On scheduling with the non-idling constraint

Philippe Chrétienne

On scheduling with the non-idling constraint*

Philippe Chrétienne‡

February 4, 2014

Abstract

In this paper, we give an overview of the main results obtained on the complexity of scheduling under the non-idling constraint, i.e, when the jobs assigned to each machine must be processed with no intermediate delay. That constraint is met in practice when the cost of intermediate idle time is too high due to the idle time itself and/or the machine restarting. The non idling constraint is a strong constraint that often needs a new solving approach and most results about classical scheduling problems do not easily extend to the non-idling variant of the problem. In this survey, we mainly consider the non-idling variants of the basic scheduling problems. So, we first present basic properties, complexity results and some algorithms concerning the one-machine non-idling scheduling problem. Then we consider the m-machine non idling scheduling problem. We show that a few basic problems may be solved by rather easy extensions of the algorithm solving their classical counterpart. However, the complexity status of the non idling version of quite easy polynomial basic problems remains an open question. We finally consider a more constrained version of non-idling, called the "homogeneously non idling" constraint, where for any subset of machines, the union of their busy intervals must make an interval and we present the structural property that leads to a polynomial algorithm for unit time jobs and a weak precedence. We conclude by giving some research directions that seem quite interesting to study both for theoretical and practical issues.

1 Introduction

This paper is an updated version of [1]. We consider scheduling problems where a set of jobs $\{J_1, \cdots, J_n\}$ must be processed on a set $\{M_1, \cdots, M_m\}$

*This is an updated version of the paper that appeared in 4OR, 12(2), 101-121 (2014)

†Sorbonne Universités, UPMC Univ Paris 06, F-75005, Paris, France

‡CNRS, UMR 7606, LIP6, F-75005, Paris, France

of identical machines. At any time, a machine cannot process more than one job and a job cannot be processed by more than one machine. Moreover at most m jobs may be processed at the same time. Each job J_i has a processing time p_i, a *release time* r_i (its execution cannot start before r_i) and may also have a *deadline* d_i. If d_i is a *hard* deadline, then the execution of J_i must not end after d_i while if d_i is a *soft* deadline, J_i may end after d_i but, in that case, J_i is late and penalized in the objective function. In addition to the release dates and deadlines, the jobs may be constrained by a set $\prec$ of precedences so that if (J_i, J_j) is such a precedence, the execution of J_j cannot start before the end of J_i. A *schedule* assigns each job a machine and a starting time on that machine so that all the constraints are satisfied. The quality of a schedule is measured by an *objective function* (e.g., the makespan, the maximum lateness,...) that must be minimized. In general, Φ is a function of the completion times of the jobs. Φ is a *regular objective function* if it non decreasing with respect to the completion time of each job. A schedule is *optimal* if the corresponding value of the objective function is minimum over the set of schedules.

The scheduling problem just described is denoted by $P|p_j, r_j, d_j, \prec|\Phi$ where the first field describe the resource constraints (here P means that the number of identical machines is a parameter of the problem), the second field describes the job constraints (here the release dates, the deadlines, and the precedence constraints) and the third field Φ is the objective function. In the particular case of a decision problem when the objective is to decide whether there exists at least one schedule, Φ is replaced by the symbol $-$.

Most studies concerning the scheduling problem $P|r_j, d_j, \prec|\Phi$ assume that no cost is incurred when a machine waits between the completion of a job and the start of the next job. Moreover, it is well-known that such waiting delays are often necessary to get optimality. This is the key feature why list algorithms, that do not allow a machine to wait for a more urgent job, do not generally provide optimal schedules. However, in some applications such as those described in [6], the cost of making a running machine stop and restart later is so high that a non-idling constraint is put on the machine so that only schedules without any intermediate delays are required.

A simple example could be the following. Let us suppose that n products $P_1, \cdots, P_n$ are respectively delivered in n distinct warehouses $W_1, \cdots, W_n$ at times $r_1, \cdots, r_n$. These products must be collected by a vehicle so that the vehicle must not wait for the arrival of any of these products. If the sum of the loading time of P_i and of the travel time from W_i to W_{i+1} is p_i, then the problem is to find a non-idling schedule of the sequence of jobs $(J_1, \cdots, J_n)$ on a single machine (here the vehicle) where the parameters of

job J_i are r_i and p_i. Indeed, a schedule must be such that, when the vehicle arrives at any warehouse W_i, product P_i must be already there. So, when arriving at warehouse W_i, the vehicle must immediately load P_i and then restart to W_{i+1}. In other words, we search for a non idling schedule. As will be shown in the paper, this problem is simple since it consists in finding a starting time of the sequence $(T_1, \cdots, T_n)$ such that these tasks may be scheduled without any intermediate idling time. In this example, the total travel time of any schedule is $\sum_{i=1}^{n} p_i$ and no objective function has been mentionned. One could be the makespan (this problem is simple and solved in the paper). Another (much more difficult to handle in the general case) could be $\sum_{i=1}^{n} f_i(C_i)$ where C_i is the completion time of task T_i and where $f_i(t)$ is a cost function attached to the completion time of task T_i.
Problems concerning power management policies may also yield similar scheduling problems [7] where for example each idling period has a cost and the total cost has to be minimized [2]. Note that the non-idling constraint will not necessarily ensure full machine utilization (i.e: no idling period from time 0 to the completion time of the last job of the machine) but will remove the cost of machine re-starts, maybe at the price of processing the first job of each machine later.

Contrary to the well-known no-wait constraint in shop scheduling where no idle time is allowed between the successive operations of a same job, the non-idling machine constraint has just begun to receive research attention in the literature. To the best of our knowledge, the first work on such problems concerns the earliness-tardiness single-machine scheduling problem with no unforced idle time, where a Branch and Bound approach has been developed [5]. More recently, some aspects of the impact of the non-idling constraint on the complexity of single-machine scheduling problems as well as the important role played by the earliest starting time of a non-idling schedule has been studied in [3]. Moreover, in [4] and [9], exact methods have been designed to solve the basic one-machine non-idling problem and in [10], approximation algorithms have been developped for the non-idling single-machine scheduling problem with release and delivery times.

2 The non-idling single-machine problem

Let $\Pi = 1|r_j, prec|\Phi$ be a one-machine scheduling problem with jobs set $J = \{J_1, \cdots, J_n\}$ where:

- the release dates r_j are compatible with *prec* (i.e: if job J_i precedes job J_j, then $r_j \geq r_i + p_i$) and satisfy $r_1 \leq \cdots \leq r_n$;

- Φ is a regular objective function;
- all numerical data are positive integers.

For any $K \subset \{1, \cdots, n\}$, the sum $\sum_{j \in K} p_j$ is denoted by P_K.
Let $\sigma = (J_{i_1}, \cdots, J_{i_n})$. The sequence σ is said to be *feasible* if there is at least one feasible schedule with sequence σ. If σ is feasible, we denote by $e(\sigma)$ the earliest feasible schedule with sequence σ. Since Φ is regular, $e(\sigma)$ is also the best feasible schedule with sequence σ. Finally we denote by Σ the set of the feasible sequences and by Σ^* the subset of the optimal sequences of Σ.
The *non-idling* scheduling problem Π_{NI} associated with Π is denoted by $1, NI|r_j, prec|\Phi$ in the 3-field notation. Π_{NI} has exactly the same instances as Π but, for each instance, the candidate schedules are those feasible schedules whose jobs are processed on the machine *with no intermediate delay*.
In the *non preemptive* case, a sequence $\sigma = (J_{i_1}, \cdots, J_{i_n})$ is said to be *NI-feasible* if there is at least one non-idling feasible schedule with sequence σ. Given a NI-feasible sequence σ, we denote by $e_{NI}(\sigma)$ the *earliest non-idling* feasible schedule with sequence σ. Since Φ is regular, $e_{NI}(\sigma)$ is also the best non-idling feasible schedule with sequence σ.
It is easily seen that the starting time $\alpha_{NI}(\sigma)$ of $e_{NI}(\sigma)$ is as follows:

$$\alpha_{NI}(\sigma) = \max\{0, \max_{k \in \{1,\cdots,n\}} \{r_{i_k} - \sum_{q=1}^{k-1} p_{i_q}\}$$

Moreover, as illustrated in Figure 1, we got in this case $e_{NI}(\sigma)$ from $e(\sigma)$ simply by right-shifting all the blocks of $e(\sigma)$ except the last one.

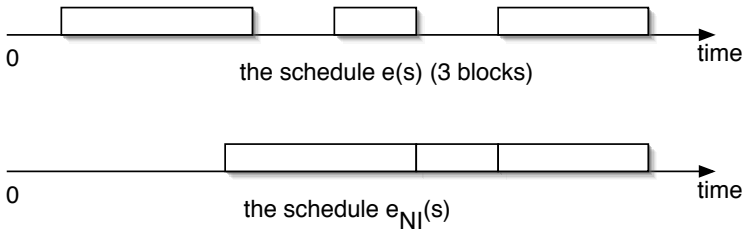

Figure 1: From $e(\sigma)$ to $e_{NI}(\sigma)$

We denote by Σ_{NI} the set of the NI-feasible sequences and by Σ^*_{NI} the subset of the optimal sequences of Σ_{NI}.

Finally, α_{NI} and β_{NI} will respectively denote the earliest starting time of a non-idling feasible schedule and the earliest starting time of an optimal non-idling feasible schedule, whenever these schedules exist.

2.1 Complexity aspects

Let us first observe that $1, NI|r_j, d_j|-$ is NP-complete in the strong sense since in the well-known reduction from 3-PARTITION, the feasible schedules of the instance of $1|r_j, d_j|-$ are non-idling [**?**].
The same argument may be used from [8] to show that the decision versions of $1, NI|r_j|L_{max}$ and $1, NI|r_j|\sum C_j$ are respectively NP-complete and NP-complete in the strong sense.
Nevertheless it is not always true that the non-idling version of an NP-complete single-machine scheduling problem is also NP-complete. For example, deciding whether non-preemptive jobs may be scheduled within given machine availability time intervals, is NP-complete while its non-idling variant is clearly polynomial.

2.2 The earliest starting time of a non-idling schedule

We know from Section 2.1 that the question whether a non-idling schedule exists is an NP-complete problem. In fact, when Π has no hard deadlines or if the time windows are regular (i.e: $(i,j) \in prec \Rightarrow (r_i \leq r_j)$ and $(d_i \leq d_j)$), then there is at least one non-idling schedule and α_{NI} is easy to compute.

Property 1. *Assume* $\Pi = 1, NI|r_j, prec|\Phi$ *either has no hard deadlines or has regular time windows and at least one NI-feasible sequence. Then* $\alpha_{NI} = \alpha_{NI}(J_1, \cdots, J_n)$.

Proof. In both cases, we know that $\Sigma_{NI} \neq \emptyset$. Let σ be a sequence of Σ_{NI} such that the two consecutive jobs J_i and J_j satisfy $i > j$. Since J_i is not an ascendant of J_j in the precedence graph, we may exchange these two jobs in the schedule $e_{NI}(\sigma)$ and get a feasible non-idling schedule starting at time $\alpha_{NI}(\sigma)$ and whose sequence σ' is obtained by exchanging the jobs J_i and J_j in σ. Since $e_{NI}(\sigma')$ is the earliest non-idling schedule with sequence σ', we get that $\alpha_{NI}(\sigma') \leq \alpha_{NI}(\sigma)$. After $O(n^2)$ such exchanges, we get that $(J_1, \cdots, J_n))$ is a NI-feasible sequence and that $\alpha_{NI}((J_1, \cdots, J_n)) \leq \alpha_{NI}(\sigma)$. □

Remark 1. *When* Π *has no hard deadlines,* α_{NI} *may be computed in polynomial time. So any problem* $1, NI|prec, r_i|f(C_{max})$*, where* f *is non-*

decreasing, is polynomial. In that case the non-idling schedule $e_{NI}(J_1, \cdots, J_n)$ is optimal.

2.3 Some specificities of non-idling scheduling

Let Π be a single-machine scheduling problem such that $\Sigma_{NI} \neq \emptyset$. Figure 2 illustrates a situation where $\beta_{NI} > \alpha_{NI}$ for a small instance with 2 jobs and the cost function $C_1 + 3C_2$.

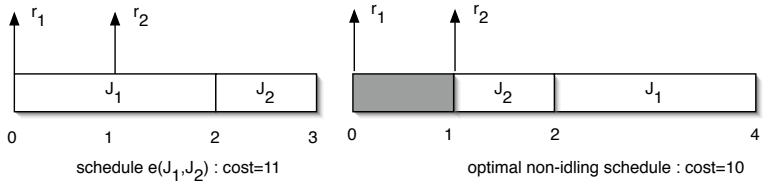

Figure 2: $\beta_{NI} > \alpha_{NI}$

We now give an example showing that a sequence of Σ^* may not be a sequence of Σ^*_{NI}. Consider the instance of $1, NI|r_j, p_j, q_j|\max(C_j + q_j)$ where q_j is the tail of job J_j and whose data are given in the array below:

i	1	2	3	4	5	6
p_i	5	3	4	3	5	3
r_i	0	10	11	11	20	30
q_i	40	7	26	24	31	8

Figure 3 shows first the schedule of Σ we get using the Jackson's rule. This schedule is optimal since its cost is 56 and in any schedule we have $C_5 + q_5 \geq 56$. So the sequence $\pi = (J_1, J_2, J_3, J_4, J_5, J_6)$ is in Σ^*. The next schedule is $e_{NI}(\pi)$ whose cost is 61 and where it clearly appears that scheduling job J_2 at time 15 is not a good decision since jobs J_3 and J_4 are also ready at that time but have greater latency times. The last schedule is $e_{NI}(J_1, J_3, J_5, J_4, J_2, J_6)$, which is an optimal non-idling schedule since its cost is 56.

Before introducing a third example, let us denote by $I(u)$ $(u \geq 0)$ the instance of Π with the same parameter values as the initial instance (say I) except for the release dates $r_j(u)$ which are defined by:

$$\forall j \in \{1, \cdots, n\}, \quad r_j(u) = \max\{r_j, l_j + u\}$$

where l_j is the value of longest path ending at J_j in the precedence graph of I. The feasible schedules of $I(u)$ are the schedules of I whose starting time is

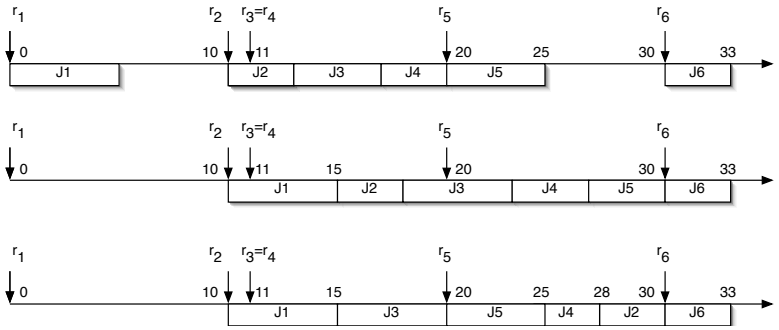

Figure 3: An instance such that $S^* \cap S^*_{NI} = \emptyset$

at least u and the notations $\Sigma(u), \Sigma_{NI}(u), \Sigma^*(u), \Sigma^*_{NI}(u), \alpha(\sigma, u)$, $\alpha_{NI}(\sigma, u)$, $e(\sigma, u)$ and $e_{NI}(\sigma, u)$ are defined in the same way as $\Sigma, \Sigma_{NI}, \Sigma^*, \Sigma^*_{NI}, \alpha(\sigma)$, $\alpha_{NI}(\sigma)$, $e(\sigma)$ and $e_{NI}(\sigma)$ for instance I but apply to the instance $I(u)$. What the following example shows is that if u^- is the smallest u such that there is a sequence σ in $\Sigma^*(u^-)$ such that the schedule $e(\sigma, u^-)$ is non-idling, then this schedule is not necessarily an optimal non-idling schedule.
Let us consider the instance described by the following array where the objective function is $f_a(C_a) + f_b(C_b) + f_c(C_c)$.

$jobs$	p_j	r_j	$f_j(t)$
a	2	0	$\begin{cases} 0 & \text{if } 2 \le t \le 5, \\ t-5 & \text{otherwise.} \end{cases}$
b	3	5	$t-8$
c	2	7	$5(t-9)$

Figure 4 first shows the schedules $e(\sigma, u)$ for the six orders and for the three cases: $u = 3$, $3 < u < 5$ and $u = 5$. In each rectangle of a schedule are inscribed the job and its individual cost in the schedule.
First, it is easy to see that $\alpha_{NI} = 3$. We see that $\Sigma^*(3) = \{(a, c, b)\}$ and that the corresponding schedule $e((a, c, b), 3)$ has cost 4 but is not a non-idling schedule. For $3 < u < 5$, we still have $\Sigma^*(u) = \{(a, c, b)\}$ and the corresponding schedule $e((a, c, b), u)$ whose cost is $u + 1$ is not a non-idling schedule. At time $u = 5$, we again have $\Sigma^*(5) = \{(a, c, b)\}$ but now the corresponding schedule $e((a, c, b), 5)$ is a non-idling schedule with cost 6.
For this instance, we thus have $u^- = 5$ but the corresponding non-idling schedule $e((a, c, b), 5)$ of $\Sigma^*(5)$ is not an optimal non-idling schedule since the schedule $e_{NI}((a, b, c))$ has cost 5.

2.4 The Earliest Non-Idling property

The question whether there is an optimal non-idling schedule starting at time α_{NI} is clearly a quite interesting matter. . A problem with that property is said to have the *ENI property*. In [3], "No-Wait" schedules (in short NW schedules) are introduced and it is proved that problems for which NW schedules are dominant have the ENI property. Moreover it is shown that preemptive problems have the ENI property.

Definition 1. *A schedule is said to be no-wait if it is non-idling whenever there is at least one non completed available job.*

Property 2. *If the NW schedules are dominant, then* Π *has the ENI property.*

Remark 2. *The dominance of NW schedules is not a necessary condition for the ENI property to be satisfied. This is illustrated by the small example of Figure 2 with two jobs where* $e(J_1, J_2)$ *is the unique NW schedule of* Σ *while the optimal non-idling schedule, which starts at time* $\beta_{NI} = \alpha_{NI} = r_2$*, is* $e(J_2, J_1)$*.*

An interesting case when the ENI property is satisfied is when preemption is allowed. The formal proof, given in [3], shows that, in this case, NW-schedules are dominant.

Property 3. *If preemption is allowed, then the ENI property is satisfied.*

Property 3 may be used to easily solve problems $1, NI|prec, r_j, pmtn|f_{max}$, $1, NI|r_j, pmtn|\sum C_j$ and $1, NI|r_j, d_j, pmtn|-$. In each case, the algorithm first computes α_{NI} and then applies the algorithm solving the unrestricted problem to the instance $I(\alpha_{NI})$. Consider for example the instance of $1, NI|r_j, d_j, pmtn|-$ whose data are written in the array below:

j	0	1	2	3	4	5	6	7	8
p_j	3	4	2	5	7	2	1	2	2
r_j	0	0	4	13	16	27	28	32	32
d_j	23	24	23	27	23	32	30	36	38

We have $\alpha_{NI} = 8$. Applying the Jackson preemptive algorithm to $I(8)$, we get the non-idling schedule illustrated by Figure 6.
Since this schedule is not feasible (job 3 is completed at time 29), we may conclude that the original instance has no feasible preemptive non-idling schedule.

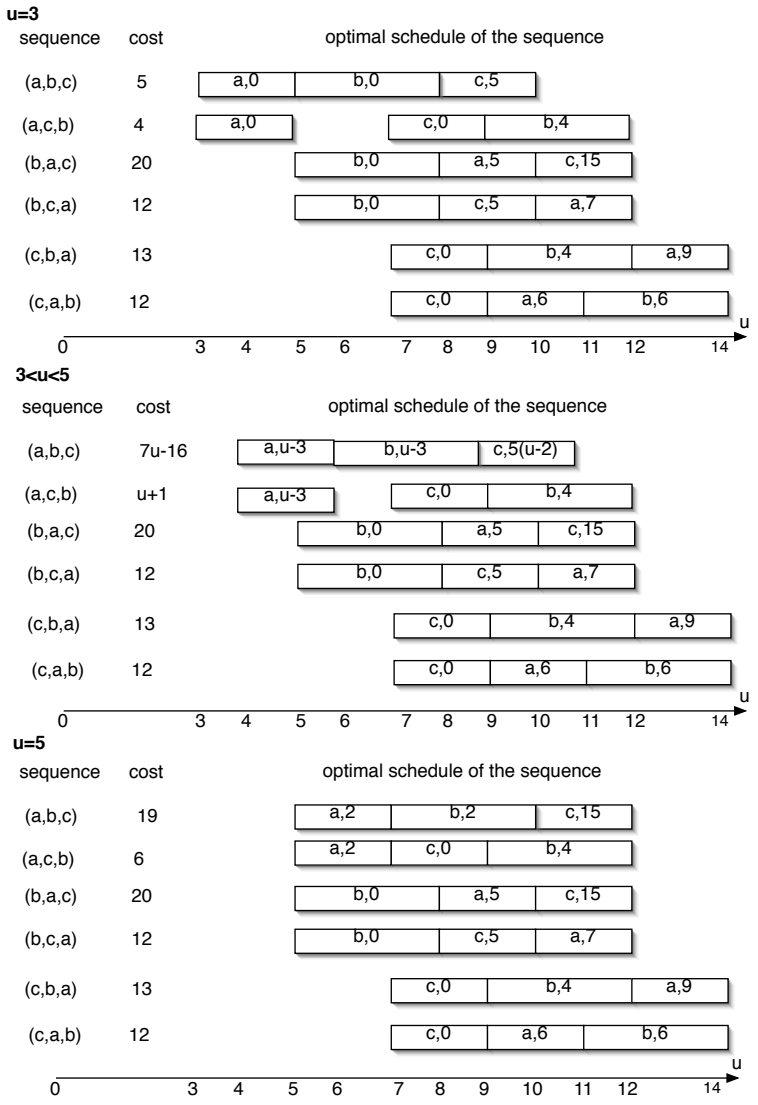

Figure 4: $(a, c, b) \in \Sigma^*(5)$,$e((a, c, b), 5)$ is non-idling but not optimal

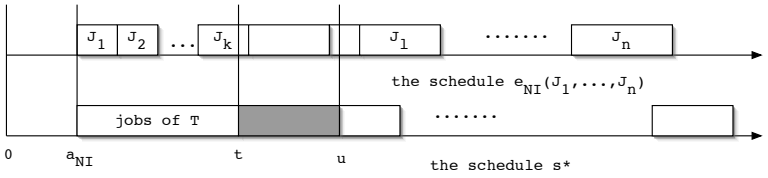

Figure 5: The schedules s^* and $e_{NI}(J_1, \cdots, J_n)$.

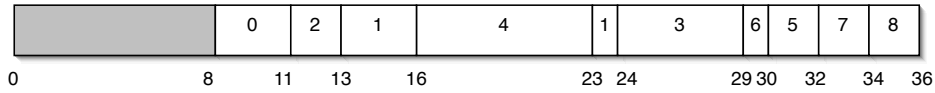

Figure 6: An optimal non-idling schedule

2.5 The critical times of an instance

2.5.1 Definition

Let us first denote by $Z(u)$ $(u \geq 0)$ the set of the sequences σ such that we get a feasible *non-idling* schedule (denoted by $z(u, \sigma)$) when performing the jobs from time u with no intermediate delays in the order of σ. The *critical times* of an instance are then defined as the times u at which $Z(u)$ increases or decreases.
Let $\sigma = (J_{i_1}, \cdots, J_{i_n})$. It is easy to see that $z(u, \sigma)$ meets the release times if and only if $u \geq \alpha_{NI}(\sigma)$. Conversely, $z(u, \sigma)$ meets the deadlines if and only if $u \leq \omega_{NI}(\sigma)$ where $\omega_{NI}(\sigma) = \min_{k \in \{1, \cdots, n\}} \{\tilde{d}_{i_k} - P_{\{i_1, \cdots, i_k\}}\}$ is (if it is non negative) the latest time at which a non idling schedule with sequence σ can start and respect the deadlines. From these definitions, we get that the sequence σ is NI-feasible if and only if $\alpha_{NI}(\sigma) \leq \omega_{NI}(\sigma)$ and that $\sigma \in Z(u)$ if and only if $\alpha_{NI}(\sigma) \leq u \leq \omega_{NI}(\sigma)$.
It follows from the definitions of $\alpha_{NI}(\sigma)$ and $\omega_{NI}(\sigma)$ that the *critical times* of an instance are the *distinct and non negative* values we may obtain from:

1. the values $r_j - P_K$ where J_j is a job and K is a subset of jobs such that $J_j \notin K$;

2. the values $d_j - P_K$ where J_j is a job and K is a subset of jobs such that $J_j \in K$.

Also note that if 0 is not such a value, then by convention it will also be considered as a critical time. If N is the number of critical times, then

we denote by $(c_1, \cdots, c_N)$ the sorted list of these values. Let $z^*(c_k)$, $k \in \{1, \cdots, N\}$ be an optimal non-idling schedule starting at time c_k. The next property, whose proof is given in [3], shows that the schedule $z^*(c_k)$ whose cost is minimum is an optimal non-idling schedule.

Property 4. *Let $(c_1, \cdots, c_N)$ be the sorted list of the critical times of an instance of Π_{NI} and let k^* be such that: $\forall k, \Phi(z^*(c_{k^*})) \leq \Phi(z^*(c_k))$. Then $z^*(c_{k^*})$ is an optimal non-idling schedule.*

2.6 Equal processing times

We assume in this section that Π_{NI} is a single-machine scheduling problem whose non preemptive jobs have the same processing time p and that the objective function $\Phi(s)$ has one of the two forms $\sum f_j(C_j)$ or $\max f_j(C_j)$ where f_j is a non decreasing function of the completion time of J_j. We first consider the special case of independent jobs and show that the corresponding problems are polynomial even if the individual cost functions are no longer regular.
It is clear from their definition that the critical times of an instance are the *distinct and non negative* values we may obtain from:

1. the value 0;
2. the values $r_j - ap$ where J_j is a job and $a \in \{0, \cdots, n-1\}$;
3. the values $d_j - bp$ where J_j is a job and $b \in \{0, \cdots, n\}$.

We thus know that N (the number of critical times) is $O(n^2)$ and that the sorted list $(c_1, \cdots, c_N)$ of the critical times may be computed in $O(n^2 \log n)$ time.
Let $u \geq 0$ and let us denote by $z^*(u)$ an optimal non-idling schedule starting at u (if such a schedule exists). Property 5 shows [3] that, if it exists, the schedule $z^*(u)$ may be computed by solving an assignment problem.

Property 5. *For any time $u \geq 0$, the schedule $z^*(u)$ may be computed in polynomial time*

It follows from Property 4 and Property 5 that the problems $1, NI|p_j = p, r_j, d_j|\Phi$ where Φ has one of the two forms $\sum f_j(C_j)$ or $\max f_j(C_j)$ are polynomial.
Let us now consider the case of dependent jobs, that is the class of problems $1, NI|r_j, p_j = p, prec|\Phi$. We get from Property 4 that a sufficient condition for the problem to be polynomial is that its special case with a fixed schedule

starting time is itself polynomial. The preceding argument may for example be applied to show that problems $1, NI|r_j, p_j = 1, prec|\ f_{max}$, $1, NI|r_j, p_j = p, prec|\ \sum C_i$ and $1, NI|r_j, p_j = p, prec|\ L_{max}$ are polynomial since their variant with a fixed schedule starting time is itself polynomial [11],[12].

3 The non-idling m-machine problem

We consider in this section the case when m identical parallel machines are available to process the jobs and when the jobs assigned to each machine must be processed without any intermediate delay. We first consider the special case with fixed job sequences on each machine. Then we show that the problems $P, NI|p_j = 1, intree|\ C_{max}$, $P2, NI|p_j = 1, prec|\ C_{max}$ and $P2, NI|p_j = 1, r_j, d_j|-$ may be solved by rather easy extensions of the algorithms solving their classical versions. We then show that the problem $P, NI|p_j = 1, prec|\ C_{max}$ is NP-hard while the complexity of $P, NI|p_j = 1, r_j, d_j|-$ is still unknown. We finally consider a stronger version of the non-idling constraint, called the Homogeneously Non-Idling constraint (HNI in short), where, for every subset of machines, the union of their busy periods must be an interval. For the special case $P, HNI|p_j = 1, r_j, d_j|-$, we provide a necessary and sufficient condition for a schedule to exist and we propose a polynomial algorithm to solve the problem.

3.1 Fixed sequences of jobs on each machine

Let us denote by L_k the sequence of jobs processed by machine M_k, by $L_k(p)$ the job with rank p in L_k and assume that L_k is compatible with $prec$ (i.e., $J_i = L_k(r)$, $J_j = L_k(s)$ and $(J_i, J_j) \in prec$ implies $r < s$). Then a schedule is entirely defined by the starting times θ_k of the first job of each sequence. If $(J_i, J_j) \in prec$, $J_i = L_k(q)$, $J_j = L_l(r)$ and $k \neq l$, then the precedence constraint (J_i, J_j) is satisfied if and only if:

$$\theta_l + P^l(r-1) \geq \theta_k + P^k(q)$$

where $P^k(r) = \sum_{s=1}^{r} p_{L_k(s)}$. Now, if we consider all the precedence constraints (J_i, J_j) such that J_i is in L_k and J_j is in L_l, all these constraints will be satisfied if and only if :

$$\theta_l - \theta_k \geq \max\{P^k(q) - P^l(r-1) | (J_i, J_j) \in prec\} = v(k, l, prec).$$

Let us define the arc-valued graph $GM = (M, A)$ where M is the set of machines, and where there is an arc (M_k, M_l) valued by $v(k, l, prec)$ if there

is at least one precedence constraint (J_i, J_j) in $prec$ with J_i in L_k and J_j in L_l. Then it is clear that at least one schedule exists if and only if GM has no positive circuit and that in this case, there is an earliest schedule (i.e., the smallest θ_k values) given by the values of the longest paths ending at each node of GM. Figure 7 shows the graph GM associated with two instances of the problem, one with no feasible schedule and one with an earliest schedule.

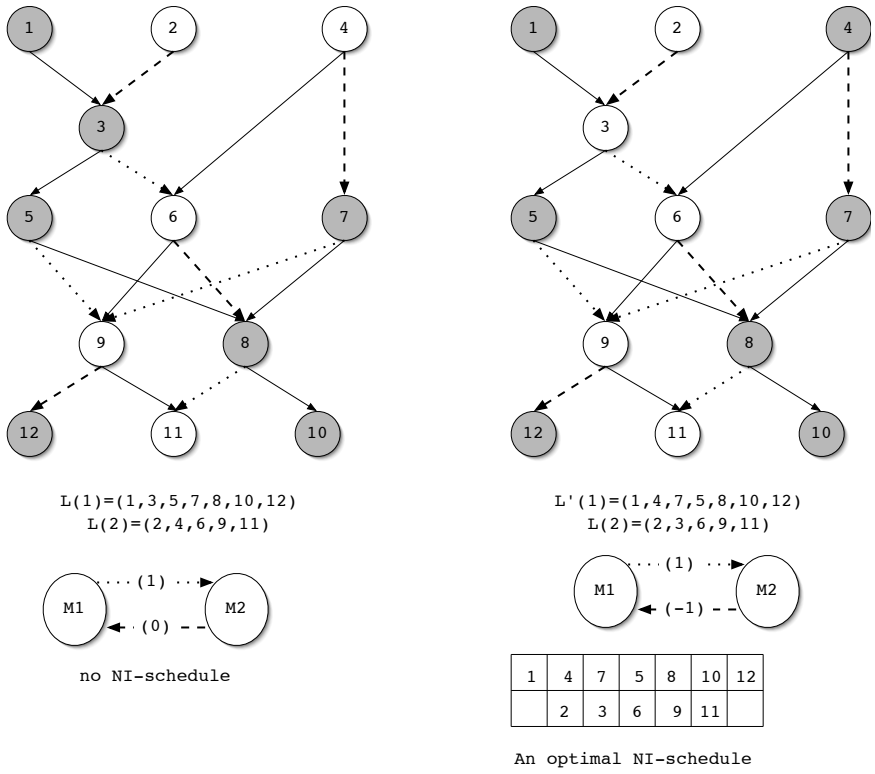

Figure 7: 2 instances with fixed sequences of jobs.

3.2 Extension of classic algorithms

In this section, we show that the well-known polynomial algorithms solving respectively the problems $P, NI|p_j = 1, intree|\ C_{max}$, $P2, NI|p_j = 1, prec|\ C_{max}$ and $P2, NI|p_j = 1, r_j, d_j|-$ may be rather easily extended to solve their non-idling versions.

3.2.1 $P, NI|p_j = 1, intree|\ C_{max}$

It is well-known that the problem $P|p_j = 1, intree|\ C_{max}$ is polynomially solved by the Hu's algorithm [16], which is a list algorithm where the priority of a job is its level in the intree. A key property of the schedule issued from that list is that the number of jobs scheduled in time slot $t+1$ is not more than that of time slot t. Since the machines are identical, by assigning the $n(t)$ jobs of slot t to the machines $M_1, \cdots, M_{n(t)}$, we get a non-idling schedule which is obviously optimal. Figure 8 shows an example.

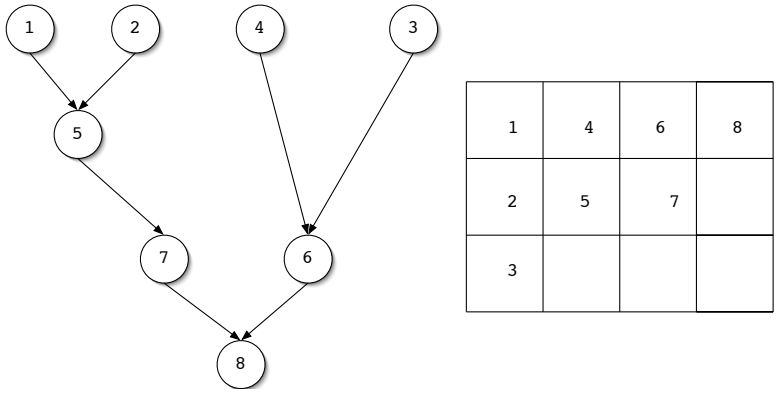

Figure 8: an instance of $P, NI|p_j = 1, intree|\ C_{max}$.

3.2.2 $P2, NI|p_j = 1, prec|\ C_{max}$

We recall that the problem $P2|p_j = 1, prec|\ C_{max}$ is polynomially solved by the Coffman-Graham algorithm [15] (CG algorithm in short).
Let I be an instance of $P2, NI|p_j = 1, prec|\ C_{max}$. The following property shows that, if I has at least one schedule, then there is an optimal one with a quite specific structure.

Property 6. *If I has at least one schedule, then there exists an optimal schedule whose structure is illustrated in Figure 9 where:*

1. *the jobs scheduled before a (called the left jobs) are the ascendants of job a in prec;*
2. *the jobs scheduled after b (called the right jobs) are the descendants of job b in prec;*

3. *the jobs scheduled after a and before b (called the middle jobs) are optimally scheduled without any idle time by the CG algorithm.*

Proof. Let O be a schedule of I. If the jobs assigned to machine M_1 and the jobs assigned to machine M_2 are processed in disjoint intervals in S (case 1 of Figure 9), then by applying the corresponding transformation, we get an at least as good schedule O^1, with no middle jobs), which has the expected structure. If the two intervals contain at least a common slot (case 2 of Figure 9), then by applying the corresponding transformation, we again get an at least as good schedule O^1 (with at least two middle jobs) which has the expected structure.
We now show that if the job a exists in O^1, then O^1 may be transformed into a schedule O^2 (with the same structure) such that the jobs scheduled before the middle jobs of O^2 are the ascendants of the job a of O^2 (if there is one) in *prec*. Assume job J_x is not an ascendant of the job a of O^1 in *prec* and is the last such job to be scheduled before a in O^1. Thus, job J_x may be scheduled on machine M_2 in the same time slot as job a and all the jobs scheduled in O^1 after J_x may be left-shifted by one time slot (see Figure 10). Iterating the process until either the job a (of the current schedule) no longer exists or all jobs scheduled before a are the ascendants of a, we get a schedule O^2 at least as good as O^1 satisfying the condition 1) of the property.
The same line of reasoning may be applied to the schedule O^2 if there is a job b in that schedule.
We thus finally get a schedule O^3 at least as good as O^2 satisfying the conditions 1) and 2) of the property. If O^3 has some middle jobs, it is clear that, since the partial schedule of O^3 restricted to these middle jobs has no idle time, the condition 3) of the property is also satisfied by O^3. □

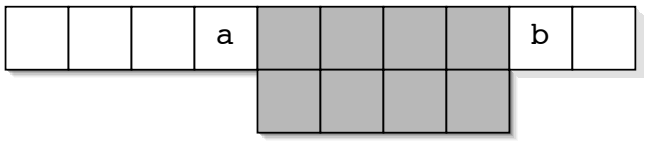

Figure 9: The dominant structure of NI-schedules.

Property 6 easily yields to a polynomial algorithm (not necessarily the best) solving $P2, NI|p_j = 1, prec|\ C_{max}$. Let (a, b) be two distinct jobs such

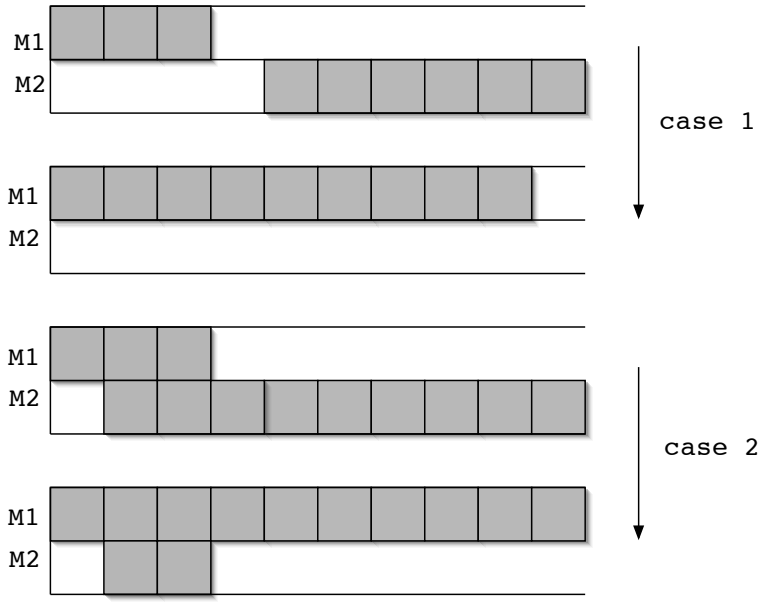

Figure 10: Getting the dominant structure.

that $(b,a) \notin prec$. Let $Asc(a)$ (resp. $Desc(b)$) be the ascendants (resp. descendants) of a (resp. b) in $prec$. Let $CG(a,b)$ be the schedule provided by the CG algorithm on the restriction of I to the jobs of $J \setminus (Asc(a) \cup Desc(b))$. Finally let

$$F(a,b) = \begin{cases} 1 & \text{if } CG(a,b) \text{ has an idle time,} \\ 0 & \text{otherwise.} \end{cases}$$

Then we get from Property 6 that the instance I has no schedule if, for every (a,b) such that $(b,a) \notin prec$, we have $F(a,b) = 0$. Otherwise, the optimal schedule of I is issued from the couple (a,b) such that $F(a,b) = 1$ and $|Asc(a)| + |Desc(b)|$ is minimum.
Since the complexity of the CG algorithm is $O(n^2)$, computing $F(a,b)$ takes $O(n^2)$ time and the overall complexity of the algorithm is $O(n^4)$.

3.2.3 $P2, NI|p_j = 1, r_j, d_j|-$

We recall that the problem $P2, NI|p_j = 1, r_j, d_j|-$ is polynomially solved by the Earliest Deadline First algorithm (EDF algorithm in short) that schedules one ready job with the minimum deadline as soon as a machine is idle and there is at least one ready job. Let I be an instance of $P2, NI|p_j = 1, r_j, d_j|-$. The following property shows that, if I has at least a schedule, then there is an optimal one with the following structure.

Property 7. *If I has at least one schedule, then there exists an optimal schedule such that the value of the starting time of the busy period of each machine is $r_j - k$ where $j \in \{1, \cdots, n\}$ and $k \in \{0, \cdots, n\}$.*

Proof. Assume that I has a schedule that executes respectively the sequences σ_1 and σ_2 on M_1 and M_2. Let $n_1 = |\sigma_1|$ and $n_2 = |\sigma_2|$. We know from Section 2 that $e_{NI}(\sigma_1)$ and $e_{NI}(\sigma_2)$ are also feasible non-idling schedules of the corresponding subsets of jobs. Since $\alpha_{NI}(\sigma_1) = \max\{0, \max_{k \in \{1,\cdots,n_1\}}\{r_{i_k} - (k-1)\}$ and $\alpha_{NI}(\sigma_2) = \max\{0, \max_{k \in \{1,\cdots,n_2\}}\{r_{i_k} - (k-1)\}$, we get a feasible non-idling schedule with the required structure. □

Let us now consider the following variant, NI-EDF, of EDF that takes as input the instance I, the starting time slots a_1, a_2 (with values satisfying Property 7) of the busy periods of M_1 and M_2 and the numbers of jobs n_1 and n_2. Let $a = \min\{a_1, a_2\}$ and $b = \max\{a_1, a_2\}$. The NI-EDF applies the following rule at each time-slot t from the leftmost one. If t belongs to exactly one of the two busy periods and there is at least one unscheduled ready job, then schedule one ready job with the minimum deadline and

consider the next time slot. If there is no ready unscheduled job at t, then stop (failure case). If t belongs to the two busy periods and there are at least two unscheduled ready jobs, then schedule the jobs with the two smallest deadlines and consider the next time slot. If there is at most one ready unscheduled job at t, then stop (failure case).
It is easy to show that the instance I has at least a schedule where the busy periods of machines M_1 and M_2 are respectively the sets of time slots $\{a_1, \cdots, a_1+n_1-1\}$ and $\{a_2, \cdots, a_2+n_2-1\}$ if and only if the the algorithm NI-EDF applied to (I, a_1, a_2, n_1, n_2) does not stop in the failure case.
Since NI-EDF is clearly polynomial, we get from Property 7 that the problem $P2, NI|p_j = 1, r_j, d_j|-$ is polynomial. It is easy to see that this result easily extends to a *fixed* number k of machines, that is $Pk, NI|p_j = 1, r_j, d_j|-$ is polynomial.

3.3 The homogeneous non-idling constraint

Up to now, the complexity of $P, NI|p_j = 1, r_j, d_j|-$ remains *an open question*. So, in order to better understand why the complexity analysis of this problem is difficult, a more constrained variant of $P, NI|p_j = 1, r_j, d_j|-$ has been studied [13] where the non-idling constraint is replaced by the homogeneously non idling (HNI in short) constraint. A schedule satisfies the HNI constraint if, *for any subset* M' of machines, the time slots at which at least one of the machines of M' is busy, make an interval. Clearly an HNI schedule is an NI schedule but the converse is not true, as shown on Figure 11 where the schedule on the left does not satisfy the HNI constraint since, for the subset $\{M_1, M_3\}$, the set of the times units when M_1 or M_3 is busy, is not an interval. On the contrary, the schedule on the right is an HNI-schedule.
So, we are given a set $J = \{J_1, \cdots, J_n\}$ of n unit-time independent jobs that are to be processed on a set $M = \{M_1, \cdots, M_m\}$ of m identical machines. Job J_i must be executed within a given time-window $F(i) = \{r_i, \cdots, d_i\}$.

A schedule (T, μ) assigns a time-unit $T(i)$ and a machine $\mu(i)$ to each job J_i so that :

1. for each job $J_i \in J, T(i) \in F(i)$;
2. for each time slot t, $|\{i|T(i) = t\}| \leq m$;
3. for any $M' \subseteq M$, the time units t at which there is at least a job J_i processed at t (i.e: $T(i) = t$) by a machine of M' ((i.e: $\mu(i) \in M'$) make a single interval.

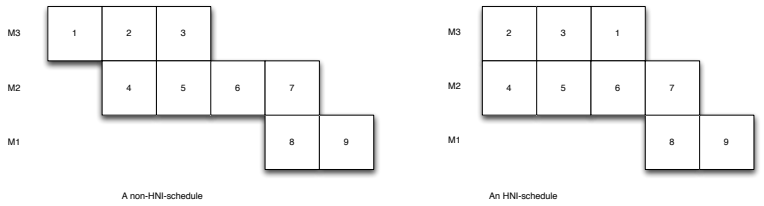

Figure 11: HNI-schedule versus non-HNI schedule

The decision problem Π_0 is to decide whether a given instance (J, F, m) has at least one schedule. If the answer is yes, the instance is said to be *feasible*.

Assume that (T, μ) is a schedule of the instance (J, F, m) and let us denote by $n_T(t)$ the number of jobs scheduled in the time slot t. It is easy to see that if the $n_T(t)$ jobs scheduled in time slot t are assigned to the machines $M_1, \cdots, M_{n_T(t)}$, then we get a new schedule with the so called *pyramidal structure* as shown on Figure 12. Thus an m-matching T, which is not a schedule, has at least a *k-hole* ($k \in \{0, \cdots, m-1\}$), i.e., a time slot t such that there are two time slots t' and t'' with $t' < t < t''$, $n_T(t') > k$, $n_T(t'') > k$ and $n_T(t) = k$. As an example, time slot $t = 2$ is a 1-hole of the m-matching on the left part of Figure 12.

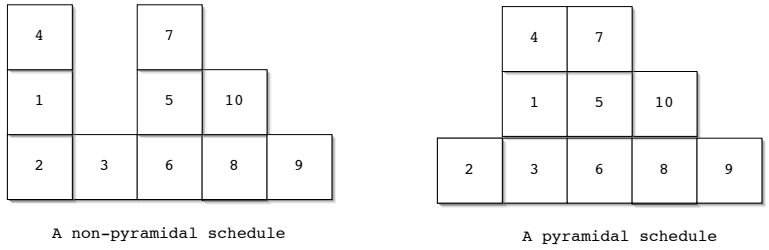

Figure 12: The pyramidal structure

If Θ is an interval (i.e: a finite set of consecutive positive integers), let us denote by $J(\Theta)$ the set of the jobs J_j such that $F(j) \subseteq \Theta$. Also, for any interval Θ, we denote by $\lambda(\Theta)$ the number $\max_{\Theta' \in Int(\Theta)} \lceil \frac{|J(\Theta')|}{|\Theta'|} \rceil$, where $Int(\Theta)$ is the set of the intervals contained in Θ. Moreover, if Θ_1 and Θ_2 are two intervals such that $\Theta_1 \cup \Theta_2$ is not an interval (two such intervals will be called *disconnected*), then $Mid(\Theta_1, \Theta_2)$ is the interval made by the time

slots between Θ_1 and Θ_2. Finally $(\Theta_1, \cdots, \Theta_s)$ is a sequence of disconnected intervals if for $i \in \{1, \cdots, s-1\}$ the ending time slot of Θ_i is strictly smaller than the starting time slot of Θ_{i+1}.

3.3.1 Existence of a schedule

Let $I = (J, F, m)$ be an instance of Π_0. Following are three basic properties of Π_0, the first two ones concern the existence of an m-matching, the third one comes from the pyramidal structure of a schedule.

Property 8. *For any interval Θ and any m-matching T of I, there is at least one time slot t of Θ such that $n_T(t) \geq \lambda(\Theta)$*

Property 9. *I has at least one m-matching if and only if, for any interval Θ, we have $|J(\Theta)| \leq m|\Theta|$*

Property 10. *Let Θ_1 and Θ_2 be two disconnected intervals. If $|J \setminus (J(\Theta_1 \cup J(\Theta_2))| < |Mid(\Theta_1, \Theta_2)| \min\{\lambda(\Theta_1), \lambda(\Theta_2)\}$, then I has no schedule.*

The necessary and sufficient condition for a schedule to exist is an extension of Property 10 to an arbitrary sequence $(\Theta_1, \cdots, \Theta_s)$ of disconnected intervals. The following theorem has been proved in [13].

Theorem 1. *$I = (J, F, m)$ has at least one schedule if and only if*

1. *for any interval Θ, $|J(\Theta)| \leq |\Theta| \times m$;*
2. *for any sequence $(\Theta_1, \cdots, \Theta_s)$ of disconnected intervals, $|J \setminus \cup_{i=1}^{s} J(\Theta_i)| \geq \sum_{i=1}^{s-1} |Mid(\Theta_i, \Theta_{i+1}| \times \min\{\lambda(\Theta_i), \lambda(\Theta_{i+1})\}$*

Proving that the two conditions are necessary is easy while proving their sufficiency is more difficult and needs to introduce the concept of a k-schedule. A k-schedule is an m-matching with a pyramidal structure up to the level k (i.e., the function $\min\{k, n_T(t)\}$ has a pyramidal structure). Figure 13 illustrates this definition.

Then, the following theorem is proved using induction on k:

Theorem 2. *$I = (J, F, m)$ has at least one k-schedule if and only if*

1. *for any interval Θ, $|J(\Theta)| \leq |\Theta| \times m$;*
2. *for any sequence $(\Theta_1, \cdots, \Theta_s)$ of disconnected intervals, $|J \setminus \cup_{i=1}^{s} J(\Theta_i)| \geq \sum_{i=1}^{s-1} |Mid(\Theta_i, \Theta_{i+1}| \times \min\{k, \lambda(\Theta_i), \lambda(\Theta_{i+1})\}$*

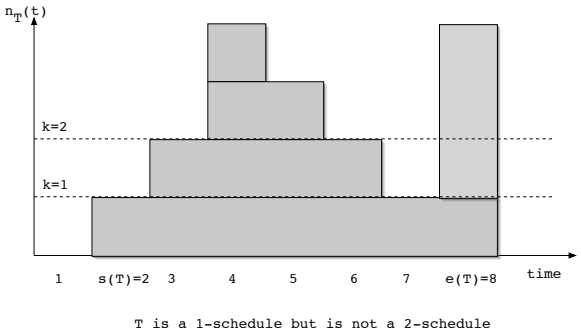

Figure 13: k-schedule

3.3.2 A polynomial algorithm solving Π_0

Let us first present the concept of *propagation path*, which will be the basic tool used by the algorithm. The *propagation graph* $G(T) = (\mathcal{H}, E(T))$ is the directed graph such that $(t, t') \in E(T)$ if $t \neq t'$ and there is at least one job J_i scheduled at t such that $t' \in F(i)$. If job J_i is scheduled at t and if $t' \in F(i)$, then J_i is said to be a *label* of the arc (t, t'). A *propagation path* of T is an elementary path $\gamma = (t_0, \cdots, t_k)$ of $G(T)$. A propagation path is *labelled* when all its arcs are assigned a label and is said to be *fitted* if it is labelled and satisfies $n_T(t_k) < m$.

If $\gamma = (t_0, \cdots, t_k)$ is a propagation path of $G(T)$ fitted with the labelling $\sigma = (J_{[0]}, \cdots, J_{[k-1]})$, we get another m-matching $T' = \mathcal{T}(T, \gamma, \sigma)$ by putting $T'(J_{[p]}) = t_{p+1}$ for all $p \in \{0, \cdots, k-1\}$. Figure 14 illustrates this transformation.

k-schedules, as defined in the preceding section, are again essential in the derivation of the algorithm. Note first that a 0-schedule always exists and that an m-schedule is a schedule. The algorithm $QuickSearchSchedule$-(J, F, m) presented below works as follows. It first checks for an m-matching T of I. If there is no m-matching, then it stops in the failure case. Otherwise it searches for the greatest k_0 such that T is a k_0-schedule and sets the variable k to k_0. Then it searches iteratively for a propagation path to transform the current k-schedule into a new one either with strictly less k-holes or with a more flat structure. If the current k-schedule is such that there is no propagation path that removes a k-hole or makes the structure more flat, then a sequence of disconnected intervals that does not satisfy Condition 2 of Theorem 2 may be found from the current k-schedule. So, in that case, we know that the instance I has no $(k+1)$-schedule. Otherwise,

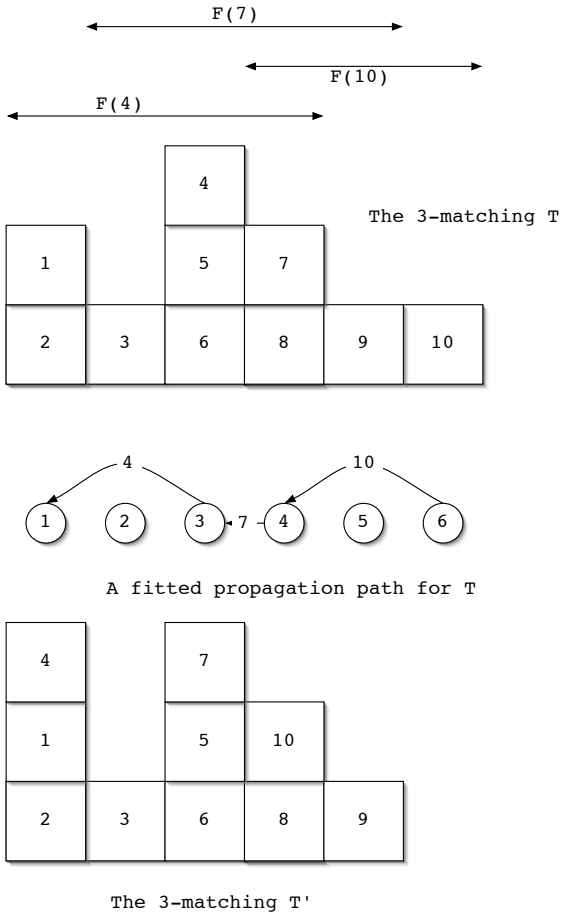

Figure 14: The new m-matching $T' = \mathcal{T}(T, \gamma, \sigma)$

the current k-schedule is a $(k+1)$-schedule and, if $k+1 < m$, k is set to $k+1$ and the process is iterated. If $k = m-1$, we have got a schedule.
In the algorithm below, the following variables concern the current k-schedule T: H is the set of k-holes, C_p^+ (resp. C_p^-) is the largest (resp. smallest) time slot of T such that $n_T(t) = p$, and $\hat{S}$ is the set of the time slots t such that $n_T(t) \geq k+2$.

function $QuickSearchSchedule(J, F, m)$;
(1) $T \leftarrow mMatching(J, F, m)$;
(2) If $(T = nil)$ then return(nil);
(3) $k \leftarrow \max\{l \in \{0, \cdots, m\} | T$ is a l-schedule$\}$;
(4) $stop \leftarrow false$;
(5) while $k < m$ and not $stop$ do
(6) while not $(H = \emptyset)$ and not $stop$ do
(7) if there is a path γ from $u \in \hat{S}$ to $v \in H$
then $[trans(T, \gamma)$;break]
(9) if there is a path γ from $u \in \cup_{p=1}^{k+1}\{C_p^+, C_p^-\}$ to $v \in H$
then $[trans(T, \gamma)$;break]
(11) if there is a path γ from $u \in \hat{S}$ to $v \in \{C_{k+1}^+ + 1, C_{k+1}^- - 1\}$
then $[trans(T, \gamma)$;break]
(13) if there is a path of $G(T)$ from u to v where:
(14) $u = C_p^+, v = C_{p-1}^+ + 1$ and $C_p^+ < C_{p-1}^+$ for some $p \in \{2, \cdots, k+1\}$
(15) or $u = C_p^+, v = C_p^+ + 1$ and $C_p^+ = C_{p-1}^+$ for some $p \in \{2, \cdots, k+1\}$
(16) or $u = C_p^-, v = C_{p-1}^- - 1$ and $C_p^- < C_{p-1}^-$ for some $p \in \{2, \cdots, k+1\}$
(17) or $u = C_p^-, v = C_p^- - 1$ and $C_p^- = C_{p-1}^-$ for some $p \in \{2, \cdots, k+1\}$
(18) then
(19) choose a path γ such that $n_T(u) - n_T(v)$ is maximum;
(20) $[trans(T, \gamma)$;break]
(21) $stop \leftarrow true$;
(22) endwhile;
(23) if not $(stop)$ then $k \leftarrow k + 1$;
(24) endwhile;
(25) if $k = m$ then return(T) else return(nil).

procedure $trans(T, \gamma)$;
(1) Let σ be a label of γ;
(2) $T \leftarrow \mathcal{T}(T, \gamma, \sigma)$.

It is worth noting that the propagation path chosen at each iteration of the inner loop (line 19) is not an arbitrary one satisfying the required starting and end node. If this choice was left free, then the algorithm would not be polynomial. On the contrary, if at each iteration, we choose the propagation path with a maximum level difference between its starting and

ending nodes, it can be shown (bit this is not that easy) that the number of iterations for a given value of k is polynomial [14].

4 Conclusion

In this paper, we have proposed a survey on the specific type of scheduling problems where intermediate delays between jobs executed by the same machine are not allowed. Even if such problems are encountered in practice, they have been investigated rather recently. In this context, the paper presents results about the non-idling variants of very basic scheduling problems. These results have been established mainly from the complexity point of view and it is clear that, in each case, more efficient algorithms could be developed. Quite many research directions are still to be investigated. From a theoretical point of view, it seems for example important to classify the problem $P, NI|p_i = 1, r_i, d_i|-$ and answer the following question: does a polynomial single-machine classical scheduling problem with a regular objective have a polynomial counterpart? If the restriction to a regular objective function is removed, we know the answer is no. Another important direction for which very few and often negative results are known, concerns approximation algorithms for non-idling problems. Practical applications also ask for studying scheduling problems with two sets of resources, one set for which the tasks assigned to the resource must be processed in the non-idling mode and one set for which they are processed in the classical mode.

References

[1] P. Chrétienne. On scheduling with the non-idling constraint, *4OR*, 12(2), 101-121 (2014).

[2] P. Baptiste. Scheduling Unit Tasks to Minimize the Number of Idle Periods: a Polynomial Time Algorithm for Off-line Dynamic Power Management, *Research Report*, Laboratoire d'Informatique CNRS LIX, 2005.

[3] P. Chrétienne, On single-machine scheduling without intermediate delays, *Discrete Applied Mathematics*, 13, 156, 2543-2550, 2008.

[4] A. Jouglet, Single-machine scheduling with no idle time and release dates to minimize a regular criterion, *Journal of Scheduling*, 15, 2, pp217-238, 2012.

[5] J.M.S. Valente and R.A.F.S. Alves. An Exact Approach to Early/Tardy Scheduling with Releases Dates. *Computers and Operations Research*,32, 2905-2917, 2005.

[6] K. Landis. Group Technology and Cellular Manufacturing in the Westvaco Los Angeles VH Department. *Project Report in IOM 581, School of Business, University of Southern california*, 1993.

[7] S. Irani and K. Pruhs. Algorithmic Problems in Power Management, *ACM Press*, vol 36, 63-76, New York, USA, 2005.

[8] J. K. Lenstra and A. H. G. Rinnoy Kan. Complexity of Scheduling under Precedence Constraints, *Oper. Res.*, 26(1), 22-35, 1978.

[9] J. Carlier, A. Moukrim and K. Guédira. Exact resolution of the nemachine sequencing problem with no machine idle time, *Computers and Industrial Engineering*, 59(2), 193-199, 2010.

[10] I. Kacem and H. Kellerer. Approximation algorithms for no idle time scheduling on a single machine with release times and delivery times, *Discrete Applied Mathematics*, Article in Press, 2011.

[11] K.R. Baker, E.L. Lawler, J.K. Lenstra and A. H. G. Rinnoy Kan. Preemptive Scheduling of a Single Machine to Minimize Maximum Cost subject to Release Dates and Precedence Constraints. *Operations Research*,31, 381-386, 1983.

[12] P. Brucker. Scheduling Algorithms, *Springer*, 2004.

[13] A. Quilliot and P. Chrétienne. Homogeneously non-idling schedules of unit-time jobs on identical parallel machine. *Discrete Applied Mathematics*,161,10-11,pages 1586-1597,2013.

[14] A. Quilliot and P. Chrétienne. A polynomial algorithm for the homogeneously non-idling scheduling problem of unit-time independent jobs on identical parallel machine. MISTA, 2013.

[15] Coffman E.G. and R.L. Graham. Optimal scheduling for two processor systems.*Acta Informatica*, 1, 200–213, 1972.

[16] Hu, T.C. Parallel sequencing and assemble line problems.*Operations Research*, 9, 841–848, 1961.