

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

SARTAJ SAHNI† AND YOONKUN 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 $P = \{P_1, P_2, \dots, P_m\}$ is a set of m processors (machines). Associated with each processor, P_i , is a speed s_i , $s_i > 0$, $1 \leq i \leq m$. Processor P_i can perform s_i units of processing in one unit of time. When $s_i = s_{i+1}$, $1 \leq i < m$, 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 , $1 \leq i \leq n$.

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 , $1 \leq i \leq v$. Assume $r_i < r_{i+1}$, $1 \leq i < v$ and $r_1 = 0$. Our algorithm works in v phases. In the i th phase tasks are scheduled from time r_i to time r_{i+1} , $1 \leq i < v$. In the v th (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 i th 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 $a_1 \geq a_2 \geq \dots \geq a_l$ be a set of task times. Let w be the minimum finish time of any preemptive schedule for these l tasks on a system $P = \{P_1, P_2, \dots, P_m\}$. Let $s_1 \geq s_2 \geq \dots \geq s_m$. Then,

$$(1) \quad w = \max \left\{ \sum_1^l a_i / \sum_1^m s_i, \max_{1 \leq j < m} \left\{ \sum_1^j a_i / \sum_1^j s_i \right\} \right\}.$$

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 $S(i) = \sum_{j=1}^i s_j$, $1 \leq i \leq m$ and $S(0) = 0$. It is assumed that $s_i \geq s_{i+1}$, $1 \leq i < m$. At the start of EQUAL, $t(i)$ is the RPT for task i , $1 \leq i \leq p$. 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 $t'(i) \geq t'(i+1)$, $1 \leq i < p$ and $\max \{ \sum_{i=1}^p t'(i) / S(m), \max_{1 \leq j < m} \{ \sum_{i=1}^j t'(i) / S(j) \} \} \leq \Delta$. Hence, the assignments determined by EQUAL for the Δ interval can be scheduled. Furthermore, at termination we shall have $t(i) \geq t(i+1)$, $1 \leq i < p$.

