Acta Informatica 35, 795–811 (1998)

Acta Informatica
© Springer-Verlag 1998

Machine scheduling with availability constraints

Eric Sanlaville[1], **Günter Schmidt**[2]

[1] LIMOS, Universite Blaise Pascal, F-63177 Aubiere Cedex, France
(e-mail: sanlavil@ucfma.univ-bpclermont.fr)
[2] Information and Technology Management, University of Saarland, D-66041 Saarbrücken, Germany (e-mail: gs@itm.uni-sb.de)

Received: 4 July 1997 / 11 March 1998

Abstract. We will give a survey on results related to scheduling problems where machines are not continuously available for processing. We will deal with single and multi machine problems and analyze their complexity. We survey NP-hardness results, polynomial optimization and approximation algorithms. We also distinguish between on-line and off-line formulations of the problems. Results are concerned with criteria on completion times and due dates.

1 Introduction

In scheduling theory the basic model assumes that all machines are continuously available for processing throughout the planning horizon. This assumption might be justified in some cases but it does not apply if certain maintenance requirements, breakdowns or other reasons that cause the machines not to be available for processing have to be considered. Another motivation for the investigation of constrained machine availability arises, if we have two sets of job requirements, one set having a fixed assignment to time intervals and the other being processed in the remaining free processing intervals. This assumption has many practical applications. In production scheduling environments orders have to be fixed in terms of starting and finishing times. When new orders come in there are already orders from the past assigned to time intervals and corresponding machines while the new ones have to be processed using the remaining free processing intervals. Another application comes from operating systems for mono- and multi-processors, where subprograms with higher priority will interfer with the current program executed. In the following we will first describe the scheduling problems in greater detail. The study is restricted to the deterministic case (for results on stochastic scheduling with non-availability see [LS95b,LS97]). We will survey possibilities of their solution in terms of polynomial and exponential time algorithms. Doing this we will distinguish between non-preemptive, unit execution time and preemptive machine scheduling problems.

2 Problem definition

A machine system with constrained availability is a set of machines (processors) which does not operate continuously; each machine is ready for processing only in specified time intervals of availability. Let $T = \{T_j | j = 1, \ldots, n\}$ be the set of tasks and $P = \{P_i | i = 1, \ldots, m\}$ be the set of machines with machine P_i only available for processing within S_i given time intervals (B_i^s, F_i^s), $s = 1, \ldots, S_i$ and $B_i^{s+1} > F_i^s$ for all $s = 1, \ldots, S_{i-1}$. B_i^s denotes the start time and F_i^s the finish time of s-th interval of availibility of machine P_i.

Each task T_j has a processing requirement of p_j time units. In set T precedence constraints among tasks may be defined. $T_i < T_j$ means that the processing of T_i must be completed before T_j can be started. The tasks in set T are called dependent if the order of execution of at least two tasks in T is restricted by this relation. Then these relations may be modelled by a precedence graph. Otherwise, the tasks are called independent.

Each machine may work only on one task at a time, and each task may be performed by only one machine at a time. Machines may be either parallel, i.e. performing the same functions, or dedicated i.e. being specialized for the execution of certain tasks. If all processors P_i from P have equal task processing speeds, then we call them identical. In case of dedicated processors there are three models of processing sets of tasks: flow shop, open shop and job shop. Later we will investigate flow shops. In such a system we assume that tasks form n subsets (chains), each subset is called a job. That is, job J_j is divided into m tasks, $T_{1j}, T_{2j}, \ldots, T_{mj}$, and task T_{ij} will be performed on processor P_i. In addition, the processing of T_{i-1j} should precede that of T_{ij} for all $i = 2, \ldots, m$ and for all $j = 1, 2, \ldots, n$. The set of jobs will be denoted by J.

Each task (job) may be characterized by the following parameters:

- an arrival time (or ready time) r_j , which is the time at which task T_j (job J_j) is ready for processing; if the arrival times are the same for all tasks from T, then it is assumed that $r_j = 0$ for all T_j;
- a due date d_j , which specifies a time limit by which $T_j(J_j)$ should be completed; usually, penalty functions are defined in accordance with due dates;

- a deadline $\tilde{d}_j$, which is a "hard" real time limit by which $T_j(J_j)$ must be completed;
- a weight (priority) w_j , which expresses the relative urgency of $T_j(J_j)$.

We want to find a feasible schedule (a time-based assignment of processors from set P to tasks from set T meeting all constraints) if one exists, such that all tasks can be processed within the given intervals of machine availability optimizing some performance criterion. Such measures considered here are completion time related and most of them refer to the maximum completion time, the sum of completion times, and the maximum lateness.

A schedule is called preemptive if each task may be preempted at any time and restarted later at no cost, perhaps on another machine. If preemption of all the tasks is not allowed we will call the schedule non-preemptive.

In the following we base the discussion on the three fields $\alpha|\beta|\gamma$ classification scheme suggested in [BEPSW96], that uses most features of the now classical scheduling theory notation. We only add some entry denoting machine availability. We omit entries which are not relevant for the problems investigated here.

The first field $\alpha = \alpha_1\alpha_2\alpha_3$ describes the machine (processor) environment. Parameter $\alpha_1 \in \{P, Q, F\}$ characterizes the type of machine used:

- $\alpha_1 = P$: identical machines (parallel machine system),
- $\alpha_1 = Q$: uniform machines (parallel machine system),
- $\alpha_1 = F$: dedicated machines (flow shop system).

Parameter $\alpha_2 \in \{\emptyset, k\}$ denotes the number of machines or stages in the system:

- $\alpha_2 = \emptyset$: the number of machines is assumed to be variable,
- $\alpha_2 = k$: the number of machines is equal to k (k is a positive integer).

In [Sch84] and [LS95a] different patterns of machine availability are discussed for the case of parallel machine systems. These are constant, zigzag, decreasing, increasing, and staircase. Let $0 = t_1 < t_2 < \ldots < t_j < \ldots < t_Q$ be the points in time where the availability of a certain machine changes and let $m(t_j)$ be the number of machines being available during time interval $[t_j, t_{j+1})$ with $m(t_j) > 0$. It is assumed that the pattern is not changed infinitely often during any finite time interval. According to these cases parameter $\alpha_3 \in \{\emptyset, NC, NC_{zz}, NC_{inc}, NC_{dec}, NC_{inczz}, NC_{deczz}, NC_{sc}\}$ denotes the machine availability.

(1) If all machines are continuously available $(t = 0)$ then the pattern is called constant ($\alpha_3 = \emptyset$).
(2) If there are only k or $k - 1$ machines in each interval available then the pattern is called zigzag ($\alpha_3 = NC_{zz}$).

(3) A pattern is called increasing (decreasing) if for all j from $IN_+ m(t_j) \geq \max_{1 \leq u \leq j-1}\{m(t_u)\}$ $(m(t_j) \leq \min_{1 \leq u \leq j-1}\{m(t_u)\})$, i.e. the number of machines available in interval $[t_{j-1}, t_j)$ is not more (less) than this number in interval $[t_j, t_{j+1})$ ($\alpha_3 \in \{NC_{inc}, NC_{dec}\}$).

(4) A pattern is called increasing (decreasing) zigzag if for all j from IN_+ $m(t_j) \geq \max_{1 \leq u \leq j-1} \{m(t_u) - 1\}$ $(m(t_j) \leq \min_{1 \leq u \leq j-1}\{m(t_u) + 1\})$ ($\alpha_3 \in \{NC_{inczz}, NC_{deczz}\}$).

(5) A pattern is called staircase if for all intervals the availability of machine P_i implies the availability of machine P_{i-1} $(\alpha_3 = NC_{sc})$. A staircase pattern is shown in the lower part of Fig. 1; note that patterns (1)-(4) are special cases of (5).

(6) A pattern is called arbitrary if none of the conditions (1)-(5) applies ($\alpha_3 = NC$). Such a pattern is shown in the upper part of Fig. 1; patterns defined in (1)-(5) are special cases of the one in (6).

Machine systems with arbitrary patterns of availability can always be translated to a composite machine system forming a staircase pattern [Sch84]. A composite machine is an artificial machine consisting of at most m original machines. The transformation process works in the following way. An arbitrary pattern is separated in as many time intervals as there are distinct points in time where the availability of at least one machine changes. Now in every interval periods of non-availability are moved from machines with smaller index to machines with greater index. If there are $m(t_j)$ machines available in some interval $[t_j, t_j + 1)$ then after the transformation machines $P_1, \ldots, P_{m(t_j)}$ will be available in $[t_j, t_j + 1)$ and $P_{m(t_j)+1}, \ldots, P_m$ will not be available, where $0 < m(t_j) < m$. Doing this for every interval we generate composite machines consisting of at most m original machines with respect to the planning horizon.

An example for such a transformation considering $m = 4$ machines is given in Fig. 1. Non-availability is represented by the grayed areas. Composite machines which do not have intervals of availability can be omitted from the problem description. Then, the number of composite machines is the maximum number of machines simultaneously available. The time complexity of the transformation is $O(Qm)$ where Q is the number of points in time, where the availability of a machine is changing. If this number is polynomial in n or m machine scheduling problems with arbitrary patterns of non-availabilty can be transformed in polynomial time to a staircase pattern. This transformation is useful as, first, availability at time t is given by the number of available composite machines and, second, some results are obtained assuming this hypothesis (see Sect. 5.2).

The second field $\beta = \beta_1, \ldots, \beta_8$ describes task (job) and resource characteristics. We will only refer to parameters β_1, β_3, β_4, β_5, and β_6. Parameter $\beta_1 \in \{\emptyset, t - pmtn, pmtn\}$ indicates the possibilities of preemption:

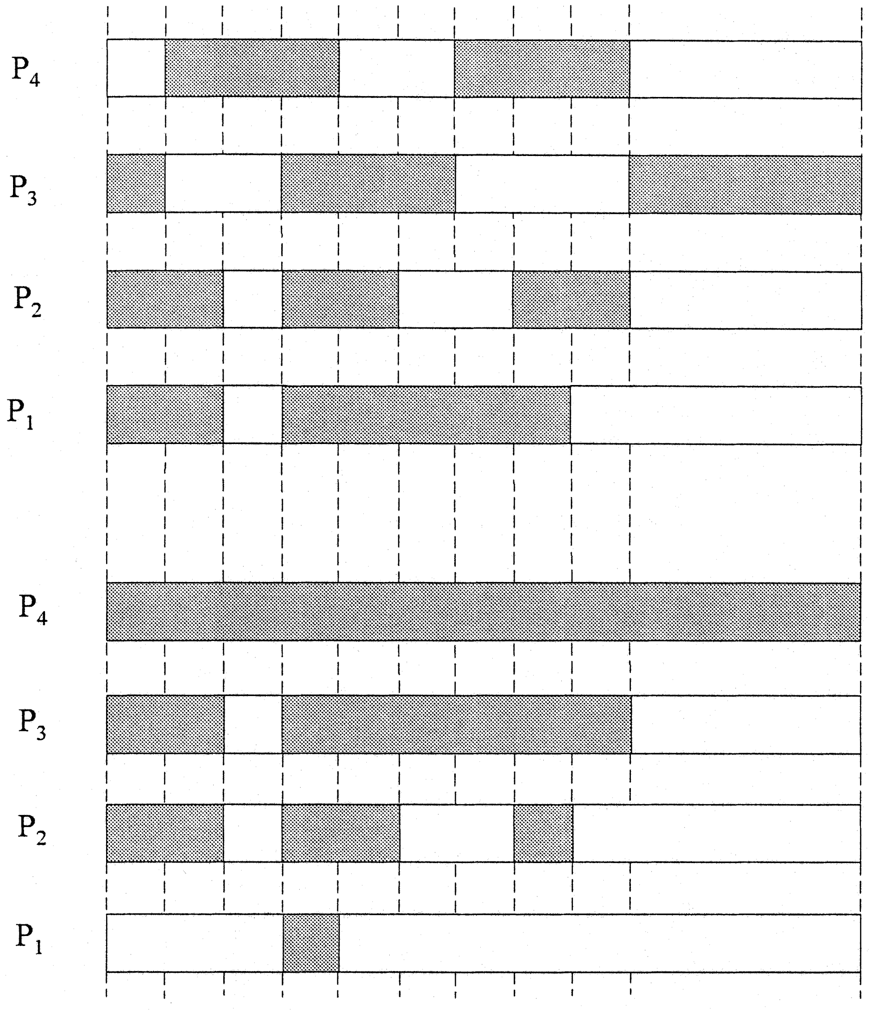


Fig. 1 Rearrangement of arbitrary patterns

- $\beta_1 = \emptyset$: no preemption is allowed;
- $\beta_1 = t - pmtn$: tasks may be preempted, but each task must be processed by only one machine;
- $\beta_1 = pmtn$: tasks may be arbitrarily preempted.

Of course the rearrangement of an arbitrary pattern to a staircase pattern is only used when preemption is allowed. In what follows the number of preemptions may be a criterion to appreciate the value of an algorithm. When the algorithm applies to staircase patterns, the number of preemptions for an arbitrary pattern is increased by at most mQ.

Parameter $\beta_3 \in \{\emptyset, prec, tree, forest, chains\}$ reflects the precedence constraints and denotes respectively independent tasks, arbitrary precedence constraints, precedence constraints forming a tree, a set of trees, or a set of chains. Parameter $\beta_4 \in \{\emptyset, r_j\}$ describes ready times:

- $\beta_4 = \emptyset$: all ready times are zero,
- $\beta_4 = r_j$: ready times differ per task.

Parameter $\beta_5 \in \{\emptyset, p_j = p\}$ describes task durations:

- $\beta_5 = \emptyset$: arbitrary processing times are assumed,
- $\beta_5 = (p_j = p)$: all processing times are equal.

Parameter $\beta_6 \in \{\emptyset, \tilde{d})\}$ describes deadlines:

- $\beta_6 = \emptyset$: no deadlines are assumed in the system (however, due dates may be defined if a due date involving criterion is used to evaluate schedules),
- $\beta_6 = \tilde{d}$: deadlines are imposed on the performance of a task set.

The third field, γ, denotes an optimality criterion (performance measure), i.e. $\gamma \in \{C_{max}, \sum C_j, \sum w_j C_j, L_{max}, \sum U_j, \sum w_j U_j, \emptyset\}$, where C_{max} refers to minimizing the makespan $(max\{C_j\})$, $\sum C_j$ to the sum of completion times, $\sum w_j C_j$ to the sum of weighted completion times, L_{max} to the maximum lateness $(\max\{C_j - d_j\})$, $\sum U_j$ to the sum (or number) of late jobs $(C_j > d_j)$, $\sum w_j U_j$ to the weighted sum of late jobs, and $\emptyset$ means testing for feasibility whenever scheduling to meet deadlines is considered.

In order to solve these problems different kind of algorithms will be applied. In some cases we will mention off-line and on-line algorithms.

- An algorithm is on-line if it proceeds sequentially and at each time t it only needs to know the number of processors available at t, the number of ready tasks at t, their remaining processing time and their deadlines or due dates.
- An algorithm is nearly on-line if it needs in addition at time t the time of the next event, that is either a new task becomes ready for processing or the number of available machines changes ([San95], extended from [LLLR79]).
- An algorithm is off-line if all problem data has to be known in advance. That is, at time 0 it needs all informations concerning machine availabilities and task characteristics.

If the machine non-availabilities are due to breakdowns on-line algorithms are needed. If the times of machine availability changes are known a little time in advance nearly on-line algorithms will suffice. Otherwise off-line algorithms will do. Note that it is often assumed in *on-line scheduling* papers (see for instance [CVW94]) that even processing times are not known before the processing begins. This is not investigated here.

We will also report on approximation algorithms in case the scheduling problem is NP-complete. Then we will distinguish between relative and absolute errors. The relative error R_H is defined as $R_H(I) = C_H(I)/C_{opt}(I)$ -1, where $C_H(I)$ is the performance of approximation algorithm H applied to some problem instance I and $C_{opt}(I)$ is the value of the corresponding optimal solution. The absolute error A_H is defined as $A_H(I) = C_H(I) - C_{opt}(I)$. For a more comprehensive description of problems and their solution in terms of complexity issues we refer to [GJ79].

3 Non-preemptive scheduling

3.1 Minimizing the sum of completion times

In case of continuous availability of the machines the problem can be solved applying the Shortest Processing Time (SPT) rule. With this rule the tasks are ordered according to non-decreasing processing times and then iteratively assigned to the machine with the smallest already assigned load. Scheduling algorithms built on this rule can be executed in $O(nlogn)$ time. If machines have only different beginning times B_i (this corresponds to an increasing pattern of availability) the problem can also be solved by the SPT rule [KM88, Lim91]. If $m = 2$ and there is only one finish time F_i^s on one machine which is smaller than infinity (this corresponds to a zigzag pattern of availability) the problem becomes NP-complete [LL93]. In the same paper Lee and Liman show that the SPT rule with the following modification leads to a tight relative error of $R_{SPT} \leq 1/2$ for $P2{,}NC||\sum C_j$ where machine P_1 is continuously available and machine P_2 has one finish time which is smaller than infinity.

Step 1: Assign the shortest task to P_1.
Step 2: Assign the remaining tasks in SPT order alternately to both machines until some time when no other task can be assigned to P_2 without violating F_2.
Step 3: Assign the remaining tasks to P_1.

Figure 2 illustrates how that bound can be reached asymptotically (when ϵ tends toward 0). The modified SPT rule leads to a large idle time for machine P_1.

If there is only a single interval of non-availability with $B_i > 0$ and $F_i < \sum_j p_j$ Adiri et al. [ABFR89] show that the problem becomes NP-complete even for $m = 1$. The SPT rule leads to a tight relative error of $R_{SPT} \leq 2/7$ for this problem [LL92]. For fixed m the SPT rule is asymptotic optimal if there is not more than one interval of non-availability for each machine [Mos94].

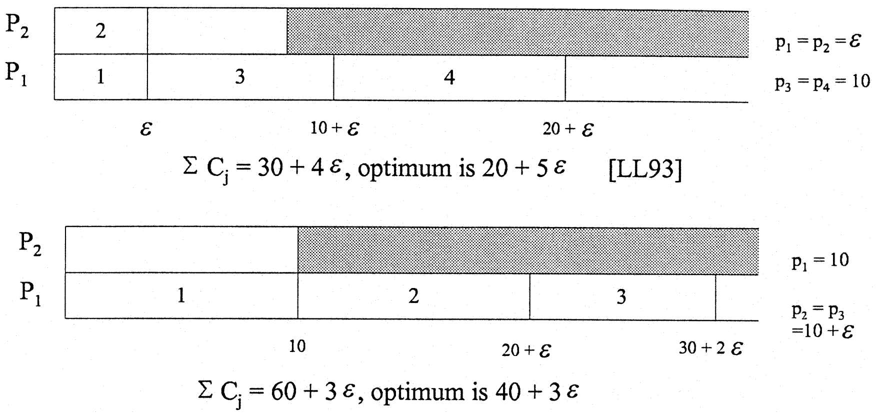


Fig. 2 Examples for the modified SPT rule

3.2 Minimizing the makespan

Ullman [Ull75] was the first to study the problem $P, NC||C_{max}$. It is NP-complete in the strong sense for m arbitrary (3-partition is a special case) even if the machines are continuously available. If machines have different beginning times B_i the Longest Processing Time (LPT) rule leads to a relative error of $R_{LPT} \leq 1/2 - 1/(2m)$ or of $R_{MLPT} \leq 1/3$ if the rule is appropriately modified [Lee91]. Both bounds are tight. The modification uses dummy tasks to simulate the different machine starting times B_i. For each machine P_i a task T_j with processing time $p_j = B_i$ is inserted. The dummy tasks are merged into the original task set and then all tasks are scheduled according to the LPT rule under an additional restriction that only one dummy task is assigned to each machine. After finishing the schedule all dummy tasks are moved to the head of the machines followed by the remaining tasks assigned to each P_i. The LPT rule runs in $O(n \log n)$ and its modification in $O((n+m)\log(n+m)+(n+m)m)$ time, respectively. Note that a LPT algorithm leads for continuously available machines to a relative error of $R_{LPT} \leq 1/3 - 1/(3m)$ [Gra69]. Kellerer [Kel98] presents a dual approximation algorithm using a bin packing approach and leading to a tight bound of $1/4$. Also for $m = 1$ the problem remains NP-complete [Lee96].

4 Unit execution time scheduling

Unit execution time (UET) scheduling is important for applications for two reasons. First, it contains several frontier problems when looking at complexity issues, and second, it models a restrictive version of preemption, when interrupting a task is only allowed at specified (integer) moments. Of course availability changes are also restricted to integer moments.

4.1 Minimizing the makespan

The problem with arbitrary precedence constraints is NP-complete even for a constant availability pattern. If the precedence graph is an inforest, the problem is still NP-complete for a decreasing pattern (see [GJTY83]). Dynamic programming algorithms may be used then.

Some list algorithms are optimal for some specific availability pattern. The list algorithm of Coffman and Graham is optimal for two machines, arbitrary precedence constraints, and an arbitrary pattern (see [LS95a]). If the graph is an interval order graph, the following property holds: consider two tasks; then the set of (non necessary immediate) successors of one is included into the set of successors of the other. The list algorithm choosing first the tasks with the largest set of successors (Most Successor First (MSF) rule) is optimal on an arbitrary availability pattern (see [LS95b]). Figure 3 shows an instance where any other choice for the first task to be scheduled leads to a sub-optimal solution (MSF priority list is $2, 1, 3, 5, 6, 4, 7$). Interval order graphs attracted much attention as any precedence graph might be transformed to an interval order graph by adding a set of precedence relations (see [PY79]). Hopefully, optimal schedules for interval order graphs may lead to satisfactory schedules for arbitrary task graphs.

Highest Level First (HLF) schedules are optimal if the precedence graph is an inforest (outforest) and the pattern is NC_{inczz} (NC_{deczz}) [DW85a], or for chains of tasks and arbitrary patterns [LS95a]. Last, using HLF policy leads to a $O(nlogn)$ " flip-flop" algorithm for scheduling opposing forests to zigzag patterns with 2 or 3 machines [DW85b]. Opposing forests are the union of out-trees and in-trees.

4.2 Minimizing the lateness

Brucker et al. [BGJ77] proved that an Earliest Due Date (EDD) rule applied to modified due dates is optimal if the precedence graph is an in-tree. Liu and Sanlaville [LS95a] prove that this remains true, with a similar modification scheme, for increasing zigzag availability patterns. In the same way, Garey and Johnson [GJ77] propose a modification scheme so that EDD builds optimal schedules on two machines, for arbitrary task graphs. This result can also be extended [LS95b] to arbitrary availability patterns; the algorithm is off-line.

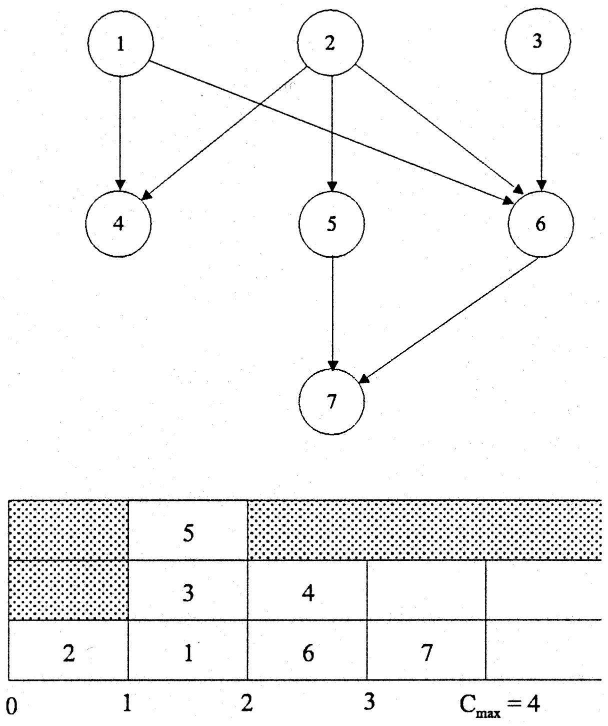


Fig. 3 Use of the MSF rule for interval order graphs

5 Preemptive scheduling

In [LLP97] the notion of resumability and non-resumability of tasks is introduced. This means that only task preemption on the same machine is considered. In such a model preemptions between different machines are not allowed. Here we assume that not only task ($\beta_1 = t - pmtn$) but also machine preemptions are possible ($\beta_1 = pmtn$). If there is a unique assignment of a task to a single machine task preemptions and arbitrary preemptions are equivalent.

5.1 Single machine problems

If we want to minimize the makespan, it is easy to see that every schedule is optimal which starts at time zero and has no unforced idle time, that is, the machine never remains idle while some task is ready for processing. Furthermore, preemption is never useful except when some task cannot be finished before an interval of non-availability occurs. This property is still

true for completion time based criteria if there is no precedence constraint and no release date, as is assumed in the rest of this section.
While the sum of completion times is still minimized by the SPT rule the problem of minimizing the weighted sum is NP-complete [Lee96]. Note that without availability constraints Smith's rule solves the problem. Maximum lateness is minimized by the Earliest Due Date (EDD) rule [Lee96]. Applying this rule all tasks are ordered and assigned to the machine according to their non-decreasing due dates. The rule can be carried out in $O(n \log n)$ time. If the number of late tasks has to be minimized the EDD rule of Moore and Hodgson's algorithm [Moo68] can be modified to solve this problem also in $O(n \log n)$ time [Lee96]. Note that the weighted problem is NP-complete already for a continuously available machine [Kar72].

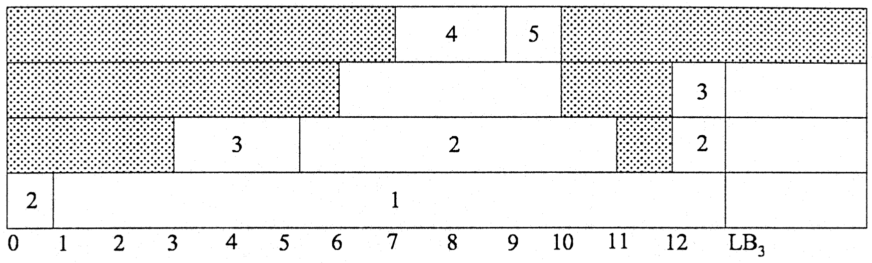
5.2 Minimizing the makespan on m machines

If all machines are only available in one and the same time interval (B, F) and tasks are independent McNaughton [McN59] shows that there exists a feasible preemptive schedule iff $\max_j\{p_j\} \leq (F - B)$ and $\sum_j p_j \leq m(F - B)$. He gives an $O(n)$ algorithm which generates at most $m - 1$ preemptions to construct this schedule. If all machines are available in an arbitrary number $S = \sum_i S_i$ of time intervals $(B_i^s, F_i^s), s = 1, \ldots, S_i$ Schmidt [Sch84] shows a feasible preemptive schedule exists if and only if the following m conditions are met:

$$\begin{cases} \forall k = 1 \rightarrow m-1, & \sum_{j=1}^{k} p_j \leq \sum_{i=1}^{k} PC_i \qquad \mathcal{P}(k) \\ & \sum_{j=1}^{n} p_j \leq \sum_{i=1}^{m} PC_i \qquad \mathcal{P}(m) \end{cases}$$

with $p_1 \geq p_2 \geq \cdots \geq p_n$ and $PC_1 \geq PC_2 \geq \cdots \geq PC_m$, where PC_i is the total processing capacity of machine P_i. Such a schedule can be constructed in $O(n + m \log m)$ time after the processing capacities PC_i are computed, with at most $S - 1$ preemptions in case of a staircase pattern (remember that any arbitrary pattern of availability can be converted into a staircase one at the price of additional preemptions). Note that in the case of the same availability interval (B, F) for all machines McNaughton's conditions are obtained from $\mathcal{P}(1)$ and $\mathcal{P}(m)$ alone. This remains true for zigzag patterns as then $\mathcal{P}(2), \ldots, \mathcal{P}(m-1)$ are always verified if $\mathcal{P}(1)$ is true (there is one availability interval for all machines but P_m). In [Sch88] the problem is generalized taking into account different task release times or deadlines. Then it can be solved in $O(nm \log m)$.

The corresponding optimization problem (minimizing C_{max}) is solved by an algorithm that first computes the lower bounds $LB_1, \ldots, LB_m$ obtained from the conditions above (see Fig. 4).



T_j	1	2	3	4	5
p_j	12	7	7	2	1
LB_j	12	11	38/3	149/12	38/3

Fig. 4 Minimizing the makespan on a staircase pattern

C_{max} must be larger than $LB_k, k = 1 \to m-1$, obtained from $\mathcal{P}(k)$. The sum of availabilities of machines $P_1, \ldots, P_k$ during time interval $[0, LB_k)$ must be larger or equal than the sum of processing times of tasks $T_1, \ldots, T_k$. The sum of all machine availabilities during time interval $[0, LB_m)$ must also be larger or equal than the sum of processing times of all tasks. In the example of Fig. 4, $C_{max} = LB_3$. The number of preemptions is $S - 2$.

When precedence constraints are added, Liu and Sanlaville [LS95a] show that problems with chains and arbitrary patterns of non-availability can be solved in polynomial time applying the Longest Remaining Path first (LRP) rule and the processor sharing procedure of [MC70]. In the same paper it is also shown that the LRP rule could be used to solve problems with decreasing (increasing) zigzag patterns and tasks forming an outforest (inforest). In case of only two machines and arbitrary patterns of non-availability this rule also solves problems with arbitrary task precedence constraints with time complexity and number of preemptions of $O(n^2)$. These results are deduced from these for Unit Execution Time scheduling by list algorithms. However, the same extension for interval order graphs is not possible. The algorithm is nearly on-line, as are all priority algorithms which extend list algorithms to preemption [Law82]. Indeed these algorithms first build a schedule admitting processor sharing. These schedules execute tasks of the same priority at the same speed. This property is respected when McNaughton's rule is applied. If a machine availability changes unexpectedly, the property does not hold any more.

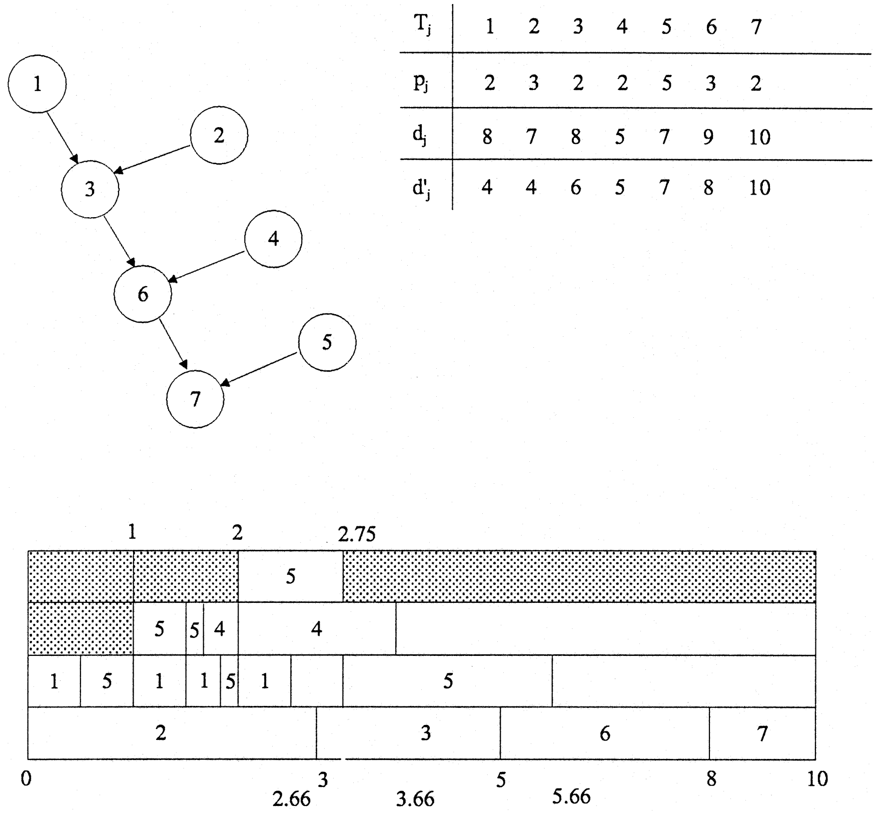
Applying the LRP rule results in a time complexity of $O(n \log n + nm)$ and a number of preemptions of $O((n+m)^2 - nm)$ which both can be improved. Therefore in [BFKS96] an algorithm is given which solves the problem with N chains in $O(N + m \log m)$ time generating a number of preemptions which is not greater than the number of intervals of availability of all machines. If all machines are only available in one processing interval and all intervals are ordered in a staircase pattern the algorithm generates feasible schedules with at most $m-1$ preemptions. Having more than two machines in the case of arbitrary precedence constraints and a tree precedence structure for an arbitrary number of machines makes the problem NP-complete [BFKS96].

5.3 Dealing with due date involving criteria

In [Hor74] it is shown that $P|pmtn, r_j, \tilde{d}_j|$ can be solved in $O(n^3 min\{n^2, \log n + \log p_{max}\})$ time. The same flow-based approach can be coupled with a bisection search to minimize the maximum lateness L_{max} (see [LLLR79], where the method is also extended to uniform machines). A slightly modified version of the algorithm still applies to the corresponding problem where the machines are not continuously available. If the number of changes of machine availabilities during any time interval is linear in the length of the interval this approach can be implemented in $O(n^3 p_{max}^3 (\log n + \log p_{max}))$ [San95].These algorithms need the knowledge of all the data at time 0 and are hence off-line. When no release dates are given but due dates have to be considered maximum lateness can be minimized using the approach suggested by [Sch88] in $O(nm \log n)$ time. The method needs just to know all possible events before the next due date.

If there are not only due dates but also release dates to be considered Sanlaville [San95] suggests a nearly on-line priority algorithm with an absolute error of $A \leq (m - 1/m)p_{max}$ if the availabilty of the machines follows a constant pattern and of $A \leq p_{max}$ if machine availability refers to an increasing zigzag pattern. The priority is calculated according to the Smallest Laxity First (SLF) rule, where laxity (or slack time) is the difference between the task's due date and its remaining processing time. The SLF algorithm runs in $O(n^2 p_{max})$ and it is optimal in the case of a zigzag pattern and no release dates.

Lawler and Martel [LM89] solve the weighted number of late jobs problem on two uniform machines, i.e. $Q2|pmtn|\sum w_j U_j$. The originality of their paper comes from the fact that they show a stronger result, as the speeds of the processors may change continuously (and even be 0) during the execution. Hence it includes as a special case availability constraints on two uniform machines. They use dynamic programming to propose pseudo-



T_j	1	2	3	4	5	6	7
p_j	2	3	2	2	5	3	2
d_j	8	7	8	5	7	9	10
d'_j	4	4	6	5	7	8	10

Fig. 5 Minimizing L_{max} on an increasing zigzag pattern

polynomial algorithms ($O(\sum w_j n^2)$, or $O(n^2 p_{max})$ to minimize the number of late jobs). Nothing however is said about the effort needed to compute processing capacity in one interval.

Liu and Sanlaville [LS95a] show that results on C_{max} minimization for inforest precedence graphs and increasing zigzag patterns can be extended to L_{max}, using Smallest Laxity First rule on the modified due dates. Figure 5 shows an optimal SLF schedule for an in-tree. The modified due date is given by $d'_j = min((d_j, d'_{s(j)} + p_{s(j)})$ where $T_{s(j)}$ is the successor of T_j when it exists.

In the same way, minimizing L_{max} on two machines with availability constraints is achieved using SLF with a different modification scheme. The proofs use again the results from UET scheduling.

Table 1 Summary of complexity results

problem	polynomial criteria	NP-complete criteria
$1, NC$		$\sum C_j$, $C_{\max}$
$1, NC\vert pmtn$	$\sum C_j$, $C_{\max}$, $L_{\max}$, $\sum U_j$	$\sum w_j C_j$, $\sum w_j U_j$ (constant availability)
P, NC	$\sum C_j$ (different beginning times)	$\sum C_j$ (different finish times)
$P2, NC\vert prec, p_j = 1$	$C_{\max}$, $L_{\max}$	
$P2, NC\vert pmtn, prec$	$C_{\max}$, $L_{\max}$	
$P, NC_{zz}\vert tree, p_j = 1$	$C_{\max}$, $L_{\max}$ (in tree)	$C_{\max}$ for NC_{dec} (in tree)
$P, NC_{zz}\vert pmtn, tree$	$C_{\max}$, $L_{\max}$ (in tree)	$C_{\max}$ for NC
$P, NC\vert p_j = 1$	$C_{\max}$ (interval order), $L_{\max}$	
$P, NC\vert pmtn$	$C_{\max}$ (chains), $L_{\max}$	
$P, NC\vert pmtn, r_j$	$C_{\max}$ $L_{\max}$	
$F2, NC$		$C_{\max}$ (single non-availability interval)

5.4 Flow shop scheduling

Lee [Lee97] has shown that flow shop scheduling is already NP-complete for two machines and there is a single interval of non-availability on one machine only. He also gives an approximation algorithm which has a relative error of 1/2 if this interval is on machine one or of 1/3 if the interval of non-availability is on machine two (the classical Flow-Shop is symmetrical, but the non-availability interval breaks the symmetry).

6 Summary and outlook

This survey shows that scheduling with availability constraints attracts more and more attention, as the importance of the applications is recognized. The results presented here are of two kinds. Either they extend classical algorithms and show that optimality is kept, or they show NP-completeness due to availability constraints. In particular, when preemption is not authorized it will logically entail NP-Completeness. Performance bounds may often be obtained,but their quality will depend on the availability patterns investigated. This should prove an interesting field for further research. We tried to summarize most of the complexity results presented in this survey in Table 1. For a given problem type, we distinguish between performance criteria entailing NP-completeness and those for which a polynomial algorithm exists.

If availability constraints come from unexpected breakdowns, fully on-line algorithms are needed. But many results of optimality concern at best nearly on-line algorithms (in case of preemptive scheduling). It is an open question to look for optimality results for fully on-line algorithms and specific availability patterns, or at least to compute performance bounds.

References

[ABFR89] Adiri, I., Bruno, J., Frostig, E., Rinnooy Kan, A.H.G.: Single machine flowtime scheduling with a single breakdown, Acta Informatica **26**, 679–696 (1989)

[BEPSW96] Blazewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling Computer and Manufacturing Processes, Berlin, Springer, 1996

[BFKS96] Blazewicz, J., Formanowicz, P., Kubiak, W., Schmidt, G.: Scheduling precedence constrained tasks on semi-identical processors, unpublished manuscript, 1996

[BGJ77] Brucker, P., Garey, M.R., Johnson, D.S.: Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness, Math. of Oper. Res. **2**, 275–284 (1977)

[CVW94] Chen, B., Van Vliet, A., Woeginger, G.J.: A lower bound for randomized on-line scheduling algorithms, Inf. Proc. Letters, **51**, 219–222 (1994)

[DW85a] Dolev, D., Warmuth, M.K.: Scheduling flat graphs, SIAM J. Comput. **14**, 638–657 (1985)

[DW85b] Dolev, D., Warmuth, M.K.: Profile scheduling of opposing forests and level orders, SIAM J. Alg. Disc. Meth. **6**, 665–687 (1985)

[GJ77] Garey, M.R., Johnson, D.S.: Two-processor scheduling with start-times and deadlines, SIAM J. on Comput. **6**, 416–426 (1977)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and Intractability, San Francisco, Freeman, 1979

[GJTY83] Garey, M.R., Johnson, D.S., Tarjan, R.E., Yanakakis, M.: Scheduling opposite forests, SIAM J. Alg. Disc. Meth. **4**, 72–93 (1983)

[Gra69] Graham, R.L.: Bounds on multiprocessing timing anomalies, SIAM J. Appl. Math. **17**, 263–269 (1969)

[Hor74] Horn, W.A., Some simple scheduling algorithms, Naval Res. Logist. Quart. **21**, 177–185 (1974)

[Kar72] Karp, R., M.: Reducibility among combinatorial problems, in: R.E.Miller, J.W.Thatcher (eds.), Complexity of Computer Communications, New York: Plenum Press, 85–103 (1972)

[Kel98] Kellerer, H.: Algorithms for multiprocessor scheduling with machine release time, IIE Transactions, to appear

[KM88] Kaspi, M., Montreuil, B.: On the scheduling of identical parallel processes with arbitrary initial processor available time, Research Report 88-12, School of Industrial Engineering, Purdue University, 1988

[LLLR79] Labetoulle, J., Lawler, E.L., Lenstra, J.K.: Rinnooy Kan, A.H.G.: Preemptive scheduling of uniform machines subject to due dates, Technical Paper BW 99/79, CWI, Amsterdam, 1979

[Law82] Lawler, E.L.: Preemptive scheduling of precedence constrained jobs on parallel machines, In: Dempster et al. (eds.), Deterministic and Stochastic Scheduling, 101–123 Dordrecht: Reidel (1982)

[LM89] Lawler, E.L., Martel, C.U.: Preemptive scheduling of two uniform machines to minimize the number of late jobs, Oper. Res. **37**, 314–318 (1989)

[Lee91] Lee, C.-Y.: Parallel machine scheduling with non-simultaneous machine available time, Discrete Appl. Maths. 30, 53–61 (1991)

[Lee96] Lee, C.-Y.: Machine scheduling with an availability constraint, Journal of Global Optimization, Special Issue on Optimization of Scheduling Applications, **9**, 363–384 (1996)

[Lee97] Lee, C.-Y.: Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint, Oper. Res. Lett., **20**, 129–139 (1997)

[LL92] Lee, C.-Y., Liman, S.D.: Single machine flow-time scheduling with scheduled maintenance, Acta Informatica **29**, 375-382, (1992)

[LL93] Lee, C.-Y., Liman, S.D.: Capacitated two-parallel machine scheduling to minimize sum of job completion times, Discrete Appl. Math. **41**, 211–222 (1993)

[LLP97] Lee, C.-Y., Lei, L., Pinedo, M.: Current trends in deterministic scheduling, Ann. Oper. Res. **70**, 1997, 1–42

[Lim91] Liman, S.: Scheduling with Capacities and Due-Dates, Ph.D. Thesis, University of Florida, 1991

[LS95a] Liu, Z., Sanlaville, E.: Preemptive scheduling with variable profile, precedence constraints and due dates, Discrete Appl. Math. **58**, 253–280 (1995)

[LS95b] Liu, Z., Sanlaville, E.: Profile scheduling of list algorithms, In: Chretienne, P. et al. (eds.), Scheduling Theory and its Applications, New York :Wiley 91–110 (1995)

[LS97] Liu, Z., Sanlaville, E.: Stochastic scheduling with variable profile and precedence constraints, SIAM J. Comput. **26**, 173–187 (1997)

[MC70] Muntz, R., Coffman, E.G.: Preemptive Scheduling of Real- Time Tasks on Multiprocessor Systems, J. Assoc. Comput. Mach. **17**, 324–338 (1970)

[McN59] McNaughton, R.: Scheduling with deadlines and loss functions, Manage. Sci. **6**, 1–12 (1959)

[Moo68] Moore, J.M.: An n job one machine sequencing algorithm for minimizing the number of late jobs, Mgmt. Sci. **15**, 102–109 (1968)

[Mos94] Mosheiov, G.: Minimizing the sum of job completion times on capacitated parallel machines, Mathl. Comput. Modelling **20**, 91–99 (1994)

[PY79] Papadimitriou, C.H., Yanakakis, M.: Scheduling interval ordered tasks, SIAM J. Comput. **8**, 405–409 (1979)

[San95] Sanlaville, E.: Nearly on line scheduling of preemptive independent tasks, Discrete Appl. Maths. **57**, 229-241 (1995)

[Sch84] Schmidt, G.: Scheduling on semi-identical processors, Z. Oper. Res. **A28**, 153-162, (1984)

[Sch88] Schmidt, G.: Scheduling Independent Tasks with Deadlines on Semi-Identical Processors, J. Oper. Res. Soc. 39, 271-277 (1988)

[Ull75] Ullman, J.D.: NP-complete scheduling problems, J. Comput. Syst. Sci. **10**, 384–393 (1975)