SIAM J. COMPUT.
Vol. 8, No. 2, May 1979

0097-5397/79/0802-0012 $01.00/0

NEARLY ON LINE SCHEDULING OF A UNIFORM PROCESSOR SYSTEM WITH RELEASE TIMES*

SARTAJ SAHNI† AND YOOKUN CHO†

Abstract. An $O(m^2n + mn \log n)$ nearly on line algorithm to preemptively schedule n independent tasks on m uniform processors is presented. It is assumed that there is a release time associated with each task. No task may be started before its release time. All tasks must be completed by a common due time (if possible). Our algorithm generates schedules having $O(nm)$ preemptions in the worst case. The algorithm can also be used to minimize maximum lateness even for the case when all jobs have the same release time but different due times.

Key words. independent tasks, uniform processors, preemptive schedule, release time, common due time, complexity

1. Introduction. A *uniform processor system* $P = \{P_1, P_2, \cdots, P_m\}$ is a set of m processors (machines). Associated with each processor, P_i, is a speed s_i, $s_i > 0$, $1 \leqq i \leqq m$. Processor P_i can perform s_i units of processing in one unit of time. When $s_i = s_{i+1}$, $1 \leqq i < m$, P is said to be a system of *identical processors*. Let T be a set of n independent tasks. Let t_i, r_i and d_i respectively be the processing requirement, release time and due time of task i, $1 \leqq i \leqq n$.

A *DD-schedule* for T is an assignment of tasks to processors such that (i) no processor is required to process more than one task at any time, (ii) no task is simultaneously processed on more than one processor, (iii) the processing of no task begins before its release time and (iv) all tasks are completed by their due times. Note that not all task sets have *DD*-schedules on a given processor system.

A nearly on line algorithm to find a *DD*-schedule (if one exists) is an algorithm which, for every distinct release time r_i, determines the schedule from 0 to r_i without knowledge of the jobs released on or after r_i.

Many researchers have studied the problem of obtaining *DD*-schedules (when they exist). Rinnooy Kan [6] shows that the problem of determining the existence of nonpreemptive *DD*-schedules is *NP*-Complete. McNaughton's algorithm [8] can be used to obtain preemptive *DD*-schedules for systems of identical processors when the task set T has only one distinct release time and one distinct due time. Gonzalez and Sahni [3] present an $O(n + m \log m)$ algorithm that works for uniform processor systems when T has only one distinct release time and one distinct due time. For the case when all tasks have the same release time (but may have different due times), Horn [4] presents an $O(n^3)$ algorithm to obtain preemptive *DD*-schedules for identical processors. A faster algorithm ($O(n \log mn)$) for this case may be found in [9]. Under the same assumptions on T, Sahni and Cho [10] obtain an $O(n \log n + mn)$ algorithm for uniform processors. Since, in all the cases cited so far all tasks are released at the same time, all the algorithms obtained are, of necessity, on line.

For the case when no restriction is placed on the task set T, Horn [4] presents an $O(n^3)$ algorithm for preemptive schedules on identical processors. Bruno and Gonzalez [1] present a similar algorithm for a system of two uniform processors. Neither of these two algorithms is on line. In fact, it is known [9] that no nearly on line algorithm exists when tasks are allowed to have arbitrary release and due times.

* Received by the editors October 10, 1977. This work was supported in part by the National Science Foundation under Grant MCS 76-21024.

† Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

Another special case that has been studied is when all tasks have the same due time. While, this case is symmetric to the case when all tasks have the same release time, the algorithms for the latter case do not result in on line algorithms for the former. Gonzalez and Johnson [2] have obtained an $O(nm)$ nearly on line algorithm for identical processors when all tasks have the same due time. Their algorithm generates DD-schedules (when they exist) having at most $O(nm)$ preemptions. In this paper we extend their result to the case of uniform processors. Our algorithm has time complexity $O(m^2n + mn \log n)$ and generates schedules with at most $O(nm)$ preemptions. In [10] we demonstrated the existence of task sets for which every DD-schedule (even those generated by off line algorithms) had at least $O(nm)$ preemptions. This algorithm can also be used to obtain schedules minimizing lateness when all jobs have the same release time but differing due times. To do this we just change the roles of due times and release times.

2. The algorithm. We first present a nearly on line algorithm that generates schedules with $O(mn + n^2)$ preemptions. Later, we shall show how to modify this algorithm so that the number of preemptions is $O(nm)$.

Assume that the n tasks to be scheduled have v distinct release times r_i, $1 \leqq i \leqq v$. Assume $r_i < r_{i+1}$, $1 \leqq i < v$ and $r_1 = 0$. Our algorithm works in v phases. In the ith phase tasks are scheduled from time r_i to time r_{i+1}, $1 \leqq i < v$. In the vth (and last) phase scheduling is done for the interval $[r_v, d]$ where d is the common due time of all tasks. The tasks available for scheduling in the ith phase are those released at or before time r_i and which haven't yet been completed, i.e., all released tasks with a nonzero remaining processing time (RPT). For each phase, i, algorithm EQUAL determines the amount each available task is to be processed. It does this by using an equalizing rule that attempts to equalize the RPTs of all tasks at the end of the phase. EQUAL utilizes the following fact which is due to Liu and Liu [7] and Horvath, Lam and Sethi [5]:

Fact 1. Let $a_1 \geqq a_2 \geqq \cdots \geqq a_l$ be a set of task times. Let w be the minimum finish time of any preemptive schedule for these l tasks on a system $P = \{P_1, P_2, \cdots, P_m\}$. Let $s_1 \geqq s_2 \geqq \cdots \geqq s_m$. Then,

$$w = \max\left\{\sum_1^l a_i \Big/ \sum_1^m s_i, \max_{1 \leqq j < m}\left\{\sum_1^j a_i \Big/ \sum_1^j s_i\right\}\right\}. \tag{1}$$

Using (1), EQUAL ensures that the amount of processing it is assigning for each task in phase i is such that all the phase i processing can be completed in $\Delta = r_{i+1} - r_i$ time (we may assume $r_{v+1} = d$). The basic strategy in EQUAL is to preferentially process the longest tasks so that the tasks remaining at the next release time are as small as possible. This is comparable to the level strategy used in [5]. The actual schedule for each phase can be constructed using the algorithm of Gonzalez and Sahni [3].

EQUAL has six parameters. Δ is the length of the interval for which scheduling is to be carried out in this phase (it is the time between two successive release times). S is an array such that $S(i) = \sum_{j=1}^{i} s_j$, $1 \leqq i \leqq m$ and $S(0) = 0$. It is assumed that $s_i \geqq s_{i+1}$, $1 \leqq i < m$. At the start of EQUAL, $t(i)$ is the RPT for task i, $1 \leqq i \leqq p$. p is the number of available jobs with nonzero RPT. For convenience, a fictitious job $p+1$ with $t(p+1) = 0$ is assumed. t' is an output array. At termination of EQUAL, $t'(i)$ is the amount job i is to be processed in the interval Δ. Also, $t(i)$ is modified to reflect the RPTs at the end of the interval. EQUAL determines $t'(i)$ such that $t'(i) \geqq t'(i+1)$, $1 \leqq i < p$ and $\max\{\sum_{i=1}^{p} t'(i)/S(m), \max_{1 \leqq j < m}\{\sum_{i=1}^{j} t'(i)/S(j)\}\} \leqq \Delta$. Hence, the assignments determined by EQUAL for the Δ interval can be scheduled. Furthermore, at termination we shall have $t(i) \geqq t(i+1)$, $1 \leqq i < p$.

EQUAL begins by initializing t' and $t(0)$ to zero. When $p<m$, only the fastest p processors need be used. Hence, in line 3, *mused* is initialized to be the actual number of processors to be used. *mlo* is the least index such that P_{mlo} is still available for processing. Processors 1 through $mlo-1$ become unavailable when $\sum_{i=1}^{mlo-1} t'(i)/S(mlo-1)=\Delta$. *nhi* is the index of the next job to be considered. Initially, $mlo=nhi=1$. $mhi=\min\{nhi-1, mused\}$. AMT is the amount of processing that has so far been assigned to processors P_{mlo} to P_{mhi}. At the start of the loop of lines 6–34, it will be the case that the RPTs of the jobs indexed $mlo, \cdots, nhi$ is the same. This RPT is *tnow*. Initially, $tnow=t(1)$ (line 5). EQUAL sequences through tasks with higher index than *nhi* determining the next smaller task i (line 8). Note that since $t(p+1)=0$ and $t(p)>0$, the loop of lines 7–9 will always terminate from line 8. *nhi* is updated in line 10 and *mhi* (line 11) is set so that processors *mlo* to *mhi* may be used for the processing of jobs *mlo* to $nhi-1$. We now attempt to equalize the RPTs of jobs *mlo* to $nhi-1$ with $t(nhi)$. Recall that the present RPTs of all these jobs is *tnow*. Hence, equalization calls for an additional total processing of $(tnow-t(nhi))*(nhi-mlo)$ units. However, only $(S(mhi)-S(mlo-1))*\Delta-$AMT units are still unassigned on P_{mlo} to P_{mhi}. INCR is the maximum amount by which each $t'(i)$, $mlo\leqq i<nhi$ is to be increased. This will result either in equalization with $t(nhi)$ or complete utilization of the processors.

ALGORITHM 2.1. The Equalizing Rule.

```
line   procedure EQUAL(Δ, S, t, m, p, t')
         //S(j) = Σ_1^j s_i and S(0) = 0 is assumed. Also t_i and s_i are in nonincreasing//
         // order and t(p+1) = 0//
 1       real t(0:p+1), t'(0:p), S(0:m)
 2       t' ← 0; t(0) ← 0 //t'(0) and t(0) needed in ADJUST; t' ← 0 initializes//
                          //all of t'//
 3       mused ← min{m, p}
 4       mlo ← nhi ← 1; AMT ← 0
 5       tnow ← t(1)
 6       while mlo ≦ mused and tnow ≠ 0 do
 7           for i ← nhi + 1 to p + 1 do //t(p+1) = 0//
 8               if t(i) < tnow then exit; endif
 9           repeat
10           nhi ← i
11           mhi ← min(nhi − 1, mused)
12           NEXTAMT ← min{(tnow − t(nhi)) * (nhi − mlo)
                                          + AMT, (S(mhi) − S(mlo − 1)) * Δ}
13           INCR ← (NEXTAMT − AMT)/(nhi − mlo)
14           TSUM ← 0
15           for i ← mlo to nhi − 1 do //test if INCR feasible//
16               t'(i) ← t'(i) + INCR
17               TSUM ← TSUM + t'(i); i' ← min {mhi, i}
18               if TSUM > (S(i') − S(mlo − 1)) * Δ then//use up P_mlo, ··· , P_i//
19                   DECR ← (TSUM − Δ * (S(i) − S(mlo − 1)))/(i − mlo + 1) //i = i'//
20                   for j ← mlo to i do
21                       t'(j) ← t'(j) − DECR
22                   repeat
23                   call ADJUST
```

```
24                  AMT ← AMT − Δ * (S(i) − S(mlo − 1)) + (INCR − DECR)
                                                        * (i − mlo + 1)
25                  mlo ← i + 1; TSUM ← 0
26                  NEXTAMT ← min {(tnow − t(nhi)) * (nhi − mlo)
                                    + AMT, (S(mhi) − S(mlo − 1)) * Δ}
27                  INCR ← (NEXTAMT − AMT)/(nhi − mlo)
28              endif
29          repeat
30          tnow ← t(nhi)
31          if t(nhi − 1) − t'(nhi − 1) ≠ tnow then AMT ← 0; mlo ← mhi + 1
32                                          else AMT ← NEXTAMT
33          endif
34      repeat
35      for i ← 1 to nhi − 1 do //update RPTs//
36          t(i) ← t(i) − t'(i)
37      repeat
38  end EQUAL
```

ALGORITHM 2.2. Subalgorithm for EQUAL.

```
line    procedure ADJUST
        //All variables used in EQUAL are available inside ADJUST//
1       if t(i) − t'(i) ≦ t(mlo − 1) − t'(mlo − 1) then return endif
2       low ← mlo; RPT ← t(i) − t'(i)
3       jobs ← i − low + 1 //Number of jobs with equal RPT//
4       while low > 1 and RPT > t(low − 1) − t'(low − 1) do
5           PRPT ← t(low − 1) − t'(low − 1)
6           pjobs ← 1
7           while t(low − pjobs − 1) − t'(low − pjobs − 1) = PRPT do
8               pjobs ← pjobs + 1
9           repeat
10          RPT ← (RPT * jobs + PRPT * pjobs)/(jobs + pjobs)
11          jobs ← jobs + pjobs
12          low ← low − pjobs
13      repeat
14      for q ← low to i do
15          t'(q) ← t(q) − RPT
16      repeat
17  end ADJUST
```

In the loop of lines 15–29 we check to see that increasing the $t'(i)$'s. $mlo \leqq i < nhi$ by INCR still leaves us with $t'(i)$'s that can be scheduled in Δ. This condition is easily tested for by the use of equation (1) and our assertion that $t'(i) \geqq t'(i+1)$ for all i, $mlo \leqq i < mhi$. If the conditional of line 18 is true then, the $t'(i)$'s cannot be increased by INCR. We compute DECR such that all $t'(j)$, $mlo \leqq j \leqq i$ can be increased by INCR − DECR and this is the maximum possible increase. This completely utilizes processors P_{mlo} to P_i. In line 23 the procedure ADJUST is invoked. This procedure ensures that the RPTs of the jobs already assigned to processors of index smaller than mlo are not less than those of the newly completed processors mlo to i. In case this is not true. ADJUST reduces the assigned processing of lower indexed jobs (thus increasing their RPTs) and

increases the $t'(j)$'s of the higher indexed jobs. Now that processors 1 through i have been completely assigned, AMT is updated to reflect the processing assigned only to processors P_{i+1} through P_{mhi}. mlo is updated to $i+1$ (the next available processor is P_{i+1}). INCR is recomputed in line 27 and an attempt is made to increase the $t'(j)$ of the jobs $mlo \leqq j < nhi$.

When the loop of lines 15–29 is exited, we will be in one of two conditions. Either, the RPTs of jobs mlo to $nhi-1$ have been equalized to $t(nhi)$ or they haven't. In the latter case, from the working of lines 12–29, it must be that all the processors with index less than $mhi+1$ have been fully assigned. Lines 31–33 do the necessary bookkeeping.

The subalgorithm ADJUST used by EQUAL is fairly straightforward. It begins with the knowledge that $t(j)-t'(j) \geqq t(j+1)-t'(j+1)$, $1 \leqq j < mlo$ and $t(j)-t'(j) = t(j+1)-t'(j+1)$, $mlo \leqq j < i$. If the condition of line 1 is true then, no adjustments need be made. Otherwise, in lines 4–13, the algorithm goes through blocks of jobs with an equal RPT. Each such block is identified by the loop of lines 7–9. The processing assignments in this block will be changed so that the RPT of this block together with all jobs seen up to index i will be the same. The new RPT is computed in line 10. One may easily verify that when the $t'(q)$'s are set as in line 15 they can still be processed in Δ units on P_{low} through P_i

From the description of ADJUST, it should be clear that when EQUAL terminates, $t(i) \geqq t(i+1)$, $1 \leqq i < p$.

Now we give an example to show how EQUAL works.

Example 2.1. Assume we have 5 jobs with RPTs 20, 19, 18, 17 and 16 respectively and 5 machines with speeds 20.1, 19.1, 17.7, 16.8 and 16.3 respectively. Further, assume $\Delta = 1$. Then $S(i) = (0, 20.1, 39.2, 56.9, 73.7, 90.0)$.

The job with RPT 20 has highest priority and will be assigned for processing. $tnow = 20$ at line 5. The **for** loop of lines 7–9 will be exited with $i = 2$. NEXTAMT = 1, INCR = 1 in lines 12 and 13. The condition of line 18 doesn't hold. $tnow$ is set to 19 and we start a new iteration of the **while** loop of lines 6–34. We have two jobs with RPT 19 now and these two jobs will be processed until their RPTs become 18. We shall follow the same procedure until we reduce the RPTs of all 5 jobs to 16. Then, we will have $mlo = 1$, $tnow = 16$ and AMT = 10. We start the 5th iteration of the **while** loop (lines 6–34). nhi becomes 6 at line 10 and $t(nhi) = 0$. NEXTAMT = 90 (line 12) and INCR = 16 (line 13). The **for** loop of lines 15–29 is entered and $t'(1) = 20$, $t'(2) = 19$ and $t'(3) = 18$. Now the condition of line 18 holds and DECR = 0.1/3. The amount of processing is reduced to $t'(1) = 20 - 0.1/3$, $t'(2) = 19 - 0.1/3$ and $t'(3) = 18 - 0.1/3$ (lines 20–22). We execute algorithm ADJUST (line 23) for the first time. The condition of line 1 holds and we return to EQUAL immediately. After executing lines 24–27, we have AMT = 1, $mlo = 4$, NEXTAMT = 33 and INCR = 16. Then the **for** loop (lines 15–29) is reiterated with $i = 4$. $t'(4)$ becomes 17 at line 16. Again the condition of line 18 holds. DECR becomes 0.2 and $t'(4)$ reduces to 16.8. ADJUST is called again and the **while** loop (lines 4–13 in ADJUST) is executed. RPT (line 10 in ADJUST) becomes 0.075 and the amount of processing for each job is adjusted to $t'(1) = 19.925$, $t'(2) = 18.925$, $t'(3) = 17.925$ and $t'(4) = 16.925$. Now we shall have AMT = 0 (line 24), $mlo = 5$, NEXTAMT = 16 and INCR = 16 (line 27). We get $t'(5) = 16$ at line 16. The condition of line 18 doesn't hold and we complete the **while** loop (lines 6–34). The final values are $t(i) = (0.075, 0.075, 0.075, 0.075, 0)$ and $t'(i) = (19.925, 18.925, 17.925, 16.925, 16)$. □

Next we prove some facts about EQUAL. These facts are needed to establish the validity of the final algorithm.

LEMMA 2.1. *Assume we have n jobs with processing time* t_i, $1 \leqq i \leqq n$, *and m machines of speed* s_i, $1 \leqq i \leqq m$. *Assume* $t_i \geqq t_{i+1}$, $1 \leqq i < n$, *and* $s_i \geqq s_{i+1}$, $1 \leqq i < m$. *Let* a_i

be the remaining processing time (RPT) of job i, $1 \leq i \leq n$, after using EQUAL *during a certain interval* Δ. *Note that* $a_i \geq a_{i+1}$, $1 \leq i < n$. *Let* b_i, $1 \leq i \leq n$ *be the RPTs after using any other valid assignment rule during the interval* Δ. *Let* $\sigma(\cdot)$ *be such that* $b_{\sigma(j)} \geq b_{\sigma(j+1)}$, $1 \leq j < n$. *Then* $\sum_1^j a_i \leq \sum_1^j b_{\sigma(i)}$, $1 \leq j \leq n$.

Proof. We first show $\sum_1^n a_i \leq \sum_1^n b_{\sigma(i)}$. This is trivially true if $\sum_{i=1}^n (t_i - a_i) = \Delta * \sum_{i=1}^m s_i$. So, assume $\sum_{i=1}^n (t_i - a_i) < \Delta * \sum_{i=1}^m s_i$. Let *mlo* be as defined at termination of EQUAL. From lines 12, 13, 19, 26 and 27, it follows that $\sum_{i=1}^{mlo-1} (t_i - a_i) = \Delta * \sum_{i=1}^{mlo-1} s_i$. Also, since $\sum_{i=1}^n (t_i - a_i) < \Delta * \sum_{i=1}^m s_i$, it follows that the RPT of jobs *mlo* to *p* is zero. I.e., $a_i = 0$, $mlo \leq i \leq n$. Since, by Fact 1 no more than $\Delta * \sum_{i=1}^{mlo-1} s_i$ of any set of $mlo - 1$ jobs can be processed in Δ, it follows that for every valid assignment $\sum_{i=1}^{mlo-1} (t_i - b_i) \leq \Delta * \sum_{i=1}^{mlo-1} s_i = \sum_{i=1}^{mlo-1} (t_i - a_i)$. Hence, $\sum_{i=1}^{mlo-1} b_i \geq \sum_{i=1}^{mlo-1} a_i = \sum_1^n a_i$. So $\sum_{i=1}^n b_{\sigma(i)} = \sum_{i=1}^n b_i \geq \sum_1^n a_i$.

Now, we shall show that $\sum_{i=1}^j a_i \leq \sum_{i=1}^j b_{\sigma(i)}$ for $1 \leq j < n$. Assume this is not true for some *j*. Let *j* be the least index such that $\sum_{i=1}^j a_i > \sum_{i=1}^j b_{\sigma(i)}$. Since $\sum_{i=1}^{j-1} a_i \leq \sum_{i=1}^{j-1} b_{\sigma(i)}$, it follows that $a_j > b_{\sigma(i)}$. Let *l* be the least integer such that $j < l \leq n$ and $a_j \neq a_l$. If no such *l* exists then, since $a_j = a_{j+1} = \cdots = a_n$ and $a_j > b_{\sigma(j)} \geq b_{\sigma(j+1)} \geq \cdots \geq b_{\sigma(n)}$, $\sum_{i=1}^n a_i > \sum_{i=1}^n b_{\sigma(i)}$. But we have just shown $\sum_{i=1}^n a_i \leq \sum_{i=1}^n b_{\sigma(i)}$. Hence, such an *l* must exist. We observe that $a_j = a_{j+1} = \cdots = a_{l-1} > a_l$ and $\sum_{i=1}^{l-1} a_i > \sum_{i=1}^{l-1} b_{\sigma(i)}$. Thus $\sum_{i=1}^{l-1} (t_i - a_i) < \sum_{i=1}^{l-1} (t_i - b_{\sigma(i)})$. One easily observes that $\sum_{i=1}^{l-1} b_i \leq \sum_{i=1}^{l-1} b_{\sigma(i)}$. So, $\sum_{i=1}^{l-1} (t_i - b_{\sigma(i)}) \leq \sum_{i=1}^{l-1} (t_i - b_i)$. If $l - 1 < m$ then, since EQUAL has failed to equalize jobs $l-1$ and *l*, it follows that $\sum_{i=1}^{l-1} (t_i - a_i) = \Delta * \sum_{i=1}^{l-1} s_i$. Furthermore, from Fact 1 it follows that in all valid assignments for Δ, the sum of the $l-1$ largest assignments is no greater than $\Delta * \sum_{i=1}^{l-1} s_i$. Hence, $\sum_{i=1}^{l-1} (t_i - b_i) \leq \Delta * \sum_{i=1}^{l-1} s_i$. This contradicts our earlier claim that $\sum_{i=1}^{l-1} (t_i - a_i) < \sum_{i=1}^{l-1} (t_i - b_i)$. Hence, *l* must be greater than *m*. But in this case job $l-1$ can always be equalized to job *l* unless this equalization requires more processing than available. Since $a_{l-1} > a_l$, it must be that $\sum_{i=1}^{l-1} (t_i - a_i) = \Delta * \sum_{i=1}^m s_i$. Since $\sum_{i=1}^{l-1} (t_i - b_i) \leq \Delta * \sum_{i=1}^m s_i$ we again obtain a contradiction. Thus, there can be no *l* for which $\sum_{i=1}^l a_i > \sum_{i=1}^l b_{\sigma(i)}$. □

LEMMA 2.2. *Let C be a set of n jobs with processing times* c_i, $1 \leq i \leq n$. *Let D be another set of of n jobs with processing times* d_i, $1 \leq i \leq n$. *Assume* c_i *and* d_i *are in nonincreasing order and* $\sum_1^j c_i \leq \sum_1^j d_i$, $1 \leq j \leq n$. *Let* c_i' *be the RPT of job i,* $1 \leq i \leq n$, *when set C is scheduled for a period* Δ *using* EQUAL. *Let* d_i' *be the RPT of job i,* $1 \leq i \leq n$, *when set D is scheduled for a period* Δ *using* EQUAL. (*Note that* $c_i' \geq c_{i+1}'$ *and* $d_i' \geq d_{i+1}'$, $1 \leq i < n$.) *Then,*

$$\sum_1^j c_i' \leq \sum_1^j d_i' \quad \text{for } 1 \leq j \leq n.$$

Proof. Assume the lemma is not true. Let *j* be the least index for which $\sum_1^j c_i' > \sum_1^j d_i'$. Then, $c_j' > d_j'$. Let *k* be the least index such that $j < k \leq n$ and $c_k' \neq c_j'$. There are two cases.

Case 1. There is no such *k*. In this case $c_j' = c_l'$ for $j < l \leq n$ and $\sum_1^n c_i' > \sum_1^n d_i'$. $c_n' > 0$ since $c_n' = c_j' > d_j' \geq 0$. Also, $\sum_1^n (c_i - c_i') < \sum_1^n (d_i - d_i')$. This means that EQUAL has assigned more total processing of the jobs in *D* than it has for the jobs in *C*. Let $x = \min\{n, m\}$. Since $c_n' > 0$, EQUAL must assign $\Delta * \sum_{i=1}^x s_i$ amount of processing for job set *C*. Also, no more than this amount can be assigned for *D*. Hence $\sum_{i=1}^n (c_i - c_i') \geq \sum_{i=1}^n (d_i - d_i')$.

Case 2. *k* as above exists. In this case $c_j' = c_{j+1}' = \cdots = c_{k-1}' > c_k'$ and $c_j' > d_j'$. Thus $\sum_{i=1}^l c_i' > \sum_1^l d_i'$ and $\sum_{i=1}^l (c_i - c_i') < \sum_{i=1}^l (d_i - d_i')$, $j \leq l < k$. $c_{k-1}' > c_k'$ can happen only if $\sum_{i=1}^{k-1} (c_i - c_i') = \Delta * \sum_{i=1}^{k-1} s_i$. But, in this case, $\sum_{i=1}^{k-1} (c_i - c_i') \geq \sum_{i=1}^{k-1} (d_i - d_i')$. □

LEMMA 2.3. *Let A and B be two sets of jobs. Let r_i, $1 \leqq i \leqq v$, be the distinct release times of the jobs in A and B. Assume that $r_i < r_{i+1}$ and that n_i jobs have release time r_i in both A and B, $1 \leqq i \leqq v$. Further, assume that the set of jobs with release time r_i in A is identical to that with release time r_i in B, $2 \leqq i \leqq v$. Let $C \cup D$ and $C \cup E$ be the job sets with release time r_1 in A and B respectively. Let $|D| = |E| = l$ and let d_i, e_i, $1 \leqq i \leqq l$, be the processing times for the jobs in D and E respectively. Assume $d_i \geqq d_{i+1}$ and $e_i \geqq e_{i+1}$, $1 \leqq i < l$. Also assume that $\sum_1^j d_i \geqq \sum_1^j e_i$, $1 \leqq j \leqq l$. If A has a DD-schedule then B also has one.*

Proof. Assume A has a *DD*-schedule S. The proof is by induction on the number of release times v. Let α_i and β_i, $1 \leqq i \leqq n$, be the processing times of the jobs in $C \cup D$ and $C \cup E$ respectively. Assume that $\alpha_i \geqq \alpha_{i+1}$ and $\beta_i \geqq \beta_{i+1}$, $1 \leqq i < n_1$.

Induction base. $v = 1$. Since $\sum_1^j d_i \geqq \sum_1^j e_i$, $1 \leqq j < l$, it follows that $\sum_1^j \alpha_i \geqq \sum_1^j \beta_i$, $1 \leqq j \leqq n_1$. Hence, from Fact 1 we conclude that the job set $C \cup E$ can be completed using no more time than $C \cup D$. So, if A has a *DD*-schedule then B must have one too.

Induction hypothesis. Assume the lemma is true for all job sets with v distinct release times for $1 \leqq v < u$.

Induction step. We shall show the lemma is true when $v = u$. A and B have n_1 jobs each at time r_1. Schedule both job sets $C \cup D$ and $C \cup E$ using EQUAL during the interval $[r_1, r_2]$. Let α_i' and β_i', $1 \leqq i \leqq n_1$ be the RPTs of these jobs at time r_2. Since $\sum_1^j \alpha_i \geqq \sum_1^j \beta_i$, $1 \leqq j \leqq n_1$, it follows from Lemma 2.2 that $\sum_1^j \alpha_i' \geqq \sum_1^j \beta_i'$, $1 \leqq j \leqq n_1$. Let A' and B' be the set of jobs remaining to be processed in A and B at time r_2. Let A'' be the corresponding set at time r_2 in the *DD*-schedule S. Further, let α_i'', $1 \leqq i \leqq n_1$, be the RPTs in S of the jobs in the set $C \cup D$ at r_2. Then, $\sum_1^j \alpha_i' \leqq \sum_1^j \alpha_i''$, $1 \leqq j \leqq n_1$, by Lemma 2.1. Since A'' has a *DD*-schedule, it follows that A' has one too (induction hypothesis). Now, since A' has a *DD*-schedule and both A' and B' have $u - 1$ release times and satisfy the conditions of the lemma, it follows that B' has a *DD*-schedule too. Hence, B has a *DD*-schedule. □

Our first nearly on line algorithm ONEDT utilizes EQUAL to schedule each phase $[r_i, r_{i+1}]$, $1 \leqq i \leqq v (r_{v+1} = d)$. It also uses two other subalgorithms ORDER and UNIFORM. ORDER sorts the jobs to be processed in each phase into nonincreasing order of their processing times. UNIFORM performs the actual scheduling of each job for the amount of processing time $t'(i)$ computed by EQUAL. Note that the conditional of lines 13 and 19 of EQUAL guarantees that this can be done. The algorithm UNIFORM is formally given in Gonzalez and Sahni [3].

THEOREM 2.1. ONEDT *generates a DD-schedule for every job set J for which such a schedule exists.*

Proof. The proof is by induction on the number of distinct release times in J and is very similar to that of Lemma 2.3.

Analysis of EQUAL. The loop of lines 7–9 contributes at most $O(p)$ to the algorithms complexity. The total number of iterations of the loop of lines 15–29 is at most p. The conditional of line 18 can be true at most m times. Each time this happens, $O(m)$ time may be spent in lines 19–27. Hence, the total contribution of lines 15–29 is $O(p + m^2)$. The complexity of EQUAL is also $O(p + m^2)$.

Analysis of ONEDT.

Time complexity. Let n_i, $1 \leqq i \leqq v$, be the number of jobs released at the distinct release times r_i, $1 \leqq i \leqq v$. Line 5 takes $O(n_i \log n_i + \sum_{j=1}^{i-1} n_j)$ time since we only sort the jobs released at r_i and merge these n_i jobs with the uncompleted jobs released from r_1 through r_{i-1}. Note that these jobs are already in nonincreasing order of their RPTs. Line 6 takes $O(N_i + m^2)$ where $N_i = \sum_1^i n_j$. Line 7 takes $O(N_i)$. Therefore the time complexity of ONEDT is $O(n \log n + nv + vm^2) = O(n^2 + nm^2)$.

Number of preemptions. We have v distinct release times. UNIFORM generates at most $2(m-1)$ preemptions [3] during the interval $[r_i, r_{i+1}]$. Additional preemptions are introduced since some of the jobs to be processed in $[r_i, r_{i+1}]$ may have been processed for some time in a previous interval. There are at most N_{i-1} such additional preemptions. Therefore, the total number of preemptions is at most $\sum_1^v (2(m-1)+N_{i-1}) = O((m+n)v) = O(mn+n^2)$. □

ALGORITHM 2.3. First algorithm for *DD*-schedules.

```
line    procedure ONEDT(d)
            //d is the common due time for all the jobs//
 1          R ← set of jobs released at time 0
 2          t ← 0
 3          loop
 4           r ← min {next release time, d}
 5           call ORDER(R)
 6           call EQUAL (r − t, S, R, m, p, T)
                    //S : cumulative speed of processors//
                    //m: number of processors//
                    //p : number of jobs//
                    //T : computed processing time//
 7           call UNIFORM (m, p, T)
 8           if r = d then exit endif
 9           Q ← set of jobs released at r
10           R ← Q ∪ R
11           t ← r
12          repeat
13          if all the jobs are completed then print ("DD-schedule exists")
14          else print ("No DD-schedule for the job set")
15          endif
16      end ONEDT
```

The total number of preemptions may be reduced by using the equalization strategy only until $2m - mlo + 1$ jobs have an equal RPT. The new algorithm, MEQUAL, to determine the scheduling assignments for each phase is described informally. MEQUAL behaves as EQUAL until *nhi* becomes greater than $2m - mlo + 1$. At this time jobs $mlo, mlo+1, \cdots, nhi-1$ have the same RPT. In case all the processing time has been assigned (line 2) then MEQUAL terminates. In this case MEQUAL is identical to EQUAL. In line 3 we increase $t'(i)$, $mlo \leqq i \leqq m$ using the equalizing rule and keeping in mind that $t'(i)$, $m < i < nhi$ has also been assigned to this interval. This step behaves as if $t(m+1) = 0$. If at the end of this step, $t(m) \neq t'(m)$ then it must be that $\sum_{i=1}^{nhi-1} t'(i) = \Delta * \sum_{i=1}^{m} s_i$ and $(t(i) - t'(i)) \geqq (t(i+1) - t'(i+1))$, $1 \leqq i \leqq m$. If $t'(m) = t(m)$ then the RPT of job m is zero. In lines 6–12 we assign as much of jobs *nhi* to p as possible. Note that since $t'(m) = t(m) \geqq t(nhi)$, the assignments made in lines 6–12 cannot violate Fact 1 with $w = \Delta$. If all of jobs *nhi* to p can be assigned then, the remaining time of the processors is allocated equally to jobs $m+1$ to $nhi - 1$. Thus, whenever MEQUAL behaves differently from EQUAL the RPTs of jobs $m+1$ to $nhi - 1$ is the same at termination. There are at least $m - mlo + 1$ such jobs.

ALGORITHM 2.4. Final equalizing rule for *DD*-schedules.

```
line    procedure MEQUAL
 1          use the equalizing rule, EQUAL, until it either terminates or
            nhi > 2m − mlo + 1 (line 10). If EQUAL terminates then return.
```

```
2       if ∑_{i=1}^{nhi-1} t'(i) = Δ * S(m) then t(i) ← t(i) − t'(i), 1 ≦ i < nhi;
        return; endif
        nhi ← 2m − mlo + 2
3       continue to increase the assignments t'(i), mlo ≦ i < m using
        the equalizing rule with the assumption t(m + 1) = 0. Now, the
        equalizing rule will terminate either with t'(m) = t(m) or
        t'(m) ≠ t(m). If t'(m) ≠ t(m) then ∑_{i=1}^{nhi-1} t'(i) = Δ * S(m)
4       if t'(m) ≠ t(m) then t(i) ← t(i) − t'(i), 1 ≦ i < nhi
                             return
                        endif
5       AMTLFT ← Δ * S(m) − ∑_{i=1}^{nhi-1} (t(i) − t'(i))
6       for i ← nhi to p do
7           if AMTLFT > t(i) then t'(i) ← t(i); AMTLFT ← AMTLFT − t(i)
8                            else t'(i) ← AMTLFT
                                  t(j) ← t(j) − t'(j), 1 ≦ j ≦ i
10                                return
11                           endif
12      repeat
13      for i ← m + 1 to nhi − 1 do
14          t'(i) ← t'(i) + AMTLFT/(nhi − m − 1)
15      repeat
16      t(i) ← t(i) − t'(i), 1 ≦ i ≦ p
17      return
18  end MEQUAL
```

Example 2.2. Assume we have 3 machines with speeds $S = \{3, 2, 1\}$, 7 jobs with processing times $t = \{10, 9, 8, 7, 6, 5, 4\}$ and $\Delta = 7$.

If we use EQUAL, the RPTs of these jobs after scheduling are $\{1, 1, 1, 1, 1, 1, 1\}$. When we use MEQUAL, we will first equalize the 6 largest jobs to have an RPT of 5. Next, MEQUAL will continue to use EQUAL but with only the first 3 jobs. The RPTs of these 3 jobs will be 0 after this step. We have used up 30 units of processing time out of 42 units we are given for the interval. The last job with RPT 4 is now assigned (lines 6–10) and is completed. We have 8 units of processing time left. We execute the jobs with initial RPTs 7, 6 and 5 this time (lines 3–5) for 8/3 more units. The final RPTs of the jobs by MEQUAL are $\{0, 0, 0, 7/3, 7/3, 7/3, 0\}$. Note that the minimum finish time for both sets is the same and is 7/6. □

The following lemma shows that the processing assignments made by MEQUAL can be completed within the given interval.

LEMMA 2.4. *$t'(i)$, $1 \leqq i \leqq p$, generated by* MEQUAL *are such that they can be completed in Δ units of processing time.*

Proof. We have 2 different cases according to when return of control is made by MEQUAL.

Case 1. The return of the control is made at one of steps 1, 2 or 4. We have assigned only using EQUAL thus far and EQUAL generates only valid sets of processing assignments.

Case 2. Return of control is made at either line 10 or line 17. In this case $t'(j) < t'(i)$, $j > m$ and $i \leqq m$. Since, $\max_{j \leqq m} \{\sum_{i=1}^{j} t'(i)/\sum_{i=1}^{j} s_i\} \leqq \Delta$ is guaranteed by EQUAL and from lines 6–15 it follows that $\sum_{i=1}^{p} t'(i) \leqq \Delta * \sum_{1}^{m} s_i$, Fact 1 guarantees that the $t'(i)$'s are a valid assignment set. □

Let t_i, $1 \leqq i \leqq p$, $t_i \geqq t_{i+1}$, $1 \leqq i < p$, be any set of p processing times. If the corresponding task set is used by EQUAL then let a_i, $1 \leqq i \leqq p$, be the RPTs. Let b_i,

$1 \leq i \leq p$, be the RPTs when MEQUAL is used. We have seen, earlier, that $a_i \geq a_{i+1}$, $1 \leq i < p$. Assume that MEQUAL does not terminate in lines 1 or 2. It is easy to see that $a_i = b_i$, $1 \leq i < mlo$. Also, there may exist a k, $mlo \leq k \leq m$, such that $b_k > b_{m+1}$. If such a k exists then we can show that $b_{mlo} \geq b_{mlo+1} \geq \cdots \geq b_k$ and $a_i \geq b_i$, $mlo \leq i \leq k$. This follows from the observation that MEQUAL tries to use all remaining space to increase the assignments of only jobs mlo to m whereas EQUAL uses this same space to increase the assignments of many more tasks. Let $\sigma(\cdot)$ be such that $b_{\sigma(i)} \geq b_{\sigma(i+1)}$, $1 \leq i < p$. From the preceding discussion and the knowledge that $b_{m+1} = b_j$, $m+1 \leq j \leq 2m - mlo + 1$, we can conclude that $a_i \geq b_{\sigma(i)}$, $1 \leq i \leq j$, and $b_{\sigma(j+1)} = b_{\sigma(j+2)} = \cdots = b_{\sigma(m)}$.

LEMMA 2.5. *Let A, B, C, D, E, d_i, e_i and l be as defined in Lemma 2.3. Assume that $d_i \geq e_i$, $1 \leq i \leq j$, and $e_{j+1} = e_i$, $j+1 < i \leq m$. Further, assume that $\sum_1^l d_i = \sum_1^l e_i$ and $l > m$. If A has a DD-schedule then B also has one.*

Proof. First assume that the number of distinct release times in A and B is 1. Sort $A = C \cup D$ and $B = C \cup E$ into nonincreasing order of processing times. Let these times respectively be α_i and β_i, $1 \leq i \leq n_1$. $\alpha_i \geq \alpha_{i+1}$ and $\beta_i \geq \beta_{i+1}$, $1 \leq i < n_1$. Let r be the largest index such that $\alpha_i \geq \beta_i$, $1 \leq i \leq r$. If $r < m$ then from the assumptions on D and E, it follows that $\beta_{r+1} = \beta_j$, $r+1 < j \leq m$. When $r \geq m$, the lemma is proved by using Fact 1 and the knowledge, $\alpha_i \geq \beta_i$, $1 \leq i \leq r$, and $\sum_1^{n_1} \alpha_i = \sum_1^{n_1} \beta_i$. When $r < m$, we use the additional information $\sum_1^j \beta_i / \sum_1^j s_i \leq \sum_1^{j+1} \beta_i / \sum_1^{j+1} s_i$, $r < j < m$, and $\sum_1^m \beta_i / \sum_1^m s_i < \sum_1^l \beta_i / \sum_1^l s_i = \sum_1^l \alpha_i / \sum_1^l s_i$.

Now assume the lemma is true for all job sets A and B with $u < v$ release times. We shall show the lemma is true for all job sets with v release times. Let α_i and β_i be as above. Use EQUAL to compute the assignments for $C \cup D$ and $C \cup E$ in the interval $\Delta = r_2 - r_1$. Let α_i' and β_i' be the respective RPTs. From the working of EQUAL, it follows that $\alpha_i' \geq \beta_i'$, $1 \leq i \leq r$ and if $r < m$ then $\beta_{r+1}' = \beta_j'$, $r+1 < j \leq m$. Let A' and B' be the job sets remaining at r_2 following the use of EQUAL in $[r_1, r_2]$. It follows that A' and B' satisfy the conditions of the lemma and have only $v-1$ release times. Also, A' has a *DD*-schedule. So, B' and hence B have *DD*-schedules. □

Let MONEDT be the algorithm resulting when line 6 of ONEDT is replaced by a call to MEQUAL.

THEOREM 2.2. MONEDT *generates a DD-schedule for every job set J for which such a schedule exists.*

Proof. The proof is similar to that of Lemma 2.5 and uses the results of the discussion preceding this lemma. □

Analysis of MONEDT. The complexity of MONEDT, is the same as that of ONEDT, i.e., $O(n^2 + nm^2)$. Its complexity can be easily reduced to $O(m^2 n + mn \log n)$ by using a heap. The changes needed to ONEDT to get the improved MONEDT are:

(i) delete line 5;
(ii) change EQUAL to MEQUAL in line 6;
(iii) maintain a max-heap of jobs with nonzero RPT;
(iv) in line 10 insert the Q into this heap;

The heap insertion of change (iv) requires $O(|Q| \log n)$ time per iteration. Hence its overall contribution to the computing time is $O(\sum (|Q| \log n)) = O(n \log n)$. We now need to modify EQUAL so that when it is called from line 1 of MEQUAL, the job times are obtained from a heap. This requires the insertion of an instruction between lines 7 and 8 to delete an element from the max-heap and to set $t(i)$. A check for $i > m - mlo + 1$ is also made. When this happens a jump to line 10 followed by a return to MEQUAL is made. Since only $O(m)$ items are deleted from the heap, the total time spent on this call to EQUAL is $O(m^2 + m \log n)$. When EQUAL is used from line 3 of

MEQUAL it works as before (i.e. using the times $t(i)$, $mlo \leqq i \leqq m$ rather than extracting times from the heap.) Hence, the time for line 3 of MEQUAL is $O(m^2)$. Lines 4 and 5 take $O(m)$ time. If the loop of lines 6–12 is iterated k_i times on the ith call to MEQUAL then the time needed is $O(k_i \log n)$ to extract the next k_i times from the heap plus $O(nhi \log n) = O(m \log n)$ time to insert the nonzero RPTs back into the heap (line 9) in case a return is made from this loop. If a return is made from line 17 instead then the total time spent in lines 13–15 is $O(m)$. Line 16 requires reinsertion of the nonzero RPTs into the heap. There can be at most $nhi - 1$ such RPTs as lines 13–17 are executed only when all jobs indexed nhi to p are fully allocated in lines 6–12. So the time needed in lines 13–17 is $O(m \log n)$. Hence, the ith call to MEQUAL takes times $O(m^2 + (m + k_i) \log n)$. If there are v release times then the total time spent in MEQUAL is $O(m_2 v + (mv + \sum k_i) \log n)$. Note that when the loop of lines 6–12 of MEQUAL are iterated, at least $k_i - 1$ jobs have a zero RPT and so are not considered in future iterations. Hence, $\sum k_i = O(n)$. Also $v \leqq n$. Therefore the time spent in MEQUAL is $O(m^2 n + mn \log n)$. The total time spent in UNIFORM is less than this. So, the overall complexity of MONEDT is $O(m^2 n + mn \log n)$.

If v is the number of release times then UNIFORM introduces at most $O(mv)$ preemptions. At most $2m + 1$ of the jobs scheduled in any interval may remain uncompleted by the end of the interval. This results in at most $2m + 1$ additional preemptions per phase (except the last phase when all jobs must be completed). The total number of preemptions is therefore $O(mv) = O(mn)$. This bound of $O(mn)$ agrees with the lower bound established in [10] on the worst case number of preemptions. □

Acknowledgment. We are grateful to an anonymous referee for suggesting the use of a heap to reduce the complexity of MONEDT from $O(n^2 + nm^2)$ to $O(m^2 n + mn \log n)$. The corresponding analysis was also provided by the referee.

REFERENCES

[1] J. BRUNO AND T. GONZALEZ, *Scheduling independent tasks with release dates and due dates on parallel machines*, Technical Report No. 213, Pennsylvania State University, State College, Dec. 1976.

[2] T. GONZALEZ AND D. JOHNSON, *A new algorithm for preemptive scheduling of trees*, Technical Report No. 222, Pennsylvania State University, State College, June 1977.

[3] T. GONZALEZ AND S. SAHNI, *Preemptive scheduling of uniform processor systems*, J. Assoc. Comput. Mach., 25 (1978), pp. 92–101.

[4] W. HORN, *Some simple scheduling algorithms*, Naval Res. Logist. Quart., 21 (1974), pp. 177–185.

[5] E. HORVATH, S. LAM AND R. SETHI, *A level algorithm for preemptive scheduling*, J. Assoc. Comput. Mach., 24 (1) (1977), pp. 32–43.

[6] A. RINNOOY KAN, *Machine scheduling problems*, Ph.D. thesis, Mathematische Centrum, Amsterdam, 1976.

[7] J. LIU AND C. LIU, *Bounds on scheduling algorithms for heterogeneous computing systems*, IFIP Proceedings, August 1974, pp. 349–353.

[8] R. MCNAUGHTON, *Scheduling with deadlines and loss functions*, Management Sci., 12 (1959), pp. 1–12.

[9] S. SAHNI, *Preemptive scheduling with due dates*, Technical Report #77-4, University of Minnesota, Minneapolis, April 1977; Operations Res., to appear.

[10] S. SAHNI AND Y. CHO, *Scheduling independent tasks with due times on a uniform processor system*, Technical Report #77-7, University of Minnesota, Minneapolis, May 1977.