ELSEVIER Discrete Applied Mathematics 57 (1995) 229–241

DISCRETE APPLIED MATHEMATICS

Nearly on line scheduling of preemptive independent tasks

Eric Sanlaville[1]

Université Paris VI, Laboratoire LITP, 4 pl. Jussieu, 75252 Paris Cedex 05, France

Received 24 August 1992; revised 27 October 1993

Abstract

We discuss the problem of scheduling preemptive independent tasks, subject to release dates and due dates, on identical processors, so as to minimize the maximum lateness. This problem was solved by a polynomial flow based algorithm, but the major drawback of this approach is its off-line character. We study a priority algorithm, the equivalent of a list scheduling method in the non-preemptive case, in which tasks are ordered according to their due dates. This algorithm is nearly on-line and of low complexity. It builds an optimal schedule when the release dates are equal. In the general case, it provides an absolute performance guarantee. These results hold when the number of available machines is allowed to vary with time in a zigzag way (the number of machines is either K, or $K - 1$).

Keywords: Parallel machines; Preemptive scheduling; Profile scheduling; Polynomial-time algorithms; Performance guarantee

1. Introduction

Let us consider the following problem: a set of preemptive independent tasks, subject to release dates and due dates, are to be scheduled on K parallel machines; the objective is the minimization of the maximum lateness. Several flow-based polynomial-time algorithms have been developed to solve it (see [5, 6]). In this paper, we consider an extension of this problem, for which the number of available processors may vary with time due to, e.g., processor failures or maintenance. The notion of profile scheduling was earlier introduced by Ullman [16]. A study of profile scheduling in the non-preemptive case may be found in the works of Dolev and Warmuth [2,3]. Schmidt [14, 15] proposed a polynomial algorithm to decide whether all due dates can be achieved in the preemptive case, but no extension to solve the minimization problem was provided, and besides, this algorithm is *off-line*. An algorithm is

[1] The work of the author was partially supported by a post-doctorate grant from INRIA, France.

SSDI 0166-218X(94)00105-7

on-line if it only needs to know the enabled tasks and the available machines at a time t to choose the assignment of the tasks at t. It is *nearly on-line* if it needs in addition the time of the next release date (definition of [6]) and, in profile scheduling, the time of the next profile change.

We consider a nearly on-line algorithm called SL (Smallest Laxity first) that at any time schedules enabled tasks according to their laxities, that is, the difference between their due date and their remaining processing time. It was named *priority algorithm* by Lawler [7], who studied it when precedence relations are allowed and the profile is constant.

In Section 2, the problem is more precisely defined and some notations are provided. We present in Section 3, the priority algorithms, the preemptive counterpart of list methods, and show how SL works in the special case of independent tasks. It is shown in Section 4 that when the profile is constant, SL provides an absolute upper bound on the optimal lateness. The result holds with a slight modification of the bound if the profile is increasing zigzag, that is, the number of available machines can decrease by at most one at a time and, between two decrements, there must be at least one increasing. Such profiles were introduced by Dolev and Warmuth [2] in the case of non-preemptive scheduling.

2. Preliminaries

An instance of *IPPS* (independent preemptive profile scheduling) is denoted by (V, p, r, d, M), and specified as follows.

Let $V = \{1, \ldots, n\}$ be the set of tasks to be scheduled. The processing time, release date and due date of task i are positive rational numbers, respectively, denoted by p_i, r_i, and d_i. $\mathscr{S}$ and $\mathscr{P}$ denote the sum and the maximum value of the processing times. There are $K \geqslant 1$ parallel and identical processors. The set of processors available to tasks varies in time, due to, e.g., failures of the processors, maintenance periods, or execution of higher-priority tasks. The availability of the processors is referred to as the profile, and is specified by the sequence $M = \{a_n, m_n\}_{n=1}^{\infty}$, where rationals $0 = a_1 < a_2 < \cdots < a_n < \cdots$ are the time epochs when the profile is changed, and m_n, $n \geqslant 1$, is the number of processors available during the time interval $[a_n, a_{n+1})$. The breadth of the profile is the maximum number of processors available, that is K. The additional notations $m(t)$ and $M(a,b)$ will be used to denote the number of available machines at time t and the total amount of processing resource available during time interval $[a,b]$, respectively. Without loss of generality, we assume that $m_n \geqslant 1$ for all $n \geqslant 1$.

In addition to constant profiles, the following two classes of profiles will be considered in the paper.

Zigzag profiles: The number of available processors is either K or $K - 1$.

Increasing zigzag profiles: The number of available processors can decrease by at most one at any time. Between successive decrements, there must be at least one increase. That is, $\forall r \in \mathbb{N}$ and $\forall n \geqslant r$, $m_n \geqslant m_r - 1$.

The performance of a schedule is measured by its maximum lateness. For the special case when there are no release dates (respectively no due dates) the corresponding symbol r (respectively d) is omitted. When the profile function is constant, M is replaced by K.

Remark. The hypothesis that all quantities are rational, always verified in practice, can be removed at the price of more complicated proofs (see [8] for technical details).

MacNaughton's algorithm produces a schedule that minimizes the makespan on a constant profile, when there are no release dates and no due dates. It is based on the lower bound $l_{\mathrm{MN}} = \max(\mathscr{P}, \mathscr{S}/K)$, which is the makespan we get by successively allocating the tasks to the first machine, the second machine,... up to time l_{MN}. This result is used in the definition of priority algorithms in Section 3.

The flow based algorithm of [6], that solves the problem on a constant profile, can be extended to variable profiles. It is still polynomial in n and $\mathscr{P}$ if the number of profile changes during any time interval I is polynomial in the length of I. The complexity is $\mathrm{O}(n^3 \min(n^2, \log n + \log \mathscr{P}))$ in the constant case, and $\mathrm{O}(n^3 \mathscr{P}^3 \cdot (\log n + \log \mathscr{P}))$ if the number of profile changes during I is linear in I (we shall implicitly keep this hypothesis in further complexity computations). This algorithm is of course completely off-line (see [12, 13]).

3. Priority schedules

Due to their easy implementation and low complexity, list scheduling algorithms have been widely studied in the framework of non-preemptive scheduling. These algorithms have their counterpart in the preemptive case. One of the most interesting examples is the algorithm by Muntz and Coffman [10], which minimizes the makespan for a set of tasks with precedence constraints in the form of an intree. A description of the way priority schedules are built can be found in [7, 8]. In this paper, we only consider the case where independent tasks are ordered according to the smallest laxity first rule: at any time, enabled tasks are ordered by non-decreasing laxity $b_i(t) = d_i - p_i(t)$, where $p_i(t)$ is the residual duration of task i at time t. Fig. 1 shows a schedule built from such a priority list in the case of a zigzag profile with breadth 3.

A non-preemptive schedule is obtained from any priority list by choosing, each time processors are available, an enabled task with highest priority. A preemptive priority schedule executes, *at each time*, the enabled tasks with highest priority. Let us illustrate the way this works by the example of Fig. 1. At time $t = 0$, three tasks are enabled but 1 and 2 have smallest laxity and are processed by the two available machines until time 1, which is the next time a change occurs among the task priorities, since the three enabled tasks now have the same laxity of value 2. They share the two available processors (we say they are executed at speed $\frac{2}{3}$) until time 2 when a new machine is available. The next event occurs at time $2 + \frac{1}{3}$, when the

i	r_i	p_i	d_i
1	0	2	3
2	0	2	3
3	0	1	3
4	1	1	5
5	2	1	4
6	3	1	4
7	3	1	4

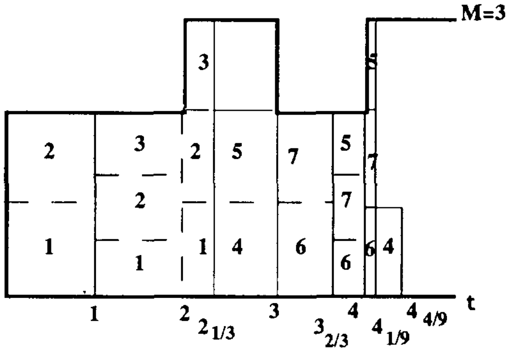

Fig. 1. Example of dynamic list schedulng.

three tasks are simultaneously completed. This process is continued until all tasks are processed. The maximum lateness is $L_5 = L_6 = L_7 = \frac{1}{9}$.

Clearly, the assignment of the tasks may change each time one of the following events occurs:
1. a task completes or a new one becomes enabled,
2. the relative priority of two enabled tasks changes,
3. a profile jump occurs.

In the interval between two successive events, the set of enabled tasks is partitioned into classes according to the task priorities (smallest laxity here). Machines are assigned to the tasks of the first class. If the number of available machines is less than the number of tasks of this class, they are *shared.* Otherwise, each task is assigned a distinct machine, and the remaining machines, if any, are assigned in the same way to the tasks of the next class.

It has been proved by Muntz and Coffman [10] that any processor sharing may be transformed using MacNaughton's algorithm into an equivalent feasible schedule whose makespan is not larger. This process is depicted in Fig. 2. Note that some task (here task 1) might finish earlier in the resulting schedule, but no task finishes later in the equivalent feasible schedule. The amount of task effectively processed during an

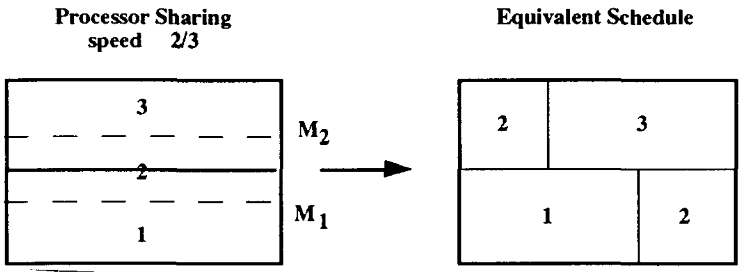

Fig. 2. Producing a feasible schedule on two machines.

interval of length l is l times v, where v is the execution speed of the task in the processor sharing schedule. In the example all three tasks have speed $\frac{2}{3}$.

The construction process of Lawler [7] is slightly different but the interested reader will be convinced the resulting schedules have equal performances. In what follows, the generalized (admitting processor sharing) and equivalent feasible versions of a priority schedule will be used indifferently for the needs of our proofs. Notice that SL produces a unique generalized schedule; however, several feasible schedules of same performance may exist.

Constructing the generalized schedule and applying McNaughton's algorithm between two events is $O(n)$. It is easily proved that the number of type 1 or 2 events is bounded by $2n$ for a set of independent tasks. Hence the overall complexity of the algorithm is $O(n^2\mathscr{P})$. It is worth noting that in the general case the resulting schedule is not optimal. In the example of Fig. 1, there is a schedule that meets all due dates: it schedules tasks 4 and 5 at speed 1 during [2, 3].

However, *SL provides an optimal schedule for zigzag profiles and no release dates*. This can be proved (see [12, 13]) by showing that SL builds a schedule respecting the conditions of Horn [5]. The use of a quite different argument entails that the priority algorithm ordering tasks by their decreasing processing times minimizes the makespan when there are no due dates and the profile is arbitrary (see [12, 13]). This is a particular application of SL for which equal fictitious due dates are added.

These optimality results are completed by the absolute guarantees we shall now provide. The interest of such bounds is immediate after the remark by Sahni in [11] that, for constant profile, release dates and due dates, no nearly on-line algorithm always providing optimal schedules ever exists. The occurrence of tasks with very high priority is, from a scheduling point of view, equivalent to a decreasing of the profile. Roughly speaking, a sudden decreasing of the number of available machines favors scheduling policies which try to reduce the width of the precedence graph but neglect to minimize its height, whereas a sudden increase is the number of available machines favors policies which try to reduce the height and keep a large number of enabled tasks.

4. Absolute upper bounds for SL on constant profiles and increasing zigzag profiles

These upper bounds are analogous to the one found by Carlier [1] in the non-preemptive case.

Denote by L_{SL} the maximum lateness of SL schedule S_{SL}, and let L^* be the optimal maximum lateness. C_i is the completion time of task i in S_{SL} and L_i its lateness. Let i_0 be a task such that $L_{i_0} = L_{\mathrm{SL}}$, and d_{i_0} is minimum. This task plays a pivotal role in the proof below. The notations C_{i_0}, r_{i_0}, and d_{i_0} are further simplified as C_0, r_0, and d_0.

4.1. Problem simplifications

The two lemmas below allow us to restrict our study to task systems such that any task has a laxity no larger than d_0 throughout the whole schedule. Their quite fastidious proofs are contained in the appendix.

Lemma 4.1. *Let V' be the task set obtained from V by removing all tasks whose initial laxity is larger than d_0. For any SL schedule S'_{SL} of V' on any profile M, we have* ,$L_{\mathrm{SL}} = L'_{\mathrm{SL}}$.

Lemma 4.2. *Suppose V is such that any task i has initial laxity smaller than d_0. The IPPS instance $(V, \hat{p}, r, \hat{d}, M)$ defined by $\hat{p}_i = p_i - \max(0, d_i - d_0)$ and $\hat{d}_i = \min(d_i, d_0)$, satisfies*

$$L_{\mathrm{SL}} - L^* \leqslant \hat{L}_{\mathrm{SL}} - \hat{L}^*.$$

4.2. Absolute performance guarantees for SL

The theorem below summarizes the results for two kinds of profiles, constant profiles and increasing zigzag profiles. The proof is given for constant profiles, with additions for increasing profiles when necessary.

Theorem 1. *Consider the maximum lateness L_{SL} provided by any SL schedule for an instance (V, p, r, d, M). If M is a constant profile of breadth K,*

$$L_{\mathrm{SL}} \leqslant L^* + \frac{K-1}{K} \cdot \mathscr{P}.$$

If M is an increasing zigzag profile,

$$L_{\mathrm{SL}} \leqslant L^* + \mathscr{P}.$$

Proof. From Lemma 4.2, we can restrict our study to the instances in which, for any task i, $d_i \leqslant d_0$.

Let b be the smallest real number in interval $[r_0, C_0)$ such that i_0 is continuously executed at speed 1 during $[b, C_0)$ (by definition of i_0, and because of the above

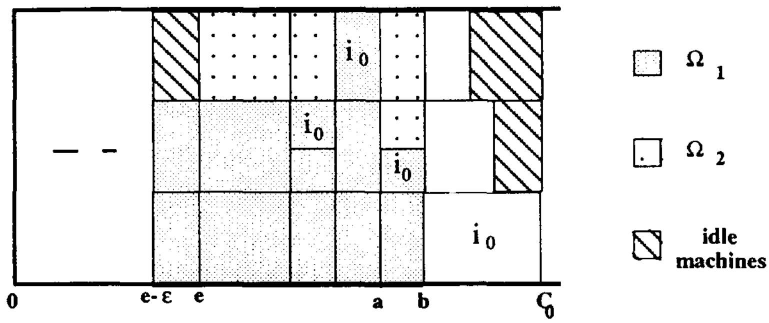

Fig. 3. Example for the first case.

restriction, $C_{\max} = C_0$ in S). The interval $[e,f)$ is the largest interval included in $[r_0, C_0)$, that contains b and during which all machines are in use. If $e = b$, it means that some $\varepsilon \in \mathbb{R}^*$ exists, such that one machine remains idle during $[b - \varepsilon, b)$, or $b = 0$. In both cases we get $b = r_0$, and S is an optimal schedule. Hence in what follows it is assumed that $e < b$.

Two cases might occur, depending on whether or not a task of smaller priority than i_0 is executed during $[e, b)$.

Case 1: Each task partially executed during $[e, b)$ has a laxity less than or equal to $d_0 - (C_0 - b)$.

Task i_0 satisfies this condition, because $d_0 - (C_0 - b)$ is an upper bound on the laxity of i_0 during $[e, b)$, as i_0 is continuously executed at speed 1 during $[b, C_0)$.

Let us consider the set of partially executed tasks during $[e, b)$. Some are not completed at time b. For the sake of simplicity, we apply the following transformation to these tasks, including i_0. Consider the task system obtained by replacing each task i by a fictitious task whose duration is equal to the amount of processing of task i executed during $[e, b)$ in S, and whose due date is equal to the laxity of i in S at b. The optimal value L^* cannot increase for the new task system, and L_{SL} keeps the same value. By a harmless abuse of notation, we identify S with the schedule of this new task system that behaves exactly like S during $[0, b)$. Let Ω denote the set of modified tasks. The subset Ω_1 contains the tasks having a release date strictly smaller than e, and symmetrically $\Omega_2 = \Omega \backslash \Omega_1$ contains the tasks having a release date larger than or equal to e. Fig. 3 illustrates these definitions. Note that i_0 may belong to Ω_1 if $r_0 < e$, and to Ω_2 otherwise. In Fig. 3, it is assumed that $r_0 < e$.

If the profile is constant, the two following properties are true:

$$\sum_{\omega \in \Omega} p_\omega \geqslant K \cdot (b - e), \tag{1}$$

$$|\Omega_1| \leqslant K - 1. \tag{2}$$

Indeed, tasks of Ω use all available machines during $[e, b)$. Furthermore, tasks of Ω_1 are available in $[e - \varepsilon, e)$ for ε small enough, and by the assumption that $b > e$, $K - 1$ machines at most are in use, each executing one task. From (1) and (2), we get

$$\sum_{\omega \in \Omega_2} p_\omega \geqslant K \cdot (b - e) - (K - 1) \cdot \mathscr{P}. \tag{3}$$

On the other hand, a lower bound on the value of L^* may be found by computing the earliest possible completion time of all tasks of Ω_2, minus the maximum value of their due dates. This extends Carlier's reduction of [1] in the non-preemptive case. So we get

$$L^* \geqslant e + \sum_{\omega \in \Omega_2} p_\omega / K - [d_0 - (C_0 - b)]. \tag{4}$$

From (3) and (4) the following inequality is proved:

$$L^* \geqslant e + (b - e) - \frac{K - 1}{K} \cdot \mathscr{P} + (C_0 - b) - d_0,$$

hence,

$$L^* \geqslant L_{\mathrm{SL}} - \frac{K - 1}{K} \cdot \mathscr{P},$$

because $L_{\mathrm{SL}} = C_0 - d_0$.

If the profile is increasing zigzag, an analogous reasoning will prove the result. We have the following two properties:

$$\sum_{\omega \in \Omega} p_\omega \geqslant K \cdot M(e, b), \tag{5}$$

$$|\Omega_1| \leqslant m(e - \varepsilon) \leqslant \min_{t \in [e, b)} m(t), \tag{6}$$

where (6) comes from the fact that M is increasing zigzag (remember $M(e, b)$ denotes the total amount of processing resource available during time interval $[e, b)$). Hence we get

$$\sum_{\omega \in \Omega_2} p_\omega \geqslant M(e, b) - \mathscr{P} \cdot \min_{t \in [e, b)} m(t). \tag{7}$$

Let b' be the time such that $M(e, b') = \sum_{\omega \in \Omega_2} p_\omega$. Computing the same minimization on Ω_2 as in the constant case, we get $L^* \geqslant e + (b' - e) + (C_0 - b) - d_0$ from (7), $b' \geqslant b - \mathscr{P}$, hence $L^* \geqslant L_{\mathrm{SL}} - \mathscr{P}$.

Case 2: Let $\bar{\imath}$ be a task partially executed during some interval $[u, v) \subseteq [e, b)$ and whose laxity is larger than $d_0 - (C_0 - b)$ in that interval (see Fig. 4).

If the profile is constant, let a be the earliest time such that i_0 is always executed at speed less than 1 during $[a, b)$. Note that v is less than or equal to a because some task of lower priority than i_0 cannot be executed if i_0 is executed at a fractional or null speed. We shall further assume that $[u, v)$ is the last such interval for $\bar{\imath}$ before a. At most

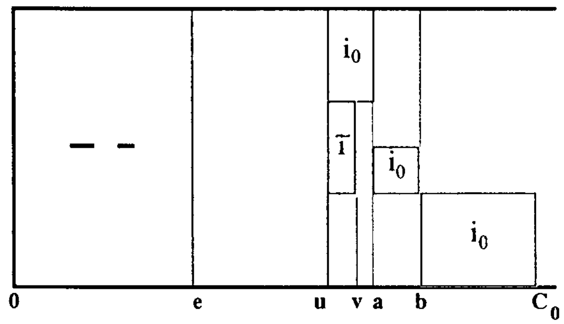

Fig. 4. Second case: example of task $\tilde{\imath}$.

$(K-1)$ enabled tasks have strictly higher priority than $\tilde{\imath}$ during $[u, v)$, and so, there are at most $(K-1)$ tasks with laxity less than or equal to $d_0-(C_0-b)$; otherwise $\tilde{\imath}$ could not be executed during $[v-\varepsilon, v)$, $\varepsilon \in \mathbb{R}_+^*$. The same reasoning as for the first case still works. The same transformation is performed for tasks partially executed during time interval $[v, b)$. The set of modified tasks is still denoted Ω. Subset Ω_1 contains the tasks of Ω enabled before v, whereas Ω_2 contains the other tasks. The following equations still hold:

$$\sum_{\omega \in \Omega} p_\Omega \geqslant K \cdot (b-v), \tag{8}$$

$$|\Omega_1| \leqslant K-1 \tag{9}$$

and applying the same reduction to the tasks of Ω_2 entails:

$$\begin{aligned} L^* &\geqslant v + \textstyle\sum_{w \in \Omega_2} p_w/K + C_0 - b - d_0 \\ &\geqslant v + (b-v) - (K-1)/K \cdot \mathscr{P} + (C_0-b) - d_0, \\ L^* &\geqslant L_{\mathrm{SL}} - (K-1)/K \cdot \mathscr{P}. \end{aligned}$$

If the profile is zigzag increasing, let $\tilde{m}$ be the minimum number of machines available during time interval $[u, v)$. At most $\tilde{m}-1$ enabled tasks have higher priority than $\tilde{\imath}$. Studying again the sets Ω, Ω_1, Ω_2, defined on time interval $[v, b)$, and considering time b' such that $M(v, b') = M(v, b) - \mathscr{P} \cdot (\tilde{m}-1)$, we obtain $L^* \geqslant e + (b'-e) + (C_0-b) - d_0$ as again, $b' \geqslant b - \mathscr{P}$, we get $L^* \geqslant L_{\mathrm{SL}} - \mathscr{P}$ and the theorem is proved. □

4.3. The bounds are asymptotically reached

We now present a family of instances for which the bounds are tight when the number of tasks grows towards infinity. The task set V^k is defined as $A_0 \cup B_0 \cup A_1 \cup B_1 \cup \cdots \cup A_k$, where:

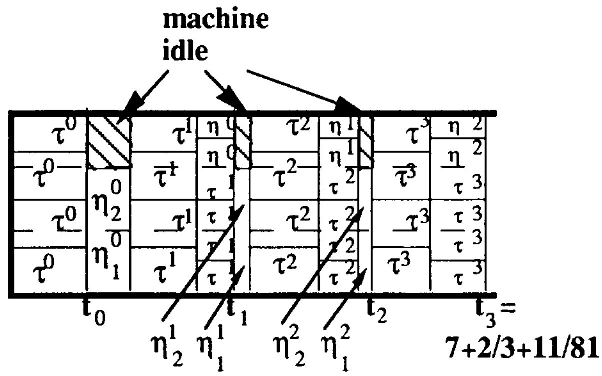

Fig. 5. Example of SL schedule, with $k = 3$ and $K = 3$.

- $A_j, j \in \{0, \ldots, k\}$, contains $(K + 1)$ tasks with release date $2j$, due date $(k + j + 1)$ and duration 1.
- $B_j, j \in \{0, \ldots, k - 1\}$, contains $(K - 1)$ tasks with release date $2j$, due date $(k + j + 2)$ and duration 1.

Consider first the constant profile of breadth K. By processing the tasks in $A_j \cup B_j$, $j \in \{0, \ldots, k - 1\}$, in $[2j, 2j + 2)$ by McNaughton's algorithm, and tasks in A_k in $[2k, 2k + (K + 1)/K)$, we get an optimal schedule S satisfying $L_{\mathrm{SL}} = 1/K = L^*$.

Now consider the schedule obtained by SL. It satisfies for $j < k$:

- any task of A_j is completed before a task of B_j is processed,
- any task of A_j is completed before a task of A_{j+1} is processed.

So one machine remains idle in some intervals. The maximum lateness is reached by some tasks in A_j, $j \leqslant k$. A simple computation yields

$$\forall j \leqslant k, \forall i \in A_j, C_i - d_i = \left[2j + 2 - \left(\frac{K - 1}{K} \right)^{(j+1)} \right] - (k + j - 1)$$

for the processor sharing version of S_{SL}. As it is true for at least one task of A_j in the feasible schedule obtained from the transformation of Section 3, and as it is maximum for $j = k$, we finally get

$$L_{\mathrm{SL}} = 1 - \left(\frac{K - 1}{K} \right)^{(k+1)}.$$

So when k tends toward infinity, we get,

$$L_{\mathrm{SL}} - L^* = \frac{K - 1}{K}.$$

The SL schedule is depicted in Fig. 5, where task τ^j, $j \leqslant k$, has due date $(k + j + 1)$ and task η^j, $j < k$, has due date $(k + j + 2)$.

Now consider, for each task set V^k, the profile M^k for which K machines are available during time interval $[0, T^k)$, and only $K-1$ machines are available afterwards. The value T^k is computed as $2k + 2 - (K-1)/K^{(k+1)} - (K-1)/K$. The same kind of computation as above permits to conclude that the maximum laxity for any SL schedule tends toward $1 + 1/K$, whereas L^* still tends toward $1/K$.

Appendix A: Proof of Lemmas 4.1 and 4.2

Let V' be the task set obtained by removing all tasks whose initial laxity is larger than d_0. Consider the schedule S_{SL} of V' on an arbitrary profile K. Let i be any task of V'. Its laxity $b_i(t)$ increases from $d_i - p_i \leqslant d_0$ to d_i. We call the critical date of task i in S_{SL} the time τ_i for which $b(\tau_i) = d_0$ (if $d_i < d_0$, τ_i is set to C_i).

Proof of Lemma 4.1. Let us consider the schedule S_{SL}. Assume the number of intervals $[t_j, t_{j+1})$, $j \geqslant 0$, defined by the events of type 1, 2 or 3, is k. We suppose $C_0 = t_{k'}$, $k' \leqslant k$. We claim that any task i is identically executed in $[0, \tau_i)$ in S_{SL} and in S'_{SL}. It suffices to prove the following property:

$\forall i \in V', \forall u \in \{1, \ldots, k'\}$, i is identically executed in $[r_i, \min(\tau_i, t_u))$ by both schedules. The property is proved by induction on u.

$u = 1$: the enabled tasks of V' have higher priority than the enabled tasks of $V \backslash V'$ at time t_0. By construction of S_{SL}, this remains true until $t = t_1$. Hence the tasks of V' are identically processed during $[t_0 = 0, t_1)$ by both schedules.

Assume the property is true for a given $u < k'$: at time t_u the enabled tasks of V' have the same residual duration in the two schedules, and the same arguments as for $u = 1$ can be followed. The assignment of tasks of V' at time t_u is identical. A task $i \in V'$, with $\tau_i > t_u$, is executed at the same speed by both schedules until time $\min(\tau_i, t_{u+1})$, because it has higher priority than any task whose laxity is larger than d_0.

As the property is true for $i = i_0$ and $u = k'$, and as $\tau_{i_0} = C_0$, the result follows. □

We are looking for an upper bound of $L_{\text{SL}} - L^*$. Hence, the study can be restricted to task systems respecting the conditions of the above lemma.

Now we consider the following transformation of S_{SL}: $\hat{S}_{\text{SL}}$ is a schedule of the IPPS instance $(V, \hat{p}, r, \hat{d}, K)$ obtained by executing a task i within $[r_i, \tau_i)$ identically as i is executed in S_{SL}. Hence, by definition of the problem, instance τ_i is the completion time of i in $\hat{S}_{\text{SL}}$. Its maximum lateness is $\hat{L}_{\text{SL}}$.

Lemma A.1.

$$\hat{L}_{\text{SL}} = L_{\text{SL}}.$$

Proof. $\hat{L}_{\text{SL}} \leqslant L_{\text{SL}}$: Consider one task i with $d_i > d_0$. As $b_i(\tau_i) = d_0$, $d_i - d_0 = p_i(\tau_i) \leqslant C_i - \tau_i$, and $\tau_i - d_0 \leqslant C_i - d_i \leqslant L_{\text{SL}}$. The result follows.

$\hat{L}_{\mathrm{SL}} \geqslant L_{\mathrm{SL}}$: It suffices to consider task i_0. It is executed identically in both schedules. □

The proof of the following lemma is identical to the first part of the proof above.

Lemma A.2. *Let $\hat{L}^*$ denote the optimal maximum lateness for $(V, \hat{p}, r, \hat{d}, K)$. Then*

$$\hat{L}^* \leqslant L^*.$$

Now consider the schedule obtained by direct applying of SL policy for $(V, \hat{p}, r, \hat{d}, K)$, denoted by S''_{SL}.

Lemma A.3. *S''_{SL} and $\hat{S}_{\mathrm{SL}}$ are identical.*

Proof. The proof is by induction as for Lemma 4.1.

Both schedules are identical during $[0, t_1)$: at $t = 0$, the task assignment is identical in both schedules. Let τ be the first time one task j finishes. It means that in S_{SL}, j has laxity d_0, so $\tau = \tau_j$. At τ_j, the only tasks assigned in $\hat{S}_{\mathrm{SL}}$ are the tasks scheduled by S_{SL} with higher priority than j at $t = 0$. This implies they are executed at speed 1 in S_{SL}, and hence in both $\hat{S}_{\mathrm{SL}}$ and S''_{SL}. Consider S''_{SL}: the completions of j and of other tasks of same priority leave some machines idle. But no task i is enabled, or it would have been scheduled before by $\hat{S}_{\mathrm{SL}}$, because the laxity of i for S_{SL} would have been less than d_0, and $\tau_j < t_1$. Hence, the task assignment at τ_j remains identical. The same argument is used when another task completes, until time t_1.

Assume $\hat{S}_{\mathrm{SL}}$ and S''_{SL} are identical during $[t_0, t_u)$, $\forall u < k$. The proof is the same as for the first interval. By induction hypothesis, enabled tasks at t_u are the same and have same priority for both schedules. Hence we are done. □

Corollary A.4 (Lemma 4.2). *Denote by $\hat{L}_{\mathrm{SL}}$ the SL schedule of $(V, \hat{p}, r, \hat{d}, m)$. Then*

$$L_{\mathrm{SL}} - L^* \leqslant \hat{L}_{\mathrm{SL}} - \hat{L}^*.$$

Proof. The result is immediate from Lemmas A.1–A.3. □

Acknowledgements

I am very grateful to the two referees for their very careful reading and their pertinent comments, specially on the presentation of the main result of the paper.

References

[1] J. Carlier, Scheduling jobs with release dates and tails on identical machines to minimize makespan, European J. Oper. Res. 29 (1987) 298–306.

[2] D. Dolev and M.K. Warmuth, Scheduling flat graphs, SIAM J. Comput. 14 (1985) 638–657.
[3] D. Dolev and M.K. Warmuth, Profile scheduling of opposing forests and level orders, SIAM J. Algebraic Discrete Methods 6 (1985) 665–687.
[4] T.F. Gonzales and D.B. Johnson, A new algorithm for preemptive scheduling of trees, J. ACM 27 (1980) 287–312.
[5] W.A. Horn, Some simple scheduling algorithms, Naval Res. Logist. Quart. 21 (1974) 177–185.
[6] J. Labetoulle, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Preemptive scheduling of uniform machines subject to release dates, Technical Paper BW 99/79, Mathematisch Centrum Amsterdam (1979).
[7] E.L. Lawler, Preemptive scheduling of precedence-constrained jobs on parallel machines, in: M.A.H. Dempster, ed., Deterministic and Stochastic Scheduling (Reidel, Dordrecht, 1982) 101–123.
[8] Z. Liu and E. Sanlaville, Preemptive scheduling with variable profile, precedence constraints and due dates, Technical Paper IBP/MASI No 92.5, Université Paris VI (1992); Discrete Appl. Math., to appear.
[9] R. McNaughton, Scheduling with deadlines and loss functions, Management Sci. 6 (1959) 1–12.
[10] R.R. Muntz and E.G. Coffman, Preemptive scheduling of real-time tasks on multiprocessor systems, J. ACM 17 (1970) 325–338.
[11] S. Sahni, Preemptive scheduling with due dates, Technical Report 77-4, Department of Computer Science, University of Minnesota, Minneapolis, MN (1977).
[12] E. Sanlaville, Conception et analyse d'algorithmes de liste en ordonnancement préemptif, Ph.D. Dissertation, Université Paris VI (1992).
[13] E. Sanlaville, Scheduling preemptive independent tasks on a variable profile, Technical Paper IBP/MASI No 92.4, Université Paris VI (1992).
[14] G. Schmidt, Scheduling on semi-identical processors, Z. Oper. Res. A28 (1984) 153–162.
[15] G. Schmidt, Scheduling independent tasks with deadlines on semi-identical processors, J. Oper. Res. Soc. 39 (1988) 271–277.
[16] J.D. Ullman, NP-complete scheduling problems, J. Comput. System Sci. 10 (1975) 384–393.