# 11 Scheduling with Limited Processor Availability[1]

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 constraints that cause the machines not to be available for processing have to be considered. In this chapter we discuss results related to deterministic scheduling problems where machines are not continuously available for processing.

Examples of such constraints can be found in many areas. Limited availabilities of machines may result from pre-schedules which exist mainly because most of the real world resources planning problems are dynamic. A natural approach to cope with a dynamic environment is to trigger a new planning horizon when the changes in the data justify it. However, due to many necessities, as process preparation for instance, it is mandatory to take results of earlier plans as fixed which obviously limits availability of resources for any subsequent plan. Consider e.g. ERP (Enterprise Resource Planning) production planning systems when a rolling horizon approach is used for customer order assignment on a tactical level. Here consecutive time periods overlap where planning decisions taken in earlier periods constrain those for later periods. Because of this arrangement orders related to earlier periods are also assigned to time intervals of later periods causing the resources not to be available during these intervals for orders arriving after the planning decisions have been taken. The same kind of problem may be repeated on the operational level of production scheduling. Here processing of some jobs is fixed in terms of starting and finishing times and machine assignment. When new jobs are released to the shop floor there are already jobs assigned to time intervals and machines while the new ones have to be processed within the remaining free processing intervals.

Another application of limited machine availability comes from operating systems for mono- and multi-processors, where subprograms with higher priority will interfere with the current program executed. A similar problem arises in multi-user computer systems where the load changes during the usage. In big massively parallel systems it is convenient to change the partition of the processors among different types of users according to their requirements for the machine. Fluctuations related to the processing capacity can be modeled by intervals of different processor availability. Numerous other examples exist where the investigation of limited machine availability is of great importance and the prac-

[1] This paper is based on O. Braun, J, Breit, G. Schmidt, Deterministic Machine Scheduling with Limited Machine Availability, Discussion paper B0403, Saarland University, 2004.

J. Blazewicz et al., *Handbook on Scheduling*, International Handbooks on Information Systems, https://doi.org/10.1007/978-3-319-99849-7_11

tical need to deal with this type of problem has been proven by a growing demand for commercial software packages. Thus, recently the analysis of these problems has attracted many researchers.

In the following we will investigate scheduling problems with limited machine availability in greater detail. The research was started by G. Schmidt [Sch84]. The review focuses on deterministic models with information about the availability constraints. Earlier surveys of this research area can be found in [SS98, Sch00, Lee04]. For stochastic scheduling problems with limited machine availability and prior distributions of the problem parameters see [GGN00, LS95b, LS97]. We will survey results for one machine, parallel machine and shop scheduling problems in terms of intractability and polynomial time algorithms. In some places also results from enumerative optimization algorithms and heuristics are analyzed. Doing this we will distinguish between non-preemptive and preemptive scheduling. We will finish with some conclusions and some suggestions for future research.

## 11.1 Problem Definition

A machine system with limited availability is a set of machines (processors) which does not operate continuously; each machine is ready for processing only in certain time intervals of availability. Let $\mathcal{P} = \{P_i \mid i = 1,\cdots,m\}$ be the set of machines with machine $P_i$ only available for processing within $S_i$ given time intervals $[B_i^s, F_i^s)$, $s = 1,\cdots,S_i$ and $B_i^{s+1} > F_i^s$ for all $s = 1,\cdots,S_{i-1}$. $B_i^s$ denotes the start time and $F_i^s$ the finish time of $s^{\text{th}}$ interval of availability of machine $P_i$.

We want to find a feasible schedule 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 and due date related and most of them refer to the maximum completion time, the sum of completion times, and the maximum lateness.

The term preemption is used as defined before. Often the notion of resumability is used instead of preemption. Under a resumable scenario a task may be interrupted when a machine becomes unavailable and resumed as the machine becomes available again without any penalty. Under the non-resumable scenario task preemption is generally forbidden. The most general scenario is semi-resumability. Let $x_j$ denote the part of task $T_j$ processed before an interruption and let $\delta \in [0,1]$ be a given parameter. Under the semi-resumable scenario $\delta x_j$ time units of task $T_j$ have to be re-processed after the non-availability interval. The total processing time for task $T_j$ is given by $x_j + \delta x_j + (p_j - x_j) = \delta x_j + p_j$.

In the following we base the discussion on the three field $\alpha \mid \beta \mid \gamma$ classification introduced in Chapter 3. We add some entry denoting machine availability and 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. In [Sch84] and [LS95a] different patterns of availability are discussed for the case of parallel machine systems (parameter $\alpha_3$). These are constant, zigzag, decreasing, increasing, and staircase. Let $0 = t_1 < t_2 < \cdots < t_j < \cdots < t_q$ 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 $\alpha_3 \in \{\varnothing, NC_{zz}, NC_{inc}, NC_{dec}, NC_{inczz}, NC_{deczz}, NC_{sc}, NC_{win}\}$ denotes the machine availability. *NC* relates to the *n*on-*c*ontinuous availability of the machines.

1. If all machines are continuously available ($t = 0$) then the pattern is called constant; $\alpha_3 = \varnothing$.

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 $\mathbb{N}$ the number of machines $m(t_j) \geq max_{1 \leq u \leq j-1}\{m(t_u)\}$ $(m(t_j) \leq min_{1 \leq u \leq j-1}\{m(t_u)\})$, i.e. the number of machines available in interval $[t_{j-1}, t_j)$ is not more (less) than this number in interval $[t_j, t_{j+1})$; $\alpha_3 = NC_{inc}$ $(NC_{dec})$.

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

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 Figure 11.1.1; grayed areas represent intervals of non-availability. Note that patterns (l)-(4) are special cases of (5).

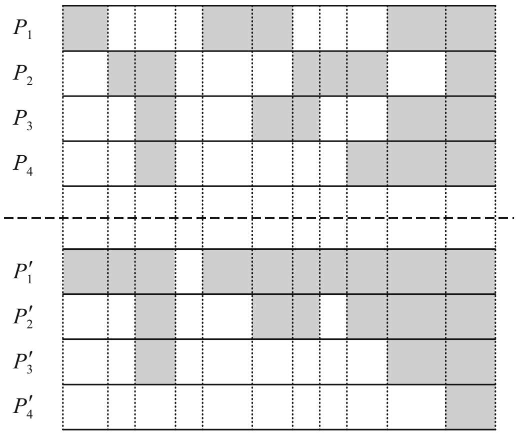


**Figure 11.1.1** *Rearrangement of arbitrary patterns.*

6. A pattern is called arbitrary if none of the conditions (l)-(5) applies; $\alpha_3 = NC_{win}$ . Such a pattern is shown in the upper part of Figure 11.1.1 for machines $P_1$, $P_2$, $P_3$, $P_4$; patterns defined in (l)-(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 or vice versa. If there are $m(t_j)$ machines available in some interval $[t_j, t_j+1)$ then after the transformation machines $P_1, \cdots, P_{m(t_j)}$ will be available in $[t_j, t_j+1)$ and $P_{m(t_j+1)}, \cdots, P_m$ will not be available, where $0 < m(t_j) < m$. Doing this for every interval we generate composite machines. Each of them consists of at most m original machines with respect to the planning horizon.

An example for such a transformation where periods of non-availability are moved from machines with greater index to machines with smaller index, considering $m = 4$ machines, is given in Figure 11.1.1 Non-availability is represented by the grayed areas. From machines $P_1$, $P_2$, $P_3$, $P_4$ composite machines $P_1'$, $P_2'$, $P_3'$, $P_4'$ are formed. Composite machines which do not have intervals of availability can be omitted from the problem description. Then the number of composite machines in each interval 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 an original machine is changing. If this number is polynomial in $n$ or $m$ machine scheduling problems with arbitrary patterns of non-availability 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.

The second field $\beta = \beta_1, \cdots, \beta_8$ describes task (job) and resource characteristics. We will only refer here to parameter $\beta_1$.

Parameter $\beta_1 \in \{\varnothing, t-pmtn, pmtn\}$ indicates the possibilities of preemption:

- $\beta_1 = \varnothing$: 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.

Here we assume that not only task ($\beta_1 = t - pmtn$) but also arbitrary (task and machine) preemptions are possible ($\beta_1 = pmtn$). If there is only one machine dedicated to each task then task preemptions and arbitrary preemptions become equivalent. For single machine and shop problems this difference has not to be considered. Of course the rearrangement of an arbitrary pattern to a staircase pattern is only used when arbitrary 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*.

The third field, $\gamma$, denotes a *single* optimality criterion (performance measure). In some recent papers *multiple* criteria scheduling models with limited machine availability are investigated, see e.g. [QBY02, LY03]. We will further investigate models with single optimality criteria.

Many of the problems considered later are solved applying simple priority rules which can be executed in $O(n\log n)$ time. The rules order the tasks in some way and then iteratively assign them to the most lightly loaded machine. The following rules as already introduced in Chapter 3 are the most prominent.

- Shortest Processing Time (*SPT*) rule. With this rule the tasks are ordered according to non-decreasing processing times.
- Longest Processing Time (*LPT*) rule. The tasks are ordered according to non-increasing processing times.
- Earliest Due Date (*EDD*) rule. Applying this rule all tasks are ordered according to non-decreasing due dates.

## 11.2 One Machine Problems

One machine problems are of fundamental character. They can be interpreted as building blocks for more complex problems. Such formulations may be used to represent bottleneck machines or an aggregation of a machine system. For one machine scheduling problems the only availability pattern which has to be investigated is a special case of zigzag with $k = 1$.

Let us consider first problems where preemption of tasks (jobs) is not allowed. If there is only a single interval of non-availability and $\sum C_j$ is the objective $(1, NC_{zz} \,||\, \sum C_j)$ [ABFR89] show that the problem is NP-hard. The Shortest Processing Time (*SPT*) rule leads to a tight relative error of $R_{SPT} \leq 2/7$ for this problem [LL92]. [SPR+05] presents a modified *SPT*-heuristic with an improved relative error of 3/17. He also develops a dynamic programming algorithm for the same problem capable of solving problem instances with up to 25000 tasks. It is easy to see that also problem $1, NC_{win} \,||\, C_{max}$ is NP-hard [Lee96].

If preemption is allowed the scheduling problem becomes easier. For $1, NC_{win} \,|pmtn|\, C_{max}$, it is obvious 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. 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 it is assumed in the rest of this section.

While the sum of completion times ($1, NC_{win} | pmtn | \sum C_j$) is minimized by the *SPT* rule the problem of minimizing the weighted sum ($1, NC_{win} | pmtn | \sum w_j C_j$) is NP-hard [Lee96]. Note that without availability constraints Smith's rule [Smi56] solves the problem. Maximum lateness is minimized by the Earliest Due Date (*EDD*) rule [Lee96]. If the number of tardy tasks has to be minimized ($1, NC_{win} | pmtn | \sum U_j$) 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 if we add release times or weights for the jobs the problem is NP-hard already for a continuously available machine ([LRB77] or [Kar72]). Details can be found in Chapter 4.

Lorigeon et al. [LBB02a] investigate a one-machine problem where each task has a release date $r_j$ and a delivery duration $q_j$. The machine is not available for processing during a single given interval. A task may only be preempted for the duration of the non-availability interval and resumed as the machine becomes available again. The objective is to find a schedule minimizing $max_j\{C_j + q_j\}$. The problem is a generalization of a well-known NP-hard problem studied by Carlier [Car82]. Lorigeon et al. provide a branch-and-bound algorithm which solves 2133 out of 2250 problems instances with up to 50 tasks.

There are also results concerning problems where an interval of non-availability is regarded as a decision variable. Qi et al. [QCT99] study a model in which the machine has to be maintained after a maximum of $\delta_1$ time units. Each such maintenance activity has a constant duration of $\delta_2$ time units. The goal is to find a non-preemptive schedule which obeys the maintenance restrictions and minimizes $\sum C_j$. The problem is proved to be NP-hard in the strong sense. Qi et al. propose heuristics and a branch-and-bound algorithm.

Graves and Lee [GL99] study several variants of the same problem. Besides processing a task requires a setup operation on the machine. If a task is preempted by an interval of non-availability an additional (second) setup is required before the processing of the task can be resumed. Maintenance activities have to be carried out after a maximum of $\delta_1$ time units. If there are at most two maintenance periods then the problem is NP-hard in the ordinary sense for the objectives $C_{max}$, $\sum C_j$, $\sum w_j C_j$, and $L_{max}$. Dynamic programming algorithms are provided to solve the problems in pseudo-polynomial time. If there is exactly one period of maintenance the problem is polynomially solvable for the objectives $\sum C_j$ (by a modification of the *SPT* rule) and $L_{max}$ (by a modification of the EDD rule). Minimizing $\sum w_j C_j$ turns out to be NP-hard in the ordinary sense. Two pseudo-polynomial time dynamic programming algorithms are provided to solve this problem.

Lee and Leon [LL01] study a problem in which a production rate modifying activity of a given duration has to be scheduled in addition to $n$ tasks. A task $T_j$ processed before the activity requires $p_j$ time units on the machine while the processing time of the same task becomes $\sigma_j p_j$ if it is scheduled after the production rate modifying activity. Preemption is not allowed. The objective is to find a starting time for the rate-modifying activity and a task sequence such that several regular functions are optimized. The problem can be solved in polynomial time for the objectives $C_{max}$ and $\sum C_j$. For the objective $\sum w_j C_j$ the authors develop pseudo-polynomial dynamic programming algorithms. For the objective $L_{max}$ the *EDD* rule is optimal for the practical case where the production rate is increased by the activity. The general case with arbitrary $\sigma_j$ is NP-hard.

## 11.3 Parallel Machine Problems

In this section we cover formulations of parallel machine scheduling problems with availability constraints.

### 11.3.1 Minimizing the Sum of Completion Times

In case of continuous availability of the machines ($P||\sum C_j$) the problem can be solved applying the *SPT* rule. 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 finite (this corresponds to a zigzag pattern of availability) the problem becomes NP-hard [LL93]. In the same paper Lee and Liman show that for $P2, NC_{ZZ}||\sum C_j$, where machine $P_2$ is continuously available and machine $P_1$ has one finish time which is smaller than infinity, the *SPT* rule with the following modification leads to a tight relative error of $R_{SPT} < 1/2$:

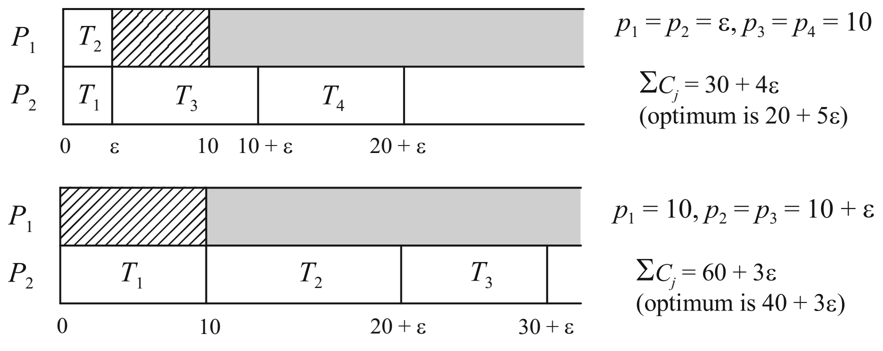
Step 1: Assign the shortest task to $P_2$.

Step 2: Assign the remaining tasks in *SPT* order alternately to both machines until some time when no other task can be assigned to $P_1$ without violating $F_1$.

Step 3: Assign the remaining tasks to $P_1$.

Figure 11.3.1 illustrates how that bound can be reached asymptotically (when $\varepsilon$ tends toward 0). In both examples, the modified *SPT* rule leads to a large idle time for machine $P_1$. For fixed $m$ the *SPT* rule is asymptotically optimal if there is no more than one interval of non-availability for each machine [Mos94].

In case there is only one interval of non-availability for each machine, the problem is NP-hard. In [LC00] a branch and bound algorithm based on the column generation approach is given which also solves the problem where $\sum w_j C_j$ is minimized.



$P_1$
$P_2$
$T_2$
$T_1$
$T_3$
$T_4$
0
ε
10
10 + ε
20 + ε
$p_1 = p_2 = \varepsilon, p_3 = p_4 = 10$
$\Sigma C_j = 30 + 4\varepsilon$
(optimum is 20 + 5ε)
$P_1$
$P_2$
$T_1$
$T_2$
$T_3$
0
10
20 + ε
30 + ε
$p_1 = 10, p_2 = p_3 = 10 + \varepsilon$
$\Sigma C_j = 60 + 3\varepsilon$
(optimum is 40 + 3ε)

**Figure 11.3.1** *Examples for the modified SPT rule.*

## 11.3.2 Minimizing the Makespan

Let us first investigate non-preemptive scheduling. J. D. Ullman [Ull75] analyses the complexity of the problem $P, NC_{win} \,||\, C_{max}$. It is NP-hard in the strong sense for arbitrary $m$ (3-partition is a special case) even if the machines are continuously available. If machines have different beginning times $B_i$ ($P, NC_{inc} \,||\, C_{max}$) the Longest Processing Time (*LPT*) rule leads to a relative error of $R_{LPT} < 1/2 - 1/(2m)$ or of $R_{MLPT} < 1/3$ if the rule is appropriately modified [Lee91]. The first bound is 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 *MLPT* rule runs in $O((n+m)\cdot log(n+m) + (n+m)\cdot m)$ time. In [LHYL97] Lee's bound of 1/3 reached by *MLPT* is improved to 1/4.

Using the bin-packing algorithm called the *MULTIFIT* it is shown in [CH98] that the bound of this algorithm is $2/7 + 2^{-k}$, where $k$ is the selected number of the major iterations in *MULTIFIT*.

Note that the *LPT* algorithm leads to a relative error of $R_{LPT} < 1/3 - 1/(3m)$ for continuously available machines [Gra69]. H. Kellerer [Kel98] presents a dual approximation algorithm using a bin packing approach leading to a tight bound of 1/4, too.

In [LSL05] a problem with two machines and one interval of non-availability is considered. For non-resumable and resumable cases the problem is solved by enumerative techniques.

Now let us investigate results for preemptive scheduling. If all machines are only available in one and the same time interval $[B, F)$ and tasks are independent the problem is of type $P|pmtn|C_{max}$ Following [McN59] it can be shown that there exists a feasible machine preemptive schedule if and only if $max_j\{p_j\} \le (F - B)$ and $\sum_j p_j \le m(F - B)$. There exists 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 $[B_i^s, F_i^s)$, $s = 1, \cdots, S_i$ and the machine system forms a staircase pattern, it is possible to generalize McNaughton's condition and show that a feasible preemptive schedule exists if and only if the following $m$ conditions are met [Sch84]:

$$\textstyle\sum_{j=1}^{k} p_j \le \sum_{i=1}^{k} PC_i \quad \forall k = 1, \cdots, m-1, \qquad (11.3.1\text{-}k)$$

$$\textstyle\sum_{j=1}^{n} p_j \le \sum_{i=1}^{m} PC_i \qquad (11.3.1\text{-}m)$$

with $p_1 \ge p_2 \ge \cdots \ge p_n$ and $PC_1 \ge PC_2 \ge \cdots \ge PC_m$, where $PC_j$ is the total processing capacity of machine $P_i$. Such a schedule can be constructed in $O(n + m{\cdot}log\, m)$ 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 (11.3.1-1) and (11.3.1-$m$) alone. This remains true for zigzag patterns as then (11.3.1-2)$, \cdots,$ (11.3.1-$m$−1) are always verified if (11.3.1-1) is true (there is one availability interval for all machines but $P_m$). The algorithm to solve the problem applies five rules which are explained now.

Let us consider two arbitrary processors $P_k$ and $P_l$ with $PC_k > PC_l$ as shown in Figure 11.3.2. Let $\Phi_k^a$, $\Phi_k^b$, and $\Phi_k^c$ denote the processing capacities of processor $P_k$ in the intervals $[B_k^1, B_l^1]$, $[B_l^1, F_l^{N(l)}]$, and $[F_l^{N(l)}, F_k^{N(k)}]$, respectively. Then obviously, $PC_k = \Phi_k^a + \Phi_k^b + \Phi_k^c$.

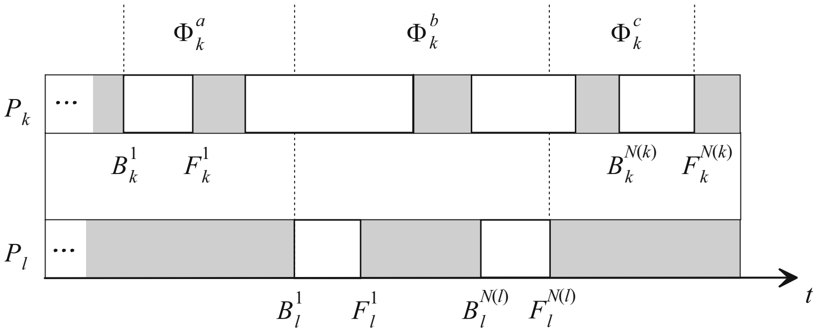


**Figure 11.3.2** *Staircase pattern for two arbitrary processors.*

Assume that the tasks are ordered according to non-increasing processing times and that the processors form a staircase pattern as defined above. All tasks $T_j$ are scheduled in the given order one by one using one of the five rules given below. Rules 1 - 4 are applied in the case where $1 \le j < m$, $p_j > \min_i \{PC_i\}$, and if there are two processors $P_k$ and $P_l$ such that $PC_l = \max_i \{PC_i \mid PC_i < p_j\}$ and $PC_k = \min_i \{PC_i \mid PC_i \ge p_j\}$. Rule 5 is used if $m \le j \le n$ or $p_j \le \min_i \{PC_i\}$. First we describe the rules, and after that we prove that their application always constructs a feasible schedule, if one exists. To avoid cumbersome notation we present the rules in a semi-formal way.

**Rule 1.** *Condition*: $p_j = PC_k$.

Schedule task $T_j$ on processor $P_k$ such that all the intervals $[B_k^r, F_k^r]$, $r = 1, \cdots, N(k)$, are completely filled; combine processors $P_k$ and $P_l$ to form a *composite processor*, denoted again by $P_k$, which is available in all free processing intervals of the original processor $P_l$, i.e. define $PC_k = PC_l$ and $PC_l = 0$.

**Rule 2.** *Condition*: $p_j - PC_l > \max\{\Phi_k^a, \Phi_k^c\}$ and $p_j - \Phi_k^b \ge \min\{\Phi_k^a, \Phi_k^c\}$.

Schedule task $T_j$ on processor $P_k$ in its free processing intervals within $[B_l^1, F_l^{N(l)}]$. If $\Phi_k^a$ (respectively $\Phi_k^c$) is minimum use all the free processing intervals of $P_k$ in $[B_k^1, B_l^1]$ ($[F_l^{N(l)}, F_k^{N(k)}]$) to schedule $T_j$, and schedule the remaining processing requirements of that task (if there is any) in the free processing intervals of $P_k$ within $[F_l^{N(l)}, F_k^{N(k)}]$ ($[B_k^1, B_l^1]$) from left to right (right to left) such that the $r$th processing interval is completely filled with $T_j$ before the $r+1$st ($r-1$st) interval is used, respectively. Combine processors $P_k$ and $P_l$ to a composite processor $P_k$ which is available in the remaining free processing intervals of the original processors $P_k$ and $P_l$, i.e. define $PC_k = PC_k + PC_l - p_j$ and $PC_l = 0$.

**Rule 3.** *Condition*: $p_j - PC_l > \max\{\Phi_k^a, \Phi_k^c\}$ and $p_j - \Phi_k^b < \min\{\Phi_k^a, \Phi_k^c\}$.

If $\Phi_k^a$ ($\Phi_k^c$) is minimum, schedule task $T_j$ on processor $P_k$ such that its free processing intervals in $[B_k^1, B_l^1]$ ($[F_l^{N(l)}, F_k^{N(k)}]$) are completely filled with $T_j$, further fill processor $P_k$ in the intervals $[B_l^r, F_l^r]$, $r = 1, \cdots, N(l)$, completely with $T_j$ and use the remaining processing capacity of $P_k$ in the interval $[B_l^1, F_l^{N(l)}]$ to schedule task $T_j$ with its remaining processing requirement such that $T_j$ is scheduled from left to right (right to left) where the $r+1$st ($r-1$st) interval is not used before the $r$th interval has been completely filled with $T_j$, respectively. After doing this there will be some time $t$ in the interval $[B_l^1, F_l^{N(l)}]$ up to (after) this time task $T_j$ is continuously scheduled on processor $P_k$. Time $t$ always exists because $p_j - \min\{\Phi_k^a, \Phi_k^c\} < \Phi_k^b$. Now move $T_j$ with its processing requirement which is scheduled after (before) $t$ on processor $P_k$ to processor $P_l$ in the corresponding time intervals. Combine processors $P_k$ and $P_l$ to a composite processor $P_k$ which is available in the remaining free processing intervals of the original processors $P_k$ and $P_l$, i.e. define $PC_k = PC_k + PC_l - p_j$ and $PC_l = 0$.

**Rule 4.** *Condition*: $p_j - PC_l \le \max\{\Phi_k^a, \Phi_k^c\}$.
Schedule task $T_j$ on processor $P_l$ such that all its intervals $[B_l^r, F_l^r]$, $r = 1, \cdots, N(l)$ are completely filled with $T_j$. If $\Phi_k^a$ ($\Phi_k^c$) is maximum, schedule task $T_j$ with its remaining processing requirement on processor $P_k$ in the free processing intervals of $[B_k^1, B_l^1]$ ($[F_l^{N(l)}, F_k^{N(k)}]$) from left to right (right to left) such that the $r$th processing interval is completely filled with $T_j$ before the $r+1$st ($r-1$st) interval is used, respectively. Combine processors $P_k$ and $P_l$ to a composite processor $P_k$ which is available in the remaining free processing intervals of the original processor $P_k$, i.e. define $PC_k = PC_k + PC_l - p_j$ and $PC_l = 0$.

**Rule 5.** *Condition*: remaining cases.
Schedule task $T_j$ and the remaining tasks in any order in the remaining free processing intervals successively from left to right starting with processor $P_k$, switch to a processor $P_i$, $i < k$ only if the $i+1$st processor is already completely filled.

To show the optimality of rules 1 - 5 one may use the following lemma and theorem [Sch84].

**Lemma 11.3.1** *After having scheduled a task $T_j$, $j \in \{1, \cdots, m-1\}$, on some processor $P_k$ according to rules* 1 *or* 2, *or on $P_k$ and $P_l$ according to rules* 3 *or* 4, *the following observations are true*:

(1) *The remaining free processing intervals of processors $P_k$ and $P_l$ are disjoint.*

(2) *Combining processors $P_k$ and $P_l$ to a composite processor $P_k$ results in a new staircase pattern.*

(3) *If all inequalities of* (11.3.1-$k$), $k = 1, \cdots, m$ *hold before scheduling task $T_j$, the remaining processing requirements and processing capacities after scheduling $T_j$ still satisfy inequalities* (11.3.1-$k$), $k = 1, \cdots, m$.

(4) *The number of completely filled or completely empty intervals is* $\sum_{i=1}^{m} N(i) - K$ *where $K$ is the number of only partially filled intervals*, $K \le j < m$. □

We are now ready to prove the following theorem. The proof is constructive and leads to an algorithm that solves our problem.

**Theorem 11.3.2** *For a system of $m$ semi-identical processors with staircase pattern of availability and a given set $\mathcal{T}$ of $n$ tasks there will always be a feasible preemptive schedule if and only if all inequalities* (11.3.1) *hold.*

*Proof.* We assume that $p_j > \min_i \{PC_i\}$ for $j = 1,\cdots,m-1$; otherwise the theorem is always true if and only if the inequality (11.3.1-$m$) holds, as can easily be seen. There always exists a feasible preemptive schedule for $T_1$. Now assume that the first $z$ tasks have been scheduled feasibly according to rules 1 - 4. We show that $T_{z+1}$ also can be scheduled feasibly:

(*i*) $1 < z < m$: after scheduling task $T_z$ all inequalities (11.3.1) hold according to Lemma 11.3.1. Then $p_{z+1} \le PC_1^z$, hence task $T_{z+1}$ can be scheduled feasibly on processor $P_1$.

(*ii*) $m \le z \le n$: after scheduling the first $m-1$ tasks using rules 1-4, $m-1$ processors are completely filled with tasks. Since $PC_2^{z-1} = PC_3^{z-1} = \cdots = PC_m^{z-1} = 0$ and $PC_1^{z-1} \ge \sum_{j=z}^{n} p_j$, task $T_z$ can be scheduled on processor $P_1$, and the remaining tasks can also be scheduled on this processor by means of rule 5. □

The following algorithm makes appropriate use of the five scheduling rules.

**Algorithm 11.3.3** *Algorithm by Schmidt* [Sch84] *for semi-identical processors.*

```
begin
Order the $m$ largest tasks $T_j$ according to non-increasing processing times and
   schedule them in the given order;
for all $i \in \{1,\cdots,m\}$ do $PC_i := \sum_{r=1}^{N(i)} PC_i^r$;
repeat
   if $j < m$ and $p_j > \min_i \{PC_i\}$
   then
      begin
      Find processor $P_l$ with $PC_l = \max_i \{PC_i \mid PC_i < p_j\}$ and processor $P_k$ with
           $PC_k = \min_i \{PC_i \mid PC_i \ge p_j\}$;
      if $PC_k = p_j$
      then call rule 1
      else
         begin
         Calculate $\Phi_k^a$, $\Phi_k^b$, and $\Phi_k^c$;
         if $p_j - PC_l > \max\{\Phi_k^a, \Phi_k^c\}$
         then
```

```
        if $p_j - \Phi_k^b \geq \min\{\Phi_k^a, \Phi_k^c\}$
        then call $rule\,2$ else call $rule\,3$;
      else call $rule\,4$;
      end;
    end
  else call $rule\,5$;
until $j = n$;
end;
```

The number of preemptions generated by the above algorithm and its complexity are estimated by the following theorems [Sch84].

**Theorem 11.3.4** *Given a system of m processors* $P_1, \cdots, P_m$ *of non-continuous availability, where each processor* $P_i$ *is available in* $N(i)$ *time intervals. Then, if the processor system forms a staircase pattern and the tasks satisfy the inequalities* (11.3.1), *Algorithm* 11.3.3 *generates a feasible preemptive schedule with at most* $(\sum_{i=1}^{m} N(i)) - 1$ *preemptions.* □

**Theorem 11.3.5** *The time complexity of Algorithm* 11.3.3 *is* $O(n + m \log m)$. □

Notice that if all processors are only available in a single processing interval and all these intervals have the staircase property the algorithm generates feasible schedules with at most $m-1$ preemptions. If we further assume that $B_i = B$ and $F_i = F$ for all $i = 1, \cdots, m$ Algorithm 11.3.3 reduces to McNaughton's rule [McN59] with time complexity $O(n)$ and at most $m-1$ preemptions.

There is a number of more general problems that can be solved by similar approaches.

(1) Consider the general problem where the intervals of $m$ semi-identical processors are arbitrarily distributed as shown in Figure 11.3.3(a) for an example problem with $m = 3$ processors. Reordering the original intervals leads to a staircase pattern which is illustrated in Figure 11.3.3(b). Now each processor $P_i'$, with $PC_i' > 0$ is a *composite processor* combining processors $P_i, P_{i+1}, \cdots, P_m$, and each interval $[B_i'^r, F_i'^r]$ is a *composite interval* combining intervals of availability of processors $P_i$, $P_{i+1}, \cdots$, $P_m$. The numbers in the different intervals of Figure 11.3.3(b) correspond to the number of original processors where that interval of availability is related to. After reordering the original intervals this way the problem consists of at most $Q' \leq Q = \sum_{i=1}^{m} N(i)$ intervals of availability. Using Algorithm 11.3.3, $O(m)$ preemptions are possible in each interval and thus $O(mQ)$ is an upper bound on the number of preemptions which will be generated for the original problem.

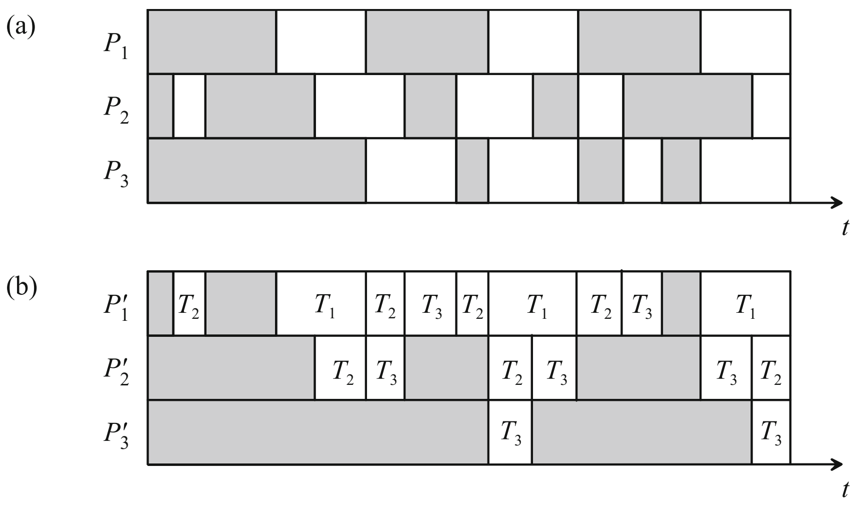


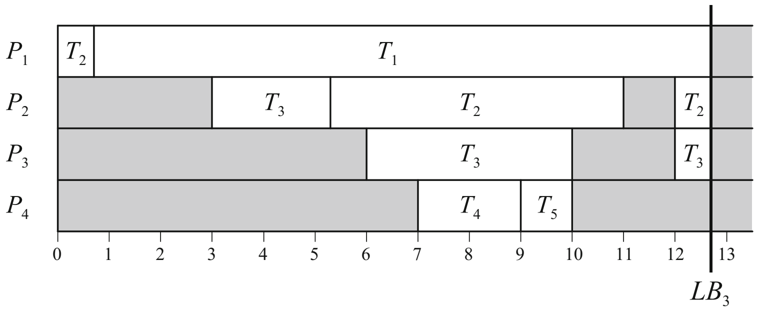
**Figure 11.3.3** *Example for arbitrary processing intervals*
**(a)** *general intervals of availability,*
**(b)** *corresponding staircase pattern.*

(2) If there is no feasible preemptive schedule for the problem at least one of the inequalities of (11.3.1) is violated; this means that the processing capacity of at least one processor is insufficient. We now increase the processing capacity in such a way that all the tasks can be feasibly processed. An *overtime cost function* might be introduced that measures the required increase of processing capacity. Assume that an increase of one time unit of processing capacity results in an increase of one unit of cost. If some inequality (11.3.1-$q$) is violated we have to increase the total capacity of the first $q$ processors by $\sum_{j=1}^{q}(p_j - PC_j)$ in case of $1 \leq q < m$; hence the processing capacity of each of the processors $P_1, \cdots, P_q$ is increased by $\frac{1}{q}\sum_{j=1}^{q}(p_j - PC_j)$. If inequality (11.3.1-$m$) is violated, the cost minimum increase of all processing capacities is achieved if the processing capacity of each processor is increased by $\frac{1}{m}(\sum_{j=1}^{n}p_j - \sum_{j=1}^{m}PC_j)$. Now Algorithm 11.3.3 can be used to construct a feasible preemptive schedule of minimum total overtime cost. Checking and adjusting the $m$ inequalities can be done in $O(m)$ time, and then Algorithm 11.3.3 can be applied. Hence a feasible schedule of minimal overtime cost can be constructed in $O(n+m\log m)$ time.

(3) If each task $T_j$ also has a deadline $\tilde{d}_j$ the problem is not only to meet start and finish times of all intervals but also all deadlines. The problem can be solved by using a similar approach where the staircase patterns and the given deadlines are considered. Since all the tasks may have different deadlines, the resulting time

complexity is $O(nm\log n)$. A detailed description of this procedure can be found in [Sch88]. It is also proved there that the algorithm generates at most $Q+m(s-1)-1$ preemptions if the semi-identical processor system forms a staircase pattern, and $m(Q+s-1)-1$ preemptions in the general case, where $s$ is the number of different deadlines. We mention that this approach is not only dedicated to the deadline problem. It can also be applied to a problem where all the tasks have different ready times and the same deadline, as these two situations are of the same structure.

The corresponding optimization problem $(P, NC_{sc} | pmtn | C_{max})$ is solved by an algorithm that first computes the lower bounds $LB_1, LB_2, \cdots, LB_m$ obtained from the conditions above (see Figure 11.3.4). $C_{max}$ cannot be smaller than $LB_k$, $k = 1, \cdots, m-1$, obtained from (11.3.1-$k$). The sum of availabilities of machines $P_1, \cdots, P_k$ during time interval $[0, LB_k)$ may not be smaller than the sum of processing times of tasks $T_1, \cdots, T_k$. The sum of all machine availabilities during time interval $[0, LB_m)$ must also be larger than or equal to the sum of processing times of all tasks. In the example of Figure 11.3.4, $C_{max} = LB_3$. The number of preemptions is $S-2$.



|  | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ |
|---|---|---|---|---|---|
| $p_j$ | 12 | 7 | 7 | 2 | 1 |
| $LB_j$ | 12 | 11 | 38/3 | 149/12 | 38/3 |

**Figure 11.3.4** *Minimizing the makespan on a staircase pattern.*

When precedence constraints are added, Liu and Sanlaville [LS95a] show that problems with chains and arbitrary patterns of non-availability (i.e. $P, NC_{win} | pmtn, chains | C_{max}$) can be solved in polynomial time applying the Longest Remaining Path (*LRP*) first 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) $(P, NC_{deczz} | pmtn, out\text{-}forest | C_{max}$ or $P, NC_{inczz} | pmtn, in\text{-}forest | C_{max})$. In case of only two machines and arbitrary (which means zigzag for $m = 2$) patterns of non-availability ($P2, NC_{win} | pmtn, prec | C_{max}$ ) 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 those obtained for unit execution time scheduling by list algorithms (see Dolev and Warmuth [DW85b, DW85a]). The *LRP* 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 machine availability changes unexpectedly, the property does not hold any more.

Applying the *LRP* rule results in a time complexity of $O(n \cdot log\, n + nm)$ and a number of preemptions of $O((n + m)^2 - nm)$ which both can be improved. Therefore in [BDF+00] an algorithm is given which solves problem $P, NC_{win} | pmtn, chains | C_{max}$ with $N < n$ chains in $O(N + m \cdot log\, m)$ 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. This result is based on the observation that preemptive scheduling of chains for minimizing schedule length can be solved by applying an algorithm for the independent tasks problem. Having more than two machines in the case of arbitrary precedence constraints or an arbitrary number of machines in the case of a tree precedence structure makes the problem NP-complete [BDF+00].

When tasks require more than one processor they are called multiprocessor tasks. In [BDDM03] polynomial algorithms are given for the following cases:

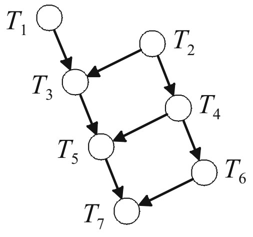
- tasks have various ready times and require either one or all processors;
- sizes of the tasks are powers of 2.

### 11.3.3 Dealing with Due Date Involving Criteria

In [Hor74] it is shown that $P | pmtn, r_j, \tilde{d}_j | -$ can be solved in $O(n^3 \cdot min\{n^2, log\, n + log\, p_{max}\})$ time. The same flow-based approach can be coupled with a bisection search to minimize 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 $O(n^3 p_{max}^3 \cdot (log\, n + log\, p_{max}))$ [San95]. When no ready times are given but due dates have to be considered, maximum lateness can be minimized for the problem $(P, NC_{win} | pmtn | L_{max})$ using the approach suggested by [Sch88] in $O(nm \cdot log\, n)$ time. The method needs to know all possible events before the next due date.

If there are not only due dates but also ready times are to be considered (problem $P, NC_{win} | r_j, \, pmtn | L_{max}$) Sanlaville [San95] suggests a nearly on-line priority algorithm with an absolute error of $A \leq (m - 1/m)p_{max}$ if the availability 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 $O(n^2 p_{max})$ time and is optimal in the case of a zigzag pattern and no release dates.

[LS95a] shows that results for $C_{max}$ minimization in cae of in-forest precedence graphs and increasing zigzag patterns ($P, NC_{inczz} | pmtn, \, \textit{in-forest} | C_{max}$) can be extended to $L_{max}$, using the *SLF* rule on the modified due dates. Figure 11.3.5 shows an optimal *SLF* schedule for the given precedence constraints.



| | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ |
|---|---|---|---|---|---|---|---|
| $p_j$ | 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 |

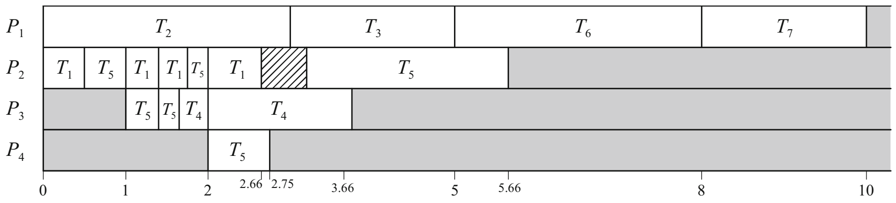


**Figure 11.3.5** *Minimizing $L_{max}$ on an increasing zigzag pattern.*

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. If there are due dates, release dates and chain precedence constraints to be considered ($P, NC_{win} | r_j, chains, pmtn | L_{max}$) the problem can be solved using a binary search procedure in combination with a linear programming formulation [BDF+00]. In case of multiprocessor tasks there exists a polynomial algorithm to minimize $L_{max}$ if the number of processors is fixed [BDDM03].

Lawler and Martel [LM89] solved the weighted number of tardy 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-polynomial algorithms ($O(\sum w_j n^2)$, or $O(n^2 p_{max})$ to minimize the number of tardy jobs). Nothing however is said about the effort needed to compute processing capacity in one interval.

If there are more than two uniform machines to be considered and the problem is to minimize maximum lateness for jobs which have different release dates ($Q, NC_{win} \mid r_j, pmtn \mid L_{max}$) the problem can be solved in polynomial time by a combined strategy of binary search and network flow [BDF+00]. In the same paper the problem is generalized taking unrelated machines, i.e. machine speeds cannot be represented by constant factors, into account. This problem can also be solved in polynomial time applying a combination of binary search and the two-phase method given in [BEP+96].

## 11.4 Shop Problems

The literature on shop scheduling problems with limited machine availability is concentrated on flow shops and open shops. We are aware of only two papers dealing with the job shop. The paper of Aggoune [Agg04b] studies the two-job special case of this problem under the makespan criterion. He proposes extensions of the well known geometric algorithm by Akers and Friedman [AF55] for problems $J, NC_{win} \mid pmtn, n = 2 \mid C_{max}$ and $J, NC_{win} \mid n = 2 \mid C_{max}$. The algorithms run in polynomial time. Braun et al. [BLS05] investigate problem $J2, NC_{win} \mid pmtn \mid C_{max}$ and derive sufficient conditions for the optimality of Jackson's rule.

### 11.4.1 Flow Shop Problems

The flow shop scheduling problem for two machines with a constant pattern of availability minimizing $C_{max}$ ($F2 \mid\mid C_{max}$ and $F2 \mid pmtn \mid C_{max}$) can be solved in polynomial time by Johnson's rule [Joh54]. C.-Y. Lee [Lee97] has shown that this problem becomes already NP-hard if there is a single interval of non-availability on one machine only. For the case where the tasks can be resumed he also gives approximation algorithms which have relative errors of 1/2 if this interval is on machine $P_1$ or of 1/3 if the interval of non-availability is on machine $P_2$. The approximation algorithms are based on a combination of Johnson's rule and a modification of the ratio rule given in [MP93]. Lee also proposes a dynamic programming algorithm for the case with one interval only.

Improved approximation algorithms for the resumable problem with one interval are presented in [CW00], [Bre04a] and [NK04]. In the first paper a 1/3-

approximation for the case with the interval on $P_1$ is presented. The second paper provides a 1/4-approximation for the case with the interval occurring on $P_2$. The third paper finally describes a fully polynomial-time approximation scheme for the general case with one interval of non-availability, no matter on which machine. Ng and Kovalyov show that these two problems are in fact symmetrical. A polynomial-time approximation scheme for the case where general preemption is allowed (not only resumability) is presented in [Bre04b].

In [KBF+02] it is shown that the existence of approximation algorithms for flow shop scheduling problems with limited machine availability is more of an exception. It is proved that no polynomial time heuristic with a finite worst case bound can exist for $F2, NC_{win} | pmtn | C_{max}$ when at least two intervals of non-availability are allowed to occur. Furthermore it is shown that makespan minimization becomes NP-hard in the strong sense if an arbitrary number of intervals occurs on one machine only. On the other hand, there always exists an optimal schedule where the permutation of jobs scheduled between any two consecutive intervals obeys Johnson's order. However, the question which jobs to assign between which intervals remains intractable.

Due to these negative results a branch and bound algorithm is developed in [KBF+02] to solve $F2, NC_{win} | pmtn | C_{max}$. The approach uses Johnson's order property of jobs scheduled between two consecutive intervals. This property helps to reduce the number of solutions to be enumerated. Computational experiments were carried out to evaluate the performance of the branch-and-bound algorithm. In the test problem instances intervals of non-availability were allowed to occur either only on $P_1$, or only on $P_2$, or on both machines. The first result of the tests was that these instances were equally difficult to solve. The second result was that the algorithm performed very well when run on randomly generated problem instances; 1957 instances out of 2000 instances could be solved to optimality within a time limit of 1000 seconds. However, it could also be shown that there exist problem instances which are much harder to solve for the algorithm. These were instances in which the processing time of a job on the second machine was exactly twice its processing time on the first machine.

In order to speed up the solution process, a parallel implementation of the branch and bound algorithm is presented in [BFKS97]. Computations have been performed on l, 2, 3, up to 8 processors. The experiment has been based on instances for which computational times of the sequential version of the algorithm were long. The maximum speed up gained was between 1.2 and 4.8 in comparison to the sequential version for 8 processors being involved in the computation.

Based on the above results in [BBF+01] constructive and improvement heuristics are designed for $F2, NC_{win} | pmtn | C_{max}$. They are empirically evaluated using test data from [KBF+02] and new difficult test data. It turned out that a combination of two constructive heuristics and a simulated annealing algorithm could solve 5870 out of 6000 easy problem instances and 41 out of 100 difficult instances. The experiments were run on a PC and the time limit to achieve this result was roughly 60 seconds per instance. The worst relative errors were 2.6%

and 44.4% above the optimum, respectively. The combination of two constructive heuristics could only solve 5812 out of 6000 easy instances and 13 out 100 difficult instances with an average computation time of 0.33 seconds and 3.96 seconds per instance, respectively. These results in [BBF+01] suggest that the heuristic algorithms are very good options for solving flow shop scheduling problems with limited machine availability.

In [Bra02] and [BLSS02] sufficient conditions for the optimality of Johnson's rule in the case of one or more intervals of non-availability (i.e. for $F2, NC_{win} | pmtn | C_{max}$) are derived. To find the results the technique of stability analysis is used and it is shown that in most cases Johnson's permutation remains optimal. These results are comparable to [KBF+02] but improve the running time for finding optimal solutions, such that instances with 10,000 jobs and 1,000 intervals of non-availability can be treated.

The non-preemptive case of the two-machine flow shop with limited machine availability is studied by [CW99]. In general, this problem is not approximable for the makespan criterion if at least two intervals of non-availability may occur. Cheng and Wang investigate the case where there are exactly two such intervals. One of them starts at the same time when the other one ends (consecutive intervals). They provide a 2/3-approximation algorithm for this problem.

[Lee99] studies the two-machine flow shop with one interval of non-availability under the semi-resumable scenario. He provides dynamic programming algorithms for this problem as well as approximation algorithms with worst case errors of 1 and 1/2, depending on whether the interval occurs on the first or on the second machine.

Quite a few papers exist on the two-machine no-wait flow shop. For constant machine availability and the makespan criterion this problem is polynomially solvable [GG64, HS96]. Espinouse et al. [EFP99, EFP01] study the case with one interval of non-availability. They show that the problem is NP-hard no matter if preemption is allowed or not, and not approximable if at least two intervals occur. They also provide approximation algorithms with a worst-case error of 1. Improved heuristics with worst-case errors of 1/2 are presented by [CL03a]. They also treat the case where each of the two machines has an interval of non-availability and these two intervals overlap. In the second paper [CL03b] provides a polynomial-time approximation scheme for this problem. [KS04] also study the case with one interval of non-availability. They provide a 1/2-approximation algorithm capable of handling the semi-resumable scenario and a 1/3-approximation algorithm for the resumable scenario. The non-preemptive $m$-machine flow shop with two intervals of non-availability on each machine and the makespan objective is studied by [Agg04a] and [AP03]. In [Agg04a] two cases are considered. In the first case, intervals of non-availability are fixed while in the second case intervals are assigned to time windows and their actual start times are decision variables. A genetic algorithm and a tabu search procedure are evaluated for test data with up to 20 jobs and 10 machines. The most important result is that flexible start times of the intervals of non-availability result in considerably shorter schedules. In [AP03] intervals of non-availability have fixed start and finish times. The proposed heuristic is based on the approach presented in [Agg04b]. The jobs in a sequence are grouped in pairs. Each pair is scheduled optimally using the algorithm in [Agg04b]. This approach is embedded into a tabu search algorithm. Experiments indicate that the heuristic is capable of finding good solutions for problem instances with up to 20 jobs.

### 11.4.2 Open Shop Problems

The literature on open shop scheduling problems (for a survey see also [BF97]) with limited machine availability is focused on the two-machine case and the objective of makespan minimization. The case with constant pattern of machine availability ($O2 \,||\, C_{max}$) can be solved in linear time by an algorithm due to [GS76].

It is essential to distinguish between two kinds of preemption. The less restrictive case is investigated by [VS95]. They use a model where the processing of a job may be interrupted and later resumed on the same machine. In the interval between interruption and resumption the job may be processed on a different machine. It is shown that under this assumption the problem is polynomially solvable even for arbitrary numbers of machines and intervals of non-availability.

In the more restrictive case the processing of a job on a machine may be interrupted by the processing of other jobs or by intervals of non-availability. In the interval between the start and the end of a task, no other task of the same job may be processed. This model is similar to the open shop with no-pass constraints as introduced by Cho and Sahni [CS81].

J. Breit [Bre00] proves that this latter problem is NP-hard even for a single interval of non-availability and not approximable within a constant factor if at least three such intervals occur. For the case with one interval there exists a pseudopolynomial dynamic programming algorithm [LBB02b] as well as a linear time approximation algorithm with an error bound of 1/3 [BSS01]. The special case in which the interval occurs at the beginning of the planning horizon is solved by a linear time algorithm due to [LP93]. M. A. Kubzin et al. [KSBS02] present polynomial-time approximation schemes for the case with an arbitrary number of intervals on one machine and a continuously available second machine, as well as for the case with exactly one interval on each machine. The non-preemptive model is studied by J. Breit et al. [BSS03]. They provide a linear time 1/3-approximation algorithm and show that the problem with at least two intervals is not approximable within a constant factor.

## 11.5 Conclusions

We reviewed results on scheduling problems with limited machine availability.

The number of results shows that scheduling with availability constraints attracts more and more researchers, as the importance of the applications is recognized. The results presented here are of various kinds. In particular, when preemption is not authorized it will logically entail NP-hardness of the problem. If one is interested in solutions for non-preemptive problems enumerative algorithms have to be applied; otherwise approximation algorithms are a good choice. Performance bounds may often be obtained, but their quality will depend on the kind of availability patterns considered. If worst case bounds cannot be found, heuristics which can only be evaluated empirically have to be applied.

Most of the results reviewed are summarized in Table 11.5.1. The table differs for a given problem type between performance criteria entailing NP-hardness and those for which a polynomial algorithm exists.

| Problem | Polynomially solvable | NP-hard |
|---|---|---|
| $1, NC_{win}$ | | $\sum C_j$, $C_{max}$ |
| $1, NC_{win} \mid pmtn$ | $\sum C_j$, $C_{max}$, $L_{max}$, $\sum U_j$ | $\sum w_j C_j$, $\sum w_j U_j$ (*constant availability*) |
| $P, NC_{inc}$ | $\sum C_j$ | |
| $P, NC_{zz}$ | | $\sum C_j$ |
| $P2, NC_{win} \mid pmtn, prec$ | $C_{max}$, $L_{max}$ | |
| $P, NC_{zz} \mid pmtn, tree$ | $C_{max}$, $L_{max}$ (*in-tree*) | $C_{max}$ (*for* $NC_{win}$) |
| $P, NC_{win} \mid pmtn, chains$ | $C_{max}$, $L_{max}$ | |
| $P, NC_{win} \mid pmtn, r_j$ | $C_{max}$, $L_{max}$ | |
| $Q, NC_{win} \mid pmtn, r_j$ | $C_{max}$, $L_{max}$ | |
| $F2, NC_{win} \mid pmtn$ | | $C_{max}$ (*single non-availability interval*) |
| $O, NC_{win} \mid pmtn$ | $C_{max}$ | |
| $O2, NC_{win} \mid pmtn(no\text{-}pass)$ | | $C_{max}$ (*single non-availability interval*) |
| $J, NC_{win} \mid n = 2$ | $C_{max}$ | |
| $J, NC_{win} \mid pmtn, n = 2$ | $C_{max}$ | |

**Table 11.5.1** *Summary of results.*

There are many interesting fields for future research.

1. As our review indicates there are many open questions in shop scheduling, e.g., for job shop models comparatively few results are available.

2. Stability analysis introduces sufficient conditions for schedules to be optimal in the case of machine availability restrictions. Extensions to open shops and job shops seem to be interesting in this field.

3. In almost all papers reviewed in this chapter machine availability restrictions are regarded as problem input. There are, however, many cases in which decision

makers have some influence on these restrictions. For example, one may think of a situation where the start time of a maintenance activity for a machine can be chosen within certain limits. In such situations machine availability restrictions become decision variables.

4. To the best of our knowledge there exist no papers dealing with limited machine availability and multiple objective functions. Such models may, however, be very interesting, especially in cases where machine availability restrictions are decision variables. For example, in a case where several machines have to undergo a maintenance activity it may be desirable to minimize the time span between start of the first and end of the last activity while a different objective function is applied for the task scheduling.

5. There are many practical cases where periods of non-availability are not known in advance. In these cases we might apply online scheduling. Some results are already available. In [AS01] it is shown that there are instances where no on-line algorithm can construct optimal makespan schedules if machines change availability at arbitrary points in time. It is also impossible for such an algorithm to guarantee that the solution is within a constant ratio $c$ if there may be time intervals where no machine is available. Albers and Schmidt also report that things look better if the algorithm is allowed to be nearly on-line. In such a case we assume that the algorithm always knows the next point in time when the set of available machines changes. Now optimal schedules can be constructed. The algorithm presented has a running time of $O(qn + S)$, where $q$ is the number of time instances where the set of available machines changes and $S$ is the total number of intervals where machines are available. If at any time at least one machine is available, an on-line algorithm can construct schedules which differ by an absolute error $c$ from an optimal schedule for any $c > 0$. This implies, that not knowing machine availabilities does not really hurt the performance of an algorithm, if arbitrary preemptions are allowed.

## References

ABFR89 I. Adiri, J. Bruno, E. Prostig, A. H. G. Rinnooy Kan, Single machine flow-time scheduling with a single breakdown, *Acta Inform*. 26, 1989, 679-696.

AF55 S. B. Akers, J. Friedman, A non-numerical approach to production scheduling Problems, *Oper. Res*. 3, 1955, 429-442.

Agg04a R. Aggoune, Minimizing the makespan for the flow shop scheduling problem with availability constraints, *Eur. J. Oper. Res*. 153, 2004, 534-543.

Agg04b R. Aggoune, Two-job shop scheduling problems with availability constraints, *Proceedings of the 14th International Conference on Automated Planning and Scheduling,* AAAI Press, 2004, 253-259.

AP03 R. Aggoune, M.-C. Portmann, Flow shop scheduling problem with limited machine availability: a heuristic approach, *International Conference on Industrial Engineering and Production Management*, l, 2003, 140-149.

| | |
|---|---|
| AS01 | S. Albers, G. Schmidt, Scheduling with unexpected machine breakdowns. *Discret Appl. Math*. 110, 2001, 85-99. |
| BBF+01 | J. Blazewicz, J. Breit, P. Formanowicz, W. Kubiak, G. Schmidt, Heuristic algorithms for the two-machine flowshop with limited machine availability, *Omega-Int. J. Manage. Sci.* 29, 2001, 599-608. |
| BDDM03 | J. Blazewicz, P. Dell'Olmo, M. Drozdowski, P. Maczka, Scheduling multiprocessor tasks on parallel processors with limited availability, *Eur. J. Oper. Res.* 149, 2003, 377-389. |
| BDF+00 | J. Blazewicz, M. Drozdowski, P. Formanowicz, W. Kubiak, G. Schmidt, Scheduling preemptable tasks on parallel processors with limited availability, *Parallel Comput*. 26, 2000, 1195-1211. |
| BEP+96 | J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, J. Weglarz, *Scheduling Computer and Manufacturing Processes*, Springer, Berlin, 1996. |
| BF97 | J. Blazewicz, P. Formanowicz, Scheduling jobs on open shops with limited machine availability, *Rairo-Oper. Res*. 36, 1997, 149-156. |
| BFKS97 | J. Blazewicz, P. Formanowicz, W. Kubiak, G. Schmidt, A note on a parallel branch and bound algorithm for the flow shop problem with limited machine availability, Working paper, Poznan University of Technology, Poznan, 1997. |
| BLS05 | O. Braun, N. M. Leshchenko, Y. N. Sotskov, Optimality of Jackson's permutations with respect to limited machine availability, *Int. Trans. Oper. Res.* 13, 2006, 59-74. |
| BLSS02 | O. Braun, T.-C. Lai, G. Schmidt, Y. N. Sotskov, Stability of Johnson's schedule with respect to limited machine availability, *Int. J. Prod. Res.* 40, 2002, 4381-4400. |
| Bra02 | O. Braun, *Scheduling Problems with Limited Available Processors and Limited Number of Preemptions*, Ph.D. thesis, Saarland University, 2002 (in German). |
| Bre00 | J. Breit, *Heuristic Scheduling Algorithms for Flow Shops and Open Shops with Limited Machine Availability*, Ph.D. thesis, Saarland University, 2000 (in German). |
| Bre04a | J. Breit, An improved approximation algorithm for two-machine flow shop scheduling with an availability constraint, *Inf. Process. Lett*. 90, 2004, 273-278. |
| Bre04b | J. Breit, A polynomial-time approximation scheme for the two-machine flow shop scheduling problem with an availability constraint, *Comput. Oper. Res.* 33, 2006, 2143-2153. |
| BSS01 | J. Breit, G. Schmidt, V. A. Strusevich, Two-machine open shop scheduling with an availability constraint, *Oper. Res. Lett.* 29, 2001, 65-77. |
| BSS03 | J. Breit, G. Schmidt, V. A. Strusevich, Non-preemptive two-machine open shop scheduling with non-availability constraints, *Math. Meth. Oper. Res.* 57, 2003, 217-234. |
| Car82 | J. Carlier, The one machine sequencing problem. *Eur. J. Oper. Res*. 11, 1982, 42-47. |

CH98 S.-Y. Chang, H.-C. Hwang, The worst-case analysis of the multifit algorithm for scheduling nonsimultaneous parallel machines, Working paper, Department of Industrial Engineering, Pohang University of Science and Technology, 1998.

CL03a T.-C. E. Cheng, Z. Liu. 3/2-approximation for two-machine no-wait flowshop scheduling with availability constraints, *Inf. Process. Lett*. 88, 2003, 161-165.

CL03b T.-C. E. Cheng, Z. Liu. Approximability of two-machine no-wait flowshop scheduling with availability constraints, *Oper. Res. Lett*. 31, 2003, 319-322.

CS81 Y. Cho, S. Sahni, Preemptive scheduling of independent jobs with release and due dates times on open, flow and job shop, *Oper. Res*. 29, 1981, 511-522.

CW99 T.-C. E. Cheng, G. Wang, Two-machine flowshop scheduling with consecutive availability constraints, *Inf. Process. Lett*. 71, 1999, 49-54.

CW00 T.-C. E. Cheng, G. Wang, An improved heuristic for two-machine flowshop scheduling with an availability constraint, *Oper. Res. Lett*. 26, 2000, 223-229.

DW85a D. Dolev, M. K. Warmuth, Profile scheduling of opposing forests and level orders, *SIAM J. Algebra. Discr.* 6, 1985, 665-687.

DW85b D. Dolev, M. K. Warmuth, Scheduling flat graphs, *SIAM J. Comput.* 14, 1985, 638-657.

EFP99 M. L. Espinouse, P. Formanowicz, B. Penz, Minimzing the makespan in the two-machine no-wait flow-shop with limited machine availability, *Comput. Ind. Eng*. 37, 1999, 497-500.

EFP01 M. L. Espinouse, P. Formanowicz, B. Penz, Complexity results and approximation algorithms for the two machine no-wait flow-shop with limited machine availability, *J. Oper. Res. Soc*. 52, 2001, 116-121.

GG64 P. C Gilmore, R. E. Gomory, Sequencing a one-state variable machine: a solvable case of the traveling salesman problem, *Oper. Res.* 12, 1964, 655-679.

GGN00 M. Gourgand, N. Grangeon, S. Norre, A review of the stochastic flow-shop scheduling problem, *Journal of Decision Systems* 9, 2000, 183-213.

GJ79 M. R. Garey, D. S. Johnson, *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

GL99 G. H. Graves, C.-Y. Lee, Scheduling maintenance and semi-resumable jobs on a single machine, *Nav. Res. Log.* 46, 1999, 845-863.

Gra69 R. L. Graham, Bounds on multiprocessing timing anomalies, *SIAM J. Appl. Math.* 17, 1969, 416-429.

GS76 T. Gonzalez, S. Sahni, Open shop scheduling to minimize finish time, *J. ACM* 23, 1976, 665-679.

Hor74 W. A. Horn, Some simple scheduling algorithms, *Nav. Res. Log.* 21, 1974, 177-185.

HS96 N. G. Hall, C. Sriskandarajah, A survey of machine scheduling problems with blocking and no-wait in process, *Oper. Res*. 44, 1996, 510-525.

Joh54 S. M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Nav. Res. Logist. Quart.* 1, 1954, 61-68.

Kar72 R. M. Karp, Reducibility among combinatorial problems, in: R. E. Miller, J. W. Thatcher (eds.), *Complexity of Computer Computations*, 1972, 85-103.

KBF+02 W. Kubiak, J. Blazewicz, P. Formanowicz, J. Breit, G. Schmidt, Two-machine flow shops with limited machine availability, *Eur. J. Oper. Res.*136, 2002, 528-540.

Kel98 H. Kellerer, Algorithms for multiprocessor scheduling with machine release time, *IIE Trans*. 31, 1998, 991-999.

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

KS04 M. A. Kubzin, V. A. Strusevich, Two-machine flow shop no-wait scheduling with a nonavailability interval, *Nav. Res. Log*. 51, 2004, 613-631.

KSBS02 M. A. Kubzin, V. A. Strusevich, J. Breit, G. Schmidt, Polynomial-time approximation schemes for the open shop scheduling problem with non-availability constraints, Paper 02/IM/100, School of Computing and Mathematical Science, University of Greenwich, 2002.

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

LBB02a T. Lorigeon, J.-C. Billaut, J.-L. Bouquard, Availability constraint for a single machine problem with heads and tails, *Proceedings of the 8th International Workshop on Project Management and Scheduling*, 2002, 240-243.

LBB02b T. Lorigeon, J.-C. Billaut, J.-L. Bouquard, A dynamic programming algorithm for scheduling jobs in a two-machine open shop with an availability constraint, *J. Oper. Res. Soc*. 53, 2002, 1239-1246.

LC00 C.-Y. Lee, Z.-L. Chen, Scheduling jobs and maintenance activities on parallel machines, *Nav. Res. Log*. 47, 2000, 145-165.

Lee91 C.-Y. Lee, Parallel machine scheduling with non-simultaneous machine available time, *Discret Appl. Math*. 30, 1991, 53-61.

Lee96 C.-Y. Lee, Machine scheduling with an availability constraint, *J. Global Optim*. 9, 1996, 363-384.

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

Lee99 C.-Y. Lee, Two-machine flowshop scheduling with availability constraints, *Eur. J. Oper. Res*. 114, 1999, 420-429.

Lee04 C.-Y. Lee, Machine scheduling with availability constraints, in: J. Y.-T. Leung (ed.), *Handbook of Scheduling*, Chapman & Hall/CRC Press, 2004, 22.1-22.13.

LHYL97 G. Lin, Y. He, Y. Yao, H. Lu, Exact bounds of the modified LPT algorithm applying to parallel machines scheduling with nonsimultaneous machine available times, *Applied Mathematics Journal of Chinese Universities* 12, 1997, 109-116.

Lim91 S. Liman, *Scheduling with Capacities and Due-Dates*, Ph.D. thesis, University of Florida, 1991.

LL92 C.-Y. Lee, S. D. Liman, Single machine flow-time scheduling with scheduled maintenance, *Acta Inform*. 29, 1992, 375-382.

LL93 C.-Y. Lee, S. D. Liman, Capacitated two-parallel machine scheduling to minimize sum of job completion time, *Discret Appl. Math*. 41, 1993, 211-222.

LL01 C.-Y. Lee, V. J. Leon, Machine scheduling with a rate-modifying activity, *Eur. J. Oper. Res*. 128, 2001, 119-128.

LLLR79 J. Labetoulle, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Preemptive scheduling of uniform machines subject to due dates, Technical Paper B W 99/79, Centrum Wiskunde & Informatica, Amsterdam, 1979.

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

LP93 L. Lu, M. E. Posner, An NP-hard open shop scheduling problem with polynomial average time complexity, *Math. Oper. Res*. 18, 1993, 12-38.

LRB77 J. K. Lenstra, A. H. G. Rinnooy Kan, P. Brucker, Complexity of processor scheduling problems, *Annals of Discrete Mathematics* 1, 1977, 343-362.

LS95a Z. Liu, E. Sanlaville, Preemptive scheduling with variable profile, precedence constraints and due dates, *Discret Appl. Math*. 58, 1995, 253-280.

LS95b Z. Liu, E. Sanlaville, Profile scheduling of list algorithms, in: P. Chretienne et al. (eds.) *Scheduling Theory and its Applications*, Wiley, 1995, 91-110.

LS97 Z. Liu, E. Sanlaville, Stochastic scheduling with variable profile and precedence constraints, *SIAM J. Comput*. 26, 1997, 173-187.

LSL05 C.-J. Liao, D.-L. Shyur, C.-H. Lin, Makespan minimization for two parallel machines with an availability constraint, *Eur. J. Oper. Res.* 160, 2005, 445-456.

LY03 C.-Y. Lee, G. Yu, Logistics scheduling under disruptions, working paper, Department of Industrial Engineering and Engineering Management, The Hong Kong University of Science and Technology, Hong Kong, 2003.

MC70 R. Muntz, E. G. Coffman, Preemptive scheduling of real-time tasks on multiprocessor systems, *J. ACM* 17, 1970, 324-338.

McN59 R. McNaughton, Scheduling with deadlines and loss functions, *Manage. Sci*. 6, 1959, 1-12.

Moo68 J. M. Moore, An *n* job one machine sequencing algorithm for minimizing the number of late jobs, *Manage. Sci*. 15, 1968, 102-109.

Mos94 G. Mosheiov, Minimizing the sum of job completion times on capacitated parallel machines, *Math. Comput. Model.* 20, 1994, 91-99.

MP93 T. E. Morton, D. W. Pentico, *Heuristic Scheduling Systems*, J. Wiley, New York, 1993.

NK04 C. T. Ng, M. Y. Kovalyov, An FPTAS for scheduling a two-machine flowshop with one unavailability interval, *Nav. Res. Logist*. 51, 2004, 307-315.

QBY02 X. Qi, J. F. Bard, G. Yu, Disruption management for machine scheduling: the case of SPT schedules, Working paper, Department of Management Science and Infomation Systems, College of Business Administration, The University of Texas, 2002.

| | |
|---|---|
| QCT99 | X. Qi, T. Chen, F. Tu, Scheduling the maintenance on a single machine, *J. Oper. Res. Soc*. 50, 1999, 1071-1078. |
| SPR+05 | C. Sadfi, B. Penz, C. Rapine, J. Blazewicz, P. Formanowicz, An improved approximation algorithm for the single machine total completion time scheduling problem with availability constraints, *Eur. J. Oper. Res.* 161, 2005, 3-10. |
| San95 | E. Sanlaville, Nearly on line scheduling of preemptive independent tasks, *Discret Appl. Math.* 57, 1995, 229-241. |
| Sch84 | G. Schmidt, Scheduling on semi-identical processors, *Zeitschrift für Operations Research* A28, 1984, 153-162. |
| Sch88 | G. Schmidt, Scheduling independent tasks with deadlines on semi-identical processors, *J. Oper. Res. Soc*. 39, 1988, 271-277. |
| Sch00 | G. Schmidt, Scheduling with limited machine availability, *Eur. J. Oper. Res*. 121, 2000, 1-15. |
| Smi56 | W. E. Smith, Various optimizers for single-stage production, *Nav. Res. Log. Quart.* 3, 1956, 59-66. |
| SS98 | E. Sanlaville, G. Schmidt, Machine scheduling with availability constraints, *Acta Inform*. 35, 1998, 795-811. |
| ST95 | D. D. Sleator, R. E. Tarjan, Amortized efficiency of list update and paging rules, *Commun. ACM* 28, 1995, 202-208. |
| U1175 | J. D. Ullman, NP-complete scheduling problems*, J. Comput. Syst. Sci*. 10, 1975, 384-393. |
| VS95 | G. Vairaktarakis, S. Sahni, Dual criteria preemptive open-shop problems with minimum makespan, *Nav. Res. Log*. 42, 1995, 103-121. |