldots, J_n$ to $M_{m+1}, M_{m+2}, \ldots, M_{m+n}$ are added in such a way that $p(J_i) = \alpha$, $1 \leq i \leq n$.

$$E_2(G) = \{(J_i, M_{m+i}) \text{ and } w(J_i, M_{m+i}) = \alpha - L_i \mid \alpha - L_i \neq 0, \quad 1 \leq i \leq n\} \qquad (5.3.3)$$

A set of edges, $E_3(G)$, is included to connect $M_1, M_2, \ldots, M_m$ to $J_{n+1}, J_{n+2}, \ldots, J_{n+m}$ in such a way that $p(M_j) = \alpha$, $1 \leq j \leq m$.

$$E_3(G) = \{(J_{n+j}, M_j) \text{ and } w(J_{n+j}, M_j) = \alpha - T_j \mid \alpha - T_j \neq 0, \quad 1 \leq j \leq m\} \qquad (5.3.4)$$

Finally, edges connecting $J_{n+1}, J_{n+2}, \ldots, J_{n+m}$ to $M_{m+1}, M_{m+2}, \ldots, M_{m+n}$ are added to make the weight of each of these vertices α. This set of edges, E_4, is of size at most $n+m$ as each (J_i, M_j) edge introduced brings the weight of either J_i or M_j to α. One may easily verify that E_4 can be so constructed.

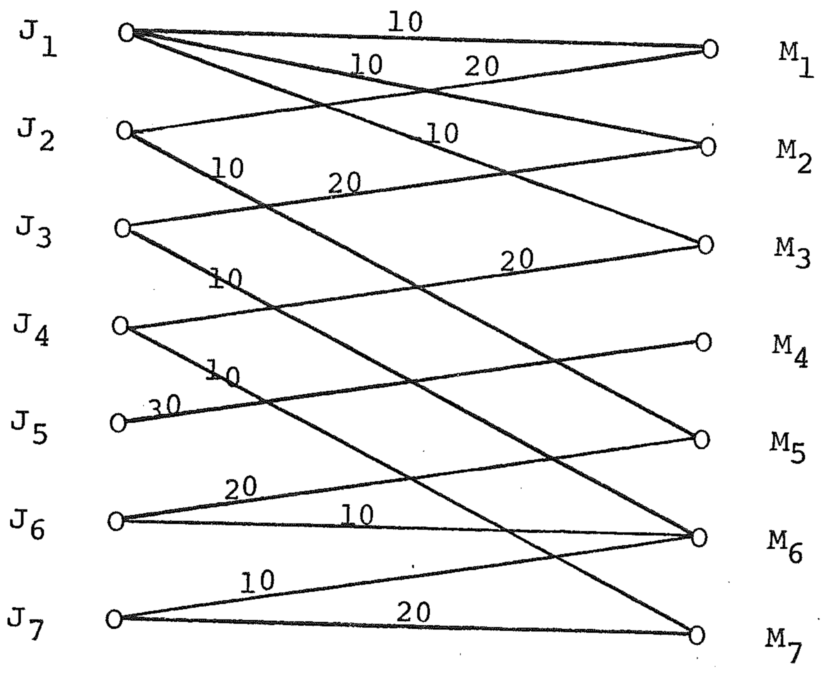
The bipartite graph $G(X,Y,E)$ is then $(\{J_1, J_2, \ldots, J_{n+m}\}, \{M_1, M_2, \ldots, M_m\}, E_1 \cup E_2 \cup E_3 \cup E_4)$. X is the set of vertices representing jobs, while Y is the set representing processors.

We illustrate this construction with an example.

Example 5.3.1 Let $m = 3$ and $n = 4$. The task times are defined by the matrix:

$$
\begin{array}{llccccc}
 & \text{job} \rightarrow & 1 & 2 & 3 & 4 & T \\
\text{processor} & 1 & 10 & 20 & 0 & 0 & 30 \\
 & 2 & 10 & 0 & 20 & 0 & 30 \\
 & 3 & 10 & 0 & 0 & 20 & 30 \\
 & & & & & & \\
 & L & 30 & 20 & 20 & 20 &
\end{array}
$$

$\therefore$ $\alpha = 30$. The bipartite graph obtained using the above construction is :



The edge set E_3 is empty as $T_j = p(M_j) = \alpha$, $1 \leq j \leq m$. ■

Before proceeding with the description of the preemptive OFT algorithm, let us review some terminology regarding matchings in graphs. The following definition and propositions are reproduced from [14].

Definition 5.3.1 Let $G = (X \cup Y, E)$ be a bipartite graph with vertex set $X \cup Y$ and edge set E. (If (i,j) is an edge in E then either $i \varepsilon X$ and $j \varepsilon Y$ or $i \varepsilon Y$ and $j \varepsilon X$.) A set $I \subseteq E$ is a matching if no vertex $v \varepsilon X \cup Y$ is incident with more than one edge in I. A matching of maximum cardinality is called a maximum matching. A matching is called a complete matching of X into Y if the cardinality (size) of I equals the number of vertices in X. $\blacksquare$

Definition 5.3.2 Let I be a matching. A vertex v is free relative to I if it is incident with no edge in I. A path (without repeated vertices) $P = (v_1,v_2)(v_2,v_3)\ldots(v_{2k-1},v_{2k})$ is called an augmenting path if its end points v_1 and v_{2k} are both free, and its edges are alternately in $E - I$ and in I. $\blacksquare$

Proposition 5.3.1 I is a maximum matching iff there is no augmenting path relative to I. $\blacksquare$

Note that when a matching I is augmented by an augmenting path P the resulting matching I' is $(I \cup P) - (I \cup P)$ and is of cardinality $1 +$ cardinality (I). Also note that the matching I' still matches all vertices that were in the matching I (two new vertices v_1 and v_{2k} are however added on.)

Proposition 5.3.2 If $G = (\{X \cup Y\}, E)$ is a bipartite graph, $|E| = e$, $|X| = n$ and $|Y| = m$, $n \geq m$ then an augmenting path relative to I starting at some free vertex v can be found in time $O(\min\{m^2, e\})$. ■

Keeping these facts about bipartite graphs and matchings in mind, let us resume the description of the preemptive OFT algorithm. Having constructed the bipartite graph G from the open shop problem as described earlier, we obtain a complete matching of $X = \{J_1, J_2, \ldots, J_{n+m}\}$ into $Y = \{M_1, M_2, \ldots, M_{n+m}\}$. Let this matching be $e_1, e_2, \ldots, e_{n+m}$. Let $r = \min_{1 \leq i \leq n+m} \{w(e_i)\}$. The jobs incident to the edges $e_1, e_2, \ldots, e_{n+m}$ are scheduled on their respective processors for a time period of r and the weight of the edges $e_1, e_2, \ldots e_{n+m}$ is decreased by r. This results in the deletion of at least one edge (i.e. the weight of at least one edge becomes zero). By scheduling a job on its respective processor we mean that if (J_i, M_j) is one of the edges in the match then job i is processed on processor j for r units of time. If $j > m$ then job i is not processed in that interval. If $i > n$ then processor j is idle in that time interval. This process is repeated until all edges are deleted. Assuming that at each iteration, a matching of size n+m can be found, all $n+m$ processors are kept busy at all times (either processing real or fictitious jobs). The total processing time needed is $\sum_{1}^{n+m} p(M_i) = (n+m)\alpha$. Hence the finish time of the schedule is $(n+m)\alpha/(n+m) = \alpha$ and the schedule is optimal. Since each time a complete matching is found, one edge is deleted, complete mat - chings have to be found at most $O(nm)$ times (note that the number of edges in G is at most $O(nm)$). Hence the maximum number of preemptions per processor is $O(nm)$. The first match can be found in time $O(nm (n+m)^{.5})$ [14]. Subsequent matches require finding augmenting paths, each of which can be determined in time $O(nm)$ (Propositions 5.3.2 with $e \simeq O(nm)$). Since a total of $O(nm)$ such paths may be needed, the total computing time for the process becomes $O(n^2m^2)$.

Example 5.3.2 Let us try out the informal computational process described above on the bipartite graph of example 5.3.1. The following complete matchings are obtained (this is not a unique set of matchings):

a) $\{(J_1,M_2),(J_2,M_1),(J_3,M_6),(J_4,M_3),(J_5,M_4),(J_6,M_5),(J_7,M_7)\}$, $r = 10$

b) $\{(J_1,M_1),(J_2,M_5),(J_3,M_2),(J_4,M_3),(J_5,M_4),(J_6,M_6),(J_7,M_7)\}$, $r = 10$

C) $\{(J_1,M_3),(J_2,M_1),(J_3,M_2),(J_4,M_7),(J_5,M_4),(J_6,M_5),(J_7,M_6)\}$, $r = 10$

This yields the following schedule:

	10	10	10
M1	J2	J1	J2
M2	J1	J3	J3
M3	J4	J4	J1
M4	J5	J5	J5
M5	J6	J2	J6
M6	J3	J6	J7
M7	J7	J7	J4

Deleting the fictitious jobs and processors, the following preemptive schedule is obtained:

	10	10	10
M1	J2	J1	J2
M2	J1	J3	J3
M3	J4	J4	J1

The schedule requires only 1 preemption i.e. on M1. Since the edge set E_3 was empty, there is no idle time on any of the processors. In general, however, this will not be the case and the deletion of the fictitious jobs will leave some idle time on the processors. ■

The success of the algorithm rests in the existence of a complete matching at each iteration. The next 3 lemmas prove that a complete match always exists. The vertices of the graph are divided into two disjoint sets $X = \{J_1, J_2, \ldots, J_{n+m}\}$ and $Y = \{M_1, M_2, \ldots, M_{n+m}\}$.

Lemma 5.3.1 At each iteration, the weight of every vertex in the bipartite graph is equal.

Proof By construction, this is certainly true for the first iteration, i.e. $p(M_i) = p(J_i) = \alpha$, $1 \leq i \leq n+m$. After a complete match is found, the weight of $n+m$ edges decreases by r . The $2(n+m)$ vertices of G are each incident to exactly one edge in the matching. Hence, the weight of each vertex decreases by r. Consequently, all vertices have the same weight at all times. ■

Lemma 5.3.2 In a bipartite graph a complete matching of vertex set Y into vertex set X exists if and only if $|A| \leq |R(A)|$ for every subset A of Y, where $R(A)$ denotes the set of vertices in X that are adjacent to the vertices in A .

Proof See Liu [28], p. 282 Theorem 11.1 . ■

Lemma 5.3.3 The conditions of Lemma 5.3.2 are valid for every bipartite graph with vertices of equal weight.

Proof Let α be the weight of a vertex. Let A be any subset of Y . Then, the sum of the weights of vertices in A is $\alpha|A|$. The corresponding sum for

$R(A)$ is $\alpha|R(A)|$. Since this sum includes all edges incident to A , we have $\alpha|A| \leq \alpha|R(A)|$ and so $|A| \leq |R(A)|$, as $\alpha > 0$. ■

Our algorithm to obtain an optimal preemptive schedule is based upon a refinement of the informal computational procedure described above. The bipartite graph constructed consists of the two vertex sets $X = \{J_1, J_2, \ldots, J_{m+n}\}$ and $Y = \{M_1, M_2, \ldots, M_m\}$. The edge set is $E_1 \cup E_3$ (cf. eq(5.3.2) and (5.3.4)). I.e. the fictitious processors of the earlier construction are dispensed with. Now, we look for complete matchings of Y into X. While before, any complete match of Y into X was acceptable, now we have to be careful about the matching that is chosen. To see this, note that if initially the matching $\{(J_2,M_1),(J_3,M_2),(J_4,M_3)\}$ is chosen for the job set of example 5.3.1 then there is no complete matching at the next iteration and consequently no schedule with finish time α can be obtained following this choice of a matching. To assist in proper choice of a complete matching we make use of an additional vector S called the slack vector. For every job i , its slack time is defined to be the difference between the amount of time remaining in the schedule and the amount of processing left for that job. In the slack time for a job becomes zero then it is essential that the job be processed continuously up to the completion of the schedule at α as otherwise the schedule lenght will be $> \alpha$. When the slack time for a job becomes zero, the job is said to have become <u>critical</u>.

<u>Example 5.3.3</u> Consider the 3 processor open shop problem with 4 jobs and the following task times:

processor \ job	1	2	3	4	T
1	10	8	5	3	26
2	6	7	9	9	31
3	7	8	3	3	21
L	23	23	17	15	$\alpha = \max_{i,j}\{T_i, L_j\} = 31$

Addition of the jobs J_5, J_6 and J_7 introduces 3 more columns into the above table $\begin{bmatrix} 5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix}$

Initially, the slack times are $\alpha - L_i$ and we have $S = (8, 8, 4, 6, 26, 31, 21)$. No job is critical. ■

We first state the algorithm and then prove its correctness. For convenience, the vector S in algorithm P instead of representing slack times actually represents the latest time a job may start so that its processing may be complete by α. Thus

SLACK(i) = S_i - current time. A job therefore becomes critical when S_i = current time.

Algorithm P

```
        // Obtain an optimal preemptive schedule for the
           m processor open shop with n jobs and proces
           sing time $t_{i,j}$, $1 \le i \le n$, $1 \le j \le m$. //
        // compute lenght, $\alpha$, of optimal schedule //
1          $T_j \leftarrow \sum_{i=1}^{n} t_{j,i}$, $1 \le j \le m$
2          $L_i \leftarrow \sum_{j=1}^{m} t_{j,i}$, $1 \le i \le n$
3          $\alpha \leftarrow \max_{i,j} \{T_j, L_i\}$
        // create fictitious jobs and compute slack
           vector //
4          $t_{j,n+j} \leftarrow \alpha - T_j$,   $1 \le j \le m$
5          $S_i \leftarrow \alpha - L_i$,   $1 \le i \le n$
6          $S_{n+j} \leftarrow T_j$,   $1 \le j \le m$
7          $n \leftarrow n + m$
        // compute initial complete matching of $Y = \{M_1,
           M_2, \dots, M_m\}$ into $X = \{J_1, J_2, \dots, J_{n+m}\}$.
           This match is obtained as a set, I, of edges
           (j,i) matching $M_j$ to $J_i$ //
8          I $\leftarrow$ INITIAL_MATCHING ;  TIME $\leftarrow$ 0  //current
                                                  time //
9          repeat
10         $\ell \leftarrow$ index of job not in matching having least
```

```
         slack time
11       (p,q) ← task and job in matching with least
                 remaining processing time.
12       $\Delta \leftarrow \min\{t_{p,q}, S_\ell - \text{TIME}\}$  // max time for
                                             for which  I  can be
                                             used //
      // schedule  I  for  Δ  time units //
13       if Δ > 0 then  [ print (Δ,I);
                          $t_{j,i} \leftarrow t_{j,i} - \Delta$  for $(j,i) \in I$
                          $S_i \leftarrow S_i + \Delta$  for all jobs $i \in I$
                          TIME ← TIME + Δ
                          if TIME = $\alpha$ then stop]
14       delete from  I  all pairs  (i,j)  such that
         $t_{j,i} = 0$
      // complete matching  I  including all critical
         jobs //
15       if there is a critical job not in  I  then
                   [ delete from  I  all pairs  (j,i)
                     such that  i  is noncritical
16                   repeat
17                       let  $J_\ell$  be a critical job
                         not in  I
18                       augment  I  using an augmen
                            ting path starting at $J_\ell$
19                   until there is no critical job
                          not in  I
```

```
20                       reintroduce into  I  all pairs
                              (j,i)  thet were deleted in
                              line 15 and such that  $M_j$
                              is still free
    // complete the match //
21      while size of $I \neq m$ do
22              let $M_j$ be a processor not in the matching
                        I
23              augment  I  using an augmenting path
                        starting at  $M_j$
24      end
25      forever
26      end of algorithm  P  ■
```

In order to prove the correctness of algorithm P we have to show the following:

(i) There exists an initial complete matching in line 8.

(ii) The matching I can be augmented so as to include the critical job J_ℓ in line 18

(iii) Augmenting to a complete match including all critical jobs can always be carried out as required in lines 21-24.

The following three lemmas show that these three requirements can always be met. α is as defined in line 3 of the algorithm.

<u>Lemma 5.3.4</u> There exists a complete matching of Y into X in line 8.

<u>Proof</u> Let A be any subset of vertices in Y. The weight of each vertex in A is α . The weight of any vertex in X is $\leq \alpha$ by definition of α . Since the weight of $R(A) \geq$ weight of A , it follows that $\alpha|A| \leq \alpha|R(A)|$ and so $|A| \leq |R(A)|$. The result now follows from Lemma 5.3.2 . ■

<u>Lemma 5.3.5</u> In line 18 there exists an augmenting path relative to I starting at J_ℓ .

<u>Proof</u> Consider the bipartite graph, G' , formed by the vertices X' and Y where X' consists of all vertices representing jobs in the matching I and the vertex J_ℓ . All edges connecting X' and Y in the original graph are included in G' . By the deletion of line 15 it follows that all vertices in X' are critical. Hence, their weight is $\alpha - t$ if t is the value of TIME when the loop of lines 16-19 is being executed. Since $\alpha - t$ is the total remaining time on all the processors, the weight of vertices in Y in the graph G' is $\leq \alpha - t$. Using the same argument as in Lemma 5.3.4, it follows that there is a complete match of X' into Y . Hence I is not a maximum matching in G' . Hence there is an augmenting path relative to

I beginning at J_ℓ . ■

Lemma 5.3.6 There is always an augmenting path relative to I beginning at M_j in line 23.

Proof At any time t , the bipartite graph formed by vertices $X = \{J_1, J_2, \ldots, J_{n+m}\}$ and $Y' = \{M_i | M_i$ is in the matching $I\} \{M_j\}$ have the following properties:

(a) The weight of vertices in Y' is $\alpha - t$

and (b) the weight of vertices in X is $\leq \alpha - t$ (as no vertex can have a slack time < 0 , see lines 11-13).

Hence, the conditions of the proof of Lemma 5.3.4 hold and there is a complete matching of Y' into X. By proposition 5.3.1 there must be an augmenting path relative to I beginning at the free vertex M_j .

Note that the complete matching obtained at the end of the "while" loop 21-24 must contain all the critical jobs as the initial matching I contained all of them and augmenting paths only add on vertices to an existing matching.

Since all processors are kept busy at all times and the total amount of processing is $m\alpha$, the finish time of the schedule generated by algorithm P is α . This schedule is therefore optimal. ■

All that remains now is to analyze the complexity of algorithm P. In carrying out this analysis we shall need a bound on the number of jobs that can become critical. This bound is provided by the next lemma. Lemma 5.3.8 itself analyzes the algorithm.

Lemma 5.3.7 The number of critical jobs at any time is $\leq m$.

Proof Since all processors are kept busy at all times, it follows that at any time t the total amount of processing remaining is $m(\alpha-t)$. If at time t there are more than m critical jobs then the processing remaining for all these critical jobs $\geq (m+1)(\alpha-t) > m(\alpha-t)$. A contradiction. Since, once a job becomes critical, it stays critical till the end of the schedule, the total number of jobs that can become critical is also $\leq m$. ■

Lemma 5.3.8 The asymptotic time complexity of algorithm P is $O(\min\{e,m^2\}(m+e) + em \log n)$ where n is the number of jobs, m the number of processors and e the number of nonzero tasks. e is assumed $\geq \max\{n,m\}$.

Proof Lines 1-7 take time $O(e)$ if the task times are maintained using linked lists (see Knuth [7]). Line 8 can be carried out in time $O(em^{.5})$ (see Hopcroft and Karp [5]). If the slack times are set up as a balanced search tree (Knuth [7]) then each execution of line 10 takes time $O(m \log n)$. At each iteration of the 'repeat forever' loop (line 9-25) either a critical job is created or a task is completed (see lines 10-13). Hence, by lemma 5.3.7, the maximum number of iterations of this loop is $e + m = O(e)$. The total contribution of line 10 is therefore $O(em \log n)$. The contribution from lines 11-12 and 14 is $O(em)$. In line 13 the change in S_i requires deletion and insertion of m values from the balanced search tree. This requires a time of $O(m \log n)$. The total contribution of line 13 is therefore $O(em \log n)$. Line 15 has the same contribution. The total computing time for algorithm P is therefore $O(em \log n$ + total from lines 16-24). Over the entire algorithm the loop of lines 16-19 is iterated at most m times. By proposition 5.3.2 an augmenting path can be found in time $O(\min\{e,m^2\})$. The total time for this loop is therefore $O(\min\{e,m^2\}m + m \log n)$. The maximum number of augmenting paths needed in the loop of lines 21-24 is $m + e$ (as one path is needed each time a critical job is found). The computing time of algorithm P then becomes $O(\min\{e,m^2\}(m+e) + em \log n)$. ∎

5.4 Complexity of Nonpreemptive Scheduling for $m > 2$

Having presented a very efficient algorithm to obtain a OFT schedule for $m = 2$ (pre and nonpreemptive) and a reasonable efficient algorithm to obtain a OFT preemptive schedule for all $m > 2$, the next question that arises is: Is there a similar efficient algorithm for the case of nonpreemptive schedules when $m > 2$. We answer this question by showing that this problem is NP-Complete [21] even when we restrict ourselves to the case when the job set consists of only one job with 3 nonzero task times while all other jobs have only 1 nonzero task time. This, then, implies that obtaining a polynomial time algorithm for $m > 2$ is as difficult as doing the same for all the other NP-Complete problems. An even stronger result can be obtained when $m > 3$. Since NP-Complete problems are normally stated as language recognition problems, we restate the OFT problem as such a problem.

LOFT: Given an open shop with $m > 2$ processors and a set of n jobs with processing times $t_{j,i}$, $1 \le j \le m$, $1 \le i \le n$ there is a nonpreemptive schedule with finish time $\le \tau$. ■

In proving LOFT NP-Complete, we shall make use of the Partition problem defined in section 2.1 and shown NP-Complete in [21].

Theorem 5.4.1 LOFT with $m = 3$, one job having 3 tasks with nonzero processing times and the remaining jobs having only 1 task with nonzero processing time is NP-Complete.

Proof It is easy to show that LOFT can be recognozed in nondeterministic polynomial time by a Turing machine. The Turing machine just guesses the optimal permutation on each of the processors and verifies that the finish time is $\leq \tau$. ■

The remainder of the proof is presented in lemma 5.4.1.

Lemma 5.4.1 If LOFT is polynomial solvable, then so also is PARTITION.

Proof From the partition problem $S = \{a_1, a_2, \ldots, a_n\}$ construct the following open shop problem, OS, with 3n+1 jobs, $m = 3$ machines and all jobs with one non-zero task except for one with 3 tasks

$$t_{1,i} = a_i \ , \quad t_{2,i} = t_{3,i} = 0 \quad \text{for} \quad 1 \leq i \leq n$$

$$t_{2,i} = a_i \ , \quad t_{1,i} = t_{3,i} = 0 \quad \text{for} \quad n+1 \leq i \leq 2n$$

$$t_{3,i} = a_i \ , \quad t_{1,i} = t_{2,i} = 0 \quad \text{for} \quad 2n+1 \leq i \leq 3n$$

$$t_{1,3n+1} = t_{2,3n+1} = t_{3,3n+1} = T/2$$

where $T = \sum_1^n a_i$ and $\tau = 3T/2$

We now show that the above open shop problem has a schedule with finish time $\leq 3T/2$ iff S has a partition.

a) If S has a partition u then there is a schedule with finish time 3T/2.

One such schedule is shown in figure 5.4.1.