Preemptive Scheduling Of A Multiprocessor System With Memories To Minimize Maximum Lateness*

Ten-Hwang Lai and Sartaj Sahni
University of Minnesota

Abstract

We develop an $O(q^2n + nlogn)$ algorithm to obtain a preemptive schedule that minimizes maximum lateness when n jobs with given due dates and memory requirements are to be scheduled on m processors ($n \geq m$) of given memory sizes. q is the number of distinct due dates. The value of the minimum maximum lateness can itself be found in O(qn + nlogn) time.

Key Words and Phrases

Preemptive scheduling, maximum lateness, memory requirements.

*This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650 and in part by the National Science Foundation under grant MCS80-005856.
Dr Ten-Hwang Lai's present address is: Computer Science Department, The Ohio State University, Columbus, Ohio.

1. Introduction

The problem of scheduling n jobs on a multiprocessor system consisting of m processors, each having its own independent memory of size μ_i has been considered by Kafura and Shen [2]. Associated with each job is a processing time t_j and a memory requirement m_j. Job j can be processed on processor i iff $m_j \le \mu_i$. No job can be simultaneously processed on two different processors and no processor can process more than one job at any given time instance. In a preemptive schedule, it is possible to interrupt the processing of a job and resume it later on a possibly different processor. In a nonpreemptive schedule, each job is processed without interruption on a single processor.

Obtaining minimum finish time nonpreemptive schedules is NP-hard even when m = 2 and $\mu_1 = \mu_2$ [1]. Hence, Kafura and Shen [2] study the effectiveness of several heuristics for nonpreemptive scheduling. For the preemptive case, they develop an O(nlogn) algorithm that obtains minimum finish time schedules (without loss of generality, we may assume n ≥ m). Their algorithm begins by first computing the finish time, f*, of a minimum finish time schedule. This is done as follows. First, the jobs and processors are reordered such that $\mu_1 \ge \mu_2 \ge \cdots \ge \mu_m$ and $m_1 \ge m_2 \ge \cdots \ge m_n$. This reordering takes O(nlogn) time (again, we assume n ≥ m). Let F_i be the set of all jobs that can be processed only on processors l, 2, ..., i because of their memory requirements. Let X_i be the sum of the processing requirements of the jobs in F_i. $X_i = 0$ *iff* $F_i = \phi$. Kafura and Shen [2] show that

$$f^* = \max\{ \max_j\{t_j\}, \max_i\{X_i/i\}\}. \qquad (1.1)$$

The jobs may now be scheduled in the above order ($m_1 \ge m_2 \ge \cdots \ge m_n$) using f* and McNaughton's rule [4].

In this paper, we extend the work of [2] to the case when each job has a due time d_j associated with it. Every job is released at a common release time. We are interested in first determining whether or not the n jobs can be preemptively scheduled in such a way that every job completes by its due time. A schedule that has this property is called a *feasible schedule*.

The existence of a feasible schedule can be determined in polynomial time using network flow techniques [3]. The complexity of the algorithm that results from this approach is $O(qn(n+qs)\log^2(n+qs))$ where q is the number of distinct due dates and s the number of different memory sizes. In fact, a feasible schedule (whenever one exists) may be obtained in this much time. In Section 2, we develop another algorithm for this problem. This algorithm is considerably harder to prove correct but has a complexity that is only O(qn + nlogn). A feasible schedule can be constructed in $O(q^2n + n\log n)$ time. In arriving at the algorithm of Section 2, we develop a necessary and sufficient condition for the existence of a feasible schedule. With the help of this condition, in Section 3, we develop an algorithm to obtain a schedule that minimizes the

maximum lateness. This algorithm is also of complexity $O(q^2n + n\log n)$.

Sahni [5] and Sahni and Cho [6 and 7] have done work related to that reported here. They have considered preemptive scheduling of n jobs with due dates when $\mu_1 = \mu_2 = \cdots = \mu_m$. For the special case when all memory sizes are the same, Sahni [5] has developed an O(nlogmn) algorithm to obtain a feasible schedule (when one exists). Sahni and Cho [6 and 7] have obtained efficient algorithms for the case when $\mu_1 = \mu_2 = \cdots = \mu_m$ and the processors run at different speeds.

2. A Fast Feasibility Algorithm

In this section, we first derive a necessary and sufficient condition for the existence of a feasible schedule. This condition is used to obtain a fast algorithm to construct a feasible schedule whenever such a schedule exists. In section 3, this necessary and sufficient condition is used to obtain a fast algorithm to minimze the maximum lateness.

Each job is characterized by a triple (t_j, d_j, m_j) where t_j is the task time of job j; d_j is its due time; and m_j its memory requirement. Let $\delta_1, \delta_2, ..., \delta_q$, $\delta_1 < \delta_2 < \cdots < \delta_q$, denote the distinct due times in the multiset $\{d_1, d_2, ..., d_n\}$. Let δ_0 be the common release time for the n jobs. Without loss of generality, we may assume that $\delta_0 < \delta_1$.

Let $\mu(1), \mu(2), ..., \mu(s)$, $\mu(1) > \mu(2) > ... > \mu(s)$ be the distinct memory sizes in the multiset $\{\mu_1, \mu_2, ..., \mu_m\}$. Let $\mu(s+1) = 0$ for convenience. Let P_k denode the set of all processors with memory size equal to $\mu(k)$; i.e., $P_k = \{ i \mid \mu_i = \mu(k)\}$, $1 \le k \le s$. Let J_k be the set of all jobs that can be processed only on processors in $P_1, P_2, ..., P_k$ because of their memory requirements; i.e., $J_k = \{ j \mid \mu(k) \ge m_j > \mu(k+1)\}$, $1 \le k \le s$. We shall refer to P_k as processor class k and J_k as job class k.

2.1 A Necessary and Sufficient Condition

It is easy to see that in every feasible schedule for the n jobs, at least

$$b(j,d) = \begin{cases} t_j & d_j \le \delta_d \\ t_j - \min\{t_j, d_j - \delta_d\} & \textit{otherwise} \end{cases}$$

amount of job j must be completed by δ_d, $1 \le j \le n$, $0 \le d \le q$. Observe that if there exist j and d such that $b(j,d) > \delta_d - \delta_0$, then there is no feasible schedule.

Of the minimum amount b(j,d) that must be completed before δ_d , at most

$$a(j,d) = \min\{b(j,d), \delta_1 - \delta_0\}$$

can be completed by δ_1.

Define B(k,d) to be the sum of the b(j,d)s for those jobs j in J_k. Define A(k,d) in a similar manner. Specifically,

$$B(k,d) = \Sigma_{j \varepsilon J_k} b(j,d),\ 1 \le k \le s,\ 0 \le d \le q,$$

and

$$A(k,d) = \Sigma_{j \varepsilon J_k} a(j,d),\ 1 \le k \le s,\ 0 \le d \le q.$$

B(k,d) gives a lower bound on the amount of jobs in J_k that must be completed by δ_d. A(k,d) gives the maximum amount of B(k,d) that can be done by δ_1.

Define a capacity function C(k,d) such that

$$C(k,d) = |P_k|(\delta_d - \delta_0),\ 1 \le k \le s,\ 0 \le d \le q.$$

C(k,d) gives the available processing capacity in processor class k from the release time δ_0 to the due time δ_d.

One readily observes that if a feasible schedule exists, then:

$$\sum_{i=1}^{k} B(i,d) \le \sum_{i=1}^{k} C(i,d) \tag{2.0}$$

for all k,d, $1 \le k \le s$, $0 \le d \le q$. While Eq. 2.0 provides a necessary condition for the existence of a feasible schedule, it does not provide a sufficient condition. We leave it to the reader to construct an instance that satisfies (2.0) but for which no feasible schedule exists.

This necessary condition can be strengthened by using the notion of a profile function. Let π be the set of all nonincreasing functions σ with domain {0, 1, 2, ..., s} and range {0, 1, ..., q}. Recall that s is the number of processor classes anq q the number of distinct due times. Thus

$$\pi = \{\ \sigma \mid \sigma: \{0,1,...,s\} \rightarrow \{0,1,...,q\} \text{ and } \sigma(k) \ge \sigma(k+1),\ 0 \le k < s\}.$$

π defines the set of *profile functions*. Each profile function σ defines a *profile* in a timing diagram (see [8]), i.e., the curve of $t = \delta_{\sigma(i)}$, i = 1,2,...,s. We shall refer to the profile defined by σ simply as the profile σ. For example, consider the case s = 4, q = 5, and the profile function σ such that $\sigma(0) = \sigma(1) = \sigma(2) = 4$; $\sigma(3) = 2$; and $\sigma(4) = 1$. Figure 2.1 displays σ pictorially.

Let σ be a profile function. In any feasible schedule, at least $B(i,\sigma(i))$ amount of processing from J_i must be done on $P_1, P_2, \cdots, P_i$ by time $\delta_{\sigma(i)}$. Since σ is nonincreasing, $B(i,\sigma(i))$ is a lower bound on the amount of processing from J_i that must be scheduled between δ_0 and the profile σ (see Figure 2.2). Since the J_i's are pairwise disjoint, it follows that $\sum_{i=1}^{s} B(i,\sigma(i))$ is a lower bound on the amount of processing that must be scheduled between δ_0 and the profile σ. Since $\sum_{i=1}^{s} C(i,\sigma(i))$ is the total processing capacity of $P_1, \cdots, P_s$ between δ_0 and the profile σ, it

Figure 2.1 Example profile.

follows that if there is a feasible schedule, then

$$\sum_{i=1}^{s} B(i, \sigma(i)) \leq \sum_{i=1}^{s} C(i, \sigma(i)) \tag{2.1}$$

for every $\sigma \in \pi$. We shall show in Theorem 1 that there is a feasible schedule iff (2.1) holds.

2.2 Obtaining A Feasible Schedule

We now ready to introduce the ideas that lead to the feasibility algorithm of section 2.3. Our algorithm begins by computing the amount w_j of each job j that is to be scheduled between δ_0 and δ_1. These w_js are determined such that they can be scheduled in the interval δ_0 to δ_1 and the remaining processing requirements $\{t_j - w_j : 1 \leq j \leq n\}$ can be feasiby met from δ_1 to δ_q.

Once the w_js are known the schedule from δ_0 to δ_1 may be obtained. The schedule for the remaining q-1 intervals is similarly obtained. The w_js are computed starting with jobs in J_1 and proceeding to J_2 etc. Observe that jobs in $J_{k+1} \cup \cdots \cup J_s$ compete with the jobs in J_k for the processing time available on P_1, ..., P_k from δ_0 to δ_1. Hence, while determining w_j, $j \in J_k$, we must also determine a value R_k that represents the amount of $P_1, \cdots, P_k$s processing capacity in the interval δ_0 to δ_1 that is to be reserved for the jobs in $J_{k+1} \cup \cdots \cup J_s$.

B(i,σ(i)) must be scheduled in the rectangle ABCD and hence to the left of the profile σ.

Figure 2.2

Considering the definition of a(j,d), it seems plausible to compute the w_js using the greedy method as below:

for d 0,1,2,... **until** satisfied **do**
set w_j to a(j,d) for every jεJ_k
end

We would like to compute R_k in a similar manner from a yet to be defined quantity Y(k,d). We shall define Z(k,d), Y(k,d), and X(k,d), 0≤k≤s, 0≤d≤q such that:

(i) In any feasible schedule, at least Z(k,d) amount of $J_{k+1} \cup \cdots \cup J_s$ must be done on $P_1, \cdots, P_k$ by time δ_d.

(ii) In any feasible schedule, at least X(k,d) amount of $J_{k+1} \cup \cdots \cup J_s$ must be processed on $P_1, \cdots, P_k$ between δ_1 and δ_d.

(iii) Y(k,d) = Z(k,d) - X(k,d). Hence, one may think of Y(k,d) as representing the maximum amount of Z(k,d) that can be done between δ_0 and δ_1.

It is important to note that $(R_k, Z(k,d), Y(k,d))$, when regarded as an attribute of $J_{k+1} \cup \cdots \cup J_s$ is the counterpart of $(w_j, b(j,d), a(j,d))$ if the latter is considered an attribute of job j, j$\in J_k$.

We now proceed to define Z(k,d). Define π_d as below:

$$\pi_d = \{\ \sigma \mid \sigma\ \varepsilon\ \pi \text{ and } \sigma(i) \le d,\ 0 \le i \le s\}.$$

From the discussion of section 2.1, it follows that for any profile $\sigma\ \varepsilon\ \pi_d$,

$$\sum_{i=k+1}^{s} B(i, \sigma(i)) - \sum_{i=k+1}^{s} C(i, \sigma(i))$$

gives a lower bound on the amount of $J_{k+1} \cup \cdots \cup J_s$ that must be done on $P_1, \cdots, P_k$ by time $\delta_{\sigma(k+1)}$ (and hence by time δ_d as $\sigma(k+1) \le d$). Hence,

$$Z(k,d) = \max_{\sigma \varepsilon \pi_d} \{ \sum_{i=k+1}^{s} B(i, \sigma(i)) - \sum_{i=k+1}^{s} C(i, \sigma(i)) \} \tag{2.2}$$

is also a lower bound on the amount of $J_{k+1} \cup \cdots \cup J_s$ that must be done on $P_1, \cdots, P_k$ by δ_d.

From (2.2) we may obtain a simple recurrence for Z(k,d). Let $\sigma' \varepsilon \pi_d$ be the σ at which

$$\sum_{i=k+1}^{s} B(i, \sigma(i)) - \sum_{i=k+1}^{s} C(i, \sigma(i))$$

is maximum. Assume that $k < s$. If $\sigma'(k+1) \neq d$, then $Z(k,d) = Z(k,d-1)$. If $\sigma'(k+1) = d$, then

$$\begin{aligned} Z(k,d) &= \sum_{i=k+1}^{s} B(i, \sigma'(i)) - \sum_{i=k+1}^{s} C(i, \sigma'(i)) \\ &= \sum_{i=k+2}^{s} B(i, \sigma'(i)) - \sum_{i=k+2}^{s} C(i, \sigma'(i)) + B(k+1,d) - C(k+1,d) \\ &= Z(k+1,d) + B(k+1,d) - C(k+1,d). \end{aligned}$$

This yields:

$$Z(k,d) = \begin{cases} 0 & \text{if } k = s \\ Z(k,0) & \text{if } d = 0 \\ \max\{Z(k,d-1),\ Z(k+1,d) + B(k+1,d) - C(k+1,d)\} & \textit{otherwise.} \end{cases} \tag{2.3}$$

Define D(k,d) as below:

$$D(k,d) = \begin{cases} 0 & d = 0 \\ C(k,1) & \textit{otherwise.} \end{cases}$$

When $d \neq 0$, D(k,d) is nothing but the processing capacity of P_k from δ_0 to δ_1.

To arrive at a formula for X(k,d), we note that for any i and d, B(i,d) - A(i,d) is a lower bound on the amount of J_i's processing that must be done between δ_1 and δ_d. The processing capacity of P_i in this interval is C(i,d) - D(i,d). So, for any profile function $\sigma \varepsilon \pi_d$'

$$\sum_{i=k+1}^{s} [B-A](i, \sigma(i)) - \sum_{i=k+1}^{s} [C-D](i, \sigma(i))$$

is a lower bound on the amount of $J_{k+1} \cup \cdots \cup J_s$ that must be done on $P_1, \cdots, P_k$ between δ_1 and δ_d. Consequently X(k,d), 0≤k≤s, 0≤d≤q, as defined below:

$$X(k,d) = \max_{\sigma \varepsilon \pi_d} \{ \sum_{i=k+1}^{s} [B - A - C + D](i, \sigma(i)) \} \tag{2.4}$$

is a lower bound on the amount of $J_{k+1} \cup \cdots \cup J_s$ that must be processed on $P_1, \cdots, P_k$ between δ_1 and δ_d.

From (2.4) the recurrence

$$X(k,d) = \begin{cases} 0 & k = s \\ X(k,0) & d = 0 \\ \max\{X(k,d-1), [X + B - A - C + D](k+1,d)\} & otherwise \end{cases} \tag{2.5}$$

may be obtained in the same way as (2.3) was obtained from (2.2).

Define:

$$Y(k,d) = Z(k,d) - X(k,d),\ 0 \le k \le s,\ 0 \le d \le q.$$

Some of the identities that we shall use in section 2.3 are stated below.

Lemma 1: If $\sum_{i=1}^{s} B(i, \sigma(i)) \le \sum_{i=1}^{s} C(i, \sigma(i))$ for every σ ε π, then

(1a) A(k,0) = B(k,0) = 0, 1 ≤ k ≤ s.

(1b) A(k,1) = B(k,1), 1 ≤ k ≤ s.

(2a) X(k,1) = 0, 0 ≤ k ≤ s.

(2b) X(k,0) = Y(k,0) = Z(k,0) = 0, 0 ≤ k ≤ s.

(2c) X(0,d) = Y(0,d) = Z(0,d) = 0, 0 ≤ d ≤ q.

(3) The functions a, b, A, B, C, D, X, Y, and Z all have nonnegative values and are nondecreasing in the second variable; i.e.,

0 ≤ f(k,d) ≤ f(k,d+1) for f = a, b, A, B, C, D, X, Y, and Z.

Proof: See Appendix. []

2.3. The Algorithm

We are now ready to describe our preemptive scheduling algorithm. The jobs will be scheduled in q phases. In phase d we determine the amount of each job j that is to be scheduled from δ_{d-1} *to* δ_d. Once this amount has been determined, the actual schedule from δ_{d-1} *to* δ_d is constructed using the Kafura-Shen algorithm.

Procedure COMPUTE_W determines the amount w_j of job j that is to be scheduled from

```
line      Procedure COMPUTE_W
          //w_j is the amount of job j to be processed from
        δ_0 to δ_1//
 1        R_0 ← 0
 2        for k ← 1 to s do //consider jobs by classes//
 3          Q_k ← { d | A(k,d) + Y(k,d) ≤ R_{k-1} + C(k,1)}
 4        if Q_k =  then print('infeasible job set')
 5                        stop  endif
 6          h_k ← max{ d | d ε Q_k}
 7          case
 8                  :(1) h_k = q: w_j ← a(j,q), j ε J_k
 9                  :(2) h_k<q, A(k,h_k+1)+Y(k,h_k)≥R_{k-1}+C(k,1):
10                set w_j for j ε J_k such that
11                      a(j,h_k) ≤ w_j ≤ a(j,h_k+1) and

12                Σ_{jεJ_k} w_j + Y(k,h_k) = R_{k-1} + C(k, 1)

13                  :(3) else:  w_j ← a(j, h_k+1), j ε J_k
14          end case
15          R_k ← R_{k-1} + C(k,1) - Σ_{jεJ_k} w_j
16        end for
17        end COMPUTE_W
```

Figure 2.3

δ_0 *to* δ_1. The w_js are determined in a way such that $\{w_j : 1 \le j \le n\}$ can be scheduled from δ_0 to δ_1 and the remaining processing requirements $\{t_j - w_j : 1 \le j \le n\}$ can be feasibly scheduled from δ_1 to δ_q. Once the w_js are determined, the amount w'_j of job j, $1 \le j \le n$ that is to be scheduled from δ_1 to δ_2 is determined by applying COMPUTE_W to $\{t_j - w_j : 1 \le j \le n\}$. Repeatedly applying COMPUTE_W in this way, one may successfully determines the w_js for each interval.

In procedure COMPUTE_W, R_k denotes the amount of idle time remaining on processor

classes 1, 2, ..., k following the scheduling of the w_js corresponding to jobs in job classes 1, 2, ..., k. (One may also think of R_k as the amount of processing time on processor classes 1, 2, ..., k that is to be reserved for jobs in job classes k+1, k+2, ..., s.) Roughly speaking, COMPUTE_W computes the w_js (job) class by class. In determining the w_js for job class k, the processing capacity available is equal to R_{k-1} + C(k,1). Initially, let w_j <- a(j,0) for j ε J_k and R_k <- Y(k,0). If $\Sigma_{j\varepsilon J_k} w_j$ + $R_k < R_{k-1}$ + C(k,1), then w_j, j ε J_k is incremented to a(j,1) and R_k incremented to Y(k,1). If it is still the case that $\Sigma_{j\varepsilon J_k} w_j$ + $R_k < R_{k-1}$ + C(k,1), then w_j, j ε J_k is incremented to a(j,2) and R_k to Y(k,2). This procedure continues until R_{k-1} + C(k,1) is used up (i.e., until $\Sigma_{j\varepsilon J_k} w_j$ + $R_k = R_{k-1}$ + C(k,1)).

When actually implementing COMPUTE_W, the subscripts on h, R, and Q may be omitted. We have kept them in the version given in Figure 2.3 so that we may easily refer to the values of h, R, and Q during different iterations of the *for* loop. One should also note that in case (2), since $A(k,h_k+1) + Y(k,h_k) \geq R_{k-1} + C(k,1)$ and $A(k,h_k) + Y(k,h_k) \leq R_{k-1} + C(k,1)$, there exist w_j, $a(j,h_k) \leq w_j \leq a(j,h_k+1)$, such that

$$\Sigma_{j\varepsilon J_k} w_j + Y(k,h_k) = R_{k-1} + C(k,1).$$

These w_js are easily determined by first setting all $w_j = a(j,h_k)$, $j \varepsilon J_k$ and then incrementing the w_js one by one (up to at most a(j,h_k+1)) until the desired equality is satisfied.

2.4. Correctness and Complexity

We now proceed to prove the correctness of the above algorithm and analyze its complexity. We have pointed out in subsection 2.1 that if there exists a feasible schedule, then $\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i))$ for every σ ε π. We shall show in the following that if $\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i))$ for every σ ε π, then the above algorithm generates a feasible schedule.

Definition. For convenience in proving Lemmas 2 and 4, we arbitrarily define $Q_0 = \{0, 1\}$, and $h_0 = 1$.

We first show that if $\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i))$ for every σεπ, then procedure COMPUTE_W wil not terminate in line 5.

Lemma 2: If $\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i))$ for every σ ε π, then

(1) $Q_k \neq \phi$ *and* $h_k \geq 1$, $0 \leq k \leq s$.
(2) $R_k \geq Y(k, h_k)$, $0 \leq k \leq s$.
(3) $R_k \leq Y(k,h_k+1)$ if $h_k \neq q$, $0 \leq k \leq s$.

Proof: We shall show (1), (2), and (3) by induction on k.

I.B. When k = 0, $Q_0 \neq \phi$, $h_0 = 1$, and $R_0 = Y(0,h_0) = Y(0,h_0+1) = 0$ (either by definition or by Lemma 1).

I.H. Assume that (1), (2), and (3) are true for k-1 where $1 \leq k\text{-}1 < s$.

I.S. We shall show that (1), (2), and (3) are true for k. To show (1), we see that

$R_{k-1} \geq Y(k-1,h_{k-1})$ *(induction hypothesis)*

$\geq Y(k\text{-}1,1)$ *(Lemma 1)*

$= Z(k\text{-}1,1)$ *(Lemma 1)*

$\geq [Z+B\text{-}C](k,1)$ *(Eq. (2.3))*

$= [Y+A\text{-}C](k,1)$ *(Lemma 1)*

Hence, $1 \in Q_k$ and so $Q_k \neq \phi$ and $h_k \geq 1$.

To prove (2) and (3), consider the three cases of COMPUTE_W (lines 7-14).

Case 1: In this case, $\sum w_j = A(k,h_k)$. From lines 3 and 6, we observe that $A(k,h_k) + Y(k,h_k) \leq R_{k-1} + C(k,1)$. Combining these two facts with the definition of R_k (line 15), we obtain $Y(k,h_k) \leq R_k$.

Case 2: From lines 12 and 15, we obtain $R_k = Y(k,h_k) \leq Y(k,h_k+1)$.

Case 3: In this case, $A(k,h_k+1) + Y(k,h_k) < R_{k-1} + C(k,1) < A(k,h_k+1) + Y(k,h_k+1)$ and $\sum w_j = A(k,h_k+1)$. From these and line 15, we immediately obtain $Y(k,h_k) < R_k < Y(k,h_k+1)$. □

Before establishing the correctness of our scheme to compute the w_js, we obtain some relationships concerning the amount of processing t'_j of job j that remains to be done following time δ_1. Note that $t'_j = t_j - w_j$, $1 \leq j \leq n$.

Definition. Define b'(j,d), a'(j,d), B'(k,d), and A'(k,d) to be the values obtained for b, a, B, and A when t'_j is used in place of t_j. Let $C'(k,d) = |P_k|(\delta_d - \delta_1)$, $1 \leq k \leq s$, $1 \leq d \leq q$, and $W(k) = \Sigma_{j \in P_k} w_j$, $1 \leq k \leq s$.

Lemma 3: If $\sum_{i=1}^{s} B(i, \sigma(i)) \leq \sum_{i=1}^{s} C(i, \sigma(i))$ for every $\sigma \in \pi$, then

(1) $B'(k,d) \leq B(k,d) - A(k,d)$, $d \leq h_k$

(2) $B'(k,d) = B(k,d) - W(k),\ d > h_k \neq q$

Proof: It is easy to see that for any job j, $1 \leq j \leq n$,

$$b'(j,d) = \max\{0, b(j,d) - w_j\}$$

When $d \leq h_k$, from Lemma 1 and lines 8-13 of COMPUTE_W, we have $a(j,d) \leq a(j,h_i) \leq w_j$. Hence,

$$b'(j,d) = \max\{0,\ b(j,d) - w_j\} \leq b(j,d) - a(j,d)$$

Hence,

$$B'(k,d) \leq B(k,d) - A(k,d)$$

When $d > h_k \neq q$, from cases (2) and (3) of COMPUTE_W, Lemma 1, and the definition of a(j,d), we see that

$$a(j,h_k) \leq w_j \leq a(j,h_k+1) \leq a(j,d) \leq b(j,d).$$

So,

$$b'(j,d) = \max\{0,\ b(j,d) - w_j\} = b(j,d) - w_j.$$

Hence,

$$B'(k,d) = B(k,d) - W(k). \quad []$$

Lemma 4: If $\sum_{i=1}^{s} B(i, \sigma(i)) \leq \sum_{i=1}^{s} C(i, \sigma(i))$ for every $\sigma \in \pi$, then the following are true for every k, $0 \leq k \leq s$ and every σ such that $\sigma(k) \geq 1$:

(l) $\sum_{i=1}^{k} B'(i, \sigma(i)) + Z(k,\sigma(k)) - R_k \leq \sum_{i=1}^{k} C'(i, \sigma(i))$,

(2) If $\sigma(k) \leq h_k$, then

$$\sum_{i=1}^{k} B'(i, \sigma(i)) + X(k, \sigma(k)) \leq \sum_{i=1}^{k} C'(i, \sigma(i)).$$

Proof: The proof is by induction on k.

I.B. When k = 0, $\sum_{i=1}^{0} B'(i, \sigma(i)) = \sum_{i=1}^{0} C'(i, \sigma(i)) = R_0 = 0$ by definitions, and Z(0,σ(0)) = X(0,σ(0)) = 0 by Lemma 1. Hence, (1) and (2) hold.

I.H. Assume that (1) and (2) are true for k-1, where k-1 is in the range $0 \le$ k-1 < s.

I.S. We proceed to establish (1) and (2) for k by considering the three cases $\sigma(k) > h_k$; $\sigma(k) \le h_k$ and $\sigma(k\text{-}1) \le h_{k-1}$; and $\sigma(k) \le h_k$ and $\sigma(k\text{-}1) > h_{k-1}$.

Case 1: $\sigma(k) > h_k$: We first obtain the following

$$\sum_{i=1}^{k-1} B'(i, \sigma(i)) + Z(k-1, \sigma(k-1)) - R_{k-1} \le \sum_{i=1}^{k-1} C'(i, \sigma(i)) \quad (I.H.)$$

$$R_{k-1} - R_k = W(k) - C(k, 1) \qquad (def.\ of\ R_k)$$

$$B'(k, \sigma(k)) + W(k) = B(k, \sigma(k)) \qquad (Lemma\ 3)$$

$$Z(k, \sigma(k)) + B(k, \sigma(k)) - C(k, \sigma(k))$$

$$\le Z(k-1, \sigma(k)) \qquad (Eq.(2.3)\ and\ \sigma(k) \ge 1)$$

$$\le Z(k-1, \sigma(k-1)) \qquad (\sigma(k-1) \ge \sigma(k)\ and\ Eq.(2.3))$$

Adding these four equalities and inequalities yields:

$$\sum_{i=1}^{k} B'(i, \sigma(i)) + Z(k, \sigma(k)) - R_k \le \sum_{i=1}^{k} C'(i, \sigma(i))$$

Case 2: $\sigma(k) \le h_k$ and $\sigma(k\text{-}1) \le h_{k-1}$: From the induction hypothesis, we have

$$\sum_{i=1}^{k-1} B'(i, \sigma(i)) + X(k-1, \sigma(k-1)) \le \sum_{i=1}^{k-1} C'(i, \sigma(i)). \qquad (2.6)$$

From Eq.(2.5) and the fact that $\sigma(k\text{-}1) \ge \sigma(k) \ge 1$, we get:

$$X(k-1, \sigma(k-1)) \ge X(k-1, \sigma(k)) \ge [X + B - A - C + D](k, \sigma(k)).$$

Using Lemma 3, this reduces to

$$X(k-1, \sigma(k-1)) \ge X(k, \sigma(k)) + B'(k, \sigma(k)) - C'(k, \sigma(k)).$$

Combining with (2.6) yields:

$$\sum_{i=1}^{k} B'(i, \sigma(i)) + X(k, \sigma(k)) \le \sum_{i=1}^{k} C'(i, \sigma(i)). \qquad (2.7)$$

Since $R_k \geq Y(k,h_k) \geq Y(k,\sigma(k))$ (Lemmas 4 and 3) we conclude that Z(k,σ(k)) - R_k ≤ X(k,σ(k)). Substituting into (2.7) yields

$$\sum_{i=1}^{k} B'(i,\sigma(i)) + Z(k,\sigma(k)) - R_k \leq \sum_{i=1}^{k} C'(i,\sigma(i)).$$

case 3: σ(k) ≤ h_k and σ($k-1$) > h_{k-1}: From the induction hypothesis, we get

$$\sum_{i=1}^{k-1} B'(i,\sigma(i)) + Z(k-1,\sigma(k-1)) - R_{k-1} \leq \sum_{i=1}^{k-1} C'(i,\sigma(i)). \tag{2.8}$$

Since $h_{k-1} \neq$ q, we obtain from Lemma 2:

$$R_{k-1} \leq Y(k-1,h_{k-1}+1) \leq Y(k-1,\sigma(k-1)). \tag{2.9}$$

From Lemma 3, Eq. (2.5), and the inequality σ(k-1) ≥ σ(k) ≥ 1, we get

$$[X + B' - C'](k,\sigma(k)) \leq X(k-1,\sigma(k)) \leq X(k-1,\sigma(k-1)). \tag{2.10}$$

Adding (2.8), (2.9), and (2.10) yields:

$$\sum_{i=1}^{k} B'(i,\sigma(i)) + X(k,\sigma(k)) \leq \sum_{i=1}^{k} C'(i,\sigma(i)).$$

Using the same reasoning as in case 2, we may now conclude the truth of (1) for k. □

Theorem 1: There exists a feasible preemptive schedule for the given n jobs if and only if

$$\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i)) \text{ for every } \sigma \in \pi_q.$$

Proof: We have already pointed out that if a feasible schedule exists, then the above inequality is satisfied for every σ ε π_q. So, we need only show that when the above inequality is satisfied for every σ ε π_q, there is a feasible schedule. Assume that

$$\sum_{i=1}^{s} B(i,\sigma(i)) \leq \sum_{i=1}^{s} C(i,\sigma(i)) \text{ for every } \sigma \in \pi_q. \tag{2.11}$$

From (2.11) it is clear that when q = 1, the t_js and Eq (1.1) yield f* ≤ δ_1 - δ_0 and so a feasible schedule exists.

For the induction hypothesis, we assume that there exists a feasible schedule when (2.11) is satisfied and q = r for some r, 1 ≤ r. We show that if (2.11) is satisfied when q = r+1, then there is a feasible schedule. From Lemma 2, we see that $Q_k \neq \phi$ for any k. Hence procedure

COMPUTE_W successfully computes the $w_j s$. Let $P(k) = \bigcup_{i=1}^{k} P_i$. It is clear from COMPUTE_W that $w_j \le a(j,d) \le \delta_1 - \delta_0$ where d = q or h_k+1 and that $\Sigma_{i \varepsilon P(k)} w_j \le |P(k)|(\delta_1-\delta_0) - R_k \le |P(k)|(\delta_1-\delta_0)$ for every k. Hence, the $w_j s$ satisfy (1.1) (i.e., $f^* \le \delta_1 - \delta_0$) and may be scheduled from δ_0 *to* δ_1 using the Kafura-Shen algorithm.

Now, consider the $t'_j s$. We know that X(s,d) = Z(s,d) = 0, $0 \le d \le r+1$. If $h_s < r+1$, then from Lemma 2, we obtain $0 \le R_s \le Y(s,h_s+1) = 0$ or $R_s = 0$. Using this in Lemma 4 yields:

$$\sum_{i=1}^{s} B'(i, \sigma(i)) \le \sum_{i=1}^{s} C'(i, \sigma(i)) \text{ for } \textit{every } \sigma \varepsilon \pi_{r+1}$$

$$\textit{such that } \sigma(s) \ge 1. \tag{2.12}$$

If $h_s = r+1$ then $\sigma(s) \le h_s$ and from Lemma 4 we once again obtain (2.12). One readily sees that (2.12) is equivalent to

$$\sum_{i=1}^{s} B'(i, \sigma(i)+1) \le \sum_{i=1}^{s} C'(i, \sigma(i)+1) \text{ for } \textit{every } \sigma \varepsilon \pi_r. \tag{2.13}$$

Following the scheduling from δ_0 *to* δ_1, we are left with the problem of scheduling the $t'_j s$ from δ_1 to δ_{r+1}. The number of distinct due times is now r (note that $t'_j = 0$ for every j such that $d_j = \delta_1$). Relabel the start time δ_1 as δ'_0 and the due times $\delta_2, \ldots, \delta_{r+1}$ as $\delta'_1, \ldots, \delta'_r$. Define b"(j,d), B"(k,d), and C"(k,d) to be the values obtained for b, B, and C when t'_j is used in place of t_j and δ'_d is used in place of δ_d. We immediately see that B"(k,d) = B'(k,d+1), and C"(k,d) = C'(k,d+1), for every k, d, $1 \le k \le s$, $0 \le d \le r$. Substituting into (2.13) yields:

$$\sum_{i=1}^{s} B''(i,\sigma(i)) \le \sum_{i=1}^{s} C''(i,\sigma(i)) \text{ for every } \sigma \varepsilon \pi_r.$$

It now follows from the induction hypothesis that the t'_js can be scheduled. []

From Theorem 1, it is clear that by repeatedly using COMPUTE_W to determine the amount to be scheduled in each interval, a feasible schedule can be obtained whenever such a schedule exists. Each time COMPUTE_W is used, we need to recompute b, a, Z, X, and Y. The time needed for this is O(nq)(note that recurrences 2.3 and 2.5 will be used to compute Z and X). The *for* loop may be executed in O(qs + n) time. We may assume that $s \le n$ and so the complexity of COMPUTE_W is O(qn). The Kafura-Shen algorithm is of complexity O(n). Hence, the overall computing time for the q phases of our scheduling algorithm is $O(q^2 n)$. An additional O(nlogn) time is needed to sort the jobs by memory size m_i. Hence, the overall complexity of our preemptive scheduling algorithm is $O(q^2 n + n \log n)$. As for preemptions, since each job may be preempted at most twice in each interval $[\delta_d, \delta_{d+1}]$, $0 \le i \le q-1$, the total number of preemptions is O(nq).

3. Minimizing maximum lateness

Let S be a preemptive schedule for (t_j, d_j, m_j), $1 \le j \le n$. Let f_j be the finish time of job j in S. If $f_j \le d_j$, $1 \le j \le n$ then S is a feasible schedule and no job is late. The *lateness* of job j is $f_j - d_j$ and the *maximum lateness* of the n jobs is $L_{\max} = \max\{f_j - d_j : 1 \le j \le n\}$. Note that $L_{\max} \le 0$ iff all jobs finish by their due times. Also, note that if $L_{\max} \le 0$ then $\delta_0 - \delta_1 \le L_{\max}$.

From the definition of $L_{\max}$, it follows that by changing the release time from δ_0 *to* $\delta_0 - L_{\max}$ we obtain a job set that can be scheduled such that no job finishes after its due time. Hence, to determine the minimum $L_{\max}$, we need to determine the least x such that the condition of Theorem 1 is satisfied when a release time of $\delta_0 - x$ is used. This x may be obtained from a form equivalent to that of Theorem 1. We observe that $\sum_{i=1}^{s} B(i,\sigma(i)) \le \sum_{i=1}^{s} C(i,\sigma(i))$ for every $\sigma \in \pi$ iff $\max_{\sigma\in\pi}\{\sum_{i=1}^{s} B(i,\sigma(i)) - \sum_{i=1}^{s} C(i,\sigma(i))\} \le 0$. It is helpful to rewrite this form seperating out the case when $\sigma(i) = 0$, $1 \le i \le s$. For this σ, we see that $\sum_{i=1}^{s} B(i,\sigma(i)) - \sum_{i=1}^{s} C(i,\sigma(i)) = \sum_{j=1}^{n} b(j,0)$. For every other σ, there is a k, $1 \le k \le s$ such that $\sigma(k) \ge 1$.

Define H_k as below:

$$H_k = \max_{\sigma\in\pi,\,\sigma(k)\ge 1}\{\sum_{i=1}^{k} B(i,\sigma(i)) - \sum_{i=1}^{k} C(i,\sigma(i))\},\ 1 \le k \le s.$$

We immediately see that

$$\max_{\sigma\in\pi}\{\sum_{i=1}^{s} B(i,\sigma(i)) - \sum_{i=1}^{s} C(i,\sigma(i))\}$$

$$= \max\{\sum_{j=1}^{n} b(j,0), H_1, H_2, \ldots, H_s\}$$

Let $x_1 = \max_{1\le j\le n}\{t_j - d_j + \delta_0\}$ and let $x_2 = \max_{1\le k\le s}\{H_k/|P(k)|\}$, where |P(k)| is the cardinality of P(k) $= \bigcup_{i=1}^{k} P_i$.

Clearly, if we change the release time to $\delta_0 - \max\{x_1, x_2\}$ then $\max\{\Sigma b'(j,0), H'_1, H'_2, \cdots, H'_s\} = 0$ (the b', H'_i values are computed with respect to the new release time $\delta_0 - \max\{x_1, x_2\}$). Hence, $\max_{\sigma\in\pi}\{\Sigma\ B'(i,\sigma(i)) - \Sigma\ C'(i,\sigma(i))\} \le 0$ and $\Sigma\ B'(i,\sigma(i)) \le \Sigma\ C'(i,\sigma(i))$ for every $\sigma \in \pi$. Moreover, $x = \max\{x_1, x_2\}$ is the least value of x for which this happens. Hence,

$$(L_{\max})_{\min} = \max\{x_1, x_2\}.$$

The H_ks may be computed in O(qs) time as follows. Define H_k^d as below:

$$H_k^d = \max_{\substack{\sigma \varepsilon \pi \\ \sigma(k) \geq d}} \{ \sum_{i=1}^{k} B(i, \sigma(i)) - \sum_{i=1}^{k} C(i, \sigma(i)) \},\ 1 \leq k \leq s,\ 1 \leq d \leq q.$$

Hence, $H_k = H_k^1$, $1 \leq k \leq s$. We immediately obtain the following recurrence for H_k^d:

$$H_k^d = \begin{cases} \max_{i \geq d} \{B(1,i) - C(1,i)\} & \text{if } k = 1 \\ \sum_{i=1}^{s} B(i,q) - \sum_{i=1}^{s} C(i,q) & \text{if } d = q \\ \max\{H_{k-1}^d + B(k,d) - C(k,d),\ H_k^{d+1}\} & \textit{otherwise.} \end{cases}$$

Using this recurrence, all the H_k^d s may be obtained in O(qs) time (excluding the time needed to determine the b(j,d)s, B(k,d)s etc.). The additional time needed to compute the B(k,d)s and C(k,d)s is O(qn + nlogn) (assuming $n \geq m$). Hence, the minimum $L_{\max}$ may be determined in O(qn + nlogn) time. Having determined the minimum $L_{\max}$, a schedule having this $L_{\max}$ value can be obtained by changing δ_0 *to* $\delta_0 - (L_{\max})_{\min}$ and using the algorithm of Section 2.

4. Conclusions

We have developed an $O(q^2 n + n \log n)$ algorithm to obtain a preemptive schedule for n jobs (t_j, d_j, m_j), $1 \leq j \leq n$ on m processors with given memory sizes. This schedule minimizes $L_{\max}$ and contains at most O(qn) preemptions. The minimum value of $L_{\max}$ can itself be obtained in only O(qn + nlogn) time.

Appendix. Proof of Lemma 1

Assume that

$$\sum_{i=1}^{s} B(i, \sigma(i)) \leq \sum_{i=1}^{s} C(i, \sigma(i)) \tag{A.1}$$

for every $\sigma \varepsilon \pi$. Using $\sigma(i) = 0$, $1 \leq i \leq s$ in (A.1), we obtain $B(i,0) = 0$ for $1 \leq i \leq s$. From this, we have $b(j,0) = 0$, $1 \leq j \leq n$ and, therefore,

$$t_j \leq d_j - \delta_0, \quad 1 \leq j \leq n \tag{A.2}$$

From Eqs. (A.1) and (A.2), it is easy to verify (1), (2), and (3) (except the case when f = Y) of Lemma 1.

The rest of this appendix is devoted to prove (3) for f = Y; i.e.,

$$0 \leq Y(k,d) \leq Y(k,d+1),\ 0 \leq k \leq s,\ 0 \leq d \leq q-1$$

To prove this inequality, since Y is defined as Z - X, we need to know the relation between X and

Z. (See Claim 2 below). But X and Z are in turn defined through A and B, so, we first establish the relationship concerning A and B:

Claim 1: $\frac{B(k,d)-A(k,d)}{\delta_d-\delta_1} \le \frac{B(k,d)}{\delta_d-\delta_0}$, $1 \le k \le s,\ 1 < d \le q$.

Proof: Since a(j,d) = min{b(j,d), $\delta_1 - \delta_0$}, a(j,d) = b(j,d) or $a(j,d) = \delta_1 - \delta_0$.

If a(j,d) = b(j,d), then

$$\frac{\delta_1-\delta_0}{\delta_d-\delta_0}\, b(j,d) \le a(j,d)$$

as $\delta_1 \le \delta_d$.

If a(j,d) = $\delta_1 - \delta_0$, then since $t_j \le d_j - \delta_0$ implies $b(j,d) \le \delta_d - \delta_0$, we get

$$\frac{\delta_1-\delta_0}{\delta_d-\delta_0}\, b(j,d) \le \delta_1-\delta_0 = a(j,d).$$

So, in both cases we have

$$\frac{\delta_1-\delta_0}{\delta_d-\delta_0}\, b(j,d) \le a(j,d).$$

Hence,

$$\frac{\delta_1-\delta_0}{\delta_d-\delta_0}\, B(k,d) \le A(k,d)$$

or

$$\frac{\delta_1-\delta_0}{\delta_d-\delta_1}\, B(k,d) \le \frac{\delta_d-\delta_0}{\delta_d-\delta_1}\, A(k,d)$$

or

$$\frac{\delta_d-\delta_0}{\delta_d-\delta_1}\, [B(k,d)-A(k,d)] \le B(k,d)$$

or

$$\frac{B(k,d)-A(k,d)}{\delta_d-\delta_1} \le \frac{B(k,d)}{\delta_d-\delta_0}.\ []$$

Claim 2: $(\delta_d-\delta_0)X(k,d) \le (\delta_d-\delta_1)Z(k,d),\ 0 \le k \le s,\ 0 \le d \le q$

Proof: The proof is by induction on k and d.

I.B. on k When k = s and $0 \le d \le q$, X(k,d) = Z(k,d) = 0.

I.H. on k Assume that the inequality is correct when $0 \le k = k'+1 \le s$ and $0 \le d \le q$.

I.S. on k When k = k' and $0 \le d \le q$, the inequality may be shown correct by induction on d.

I.B. on d When d = 0 or 1, X(k',d) = 0 and $Z(k',d) \ge 0$.

I.H. on d Assume that the inequality is correct when $q \ge d = d'-1 \ge 1$.

I.S. on d We need to show the inequality is correct for d = d' (and k = k'). From Eq. (2.5), we see that there are two possibilities for X(k',d').

Case(i) X(k',d') = X(k',d'-1): In this case,

$$(\delta_{d'-1}-\delta_0)X(k',d') = (\delta_{d'-1}-\delta_0)X(k',d'-1)$$

$$\le (\delta_{d'-1}-\delta_1)Z(k',d'-1) \quad (\textit{I.H. on } d)$$

$$\le (\delta_{d'-1}-\delta_1)Z(k',d') \quad (\textit{Eq. } (2.3))$$

Since $\delta_{d'-1} - \delta_0 > \delta_{d'-1} - \delta_1 \ge 0$, we get

$$X(k',d') \le Z(k',d')$$

or

$$(\delta_{d'}-\delta_{d'-1})X(k',d') \le (\delta_{d'}-\delta_{d'-1})Z(k',d').$$

Adding this inequality to the previous one yields:

$$(\delta_{d'}-\delta_0)X(k',d') \le (\delta_{d'}-\delta_1)Z(k',d').$$

Case(ii) X(k',d') = [X+B-A-C+D](k'+1,d'): Now, we obtain:

$$\frac{X(k',d')}{\delta_{d'}-\delta_1} = \frac{[X+B-A-C+D](k'+1,d')}{\delta_{d'}-\delta_1}.$$

Using I.H. on k, Claim 1, and the equality

$$\frac{C(k'+1,d')-D(k'+1,d')}{\delta_{d'}-\delta_1} = |P_{k'+1}| = \frac{C(k'+1,d')}{\delta_{d'}-\delta_0},$$

we obtain:

$$\frac{X(k',d')}{\delta_{d'} - \delta_1} \leq \frac{Z(k'+1,d') + B(k'+1,d') - C(k'+1,d')}{\delta_{d'} - \delta_0}$$

$$\leq \frac{Z(k',d')}{\delta_{d'} - \delta_0} \quad (Eq.\ (2.5)) \quad []$$

We are now ready to show $Y(k,d) \geq Y(k,d-1) \geq 0$, $0 \leq k \leq s$, $1 \leq d \leq q$.

Claim 3: $Y(k,d) \geq 0$, $0 \leq k \leq s$, $0 \leq d \leq q$.

Proof: For $k \geq 1$, this follows from Claim 2 and the fact $\delta_k - \delta_0 > \delta_k - \delta_1 \geq 0$. For $k = 0$, this follows from part (2b) of Lemma 1. []

Claim 4: $Y(k,d) \geq Y(k,d-1)$, $0 \leq k \leq s$, $1 \leq d \leq q$.

Proof: The proof is by induction on k.

I.B. When $k = s$ and $1 \leq d \leq q$, $Y(k,d) = Y(k,d-1) = 0$.

I.H. Assume that $Y(k,d) \geq Y(k,d-1)$ for $1 \leq k' < k \leq s$ and $1 \leq d \leq q$.

I.S. We need to show that $Y(k',d) \geq Y(k',d-1)$ for $1 \leq d \leq q$.

Let d be in the range $1 \leq d \leq q$.

Case (i) $Z(k',d) \geq Z(k',d-1)$ and $X(k',d) = X(k',d-1)$:

In this case, it is readily seen that $Y(k',d) \geq Y(k',d-1)$.

Case (ii) $Z(k',d) = Z(k',d-1)$ and $X(k',d-1) < X(k',d) = [X+B-A-C+D](k'+1,d)$:

This case is not possible. To see this, suppose that this case is possible. Let $k'' \geq k'$ be the largest k'' for which

$$Z(k'',d) = Z(k'',d-1) \text{ and } X(k'',d-1) < [X+B-A-C+D](k''+1,d) \quad (A.3)$$

for some d. Let d'' be the smallest d for which (A.3) holds. Note that $d'' > 0$. So,

$Z(k'',d'') = Z(k'',d''-1)$ and $X(k'',d''-1) < [X+B-A-C+D](k''+1,d'')$

Since X(k",d"-1) ≥ 0, it follows from Eqs. (2.5) and (A.4) that X(k",d") > 0. Assume that

$$Z(k'',d'') = Z(k'',d''-1) = Z(k'',d''-2) = \cdots = Z(k'',d^*) \neq Z(k'',d^*-1).$$

Then, it follows from our choice of k" and d" that

$$X(k'',d''-1) = X(k'',d''-2) = \cdots = X(k'',d^*).$$

If d* = 0, then 0 = Z(k",d*) = Z(k",d") ≥ X(k",d") > 0. Hence, d* ≠ 0. Now, from the choice of d*, we get

$$\begin{aligned} Y(k'',d^*) &= Z(k'',d^*) - X(k'',d^*) \\ &\leq [Z+B-C](k''+1,d^*) - [X+B-A-C+D](k''+1,d^*) \\ &= [Y+A-D](k''+1,d^*). \end{aligned} \tag{A.5}$$

Also,

$$X(k'',d''-1) < [X+B-C-A+D](k''+1,d''),$$

and

$$Z(k'',d''-1) \geq [Z+B-C](k''+1,d'').$$

So,

$$Y(k'',d^*) = Y(k'',d''-1) > [Y+A-D](k''+1,d'').$$

Substituting into (A.5) yields:

$$[Y+A-D](k''+1,d^*) > [Y+A-D](k''+1,d'')$$

or,

$$\begin{aligned} Y(k''+1,d^*) &> [Y+A-D](k''+1,d'') - [A-D](k''+1,d^*) \\ &\geq Y(k''+1,d'') \quad \textit{(Lemma 1 and def. of D)} \end{aligned}$$

But, k" + 1 > k' and so from I.H., it follows that

$$Y(k''+1,d'') \geq Y(k''+1,d''-1) \geq \cdots \geq Y(k''+1,d^*).$$

So, case (ii) is not possible.

Case (iii) Z(k',d) = [Z+B-C](k'+1,d) and X(k',d) = [X+B-A-C+D](k'+1,d):

Now, Y(k',d) = [Y+A-D](k'+1,d). Suppose that

$$Z(k',d-1) = Z(k',d-2) = \ldots = Z(k',d^*) \neq Z(k',d^*-1) \quad (A.6)$$

From the proof of case (ii), it follows that $X(k',d-1) = X(k',d-2) = \ldots = X(k',d^*)$. So, $Y(k',d-1) = Y(k',d^*)$. If $d^* = 0$, then $Y(k',d-1) = Y(k',0) = 0 \leq Y(k',d)$. If $d^* \neq 0$, then

$$\begin{aligned}
Y(k',d-1) &= Y(k',d^*) \\
&\leq [Y+A-D](k'+1,d^*) \qquad (\textit{Eqs. A.}6,\ 2.3,\ 2.5) \\
&\leq Y(k'+1,d)+[A-D](k'+1,d^*) \quad (\textit{I.H.}) \\
&\leq [Y+A-D](k'+1,d) \qquad (\textit{Lemma}\ 1) \\
&= Y(k',d). \quad []
\end{aligned}$$

References

1. M. Garey and D. Johnson, "Computers and intractability, a guide to the theory of NP-Completenss," W. H. Freeman and Co., San Francisco, l979.
2. D. Kafura and V. Shen, "Task scheduling on a multiprocessor system with independent memories," *SICOMP*, Vol. 6, No. 1, l977, pp. l67-l87.
3. T. Lai and S. Sahni, "Preemptive scheduling of a multiprocessor system with memories to minimize maximum lateness," Tech. Report 81-20, Computer Science Department, University of Minnesota, Minneapolis, 1981.
4. R. McNaughton, "Scheduling with deadlines and loss functions," *Manag-Sci*, 12, 7, l959.
5. S. Sahni, "Preemptive scheduling with due dates," *Op. Res.*, , Vol. 27, No. 5, l979, pp. 925-934.
6. S. Sahni and Y. Cho, "Scheduling independent tasks with due times on a uniform processor system," *JACM*, Vol. 27, No. 3, l980, pp. 550-563.
7. S. Sahni and Y. Cho, " Nearly on line scheduling of a uniform processor system with release times," *SICOMP* Vol. 8, No. 2, 1979, pp. 275-285.
8. E. Coffman, Jr., "Computer and job shop scheduling theory", John Wiley and Sons, Inc., New York, 1976.