DEADLINE SCHEDULING OF TASKS WITH READY TIMES AND RESOURCE CONSTRAINTS

Jacek BŁAŻEWICZ
Institute of Control Engineering, Technical University of Poznań, Ul. Piotrowo 3, Pl–60 965, Poznań, Poland

Received 10 April 1978; revised version received 20 September 1978;

Scheduling to meet deadlines, multiprocessor environment, scheduling under resource constraints

1. Introduction

The problem of scheduling tasks to meet deadlines may be stated as follows. There are given: a set of m identical *processors,* as set of n *tasks* $\mathcal{T} = \{T_1, T_2, ..., T_n\}$, a set of additional *resources* $\mathcal{R} = \{R_1, R_2, ..., R_s\}$, where m_l units of resource R_l, $l = 1, 2, ..., s$, are available. For every task T_i we have specified: an *execution time* $\tau_i > 0$, a *ready time* $r_i \geq 0$, a *deadline* $d_i > 0$, and a vector of *resource requirements* $R(T_i) = [R_1(T_i), ..., R_s(T_i)]$, where $R_l(T_i) \leq m_l$, $l = 1, 2, ..., s$, denotes the amount of resource R_l needed throughout the execution of T_i. There is also specified a *partial order* $<$ on $\mathcal{T} \cdot T_i < T_j$ means that T_j may only be started after T_i is completed. Our objective is to find a schedu , whenever one exists, such that no task will be late.

This problem has received much attention in recent years and many papers have been devoted to solving particular subproblems [2,3,6,7,9,10,13–16] (see [4] or [11] for a survey). In all of these papers it has been assumed that no additional resources are required during task execution. Lastly, the problem of scheduling to meet deadlines under resource constraints was considered in [5]. Table 1 summarizes the results obtained in the paper mentioned above. (The reader who is unfamiliar with the concept of NP-completeness is referred to [1,8,12] for an introduction.) For all the problems in Table 1 it is assumed that tasks are nonpreemptable, and $r_i = 0$, $i = 1, 2, ..., n$. It follows that the only problem, for which a polynomial in time algorithm so far exists, is the problem of nonpreemptive scheduling unit execution time tasks with empty partial order, on m processors

Table 1
Complexity of scheduling problems to meet deadlines under resource constraints

Problem complexity	m	$<$	s	m_l	$R_l(T_i)$	References
(1) $O(n^2)$	–	$\emptyset$	$s = 1$	–	$R_1(T_i) = 0$ or 1 for all i	[5]
(2) Open	2	$\emptyset$	–	–	–	
(3) NP-complete	2	forest	$s = 1$	–	–	[5]
(4) Open	–	in-tree	$s = 1$	–	$R_1(T_i) = 0$ or 1 for all i	
(5) NP-complete	–	out-tree	$s = 2$	$m_1 = 1$	–	[7]
(6) NP-complete	2	–	$s = 1$	$m_1 = 1$		[5]
(7) NP-complete	3	$\emptyset$	$s = 1$	–	–	[5]

(m arbitrary), when each task enters the system at time $t = 0$ and may require 0 or 1 unit of additional resource. In this paper, we give an algorithm for a more general case, allowing for nonequal ready times. Its complexity is $O(n^2)$.

2. Algorithm

Before describing the algorithm we will give some necessary definitions. A *valid schedule* is such an assignment of processors and additional resources to tasks that

– no task is started prior to its ready time,

– each task once started is executed to completion,

– at every moment not more than m tasks are being executed,

– at every moment the sum of resource requirements of tasks being executed does not exceed the available amounts of resources.

A valid schedule will be called an *optimal* one, if all tasks are completed on time. A scheduling algorithm will be called *optimal* if it finds an optimal schedule, whenever one exists. Task T_i will be called *active* (at moment t) if $r_i \leqslant t$.

In what follows, we describe how to solve the scheduling problem characterized by the following data: n and m arbitrary, partial order empty, $s = 1$, $\mathcal{T}_i = 1$ and $R_1(T_i) = 0$ or $1, i = 1, 2, ..., n$, r_i and d_i arbitrary (integer). Find a nonpreemptive schedule, if any, with no task late. We denote this problem by P.

A scheduling algorithm for solving this problem uses modified deadlines d_i^*, $i = 1, 2, ..., n$, assigned to tasks in the way described by Algorithm 1. In Algorithm 1, for a given t, set A_t is defined as follows: $A_t = \{T_i : (d_i^* = t) \wedge (R_1(T_i) = 1)\}$.

Algorithm 1

Step 1. Order tasks in nondecreasing order of their original deadlines. Initially assign, for all tasks, $d_i^* := d_i$. Set $t := \max_i\{d_i\}$.

Step 2. If $|A_t| \leqslant m_1$, then go to Step 3. From the set A_t, take $|A_t| - m_1$ tasks with the earliest ready times and calculate their modified deadlines according to the formula:

$d_i^* := d_i^* - 1$.

Step 3. Set $t := t - 1$. If $|A_t| = 0$ repeat this step until either $|A_t| > 0$ or $t = 0$. In the former case go to Step 2 in the latter end the algorithm (if $|A_0| \neq 0$ then no optimal schedule exists).

It follows, that modified deadlines may be assigned to tasks in $O(n^2)$ time in a worst case. We now prove two lemmas necessary for developing an optimal scheduling algorithm.

Lemma 1. *A valid schedule meets all the original deadlines if it meets all the modified deadlines.*

Proof. The proof is obvious, because $d_i^* \leqslant d_i$, $i = 1, 2, ..., n$.

Lemma 2. *There exists an optimal schedule only if there exists a valid schedule meeting all the modified deadlines.*

Proof. We have to prove that an existence of an optimal schedule implies the existence of a schedule which meets all modified deadlines. Let Π be an optimal schedule in which some modified deadlines are not respected. Let t be the latest moment at which a modified deadline of some task (say T_i) is not respected. (Thus, T_i is completed at moment t.) It follows from Algorithm 1 that $R_1(T_i) = 1$. Since $d_i^* < d_i$, there exist k T_j's ($k > m_1$), for which $R_1(T_j) = 1$, and $d_j \geqslant t$ and $d_j^* \leqslant t$. From among these tasks, m_1 tasks have their modified deadlines equal to t, and at least one of them (say T_k) is completed prior to t. Since $d_k^* = t$ and $d_i^* < t$, we have $d_i^* < d_k^*$ and $r_i \leqslant r_k$. Thus, exchanging T_i and T_k, we do not cause d_k^* to be exceeded and r_i to be violated.

Such a proceeding may be repeated for other tasks, if any, which fail to meet their modified deadline at moment t. If there is no such task at moment t, we can proceed in a similar way for $t' < t$ as the latest moment at which some task fails to meet its modified deadline; etc.

Finally, we get Π' which is valid, and in which all tasks meet their modified deadlines.

It follows from the above lemmas that we need only to construct an algorithm which finds a schedule, whenever one exists, which meets all modified deadlines. The following Algorithm 2 has the desired property.

Algorithm 2

Step 1. Form the list of uncompleted tasks (include also those which will arrive) in nondecreasing order of d_i^*. Renumber tasks according to this order. Set $t := 0$ and $k := 0$.

Step 2. Assign the first nonassigned active task (say T_i) from the list to the next free processor. If $R_1(T_i) = 1$ set $k := k + 1$. Repeat this step until either $k = m_1$ or all processors are busy. If $k = m_1$ then go to Step 3 else go to Step 4.

Step 3. Assign to free processors the nonassigned active tasks with the earliest deadlines, for which $R_1(T_i) = 0$. Go to Step 5.

Step 4. Remove a task (say T_j) which, from among all assigned tasks with $R_1(T_i) = 0$ has the latest deadline, from the processor to which it was assigned. Assign to this processor the first nonassigned active task on the list (say T_l) with resource requirement $R_1(T_l) = 1$, if for every nonassigned T_i (including task T_j), $i < l$, the following condition is fulfilled [1].

$$t + 1 + \lceil |B_i|/m \rceil \leqslant d_i^* \qquad (1)$$

where B_i denotes the set of nonassigned tasks with modified deadlines $\leqslant d_i^*$.

If this condition is not fulfilled for some i, do not change the assignment and go to Step 5. Set $k := k + 1$. If $k < m_1$ then repeat Step 4.

Step 5. Remove the assigned tasks from the list and set $t := t + 1$ and $k := 0$. If there are any active tasks on the list then go to Step 2. If there are only non-active tasks on the list then set $t := \min_{k \in N}\{r_k\}$, where N denotes the set of indexes of these tasks and go to Step 2.

Check to determine whether the obtained schedule meets all task deadlines.

We now prove the optimality of Algorithm 2.

Theorem 3. *Algorithm 2 is optimal for scheduling problem* P.

Proof. Following our observation, preceding Algorithm 2, we have only to prove that this algorithm finds a schedule, whenever one exists, in which all modified deadlines are respected.

Let Π be a schedule obtained after using Algorithm 2. Let us assume, moreover, that Π' is a schedule which, among all schedules that meet all modified deadlines, maximizes the first time at which Π' disagrees with Π. Let this moment be t. We may assume without loss of generality for Π' (and Π, for which it is already guaranteed by Algorithm 2) and any two active tasks T_i and T_j with equal resource requirements that, if T_i is scheduled earlier than T_j, we have $d_i^* \leqslant d_j^*$ (otherwise T_i and T_j can be safely exchanged). Call this assumption (†). Let, for $k = 0, 1, p_k$ (respectively p'_k) be the set of tasks T_i assigned in Π (respectively Π') at moment t with $R_1(T_i) = k$. Because of Step 4 (and condition (1)) $p_1 \subset p'_1$ is impossible. Now, assume $T_i \in p_1 - p'_1$, i minimal. As Algorithm 2 selects the tasks with resource requirement = 1 ordered with respect to their modified deadlines, this implies together with (†) $p'_1 \subseteq p_1$, and therefore $|p'_1| < m_1$. Now, let $T_j \in p'_0 - p_0$ be such that it has the maximal modified deadline in this set. If $d_i^* \geqslant$ the moment at which T_i is assigned in Π' we may simply exchange T_i and T_j, otherwise condition (1) guarantees that T_i can be moved (in Π') to moment t and T_j to some moment t' with $t < t' \leqslant d_j^*$. In either case, we may conclude $p_1 = p'_1$. But from this and (†) it easily follows that also $p_0 = p'_0$, contradicting the definition of t.

It may be verified that Algorithm 2 may be implemented in at most $O(n^2)$ time. Thus, the complexity of the described scheduling problem is also $O(n^2)$.

Acknowledgement

The author would like to acknowledge helpful comments made by the referee, concerning the first draft of this paper.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, Reading, MA, 1974).

[1] $\lceil x \rceil$ denotes the least integer not less than x.

[2] J. Błażewicz, Scheduling dependent tasks with different arrival times to meet deadlines, in: E. Gelenbe, H. Beilner (eds.) Modelling and Performance Evaluation of Computer Systems (North-Holland, Amsterdam, 1977) 57–65.
[3] J. Błażewicz, Simple algorithms for multiprocessor scheduling to meet deadlines, Information Processing Letters 6 (3) (1977) 162–164.
[4] J. Błażewicz, Deadline scheduling of tasks – a survey, Foundations of Control Engineering, 1 (4) (1977) 203–216.
[5] J. Błażewicz, Scheduling with deadlines and resource constraints, Rep. PR-25/77, Inst. of Control Eng. Tech. Univ. of Poznań, 1977.
[6] P.J. Brucker, Sequencing unit-time jobs with treelike precedence on m machines to minimize maximum lateness, Proc. IX Internat. Symposium on Mathematical Programming, Budapest, 1976.
[7] P. Brucker, M.R. Garey and D.S. Johnson, Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness, Math. Operations Res. 2 (3) (1977) 275–284.
[8] E.G.J. Coffman, (ed.), Computer and Job/Shop Scheduling Theory (J. Wiley and Sons, New York, 1976).
[9] M.R. Garey and D.S. Johnson, Scheduling tasks with nonuniform deadlines on two processors, J. ACM 23 (3) (1976) 461–467.
[10] M.R. Garey and D.S. Johnson, Two-processor scheduling with start-times and deadlines, SIAM J. Comput. 6 (1977) 416–426.
[11] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, Report BW 82/77, Mathematisch Centrum (1977).
[12] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher (eds.), Complexity of Computer Computation (Plenum Press, New York, 1972) 85–104.
[13] J. Labetoulle, Some theorems on real time scheduling, in: E. Gelenbe, R. Mahl (eds.), Comp. Architecture and Networks (North-Holland, Amsterdam, 1974) 285–298.
[14] J. Labetoulle, Real time scheduling in a multiprocessor environment. To appear.
[15] B.J. Lageweg, J.K. Lenstra and A.H.G. Rinnooy Kan, Minimizing maximum lateness on one machine: Computational experience and some applications, Statistica Neerlandica 30 (No. 1) (1976) 25–41.
[16] C.L. Liu and J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20, (1) (1973) 46–61.