# 4

# Profile Scheduling by List Algorithms

**Zhen LIU, INRIA, Centre Sophia Antipolis Eric SANLAVILLE, Université Pierre et Marie Curie**

**Abstract**: The notion of profile scheduling was first introduced by Ullman in 1975 in the complexity analysis of deterministic scheduling algorithms. In such a model, the number of processors available to a set of tasks may vary in time. Since the last decade, this model has been used to deal with systems subject to processor failures, multiprogrammed systems, or dynamically reconfigured systems. The aim of this paper is to overview optimal polynomial solutions for scheduling a set of partially ordered tasks in these systems. Particular attentions are given to a class of algorithms referred to as list scheduling algorithms. The objective of the scheduling problem is to minimize either the maximum lateness or the makespan. Results on preemptive and nonpreemptive deterministic scheduling, and on preemptive stochastic scheduling, are presented.

**Keywords:** Deterministic Scheduling, Stochastic Scheduling, Profile Scheduling, List Schedule, Priority Schedule, Precedence Constraints, Lateness, Makespan.

## 4.1 Introduction

Consider the problem of scheduling a set of partially ordered tasks represented by a directed acyclic graph, referred to as *task graph*, where vertices represent tasks and arcs represent precedence relations. Tasks are executed, subject to precedence constraints, on a set of parallel identical processors. The number of available processors, referred to as *profile*, may vary in time. Task and processor assignments must be nonredundant,

Scheduling Theory and Its Applications. P. Chretienne, E. G. Coffman, J. K. Lenstra, Z. Liu (Eds.)

i.e., at any time, a task can be assigned to at most one processor, and a processor can execute at most one task. Each task has a due date. The objective is to minimize the maximum lateness or, when due dates are not taken into consideration, the makespan.

The notion of profile scheduling was first introduced by Ullman [31] and later used by Garey et al. [15] in the complexity analysis of deterministic scheduling algorithms. In this paper we use the notion of profile to deal with systems subject to processor failures, multiprogrammed systems, or dynamically reconfigured systems. In such cases, the number of processors available to a set of tasks may vary in time.

The problem of minimizing the maximum lateness and the makespan is in general NP-hard. We are interested in simple on-line or nearly on-line algorithms which yield optimal solutions under specific assumptions. Results on three problems will be presented here, with each result accounting for different task characteristics. The reader is referred to the survey paper by Lawler et al. [20] for results on optimal scheduling under a constant profile, i.e., when the number of available processors is constant. The first results on optimal polynomial solutions for profile scheduling problems are due to Dolev and Warmuth [9, 10, 11].

The paper is organized as follows. Section 2 is devoted to notation. Section 3 deals with scheduling of nonpreemptive Unit Execution Time (UET) tasks; optimality results concerning list schedules on a variable profile are surveyed. Section 4 is concerned with scheduling of preemptive Real Execution Time (RET) tasks. We exhibit a tight relation between optimal list schedules and optimal priority schedules, which are counterparts of list schedules for preemptive scheduling. Section 5 focuses on tasks whose running times are independent and identically distributed random variables with a common exponential distribution. We prove the optimality of some list policies which stochastically minimize the makespan in several subproblems.

## 4.2 Notation

A task graph $G = (V, E)$ is a directed acyclic graph, where $V = \{1, 2, \ldots, |V|\}$ is the set of vertices representing the tasks, $E \subset V \times V$ is the set of arcs representing the precedence constraints: $(i, j) \in E$ if and only if task $i$ must be completed before task $j$ can start. Denote by $p_i$ and $d_i$ the respective processing time and due date of task $i \in V$.

Let $p(i)$ and $s(i)$ be the respective sets of immediate predecessors and successors of $i \in V$, i.e.,

$$p(i) = \{j : (j, i) \in E\}, \qquad s(i) = \{j : (i, j) \in E\}.$$

Let $S(i)$ be the set of (not necessarily immediate) successors of $i \in V$, i.e.,

$$\forall i : \quad \text{if} \quad s(i) = \emptyset \quad \text{then} \quad S(i) = \emptyset \quad \text{else} \quad S(i) = s(i) \bigcup \left( \bigcup_{j \in s(i)} S(j) \right).$$

A task without a predecessor (successor) is called an *initial* (*final*) task. Denote by $l_i$ the level of $i$, i.e., the number of arcs in a longest path from task $i$ to some final task. Denote by $h_i$ the height of $i$, i.e., the sum of the processing times of the tasks in a longest path from task $i$ to some final task, with final vertex included.

In the sequel, we will consider different classes of precedence graphs. The three most interesting classes are the following:

**interval order** $G \in \mathcal{G}_{io}$: Each vertex $i$ corresponds to an interval $b_i$ of the real line such that $(i, j) \in E$ if and only if $x \in b_i$ and $y \in b_j$ imply $x < y$. These graphs have the following characteristic property:

$$\forall i, j \in V, \quad \text{either } S(i) \subseteq S(j), \quad \text{or } S(j) \subseteq S(i).$$

**in-forest** $G \in \mathcal{G}_{if}$: Each vertex has at most one immediate successor: $|s(i)| \leq 1$, $i \in V$. A vertex $i \in V$ is called a leaf of in-forest $G$ if $p(i) = \emptyset$. A vertex $i \in V$ is called a root of in-forest $G$ if $s(i) = \emptyset$.

**out-forest** $G \in \mathcal{G}_{of}$: Each vertex has at most one immediate predecessor: $|p(i)| \leq 1$, $i \in V$. A vertex $i \in V$ is called a leaf of out-forest $G$ if $s(i) = \emptyset$. A vertex $i \in V$ is called a root of out-forest $G$ if $p(i) = \emptyset$.

There are $K \geq 1$ parallel and identical processors. The set of processors available to tasks varies in time, due to, e.g., failures of the processors or the 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 $0 = a_1 < a_2, < \ldots < a_n < \ldots$ are the time epochs when the profile is changed, and $m_n$, $n \geq 1$, is the number of processors available during time interval $[a_n, a_{n+1})$. Without loss of generality, we assume that $m_n \geq 1$ for all $n \geq 1$ . We will assume that the profile is not changed infinitely often during any finite time interval: for all $x \in I\!R^+$, there is some finite $n \geq 1$ such that $a_n > x$.

The following three classes of profiles will often be referred to in the paper.

**zigzag profiles** $M \in \mathcal{M}_z$: the number of available processors is either $K$ or $K - 1$.

**increasing zigzag profiles** $M \in \mathcal{M}_{iz}$: 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, $\forall j$ and $\forall n \geq j$, $m_n \geq m_j - 1$.

**decreasing zigzag profiles** $M \in \mathcal{M}_{dz}$: a symmetrical definition applies; that is, $\forall j$ and $\forall n \geq j$, $m_n \leq m_j + 1$. Such a profile is illustrated in Figure 4.1.

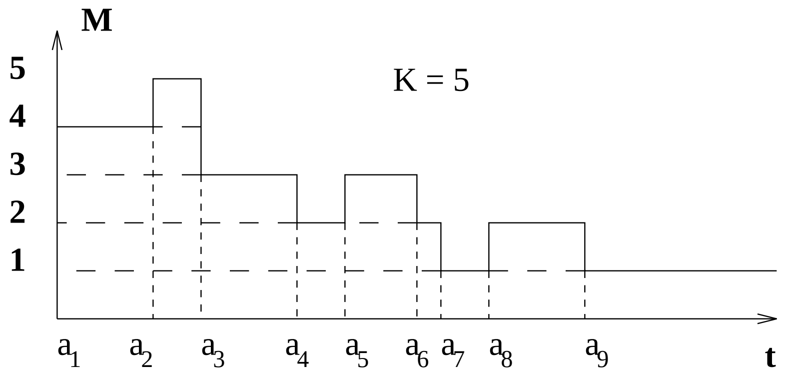


**Figure 4.1** Decreasing zigzag profile

A scheduling algorithm (or policy) decides when an enabled task, i.e. an unassigned, unfinished task all of whose predecessors have finished, should be assigned to one of the available processors. A schedule is feasible if assignments are nonredundant and the constraints relative to the precedence relation and to the variable profile are respected. Scheduling can be either preemptive, i.e., the execution of a task can be stopped and later resumed on any processor without penalty, or nonpreemptive: once begun, the execution of a task continues on the same processor until its completion.

Let $S$ be an arbitrary feasible schedule of task graph $G$ under profile $M$. Let $C_i(S)$ be the completion time of task $i$ under $S$. The lateness of task $i$ is defined as $L_i(S) = C_i(S) - d_i$. The maximum lateness of schedule $S$ is denoted by $L_S(G, M) = \max_{i \in V} L_i(S)$.

When all due dates are set to zero, the maximum lateness becomes the makespan. Denote by $C_S(G, M) = \max_{i \in V} C_i(S)$ the makespan of $(G, M)$ obtained by schedule $S$.

We extend the classical notation scheme of Graham et al. [17] for scheduling problems to the case of variable profiles. We use $P(t)$ (resp. $Q(t)$, $R(t)$) to denote machine environment with identical (resp. uniform, unrelated) parallel processors whose number varies in time. For example, $P(t) \mid p_i = 1, prec \mid C_{\max}$ denotes the nonpreemptive scheduling for makespan minimization of UET tasks subject to precedence constraints on identical parallel processors with variable profile.

## 4.3 Nonpreemptive Profile Scheduling of UET Tasks

In the framework of nonpreemptive profile scheduling, we will consider UET tasks and integer profiles (those in which profiles change only at integer time epochs). The problems under consideration can be denoted by $P(t) \mid p_i = 1, prec \mid C_{\max}$ or $P(t) \mid p_i = 1, prec \mid L_{\max}$.

List algorithms are often used in nonpreemptive scheduling. These algorithms put the enabled tasks in an ordered list. Each time a processor becomes available, the task at the head of the list is assigned to that processor. The list can be dynamically updated. The Highest Level First (HLF) and the Earliest Due Date (EDD) algorithms are well known examples. The reader is referred to [6] for properties of list algorithms.

### 4.3.1 Complexity Issues

In [31], Ullman exhibited a polynomial reduction from 3-SAT to $P(t) \mid p_i = 1, prec \mid C_{\max}$, and then showed that $P \mid p_i = 1, prec \mid C_{\max}$ had the same complexity (It suffices to “fill” the unavailable processors with dummy tasks). This is immediately generalized to the lateness minimization $L_{\max}$.

Garey et al. [15] also used profile scheduling for the study of opposing forests (union of in-forest and out-forest). They proved NP-completeness for $P(t)\,decreasing \mid p_i = 1, intree \mid C_{\max}$ (the decreasing profile refers to the case where the number of available processors is decreasing in time), and consequently for $P \mid p_i = 1, opposing\ forest \mid C_{\max}$. For such precedence graphs, however, polynomial algorithms have been found for bounded profiles, i.e., $K$ is fixed (see discussions below).

### 4.3.2 Minimization of the Makespan by List Scheduling

Consider first the minimization of the makespan. We focus on the Highest Level First (HLF) algorithm: tasks are ordered by decreasing level. Note that HLF schedules differ only in the way ties are broken.

*Forests.* When the task graph is a forest, Hu [18] proved that HLF yields an optimal schedule when the task graph is an in-forest and the profile is constant. Bruno [2] extended this result to out-forests.

For variable profile, Dolev and Warmuth [9, 10, 11] showed that HLF remains optimal in some special cases of profiles. For that, they first showed the so-called *elite theorem*. Define the height of a connected component of $G$ to be the highest level of its vertices. Let the components be ordered by decreasing height, and $h(k)$ be the height of the $k$-th component. Suppose profile $M$ is bounded by $K$. The *median* of $(G, M)$ is defined as $\mu = h(K) + 1$. Consider

- H(G): set of components of $G$ whose heights are strictly greater than $\mu$ (High Part);
- E(G): set of initial tasks of $H(G)$ whose levels are strictly greater than $\mu$ (Elite);
- L(G): graph obtained from $G$ by removing the components of $H(G)$ (Low Part).

When the graph contains less than $K$ components, the median has value 0 and $G = H(G)$. These definitions are illustrated in Figure 4.2 where $K = 3$.

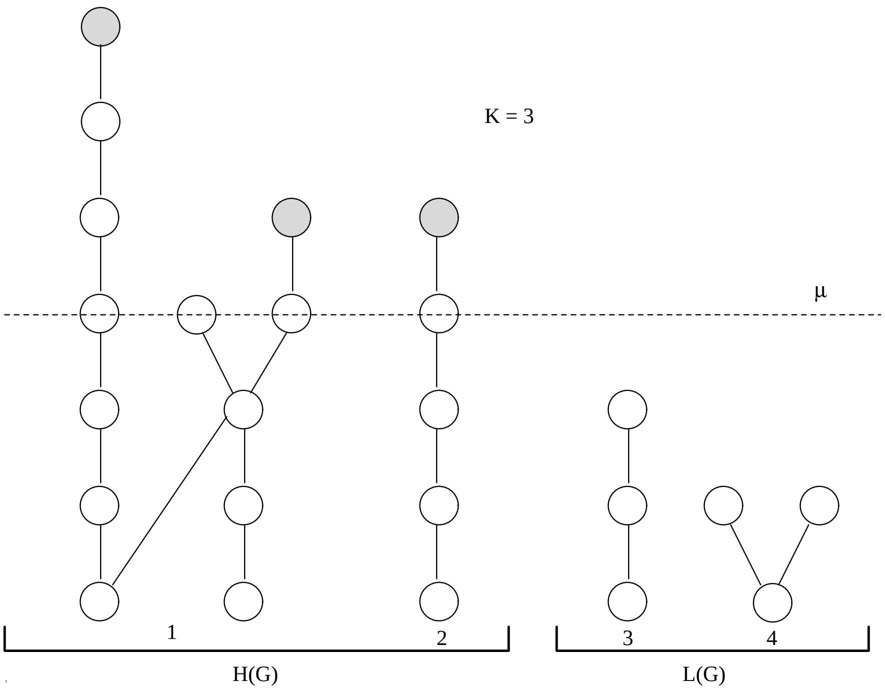


**Figure 4.2** Elite of a task graph

We now state the elite theorem of [10]:

**Theorem 4.3.1** *Consider a task graph $G$ and an integer profile $M$.*

1. *If $|E(G)| > m_1$, then there is an optimal schedule such that $m_1$ tasks of $E(G)$ are executed during the first time unit.*
2. *If $|E(G)| \leq m_1$, then for each set $E$ of $m_1$ tasks of maximum levels, there exists an optimal schedule executing the tasks of $E$ during the first time unit.*
3. *If $E(G) = \emptyset$, then any HLF schedule is optimal.*

As a consequence, we obtain

**Corollary 4.3.1** *HLF minimizes the makespan*

- *when the task graph is an out-forest and the profile is decreasing zigzag;*
- *or when the task graph is an in-forest and the profile is increasing zigzag.*

In the first case, the elite contains less than $K$ tasks, and $m_1 \geq K - 1$. Part 2 of elite theorem states that there exists an optimal schedule that coincides with any $HLF$ schedule during the first time unit. The same reasoning is used during each time unit until the graph is empty. To prove the second case, it suffices to reverse the graph and the profile.

For bounded profiles, Dolev and Warmuth [11] provided a backward dynamic programming method whose time complexity is $O(n^{K-1} \cdot \log n)$ for scheduling an in-tree of size $n$ on an arbitrary profile bounded by $K$. This was based on two observations. First, if an optimal schedule is known for $H(G)$, then according to the *merge theorem* of [9], an optimal schedule for $G$ may be obtained in linear time with respect to $|L(G)|$. Second, If we consider all subgraphs of some in-forest with at most $K-1$ components, which could be obtained during the execution under any schedule, these subgraphs may be partitioned into at most $n^{K-1}$ equivalence classes. Here, two graphs $G = (V, E)$ and $G' = (V', E')$ are equivalent if there is a bijection $\phi$ between $V$ and $V'$ such that the set of successors of vertex $\phi(v)$ is $\{\phi(v_1), \ldots, \phi(v_l)\}$, where $\{v_1, \ldots, v_l\}$ is the set of successors of $v$. The dynamic programming method decides whether a schedule of duration $D$ exists, building schedules backward from $D$-th time unit. The $\log n$ factor comes from a bisection search to get the right value for $D$.

This method is then applied to out-forests under bounded profiles by reversing the directions of the precedence constraints and the profile. Further, such a dynamic programming algorithm is used to solve the scheduling problem of opposing forests as mentioned in Section 4.3.1.

In the special case of chains, i.e. $\mathcal{G}_{ch} \stackrel{\text{def}}{=} \mathcal{G}_{if} \bigcap \mathcal{G}_{of}$, Liu and Sanlaville [21] showed

**Theorem 4.3.2** *If $G$ is a union of chains and $M$ an arbitrary integer profile, then any HLF minimizes the makespan.*

This was proved by an interchange argument, i.e., from an optimal non-HLF schedule, one can interchange the assignment decisions for two subchains so that the number of non-HLF decisions is decreased by at least one. Iterating this procedure for at most $n^2$ times yields an optimal HLF schedule. Note that all HLF schedules have the same makespan in that case.

*Interval order graphs.* Papadimitriou and Yannakakis [27] have shown that task graphs with interval order structure have the following property: for any two vertices in the graph, their sets of (all) successors are comparable by inclusion relation:

one set is included into the other. This property is essential in establishing the optimality of Most Successors First (MSF) schedules (forming a subset of HLF schedules) which respect the order of inclusion of the sets of successors. This result was obtained by Papadimitriou and Yannakakis [27] for constant profile and extended to variable profile in Sanlaville [29].

**Theorem 4.3.3** *If $G$ is a task graph with an interval order structure and $M$ an arbitrary integer profile, then any MSF schedule minimizes the makespan.*

This result can again be proved using an interchange argument. Consider an optimal non-MSF schedule $\rho$ for graph $G$. Let $\tau$ be the first time when $\rho$ schedules some vertex $v$ instead of vertex $u$ if MSF rule is applied. Note that by definition, all the predecessors of $u$ and $v$ have finished execution by time $\tau$. Let $\pi$ be the schedule obtained from $\rho$ by interchanging the execution times for $u$ and $v$. Since $G$ has an interval order structure, $S(v) \subset S(u)$, it is easy to see that $\pi$ satisfies the precedence constraints of $G$. Moreover, the number of non-MSF decisions in $\pi$ is decreased by one. Iterating this procedure for at most $n^2$ times yields an optimal MSF schedule.

*Arbitrary graphs with profiles bounded by 2.* Coffman and Graham [7] proved that a subset of HLF schedules, referred to as Lexicographic Order Schedules (LOS) in this paper, minimizes the makespan when the profile is constant and equal to two. LOS is based on a static list of tasks defined by the lexicographic order as follows. Let there be $f$ final tasks. Assign labels $1, \cdots, f$ to these final tasks in an arbitrary way. Suppose now that $k \geq f$ tasks have already been labeled by $1, 2, \ldots k$. Consider all the tasks whose successors are all labeled. Assign label $k+1$ to the task such that the decreasing sequence of the labels of its immediate successors is lexicographically minimal (ties are broken arbitrarily). Coffman and Graham [7] showed that LOS schedules minimize the makespan of an arbitrary graph in a 2-processor system. The main idea of their proof is to cut the Gantt chart of the LOS schedule into several segments such that each segment is composed of two sets of tasks $E_i$ and $F_i$, where $F_i$ is a singleton, and that all tasks of $E_i$ are predecessors of $E_{i+1}$. The optimality of LOS is extended in Sanlaville [29] to variable profiles with at most 2 processors by a minor modification of the proof of [7], which consists in assigning fictitious tasks to unavailable processors.

**Theorem 4.3.4** *Any LOS schedule is optimal for makespan minimization of any task graph $G$ under any profile $M$ bounded by 2.*

*Summary and discussion on makespan minimization.* These results are summarized in Figure 4.3, where "constant" and "arbitrary" stand for constant and arbitrary profiles. A gray area means that this particular subproblem is NP-hard for arbitrary $K$. Since constant profiles are special cases of (increasing and decreasing) zigzag profiles which are in turn special cases of arbitrary profiles, the implication relation for subproblems is clear.

Note that when the number of processors $K$ is fixed, the complexity remains an open problem, except for the case $K = 2$ for which LOS algorithm of Coffman and Graham [7] provides optimal schedules. However, for any fixed $K \geq 3$, the scheduling problem is still of unknown complexity despite extensive research.

Bartush et al [1] presented an interesting work on this problem, viz. scheduling

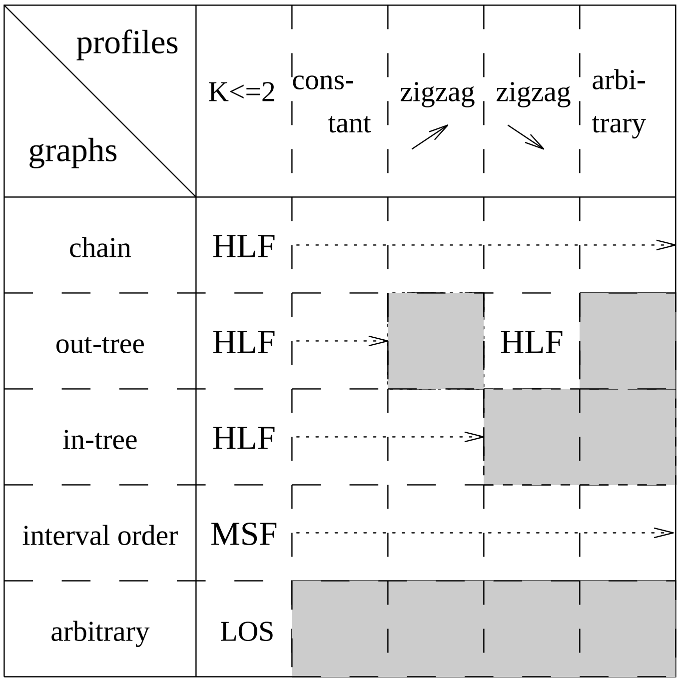


**Figure 4.3** Results on nonpreemptive profile scheduling of UET tasks ($C_{\max}$)

with fixed number of processors. The authors provided a general 2-phase solution scheme. In the first phase, they tried to generalize the consistency notion of [4, 14]. They compute a so-called "test choice" for the problem instance under consideration. This is derived from bounds on the execution periods of some special tasks. In the second phase, they construct an optimal schedule in $O(n^K)$ time. This is based on sophisticated dominancy criteria and on the study of special partially ordered sets. The time complexity of the first phase is simple for the 2-processor case, and is polynomial for all known polynomial cases (trees, interval-order graphs, etc...). However, its complexity remains unknown in the general case. It seems that no further research results concerning this approach have been reported since 1989.

### 4.3.3 Minimization of the Maximum Lateness by List Scheduling

Consider now the minimization of the maximum lateness. We analyze the Earliest Due Date (EDD) algorithm. The optimality of the EDD schedules will rely on a two-step method:

1. modify the due dates so that they satisfy some consistency relation,
2. apply EDD to the modified due dates.

Roughly speaking, the consistency between due dates requires that whenever the due date of a task is met in some schedule, the due dates of its successors can be met too, provided there are enough processors. Such a consistency relation is not verified if, for instance, the due date of some of its predecessors.

Initially, for any constant profile, Brucker, Garey and Johnson [4] proved that if $G$ is an in-forest, EDD yields optimal schedules provided the modified due dates $d'_i$ are defined as follows:

$$d'_i = \begin{cases} d_i, & s(i) = \emptyset; \\ \min(d_i, d'_{s(i)} - 1), & s(i) \neq \emptyset, \end{cases} \tag{4.1}$$

where, by a harmless abuse of notation, $s(i)$ denotes the unique successor of task $i$. Note that such a modification has the purpose of obtaining consistent due dates. It is shown by Liu and Sanlaville [21] that the optimality of EDD still holds for increasing zigzag profiles:

**Theorem 4.3.5** *For any in-forest $G \in \mathcal{G}_{if}$ and any increasing zigzag profile $M \in \mathcal{M}_{iz}$, EDD schedule defined on modified due dates minimizes the maximum lateness.*

The proof proceeds by first establishing that EDD defined on the modified due dates meets all the original due dates if and only if such a feasible schedule exists, and then by using a standard argument to show the optimality of EDD schedules.

In a slightly more complicated way, Garey and Johnson [13, 14] showed the existence of modification schemes for the due dates, such that EDD applied to these due dates yields optimal schedules on two processors, even when release dates are associated with the tasks. In the latter case, the algorithm is not on-line. These results can be extended to variable profiles (see [29]). However, even without release dates, the algorithms are then off-line because the computation of the modified due dates depends not only on the release dates but also on the profile.

For profile of width $K \geq 3$ and general task graphs, there is no simple way to design some modification scheme so that EDD applied to the modified due dates becomes optimal. The general problem is NP-hard. In fact, even for out-forests with constant profiles, the minimization of maximum lateness is NP-hard, as was shown in [4]. Unlike the makespan minimization, the scheduling of out-forests cannot be achieved by analyzing the "reverse" problem. Indeed, reversing problem $P(t)\ dec.\ zig. \mid p_i = 1, out\,tree \mid L_{\max}$ yields problem $P(t)\, inc.\, zig. \mid p_i = 1, r_i, in\,tree \mid C_{\max}$, where release dates $r_i$ are added to the tasks.

## 4.4 Preemptive Profile Scheduling

We now consider preemptive scheduling. The task processing times and profile change epochs are arbitrary real numbers. We are interested in optimal priority algorithms. In what follows, we first describe such algorithms, and then present a tight relation between optimal nonpreemptive list algorithms and optimal preemptive priority algorithms. Finally, we present simple optimal priority algorithms for specific problems.

### 4.4.1 Priority Scheduling Algorithms

Parallel to list algorithms, (dynamic) priority algorithms are used in preemptive scheduling. At any time, enabled tasks are assigned to available processors according to a priority list which can change in time and can depend on the partial schedule already constructed. A general description is given below (see Muntz and Coffman [25] and Lawler [19]):

- At any time $t$, enabled tasks are ordered according to their priorities, thus forming subsets $V_1, \ldots, V_k$, where all tasks of $V_j$ have the same priority and greater priority than tasks in $V_{j+1}$.
- Suppose that tasks in $V_1, \ldots, V_{r-1}$, $r \leq k$, are assigned. Let $\tilde{m}_r(t)$ be the number of remaining free processors. If $\tilde{m}_r(t) \geq |V_r|$, then one processor is assigned to each of the tasks in $V_r$, and the algorithm deals with the next subset. Otherwise, the $\tilde{m}_r(t)$ processors are shared by the tasks of $V_r$ so that each task in $V_r$ is executed at speed $v_r = \tilde{m}_r(t)/|V_r|$.
- This assignment remains unchanged until one of the following events occurs:

  1. A task completes (or a new task is enabled, when release dates are considered);
  2. The priority order of tasks is changed;
  3. The profile changes.

  At such moments the processor assignment is re-computed.

In the above scheme, the processor sharing can be achieved by McNaughton's wrap-around algorithm (cf. [23]) which is linear in the number of tasks scheduled in each time interval. An example is illustrated in Figure 4.4 where three tasks are executed at speed 2/3 on two processors during a unit length interval.

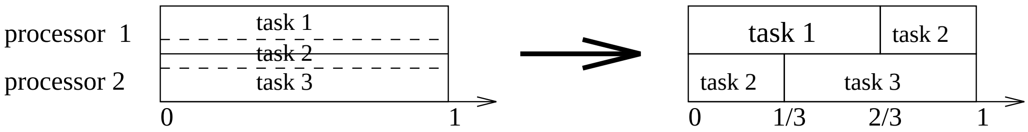


**Figure 4.4** An example of processor sharing with McNaughton's algorithm

Note that other processor sharing schemes may be used. However, they will generate the same latenesses of the tasks provided the corresponding task processing speeds are the same in these processor sharing schemes. Therefore, we will not make any difference between them.

Denote by $p_i^S(t)$ the remaining processing requirement of task $i$ at time $t$ in schedule $S$. Define the laxity of task $i$ at time $t$ in this schedule as $b_i^S(t) = d_i - p_i^S(t)$. We will consider SLF (Smallest Laxity First) algorithms. Figure 4.5 shows an SLF schedule for a set of independent tasks whose characteristics are indicated in Table 4.1.

| $i$ | $r_i$ | $p_i$ | $d_i$ |
|---|---|---|---|
| 1 | 0 | 3 | 5 |
| 2 | 0 | 2 | 2 |
| 3 | 2 | 2 | 5 |
| 4 | 2 | 1 | 3 |

**Table 4.1** A set of independent tasks

The schedule is optimal, which is not true in general when the maximum lateness is under consideration.

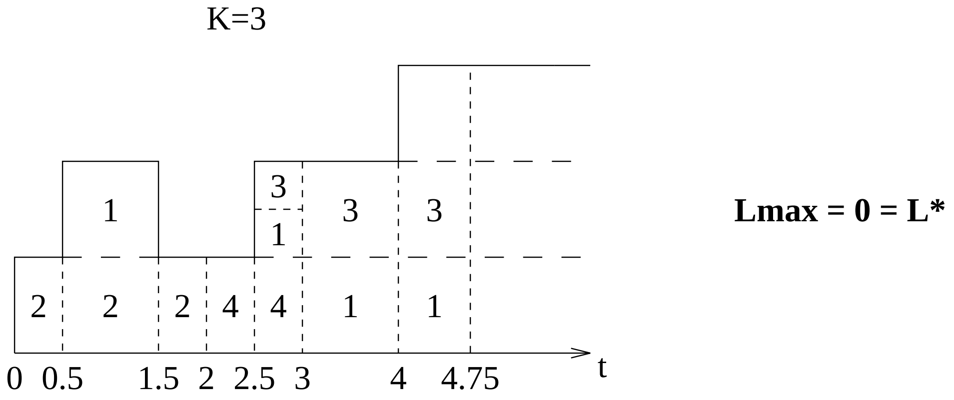


**Figure 4.5** An example of SLF schedule

Priority schedules are also used for the minimization of makespan. Define the length of the remaining longest path of task $i$ at time $t$ in preemptive schedule $S$ as $r_i^S(t) = h_i + p_i^S(t)$. In an LRP (Longest Remaining Path first) schedule, tasks are ordered by decreasing length of the remaining longest path. This corresponds to the HLF rule when tasks have unit execution times.

### 4.4.2 Relation between Optimal Nonpreemptive List Algorithms and Preemptive Priority Algorithms

It was shown in Liu and Sanlaville [21] that there is a tight relation between the conditions under which nonpreemptive list algorithms (EDD, HLF) are optimal and those under which preemptive priority algorithms (SLF, LRP) are optimal. In order to state these results, we need the following notions of closure.

A class $\mathcal{G}$ of graphs is said to be closed under expansion if the following property is true for any graph $G = (V, E) \in \mathcal{G}$: For any vertex $i \in V$, if $G'$ is the graph obtained from $G$ by replacing vertex $i$ with a chain of two vertices $i_1$ and $i_2$ such that:

$$p(i_1) = p(i), \quad s(i_1) = \{i_2\}, \quad \text{and} \quad p(i_2) = \{i_1\}, \quad s(i_2) = s(i),$$

then $G'$ still belongs to the class $\mathcal{G}$.

A class $\mathcal{M}$ of profiles is said to be closed under translation if for any profile $M = \{a_r, m_r\}_{r=1}^{\infty}$ in $\mathcal{M}$, all profiles $M' = \{a'_r, m_r\}_{r=1}^{\infty}$ belong to $\mathcal{M}$, provided $\{a'_r\}_{r=1}^{\infty}$ is an increasing sequence of real numbers.

**Theorem 4.4.1** *Let $\mathcal{M}$ be a class of profiles which is closed under translation and $\mathcal{G}$ a class of graphs which is closed under expansion. If for any integer profile $M \in \mathcal{M}$ and for any $G \in \mathcal{G}$ with UET tasks and integer due dates, there exists an EDD schedule minimizing the maximum lateness of $G$ within the class of nonpreemptive policies, then for any $M \in \mathcal{M}$ and any $G \in \mathcal{G}$, the SLF schedule minimizes the maximum lateness of $G$ within the class of preemptive schedules.*

The proof proceeds in two steps. In the first step, we prove the result for the case where the pair $(G, M)$ has commensurable timing, i.e., the task processing times and due dates, and the profile change times are mutually commensurable. Real numbers $x_1, \cdots, x_r \in I\!R$ are said to be mutually commensurable if there exist $w \in I\!R$ and $r$ integers $\alpha_1, \cdots, \alpha_r$ such that $x_i = \alpha_i w$ for all $i = 1, \cdots, r$. In the second step, we extend the result to the general case with arbitrary real timing.

The scheme of the proof in the first step is similar to that of Muntz and Coffman [25]. Roughly speaking, we show that when graph $G$ is sufficiently expanded, *i)* an optimal preemptive solution for a pair $(G, M)$ may be approached arbitrarily closely by considering optimal nonpreemptive schedules, and, *ii)* nonpreemptive EDD schedules coincide with the preemptive SLF schedule for $(G, M)$. Putting these two points together yields the desired result. Note that in an expansion, if vertex $i$ is split into two vertices $i_1$ and $i_2$, then their processing times and due dates are defined as follows:

$$p_{i_1} = p_{i_2} = p_i/2, \quad \text{and} \quad d_{i_2} = d_i, \;\; d_{i_1} = d_i - p_{i_2}.$$

In the second step, we show that when $(G, M)$ has real timings, the absolute difference between the maximum lateness of SLF schedule and the optimal one is bounded by an arbitrarily small constant. This implies that SLF schedule does yield an optimal solution.

If the complements of the task heights are taken as due dates, i.e., $d_i = -h_i$ for all $i \in V$, then the EDD (resp. SLF) rule coincides with the HLF (resp. LRP) rule. It can also be shown that in such a case the maximum lateness coincides with the makespan [21]. Therefore,

**Theorem 4.4.2** *Let $\mathcal{M}$ be a class of profiles which is closed under translation and $\mathcal{G}$ a class of graphs which is closed under expansion. If for any integer profile $M \in \mathcal{M}$ and for any $G \in \mathcal{G}$ with UET tasks, there exists an HLF schedule minimizing the makespan of $G$ within the class of nonpreemptive policies, then for any $M \in \mathcal{M}$ and any $G \in \mathcal{G}$, the LRP schedule minimizes the makespan of $G$ within the class of preemptive schedules.*

Note that the above results actually hold in a more general case where the task executions are subject to release dates.

In the remainder of this section, we apply these results together with the results of previous section concerning optimal nonpreemptive scheduling in order to obtain optimal preemptive schedules.

### 4.4.3 Applications

We first consider the maximum lateness of in-forests. For a given in-forest $G \in \mathcal{G}_{if}$ with processing times $p_1, \cdots, p_n$ and due dates $d_1, \cdots, d_n$, we define an in-forest $G' \in \mathcal{G}_{if}$ such that $G'$ has the same set of tasks, the same precedence constraints and the same processing times. The due dates in $G'$ are modified as in (4.1). It can be shown [21] that such a modification on the due dates does not change the maximum lateness of any feasible schedule. It then follows from Theorems 4.3.5 and 4.4.1 that

**Corollary 4.4.1** *If $G \in \mathcal{G}_{if}$ is an in-forest, and $M \in \mathcal{M}_{iz}$ is an increasing zigzag profile, then the SLF schedule defined on the modified due dates minimizes the maximum lateness within the class of preemptive schedules.*

Note that this result extends Theorem 7.3 of Lawler [19] to the case of increasing zigzag variable profile. It is possible to apply Theorem 4.4.1 to the case of arbitrary task graph and constant profile with two processors. In such a case, a new proof of Theorem 8.3 of Lawler [19] may be obtained for the case of identical processors (see [29]).

Consider now the makespan minimization problem. For the simplest case of the task graphs: the chains, it follows from Theorems 4.3.2 and 4.4.2 that

**Corollary 4.4.2** *For any graph consisting of chains and for any profile, the LRP schedule is an optimal preemptive schedule for makespan minimization.*

Note that in the preemptive case, scheduling problems for a union of disjoint chains and for a set of independent tasks are equivalent.

In the case of forests, as a consequence of Corollary 4.3.1 and Theorem 4.4.2, we obtain

**Corollary 4.4.3** *If $G$ is an in-forest and $M$ is an increasing zigzag profile, or if $G$ is an out-forest and $M$ is a decreasing zigzag profile, then the LRP schedule minimizes the makespan within the class of preemptive schedules.*

Observe that this result extends a result of Muntz and Coffman [25] to the zigzag variable profiles.

For an arbitrary task graph, since LOS schedule belongs to the class of HLF schedules, Theorems 4.3.4 and 4.4.2 allow us to conclude that

**Corollary 4.4.4** *LRP is an optimal preemptive schedule for makespan minimization of any task graph $G$ under any profile $M$ bounded by* 2.

This last result extends a result of Muntz and Coffman [24] to variable profiles.

Note that in order to apply Theorems 4.4.1 and 4.4.2, the class of task graphs under consideration should be closed under expansion. Thus, we cannot apply Theorems 4.3.3 and 4.4.2 to obtain the optimality of LRP for graphs with interval order structure, as this class of graphs does not fulfill the condition.

## 4.5 Stochastic Profile Scheduling

### 4.5.1 Problem Description

In this section, we consider the problem of stochastic scheduling under variable profile. We assume that the task processing times are independent and identically distributed random variables having a common exponential distribution. These processing times are independent of the profile $M = \{a_n, m_n\}_{n=1}^{\infty}$ which is a sequence of random vectors.

We assume that the scheduler has no information on the samples of the (remaining) processing times of the tasks. At any time $t$, $a_n \leq t < a_{n+1}$, the scheduler may not have any information on the truncated sequence $\{a_l, m_l\}_{l=n+1}^{\infty}$. In other words, the scheduler may not know either the future time epochs when the profile changes or the number of available processors at any future time. Within such a framework, dynamic preemptive scheduling is necessary.

We are interested in the stochastic minimization of makespan. A policy $\pi_*$ is said to stochastically minimize the makespan of $(G, M)$ within the above described class of policies if for any policy $\pi$ in that class, the makespan of $\pi_*$ is stochastically smaller than that of $\pi$, where a random variable $X \in I\!R$ is said to be stochastically smaller than a random variable $Y \in I\!R$, if for all $x \in I\!R$, $P[X \le x] \ge P[Y \le x]$.

### 4.5.2 Optimal Algorithms for Constant Profiles

When the task graph is an in-forest, and the profile is a constant 2, Chandy and Reynolds [5] proved that the HLF policy minimizes the expected makespan. Bruno [3] subsequently showed that HLF stochastically minimizes the makespan. Pinedo and Weiss [28] extended this last result to the case where tasks at different levels may have different expected task running times. Frostig [12] further generalized the result of Pinedo and Weiss to include increasing likelihood ratio distributions for the task running times. These results do not hold for systems with three processors, see counterexamples in [5]. However, Papadimitriou and Tsitsiklis [26] proved that for any arbitrarily fixed number of processors, HLF is asymptotically optimal as the number of tasks tends to infinity, provided the task processing times have a common exponential distribution.

Coffman and Liu [8] investigated the stochastic scheduling of out-forests on identical parallel processors with constant profile. For the uniform out-forests where all the subtrees are ordered by an embedding relation (see definition below), they showed that an intuitive priority scheduling policy induced by the embedding relation, referred to as the Largest Tree First (LTF) policy in this paper, stochastically minimizes the makespan when there are two processors. If in addition, the out-forests satisfy a uniform root-embedding constraint, then the greedy policy stochastically minimizes the makespan for an arbitrary number of processors.

### 4.5.3 Optimal Algorithms for Variable Profiles

Stochastic profile scheduling was first investigated by Liu and Sanlaville [22]. They considered three kinds of task graphs: interval-order graphs, in-forests and out-forests. The results we are going to present in the remainder of this section are due to [22] and were actually obtained in a more general framework: uniform processors, where the processors may have different speeds.

*Interval order graphs.* As in the deterministic UET case, MSF (Most Successor First) is optimal when the task graph has an interval-order structure.

**Theorem 4.5.1** *For any interval-order graph $G \in \mathcal{G}_{io}$ and any profile $M$, MSF stochastically minimizes the makespan of $G$.*

The proof uses uniformization technique, i.e., we can consider a coupled processing model where all processors $1, \cdots, K$, whenever they are available, are continually executing tasks. When a completion occurs, and no task was assigned to some processor, it corresponds to the completion of a fictitious task on this processor. When a task is assigned to a processor, it is assigned a running time equal to the remainder of the running time already underway at that processor. Owing to the memoryless property of exponential distributions, we can see that this coupled model is equivalent in law to the initial one.

Let $G = (V, E)$ be an interval-order graph, and $T(G) = \{T_1, T_2, \cdots, T_g\}$ be a partition of $V$ obtained by the equivalence relation on the sets of successors: for all $1 \le i \le g$, $u, v \in T_i$ if and only if $S(u) = S(v)$. The sets $T_1, \cdots, T_g$ are labeled in such a way that for all $1 \le i < j \le g$, $u \in T_i$ and $v \in T_j$ imply $S(u) \supset S(v)$. We define a majorization relation: Let $G^1 = (V^1, E^1)$ and $G^2 = (V^2, E^2)$ be two subgraphs of $G$ obtained by successively deleting vertices of $G$ having no predecessor in $G$ or in the previously obtained subgraphs. It is easy to see that $G^1$ and $G^2$ are in $\mathcal{G}_{io}$. Let $T_i^j = T_i \cap V^j$, $j = 1, 2$, $i = 1, \cdots, g$. Graph $G^1$ is said to be majorized by $G^2$, referred to as $G^1 \prec_s G^2$, if and only if

$$\forall i,\ 1 \le i \le g: \qquad \sum_{k=1}^{i} |T_k^1| \le \sum_{k=1}^{i} |T_k^2|,$$

Using now the uniformization technique and the above notion of majorization, one proves the following properties:

- Let $G^1 = (V^1, E^1)$ and $G^2 = (V^2, E^2)$ be two subgraphs of $G \in \mathcal{G}_{io}$ obtained by successively deleting vertices of $G$ having no predecessor in $G$ or in the previously obtained subgraphs. If $G^1 \prec_s G^2$, then under MSF policy, the makespan of $G^1$ is stochastically smaller than that of $G^2$.
- Let $G \in \mathcal{G}_{io}$ be a task graph. Let $\pi$ be a policy which follows the MSF rule all the time except at the first decision epoch. Then, the makespan of $G$ under MSF is stochastically smaller than that under $\pi$.

This last property together with a backward induction allow us to conclude Theorem 4.5.1.

*In-forests.* When the task graph is an in-forest, we have

**Theorem 4.5.2** *For any in-forest $G \in \mathcal{G}_{if}$ and any profile $M$ bounded by 2, HLF stochastically minimizes the makespan of $G$.*

The scheme of the proof is similar. However, we have to use another majorization relation, referred to as "flatter than" in [5]. Let $G^1 = (V^1, E^1)$ and $G^2 = (V^2, E^2)$ be two in-forests. Forest $G^1$ is said to be flatter than $G^2$, denoted by $G^1 \prec_f G^2$, if and only if

$$\forall i,\ i \ge 0: \qquad \sum_{k \ge i} N_k(G^1) \le \sum_{k \ge i} N_k(G^2),$$

where $N_k(G)$ denotes the number of vertices at level $k$ of graph $G$.

*Out-forests.* Suppose now that the task graph is an out-forest. Even for a profile bounded by two, examples may easily be found for which the HLF policy is not optimal (even in term of expected makespan), see [8]. Instead of HLF, the greedy policy LTF introduced in [8] turns out to be optimal in a subclass of out-forests.

Let $G = (V, E) \in \mathcal{G}_{of}$ be an out-forest. Vertex $v \in V$ and all its successors form a subtree of $G$, denoted by $T_G(v)$ or simply $T(v)$ when there is no ambiguity. We denote by $|T(v)|$ the size of $T(v)$, i.e. its number of vertices.

The Largest Tree First (LTF) policy is defined as follows: at any decision epoch, LTF assigns the task $v$ whose subtree $T(v)$ is the largest among all subtrees of the enabled tasks to an available processor. In general, policy LTF is not optimal within the class of out-forests $\mathcal{G}_{of}$. Counterexamples were provided in [8]. However, within the classes of uniform and r-uniform out-forests (introduced in [8]), a policy is optimal if and only if it is LTF.

Let $T_1, T_2 \in \mathcal{G}_{of}$ be two out-trees. Out-tree $T_1$ is said to embed out-tree $T_2$, or $T_2$ is embedded in $T_1$, denoted by $T_1 \succ_e T_2$ or $T_2 \prec_e T_1$, if $T_2$ is isomorphic to a subgraph of $T_1$. Formally, $T_1$ embeds $T_2$ if there exists an injective function $f$ from $T_2$ into $T_1$ such that $\forall u, v \in T_2$, $v \in s(u)$ implies $f(v) \in s(f(u))$. Function $f$ is called an embedding function.

Let $r_1$ and $r_2$ be the roots of the out-trees $T_1$ and $T_2$, respectively. If $T_1 \succ_e T_2$ and if there is an embedding function $f$ such that $f(r_2) = r_1$, then $f$ is a root-embedding function, and we write $T_1 \succ_r T_2$ or $T_2 \prec_r T_1$.

An out-forest $G \in \mathcal{G}_{of}$ is said to be uniform (respectively r-uniform) if all its subtrees $\{T(v),\ v \in G\}$ can be ordered by the embedding (respectively root-embedding) relation. The class of uniform (respectively r-uniform) forests is denoted by $\mathcal{G}_{uof}$ (respectively $\mathcal{G}_{rof}$). It is clear that $\mathcal{G}_{rof} \subset \mathcal{G}_{uof} \subset \mathcal{G}_{of}$.

The graph illustrated in Figure 4.6 is a uniform out-forest. However, it is not r-uniform. An example of r-uniform out-forest is given in Figure 4.7.

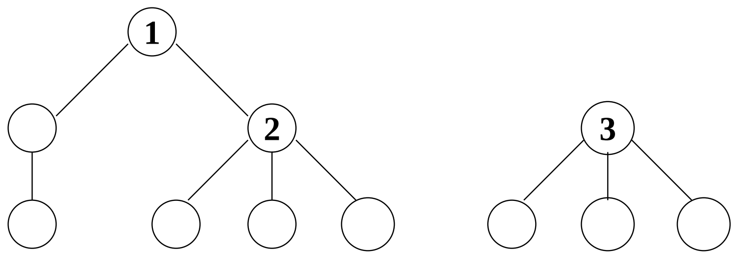


**Figure 4.6** An example of uniform out-forest

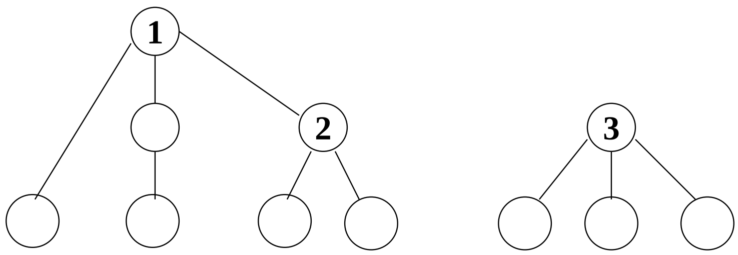


**Figure 4.7** An example of r-uniform out-forest

**Theorem 4.5.3** *LTF stochastically minimizes the makespan of out-forest $G$,*

- *if* $G \in \mathcal{G}_{uof}$ *is uniform and* $M$ *is bounded by* 2*,*
- *or if* $G \in \mathcal{G}_{rof}$ *is r-uniform and* $M$ *is arbitrary.*

The scheme of the proof is again similar to that of Theorem 4.5.1, with the majorization relation being defined as the embedding relation between uniform out-forests: Let $G^1 = (V^1, E^1)$ and $G^2 = (V^2, E^2)$ be two uniform out-forests. Assume that the vertices of $G^1$ and $G^2$ are indexed in such a way that

$$T_{G^1}(1) \succ_e T_{G^1}(2) \succ_e \cdots \succ_e T_{G^1}(|V^1|).$$

$$T_{G^2}(1) \succ_e T_{G^2}(2) \succ_e \cdots \succ_e T_{G^2}(|V^2|).$$

Out-forest $G^1$ is embedded in $G^2$, referred to as $G^1 \prec_e G^2$, if and only if

$$|V^1| \leq |V^2|, \quad \text{and} \quad \forall i,\ 1 \leq i \leq |V^1| : \quad T_{G^1}(i) \prec_e T_{G^2}(i).$$

Similarly, $G^1 \prec_r G^2$ if and only if

$$T_{G^1}(1) \succ_r T_{G^1}(2) \succ_r \cdots \succ_r T_{G^1}(|V^1|),$$

$$T_{G^2}(1) \succ_r T_{G^2}(2) \succ_r \cdots \succ_r T_{G^2}(|V^2|),$$

$$|V^1| \leq |V^2|, \quad \text{and} \quad \forall i,\ 1 \leq i \leq |V^1| : \quad T_{G^1}(i) \prec_r T_{G^2}(i).$$

## REFERENCES


[1] M. Bartusch, R.H. Mohring, and F.J. Radermacher, "M-machine unit time scheduling: a report on ongoing research", *Lecture notes in economics and mathematical systems*, **304** (1988), pp 165-212, Springer, Berlin.

[2] J.L. Bruno, "Deterministic and stochastic problems with tree-like precedence constraints", NATO conference, Durham England, July 1981.

[3] J.L. Bruno, "On scheduling tasks with exponential service times and in-tree precedence constraints", *Acta Informatica*, **22** (1985), pp 139–148.

[4] P. Brucker, M. R. Garey and D. S. Johnson, "Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness", *Math. of Oper. Res.*, **2** (1977), pp 275–284.

[5] K. M. Chandy and P. F. Reynolds, "Scheduling partially ordered tasks with probabilistic execution times", *Operating System Review*, **9** (1975), pp 169–177.

[6] E. G. Coffman, Jr. (ed.) *Computer and job-shop scheduling theory*, Wiley, New York, 1976.

[7] E. G. Coffman, Jr. and R. L. Graham, "Optimal scheduling for two-processor systems", *Acta Informatica*, **1** (1972), pp 200–213.

[8] E. G. Coffman, Jr. and Z. Liu, "On the optimal stochastic scheduling of out-forests", *Opns Res.*, **40** (1992), pp S67–S75.

[9] D. Dolev and M. K. Warmuth, "Scheduling precedence graphs of bounded height", *J. of Algorithms*, **5** (1984), pp 48–59.

[10] D. Dolev and M. K. Warmuth, "Scheduling flat graphs", *SIAM J. on Comput.*, **14** (1985), pp 638–657.

[11] D. Dolev and M. K. Warmuth, "Profile scheduling of opposing forests and level orders", *SIAM J. Alg. Disc. Meth.*, **6** (1985), pp 665–687.

[12] E. Frostig, "A stochastic scheduling problem with intree precedence constraints", *Opns Res.*, **36** (1988), pp 937–943.

[13] M. R. Garey and D. S. Johnson, "Scheduling tasks with nonuniform deadlines on two processors", *J. of the ACM*, **23** (1976), pp 461–467.
[14] M. R. Garey and D. S. Johnson, "Two-processor scheduling with start-times and deadlines", *SIAM J. on Computing*, **6** (1977), pp 416-426.
[15] M. R. Garey, D. S. Johnson, R. E. Tarjan et M. Yannakakis, "Scheduling opposite forests", *SIAM J. Alg. Disc. Meth.*, **4** (1983), pp 72–93.
[16] T. F. Gonzales and D. B. Johnson, "A new algorithm for preemptive scheduling of trees", *J. of the ACM*, **27** (1980), pp 287–312.
[17] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey", *Ann. Discr. Math.*, **5** (1979), pp 287–326.
[18] T.C. Hu, "Parallel sequencing and assembly line problems", *Opns Res.*, **9** (1961), pp 841–848.
[19] E. L. Lawler, "Preemptive scheduling of precedence constrained jobs on parallel machines", in *Deterministic and Stochastic Scheduling*, Dempster et al. (editors), Reidel, 1982, pp 101–123.
[20] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, "Sequencing and scheduling: algorithms and complexity", Report BS-R8909, CWI, Amsterdam, Holland, 1989.
[21] Z. Liu and E. Sanlaville, "Preemptive scheduling with variable profile, precedence constraints and due dates", Rapport de Recherche MASI No. 92.5, Univ. P. et M. Curie, Paris, 1992, to appear in *D.A.M.*
[22] Z. Liu and E. Sanlaville, "Stochastic scheduling with variable profile and precedence constraints". Rapport de Recherche INRIA, No. 1525, 1991, Submitted for publication.
[23] R. McNaughton, "Scheduling with deadlines and loss functions", *Man. Sci.*, **6** (1959), pp 1–12.
[24] R. R. Muntz and E. G. Coffman, Jr., "Optimal preemptive scheduling on two-processor systems", *IEEE Trans. on Comp.*, **C-18** (1969), pp 1014–1020.
[25] R. R. Muntz and E. G. Coffman, Jr., "Preemptive scheduling of real-time tasks on multiprocessor systems", *J. of the ACM*, **17** (1970), pp 325–338.
[26] C. H. Papadimitriou and J. N. Tsitsiklis, "On stochastic scheduling with in-tree precedence constraints", *SIAM J. Comput.*, **16** (1987), pp 1–6.
[27] C. H. Papadimitriou and M. Yannakakis, "Scheduling interval-ordered tasks", Report 11.78, center for research in computer technology Harvard, Cambridge Ma, 1978.
[28] M. Pinedo and G. Weiss, "Scheduling jobs with exponentially distributed processing times and intree precedence constraints on two parallel machines", *Opns Res.*, **33** (1985), pp 1381–1388.
[29] E. Sanlaville, *Conception et analyse d'algorithmes de liste en ordonnancement préemptif.* Thèse de l'université P. et M. Curie, Paris, 1992.
[30] G. Schmidt, "Scheduling independent tasks with deadlines on semi-identical processors", *J. Opnl Res. Soc.*, **39** (1988), pp 271–277.
[31] J. D. Ullman, "NP-complete scheduling problems" *J. Comp. Sys. Sci.*, **10** (1975), pp 384–393.

**Zhen LIU** : INRIA, Centre Sophia Antipolis, 2004 Route des Lucioles, B.P. 93, 06902 Sophia Antipolis, FRANCE
**Eric SANLAVILLE** : Laboratoire LITP, Université Pierre et Marie Curie, 4, place Jussieu, 75252 Paris Cedex 05, FRANCE
The work of this author was partially supported by INRIA while visiting the GERAD laboratory, Montréal, Canada.