Les cahiers Leibniz

Polynomial time algorithms for makespan minimization on one machine with forbidden start and completion times

Christophe Rapine, Nadia Brauner

Laboratoire G-SCOP
46 av. Félix Viallet, 38000 GRENOBLE, France
ISSN : 1298-020X

n$^{\text{o}}$ 181

April 2010

Site internet : http://www.g-scop.inpg.fr/CahiersLeibniz/

Polynomial time algorithms for makespan minimization on one machine with forbidden start and completion times

Christophe Rapine*[,a], Nadia Brauner[a]

[a]*Laboratory G-SCOP, 46 avenue Félix Viallet, 38031 Grenoble Cedex, France*

Abstract

We consider independent jobs to sequence on a single resource under a special unavailability constraint: a set of *forbidden* instants is given, where no job is allowed neither to start not to complete. We show that a schedule without idle time always exists if the number of *forbidden* instants is less than the number of distinct processing times appearing in the instance. We derive a quite fast algorithm to find such a schedule, based on an hybridization between list algorithm and local exchange. We then improve our approach to propose a strongly polynomial time algorithm under a High Multiplicity encoding. As a corollary minimizing the makespan for a constant number of forbidden instants is polynomial.

Key words: Scheduling, Makespan criterion, Availability constraints, High multiplicity

1. Introduction

We call a *forbidden start & end instant* a point in time where no job is allowed neither to start nor to complete. Such forbidden start & end instants (FSE for short, or simply *forbidden*) arise when jobs need additional resources at launch and at completion. These additional resources may not be continuously available, as they can be shared with other yet planed activities. For instance consider the situation where the jobs are processed by an automated device during a specified amount of time, but a qualified operator is required at setup and at completion. While the device is continuously available, operators have day-off and holidays. This creates some forbidden days when the jobs can be performed by the device, but none can start nor complete. We encountered this problem in chemical industry [12] through an industrial collaboration with the *Institut Français du Pétrole* (IFP), a large research center in fields of energy and transport. The jobs consisted in chemical experiments whose durations typically last between 3 days and 3 weeks. The intervention of a chemist is required at start and termination: at start, the chemist basically fills up the device and launches the process. At termination he has to stop the chemical reactions for the analysis of the experimental results.

The additional resource can also be for instance a special handling tool, expansive enough for the company not to own it but to call a subcontractor. For very large products like turbines for hydropower plants, a crane lifting is needed to put in and get out the product from the shop floor. Due to strict deadlines, teams are often organized to work continuously, and thus a finished product must immediately get out in order to start the next one. However an overcost is typically paid to rent the crane lifting on week-ends. The objective is then to find a schedule of minimal duration for the different products such that ideally no overcost is paid. One can imagine other additional resources such as energy (burning up or cooling down operations), water, etc., with consumption restriction or overcost during some periods.

In this article we consider the scheduling problem of a set of n jobs to sequence on a single resource in presence of FSE instants. The objective is to minimize the completion time of the last job, also called the *makespan* of the schedule. As a variant we also investigate the case where a penalty c_j is associated to the jth FSE instant: this cost is paid if a job starts or ends at this instant. Another penalty d is paid per day late to complete all the jobs. This corresponds to the previous subcontractor problem. The objective is then to find a schedule of minimal cost. We show that both problems are polynomial for a fixed number of FSE instants.

The remaining of the paper is organized as follows: in Section 2 we present a literature review on scheduling with forbidden instants, together with some notations and definitions. Section 3 is devoted to establish that a schedule without idle time always exists if the number of FSE instants is smaller than the number of distinct processing times appearing in the instance. We call such an instance a *large diversity* instance. Based on these results we propose in Section 4 polynomial time algorithms to solve the problem in case of a constant number of forbidden instants. Finally we prove in Section 5 that the problem remains polynomial under a *high multiplicity* encoding for *large diversity* instances.

*Corresponding author

2. Notations and preliminary definitions

We consider a set of n independent jobs, with processing times $p_1, \ldots, p_n$, to be sequenced on a single resource where k Fse instants appears. A schedule is feasible if no job starts nor completes its processing during a forbidden instant. Preemption of jobs is not allowed. All data are assumed to be integers. In addition the starting time and completion time of any job are also restricted to take integer values. With the objective of minimizing the duration of the schedule, we designate the problem as $1|\text{Fse}|C_{\max}$. Figure 1 gives an example of a feasible schedule for the instance composed of 4 jobs $\{a, b, c, d\}$ of duration $p_a = 5$, $p_b = 3$ and $p_c = p_d = 2$. Two forbidden instants are present, at time 7 and 10. The jobs are scheduled according to the sequence (a, b, c, d): due to forbidden instant 10, processing of job c has to be delayed up to time 9, resulting in a makespan of 13. Notice that the sequence (c, d, a, b) would give a schedule without idle time. As we have 3 different processing times and only 2 forbidden instants, next section will assert the existence of such an idle-free schedule.

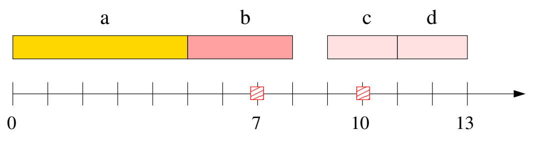

Figure 1: The Gantt chart of a feasible schedule for $1|\text{Fse}|C_{\max}$ following the sequence (a, b, c, d). Forbidden instants are represented on the time axis by dashed rectangles. The schedule completes at time 13

In scheduling theory, machine non-availability problems have been largely investigated (see for instance Lee [13] for a survey). Machine non-availabilities correspond to periods where the machine can not process any job, typically due to a preventive maintenance. In contrast a Fse instant only prevents a job completion or a job start: the machine can go on processing its current job. In [4, 15], the authors study a scheduling problem with similar constraints: an *operator non-availability* (Ona) period is defined as an open time interval in which no job can start nor end. With the makespan as objective criterion, they prove the problem to be $\mathcal{NP}$-hard and not in APX even if the duration of any Ona period is smaller than the processing time of any job. Notice that if processing times are integers and jobs are required to start at integer instants, an Ona period $(s, s+t)$ can be represented by the finite, but exponentially large, set of forbidden instants $\{s+1, s+2, \ldots, s+t-1\}$. However we do not consider in this article any special condition on the distribution of the forbidden instants. Other previous works have considered additional resources for task setup or ending operations. Cheng *et al.* [5] study a 2-machine flowshop scheduling problem where the same operator performs setup and dismounting operations on both machines following a cyclic movement pattern. This problem is connected with the server model (see for instance [8, 1]), where a server has to do some setup operations before the processing of a job starts on a machine, or is required to unload the machine, see Ou *et al.* [14], where an example is given from logistics. A server model deals with the problem of sharing an additional resource among several machines, creating machine interferences [10], while we consider in this paper unavailability constraints on the additional resource. Our problem could correspond to the situation where a schedule has yet been fixed for the server, and we have to incorporate a new machine with allotted jobs in the planning. If setup time is one unit, the fixed schedule of the server is seen by this machine precisely as forbidden instants. Note that problem $P2, S1|s_j = 1|C_{\max}$ with a single server shared by two parallel machines has been proved $\mathcal{NP}$-hard by Hall *et al.* [8], while Kravchenko and Werner [11] propose a pseudo-polynomial time algorithm. More precisely they show that an optimal schedule for $P2||C_{\max}$ can be converted into an optimal schedule for $P2, S1|s_j = 1|C_{\max}$. This procedure runs in time $\mathcal{O}(n \log n)$ and is based on local job exchanges. We use quite similar technics in this paper, except that forbidden instants are fixed. The most relevant work to our problem is certainly Billaut & Sourd [2]. They consider the scheduling of independent jobs on a single machine where a set of time slots is forbidden for starting the processing of a job. We call in this paper such an instant a Fs instant (for *forbidden start*). They prove that minimizing the makespan is polynomially solvable if the number of forbidden start times is a constant, and $\mathcal{NP}$-hard in the strong sense if this number is part of the input. Their algorithm runs in time complexity $\mathcal{O}(n^{2k^2+2k-1})$, where k denotes the number of Fs instants. They also establish that an idle-free schedule exists if at least $2k(k+1)$ distinct processing times appears in the instance. Interestingly enough, they prove that for the special case of 2 forbidden start instants, having 3 distinct processing times is a sufficient condition to assess the existence of a schedule without idle time. This article generalizes and improves their results for Fse instants: we prove that having $k+1$ distinct processing times is sufficient to ensure the existence of an idle-free schedule in presence of k Fse instants. In this case we derive a fast algorithm to compute such an optimal schedule: the algorithm is an original hybridization between list scheduling and local search. The overall complexity to solve the problem for a constant number of forbidden instants is reduced as a consequence of our result to $\mathcal{O}(n^k)$. In addition the proofs used new ideas and are far shorter. Finally we also propose a polynomial time algorithm for the High Multiplicity version of the problem, while Billaut & Sourd only considered the case $k = 2$. Before presenting our notations and the concept of valid partition and exchangeable jobs, we quote the following theorem from Billaut & Sourd [2]:

Theorem 1 *The scheduling problem $1|\text{Fse}|C_{\max}$ is $\mathcal{NP}$-hard in the strong sense*

Proof. The reduction from 3-Partition is the same as in Billaut & Sourd [2]. As they noticed the many to one reduction used in the proof can easily be transformed into a gap reduction by appending a large (but polynomial) number of consecutive forbidden instants at the end of the schedule. This proves that $1|\text{Fse}|C_{\max}$ is not in APX if $\mathcal{P} \neq \mathcal{NP}$. $\square$

Throughout this paper we denote by N the set $\{1, \ldots, n\}$ of jobs indices and by $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ the set of forbidden instants, indexed in increasing order. As usual a sequence f defined on N is extended to any subset $S \subseteq N$ by letting $f(S) = \sum_{i \in S} f_i$. The number of distinct processing times will play a central role in our analysis. We say that 2 jobs are of the same type is they have the same processing time. We denote by $\langle S \rangle$ the set of (distinct) types appearing in a subset S of jobs. By slight abuse of language, we will often confuse "type" and "job" in the sense that we will speak about the processing time of a type, instead of the processing time of a representative (job) of this type. For a subset S of jobs (and by extension of types), we denote by $\min S$ and $\max S$ the minimal and maximal value of p_i over S, respectively.

We will prove in next section that if the number of types is greater than the number of forbidden instants, then there exists a schedule without idle time. We need first some additional definitions. We call a partition $S \cup U$ of N a *valid partition* if and only if there exists a schedule for S starting at time 0, without idle time. A valid partition has to be thought as the set S of scheduled jobs and the set U of unscheduled jobs at some step of a constructive algorithm. Such a constructive algorithm, presented in Section 4, will naturally try at each step to append a new job to its current partial schedule. We say that S is *blocked* if none of the unscheduled jobs can be appended to S without violating the forbidden instants, *i.e.* $p(S) + p_i \in \Gamma$ for all $i \in U$. In this case local exchanges between jobs of S and U will be needed to "unblock" the partial schedule. We introduce the following definition:

Definition 1 *We call a job (a type) $s \in S$ exchangeable if $S \cup \{\underline{u}\} \backslash \{s\}$ defines a valid partition, where $\underline{u}$ is a job of duration $\min U$. We denote by $E(S) \subseteq \langle S \rangle$ the subset of exchangeable types that are not in $\langle U \rangle$*

Clearly a necessary condition for a job x to belong to $E(S)$ is that $p(S) - p_x + \min U$ does not coincide with a forbidden instant. In next section Property 1 proves that it is in fact a sufficient condition in some circumstances. Figure 2 shows an example of partial schedule, defining the valid partition $S = \{1, 2, 3\}$. The remaining jobs to schedule are $U = \{4, 5\}$. They are represented above the Gantt chart to stress the fact that the partition is blocked. In the same way we have represented job 2 under the time axis to visualize that it is not exchangeable with job 4: an idle-free schedule of set $\{1, 3, 4\}$ would complete on a forbidden instant. As job 3 is of same type as job 4, set $E(S)$ is reduced to singleton $\{1\}$.

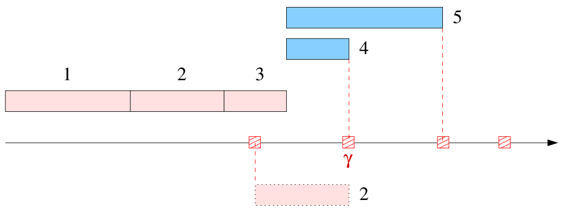

Figure 2: An example of blocked valid partition $S = \{1, 2, 3\}$. Set $U = \{4, 5\}$ has been represented above the Gantt chart to visualize the corresponding forbidden instants. Under the time axis job 2 is represented to visualize that it is not exchangeable with job 4.

For x in S, we note $S_x = S \backslash \{x\} \cup \{\underline{u}\}$ the partition obtained by unscheduling a job of type x and by scheduling a job $\underline{u}$ of type $\min U$. By definition for $x \in E(S)$, set S_x defines a valid partition, that differs from S only by jobs x and $\underline{u}$. Of course their respective idle-free schedule may be totally different.

3. Existence of an idle free schedule

We now state that it is possible to schedule a set of jobs without idle-time if the number of distinct processing times is greater than the number of forbidden instants. Another necessary condition is of course that the sum of the processing times does not coincide with a forbidden instant. For the same reason we require in this part that $0 \notin \Gamma$. For a subset S of jobs, let $\Gamma(S) = \{\gamma \in \Gamma \mid \gamma \leq p(S)\}$ be the set of forbidden instants appearing before time $p(S)$.

Theorem 2 *If $|\langle N \rangle| > |\Gamma(N)|$ and $0, p(N) \notin \Gamma$, then there exists a feasible schedule for N without idle time.*

The remaining of this section is devoted to prove Theorem 2. Note that the assumptions of Theorem 2 are the weakest possible to ensure the existence of an idle-free schedule. Indeed if $|\langle N \rangle| = |\Gamma(N)|$, for some instances idle times will be forced in any schedule: consider for example set $\Gamma = \{p_i | i \in \langle N \rangle\}$. Necessarily an idle time appears in any feasible schedule at time 0 although it is not a forbidden instant.

Theorem 2 can be rephrased as $N \cup \emptyset$ is a valid partition. We prove this result by induction on $|\Gamma(N)|$. If $|\Gamma(N)| = 0$, certainly Theorem 2 holds. Now consider that $|\Gamma(N)| = K > 0$ and assume the result true for any set S of jobs such that $|\Gamma(S)| < K$. First, we can use the induction hypothesis to give the following characterization of exchangeable jobs:

Property 1 (Characterization of $E(S)$) *Consider a valid partition $S \cup U$ such that S is blocked. We have $E(S) = \{x \notin \langle U \rangle \mid p(S_x) \notin \Gamma\}$*

Proof. Clearly if $x \in E(S)$, then $p(S_x)$ can not coincide with an instant of Γ by definition of a valid partition. Conversely consider a type $x \notin \langle U \rangle$ such that $p(S_x) \notin \Gamma$. Let $\underline{u}$ be a job of U with the smallest execution time, and let $\underline{\gamma}$ be the instant $p(S) + p_{\underline{u}}$. Since S is blocked there exists at least $|\langle U \rangle|$ forbidden instants in interval $[\underline{\gamma}, p(N)]$, as $p(S) + p_u \in \Gamma$ for all $u \in \langle U \rangle$. Thus we have $|\Gamma(S_x)| \leq |\Gamma(N) \cap [0, \underline{\gamma})| \leq |\Gamma(N)| - |\langle U \rangle|$. In particular we have $|\Gamma(S_x)| < K$. Now let us bound the number of types in S_x. Clearly we have $S_x \cup U \supseteq N \backslash \{x\}$ and $S_x \cap U \supseteq \{\underline{u}\}$. As for any sets A and B we can write $|\langle A \cup B \rangle| = |\langle A \rangle| + |\langle B \rangle| - |\langle A \cap B \rangle|$, it results that $|\langle S_x \rangle| \geq |\langle N \rangle| - |\langle U \rangle|$. We have established that $|\langle S_x \rangle| > |\Gamma(S_x)|$ and $|\Gamma(S_x)| < K$: the induction hypothesis shows that S_x is a valid partition. $\square$

The benefit of exchanging jobs is to modify the set of types of unscheduled jobs. We may hope that after a series of exchanges we will be able to append a new job to the current schedule. Lemma 1 gives a sufficient condition for this situation to happen:

Lemma 1 *Consider a valid partition $S \cup U$. If S is blocked, then set $E(S)$ is not empty. In addition if $\max E(S) < \min U$, then there exists $x \in E(S)$ such that S_x is not blocked.*

Proof. We use a simple counting argument. First we define $F(S)$ as $\langle N \rangle \backslash (\langle U \rangle \cup E(S))$. This set contains the types that neither belongs to $\langle U \rangle$ nor are exchangeable. By construction we get a partition $\langle U \rangle \cup F(S) \cup E(S)$ of $\langle N \rangle$. We then define the following function $\varphi : \langle N \rangle \mapsto \mathbb{R}$, where $\overline{u}$ is a job of type $\max U$ and $\underline{u}$ is a job of type $\min U$:

$$\varphi(i) = \begin{cases} p(S) + p_i & \text{if } i \in \langle U \rangle \\ p(S_i) & \text{if } i \in F(S) \\ p(S_i) + p_{\overline{u}} & \text{if } i \in E(S) \end{cases}$$

Clearly φ is injective on each part of the partition, as p is injective on $\langle N \rangle$. In addition, denoting $\underline{\gamma}$ and $\overline{\gamma}$ the instant $p(S) + p_{\underline{u}}$ and $p(S) + p_{\bar{u}}$, respectively, we have $\varphi(F(S)) \subseteq [0, \underline{\gamma})$ and $\varphi(\langle U \rangle) \subseteq [\underline{\gamma}, \overline{\gamma}]$. Thus φ is injective on $\langle U \rangle \cup F(S)$. The fact that S is blocked along with Property 1 shows that the image of this set under φ is included into $\Gamma(N)$. Thus condition $|\langle N \rangle| > |\Gamma(N)|$ necessarily involves that $\langle U \rangle \cup F(S) \subset \langle N \rangle$, which proves the first assertion of Lemma 1. Now consider that conditions $\max E(S) < \min U$ holds. It results that $\varphi(E(S)) \subseteq (\overline{\gamma}, p(N)]$, and thus φ is an injection on $\langle N \rangle$. The same counting argument shows that there necessarily exists a type $x \in E(S)$ such that $p(S_x) + p_{\overline{u}} \notin \Gamma(N)$. Thus partition S_x is not blocked as a job of type $\max U$ can be appended to it. This is true as long as type $p_{\overline{u}}$ is still present in U_x after the exchange, or, put in other words, as long as set U is not reduced to singleton $\{\underline{u}\}$. Writing down that $p(N) \notin \Gamma$ immediately conduces to $|U| \geq 2$, which concludes the proof. $\square$

Now to conclude proof of Theorem 2, assume for the sake of contradiction that $N \cup \emptyset$ is not a valid partition. Among all valid partitions (at least $\emptyset \cup N$ is one), we choose $S \cup U$ maximizing $|S|$, and of minimal size $p(S)$ among all partitions of maximal cardinality. The maximality of $|S|$ implies that S is blocked. We claim that necessarily $\max E(S) < \min(U)$: indeed for any type $x \in E(S)$, set S_x defines a valid partition of cardinality $|S|$ and of size $p(S_x) = p(S) + p_{\underline{u}} - p_x$; the minimality of $p(S)$ among the partitions of the same cardinality imposes that $p_x \leq p_{\underline{u}}$. Thus we are in the conditions stated by Lemma 1. It implies that there exists x such that $S_x \cup \{\bar{u}\}$ defines a valid partition, with $\bar{u}$ a job of type $\max U$. This contradicts the maximality of $|S|$, and achieves the proof of Theorem 2.

4. Polynomial time algorithms

Theorem 2 proves the existence of a schedule without idle time if a set of jobs contains more types than the number of forbidden instants. In this part, we derive an algorithm to construct such a schedule. The algorithm is a simple hybridization between list scheduling algorithm and local search. Basically the algorithm is constructive and tries at each step to schedule a new job in a greedy manner. If at some step no job can be scheduled without creating some idleness, we then perform some job exchanges until the current partial schedule is no more blocked. The priority given to the jobs in the greedy allocation does not matter, thus we consider an arbitrary list L. Nevertheless, for efficiency reasons, we assume that jobs of the same type have the same priority.

We have to show how a small number of exchanges always permit to obtain a partial schedule that is no more blocked. The basic idea is to exchange at each step the smallest unscheduled job $\underline{u}$ with the largest exchangeable (scheduled) job $\bar{x}$. We call such an exchange a *min-max-exchange*. Using the vocabulary of previous section, if (S, U) is a blocked partition, a *min-max* exchange consists then in exchanging a job $\underline{u}$ of processing time $\min U$ with a job $\overline{x}$ of processing time $\max E(S)$. These exchanges are performed as long as S remains blocked and that $\min U < \max E(S)$. If S is blocked but $\min U > \max E(S)$, we then select a job $x \in E(S)$ such that S_x is not blocked. We call this later exchange an *unblocking* exchange. Notice that Lemma 1 ensures that, if S is blocked, there does exist exchangeable jobs. It also ensures that if $\max E(S) < \min U$, an *unblocking* exchange is always possible. Next lemma bounds the number of successive exchanges:

Lemma 2 *At most $k+1$ exchanges are needed to obtain a non-blocked partition from any blocked partition.*

Proof. Let $(S_1, U_1), \ldots, (S_l, U_l), (S_{l+1}, U_{l+1})$ be the sequence of valid partitions constructed by iterating *min-max* exchanges from an initial blocked partition (S_0, U_0). By construction for each index $i \leq l-1$, partition (S_i, U_i)

is blocked and verifies $\min U_i < \max E(S_i)$. The last partition (S_{l+1}, U_{l+1}) is no more blocked. Notice that the $(l+1)$th exchange may be an *unblocking* exchange, while the l first ones are *min-max* exchanges. For short let $\alpha_i = |\langle U_i \rangle|$ be the number of types in U_i. We also denote by β_i the number of types of processing times smaller or equal to $\min U_i$, *i.e.* β_i is the rank of type $\min U_i$ following the non-decreasing order. It is easy to check that α_i and β_i are non-decreasing function. More precisely for all index $i = 0, \ldots, l-1$:

(1) either $\beta_{i+1} = \beta_i$ and $\alpha_{i+1} = \alpha_i + 1$,

(2) or $\alpha_{i+1} = \alpha_i$ and $\beta_{i+1} > \beta_i$.

To see this, recall that the exchange performed to transform $(S_i, U_i) \to (S_{i+1}, U_{i+1})$ is then a *min-max* exchange. Let $\overline{x}_i$ be the job selected in $\max E(S_i)$ to exchange with $\underline{u}_i$ of duration $\min U_i$. By definition of $E(S_i)$, type $\overline{x}_i$ does not belong to U_i. In addition $\overline{x}_i$ has a larger processing time than $\underline{u}_i$. If U_i contains more than one job of type $\underline{u}_i$, after the exchange U_{i+1} contains the same types as U_i plus the new type $\overline{x}_i$, *i.e.* we are in situation (1). Otherwise $\underline{u}_i$ is the only job of type $\min U_i$ in U_i. Then clearly we have $\min U_{i+1} > \min U_i$, which corresponds to situation (2).

Now let $\mu_i = \alpha_i + \beta_i$. From what precedes μ_i is an increasing series on $[0, \ldots, l]$, which involves that $\mu_l \geq \mu_0 + l$. Observe that for any subset V of $\langle N \rangle$, the rank of its smallest type can be at most $|\langle N \rangle| - |\langle V \rangle| + 1$. Hence the function that associates to V the index of its smallest type plus its cardinality lies between 2 and $|\langle N \rangle| + 1$. It results that $l \leq \mu_l - \mu_0 \leq |\langle N \rangle| - 1$, which establishes that at most $l + 1 \leq |\langle N \rangle|$ exchanges are performed.

To decrease the number of exchanges from $|\langle N \rangle|$ to $k+1$, we can restrict our attention to set $\mathcal{F}$ of the first $(k+1)$ types. More precisely in each *min-max* exchange, we select a job x of greatest processing time in set $E(S) \cap \mathcal{F}$. Notice that if a partition is blocked, this set can not be empty. In addition if $\min U \notin \mathcal{F}$, an *unblocking* exchange can then be performed. Otherwise *min-max* exchanges will swap only jobs of $\mathcal{F}$. By the previous argument the number of exchanges is then bounded by $|\langle \mathcal{F} \rangle| = k + 1$. □

If we design an algorithm on this simple principle, alternating greedy allocation and *min-max* exchanges, one difficulty arises when an exchange occurs at some step between a job x and $\underline{u}$. Indeed if Theorem 2 ensures that S_x can be scheduled without idle time, it gives no clue on how to find such a schedule. To avoid (expensive) recursive calls, we introduce the following notion:

Definition 2 *A subset $N' \subset N$ defines a L-partition if:*

(1) *set N' defines a valid partition,*

(2) $|\Gamma(N')| < |\Gamma(N)|$,

(3) *the remaining jobs of $N \backslash N'$ can be scheduled in time interval $[p(N'), p(N)]$ by the list scheduling algorithm of list L.*

We represent such a L-partition by the couple (N', π), where π is the sequence of jobs given by the list scheduling algorithm. The L-partition problem consists, given N and L, to find a L-partition (N', π). We can immediately notice the following fact :

Remark 1. Given an algorithm for the L-partition problem running in time $t(k, n)$, a schedule without idle time for N can be found in time $kt(k, n)$.

Consider Algorithm 1: as announced it mixes greedy allocation with local exchanges of jobs. To deliver a L-partition, it simply memorizes the valid partition obtained after the last exchange. Before establishing the correctness and the time complexity of the algorithm, we demonstrate it on the instance of Figure 1. Using list $L = (a, b, c, d)$, jobs a and b are successively scheduled. We are then in situation of Figure 3 with a blocked partition $S = \{a, b\}$, $U = \{c, d\}$.

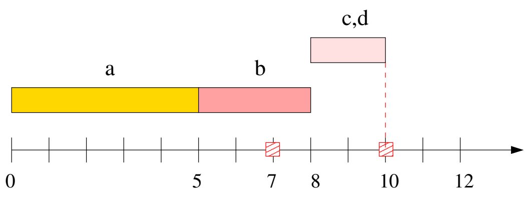

Figure 3: The partial schedule obtained for list $L = (a, b, c, d)$ on the instance of Figure 1

The set of exchangeable jobs $E(S)$ is reduced to singleton $\{a\}$ since exchanging b and c would result in a non valid partition completing on forbidden instant 7. We then perform a *min-max* exchange between job a and c. We are in situation of Figure 4 with a new valid partition $S' = \{b, c\}$.

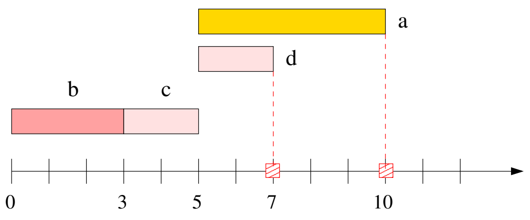

Figure 4: The partial schedule obtained after exchanging a and c

Partition S' is still blocked. As job c and d are of the same type, only job b is exchangeable with d. The *min-max* exchange of jobs b and d results in partition $S'' = \{c, d\}$ which is not blocked. Jobs a and b are then appended to the schedule by the list algorithm. We obtain the idle-free schedule given in Figure 5. The algorithm returns the L-partition (S'', ab). In this example the L-partition directly provides a feasible schedule as no forbidden instant appears before time $p(S'')$.

Note that by definition of *min-max* exchanges the size $p(S)$ of the current valid partition decreases between 2 exchanges. Hence we can not assert that if S is a blocked partition, after a series of exchanges the resulting non-blocked

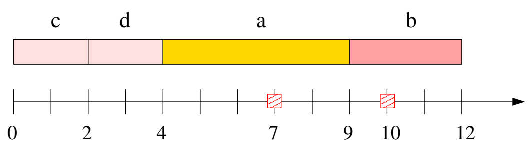

Figure 5: The final schedule. The corresponding L-partition is $(\{c,d\}, ab)$

partition S' will obey $|\Gamma(S')| > |\Gamma(S)|$. Such a property would have bounded the total number of exchanges to obtain a L-partition by $\mathcal{O}(k^2)$. However we can assert that this number is at most $\mathcal{O}(kn)$, as a new job is scheduled after each series of exchange. We have the following lemma:

Lemma 3 *If N is a set of jobs verifying conditions of Theorem 2, a L-partition of N can be found in time $\mathcal{O}(k^2 n)$.*

Proof. We first show that, if Algorithm 1 terminates, it returns a L-partition. If no exchange occurred, by definition $N' = \emptyset$ defines a L-partition, *i.e.* the greedy list scheduling directly finds a schedule without idleness. Otherwise the algorithm returns the set obtained after the last exchange. At this step set S is blocked, and a job $x \in E(S)$ is exchanged with $\underline{u}$. Notice that after the exchange, at least job x remains to schedule, thus $S_x \subset N$. Moreover by construction S_x is a valid partition, and the remaining jobs can be scheduled according to list L without idle time since no other partition is blocked afterwards. Finally, since S was blocked, necessarily time $p(S_x) + p_x$ is a forbidden instants. Hence we get a L-partition.

The fact that the algorithm always terminates is ensured by Lemma 2: let us call a *step* either an exchange between two jobs or the allocation of a job. Clearly we have exactly n allocation steps. Since at least one allocation is done after each sequence of exchanges, Lemma 2 bounds the number of steps by $\mathcal{O}(kn)$. To obtain the time complexity of the algorithm, we show that both allocation and exchange step can be done in time $\mathcal{O}(k)$. Note that L can be described has a vector of integers of dimension $|\langle N \rangle|$. More generally each subset can be represented by a vector of types. For an allocation steps it is clearly sufficient to scan the first $k+1$ non null types in L. With a proper list data structure, this can be done in time $\mathcal{O}(k)$. For an exchange step, we scan only the $k+1$ smallest types of set S as explained in Lemma 2 to determine the set $E(S) \cap \mathcal{F}$. Checking if a type is exchangeable can be done in constant time due to Property 1. Thus an exchange step can also be achieve in time $\mathcal{O}(k)$. $\square$

The condition that the number of types is greater than the number of forbidden instants plays as we have seen an import part in the analyze of the problem. This motivates the following definition:

Definition 3 *An instance (N, Γ) is said to be of large diversity if $|\langle N \rangle| > |\Gamma|$, of small diversity otherwise.*

Algorithm 1 L-partition Algorithm

```
Require: a set $N$ of jobs and a set $\Gamma$ of forbidden instants
Ensure: a $L$-partition $(N',\pi)$
  mark all jobs unscheduled
  set $S = \emptyset$ ; $N' = \emptyset$ ; $\pi = \emptyset$ // initialization
  while an unscheduled job remains do
    // perform local exchanges while $S$ is blocked
    while $S$ is blocked do
      if $\max E(S) > \min U$ then // min-max exchange
        select $x \in E(S)$ of largest processing time
      else // unblocking exchange
        select $x \in E(S)$ such that $S_x$ is not blocked
      end if
      mark $x$ unscheduled and $\underline{u}$ scheduled.
      set $S := S_x$ ; $N' = S_x$ ; $\pi = \emptyset$
    end while
    // greedy list allocation
    Select the first unscheduled job $u$ in $L$ such that $p(S)+$
    $p_u \notin \Gamma$; mark $u$ scheduled
    set $S := S \cup \{u\}$; $\pi = \pi\{u\}$
  end while
  return $(N', \pi)$
```

Theorem 3 *Any instance of $1|\textsc{Fse}|C_{\max}$ of large diversity can be solved optimally in time $\mathcal{O}(k^3 n)$.*

Proof. If neither instant 0 nor $p(N)$ belongs to Γ, Theorem 2 proves that an idle free schedule exists. This schedule can be found in time $\mathcal{O}(k^3 n)$ by Algorithm L-partition due to Lemma 3 and Remark 1. Otherwise let k_1 be the first integer instant which is not forbidden, and let k_2 be the first integer instant greater than $p(N) + k_1$ which is not forbidden. Obviously no feasible schedule can complete before time k_2. Let $k' = k_2 - k_1 - p(N)$. We start the schedule at time k_1 adding to N a dummy job of duration k'. Thus we are in conditions of Theorem 2. We find an idle free schedule completing at time k_2. Replacing each dummy job processing by an idle slot, we obtain a feasible (and optimal) schedule for N. $\square$

The general case needs to consider what happens for *small diversity* instances. As problem $1|\textsc{Fse}|C_{\max}$ is $\mathcal{NP}$-hard, we can not hope to obtain a polynomial time algorithm, not even a constant approximation algorithm, if one believes that $\mathcal{P} \neq \mathcal{NP}$. However next lemma involves that any list scheduling algorithm has an *absolute* error of at most $2k$:

Lemma 4 *Any list scheduling algorithm delivers a schedule of length at most $p(N) + 2k$.*

Proof. Consider the last job, say l, of the schedule. Let t be an idle instant. Due to the greedy allocation of the algorithm, necessarily either t or $t + p_l$ belongs to Γ, otherwise l should have been scheduled at time t. Hence the set of idle instants is included in $\{\gamma_j, \gamma_j - p_l \mid j \in 1, \dots, k\}$, which is of cardinality at most $2k$. □

The bound of Lemma 4 is tight: consider an instance with only jobs of duration 1 and forbidden instants occurring at each odd instant till time $2k-1$. This example of course does not prove that $2k$ is a tight bound for the absolute error. It could be investigated if a better bound can be found, at least for particular lists. In this paper we use this result to bound the value of the optimal makespan and derive a polynomial time algorithm in the case where the number of forbidden instants is a constant. We denote by $1|k-\text{FSE}|C_{\max}$ this problem, when k is not part of the input.

Theorem 4 *Problem $1|k-\text{FSE}|C_{\max}$ is polynomial, i.e. if the number of forbidden instants is a constant, an optimal schedule can be found in polynomial time.*

Proof. Consider an instance (N,Γ) of problem $1|k-\text{FSE}|C_{\max}$. If $|\langle N\rangle| > |\Gamma|$, Theorem 3 proves that we can find an optimal schedule in linear time $\mathcal{O}(n)$. Otherwise, in case of a small diversity instance, we have only a constant number of types. For short let us denote by $q \leq k$ the number $|\langle N\rangle|$ of distinct processing times appearing in the instance. We represent any subset $S \subseteq N$ by its multiplicity vector $(s_1,\dots,s_q)$, where s_i is the number of jobs of type i present in S. As in Billaut & Sourd [2] we use dynamic programming to compute predicate $\mathcal{P}(S,K)$ which equals true if and only if there exists a partial schedule for S completing at time $p(S)+K$. At the end we output the smallest value of K such that predicate $\mathcal{P}(N,K)$ is verified. Due to Lemma 4 we can restrict our attention to values of $K \in \{0,\dots,2k-1\}$ (a schedule of length at most $p(N)+2k$ can be obtained by any list scheduling algorithm in linear time). Thus the number of states of the dynamic program to compute all predicates $\mathcal{P}(S,K)$ for $S \subseteq N$ and $K \leq 2k-1$ is $2k\Pi_{i=1}^{q}(n_i+1)$, where $(n_1,\dots,n_q)$ is the multiplicity vector of N. Each state $\mathcal{P}(S,K)$ can be computed in time $\mathcal{O}(q)$ by considering which type (or idle slot) can be scheduled in last position. Hence the running time of the dynamic program is in $\mathcal{O}(2kq(n/q+1)^q)$. For k a constant, this time complexity is bounded by $\mathcal{O}(n^k)$, which is a polynomial in n. □

Theorem 4 appeals to some comments. If the problem is polynomial for a constant number k of forbidden instants, an algorithm of time complexity in $\mathcal{O}(n^k)$ can be used in practice only for small values of n and k. Note that in fact the dynamic programming algorithm runs in time $\mathcal{O}(n^q)$, where q is the number of types in the instance. Thus for instances with a small number of types $(2,3,\dots)$ the algorithm is efficient in practise. This time complexity grows up to $\mathcal{O}(n^k)$ as the number of types increases to k, and then drops to $\mathcal{O}(n)$ if we have more than $k+1$ types. This gap legitimates in our opinion future researches for a more efficient algorithm for small diversity instances.

5. High Multiplicity

The term *High Multiplicity* (HM for short) was introduced by Hochbaum and Shamir [9] to refer to a compact encoding of instances where identical jobs appear many times. Compared to a traditional encoding where each job is described, in a HM encoding each *type* is described only once, along with its multiplicity (the number of jobs of this type). Thus the size of a HM encoding depends linearly on the number of types but only logarithmically on the number of jobs. As a consequence a polynomial time algorithm under the standard encoding may become exponential under a HM encoding of the instances, which is the case of our algorithms. HM scheduling and more generally HM combinatorial optimization has become an active domain in recent years [3, 6, 7].

Let $x=(N,\Gamma)$ be a large diversity instance. We denote for short by $q=|\langle N\rangle|$ the number of types. Indexing types by decreasing order of their processing times, set N is represented in HM encoding by its multiplicity vector $(n_1,\dots,n_q)$ together with the processing vector $(p_{\langle 1\rangle},\dots,p_{\langle q\rangle})$, where n_i and $p_{\langle i\rangle}$ are the number of jobs of the ith type and its processing time, respectively. The size $|x|$ of the instance under a HM encoding is thus in $\Omega(q(\log n+\log p_{\langle 1\rangle})+k\log\gamma_k)$. Hence $|x|$ can be in $\mathcal{O}(q\log n)$ while algorithm L-partition runs in $\mathcal{O}(k^3n)$, which can be exponential with respect to $|x|$. Note that an idle-free schedule π is simply a permutation of the jobs. In HM scheduling, it may be not obvious to determine if there exists (optimal) schedules with a compact encoding, *i.e.* polynomial in $|x|$. For $1|\text{FSE}|C_{\max}$ it is readily that the schedule of the jobs between two forbidden instants is meaningless, and thus the jobs of the same type can be scheduled consecutively. As a consequence any idle-free schedule has a polynomial encoding as a sequence of couples (i_l,α_l), where i_l designates a type and α_l the number of jobs of this type scheduled consecutively. We use this representation inside this section.

To achieve a polynomial time algorithm, we need two ingredients. Firstly, we can not afford to allocate only one job at a time. Secondly, we have to design a more efficient approach than L-partition. To cope with the latter point, consider a partial schedule π, completing at time t. We say that π is an *optimal prefix* if there exists an optimal schedule of the form $\pi\sigma$. In this situation, the problem reduces to finding an optimal schedule starting at time t on the remaining set N' of jobs. Notice that algorithm L-partition finds an optimal suffix, with the drawback of computing a potentially long sequence of valid partitions to obtain it. Now how can we assert that a partial schedule π is an optimal prefix? The response is quite simple: due to Theorem 2, a sufficient condition if that π is idle-free, and that the remaining instance $(N',\Gamma'=\Gamma\cap[t,+\infty])$ is a large diversity instance. Based on these ideas, we derive Algorithm 2.

We give here a comprehensive description of this algorithm. The basic idea is to reduce the instance to have

Algorithm 2 Optimal Prefix Algorithm

```
Require: a large diversity instance $(N,\Gamma)$ with types indexed in decreasing order.
Ensure: an optimal prefix $\pi$
  set $m_i = n_i - 1$ if $i \leq |\Gamma| + 1$, $m_i = n_i$ otherwise
  set $i = 1$ ; $t = 0$ ; $\pi = \emptyset$
  while $i \leq |\langle N \rangle|$ and $t + m_i p_{\langle i \rangle} < \gamma_1$ do
    // Append to $\pi$ all the $m_i$ jobs of type $i$
    $\pi = \pi(i, m_i)$ ; $t = t + m_i p_{\langle i \rangle}$ ; $i = i + 1$
  end while
  if $i > |\langle N \rangle|$ then
    return $\pi$ // Only $|\Gamma| + 1$ jobs remains to schedule
  end if
  // Append as many as possible jobs of type $i$ before $\gamma_1$
  $\alpha = \lceil (\gamma_1 - t)/p_{\langle i \rangle} \rceil - 1$ ; $\pi = \pi(i, \alpha)$ ; $t = t + \alpha p_{\langle i \rangle}$ ;
  // Extend $\pi$ to complete after time $\gamma_1$
  for all $l \in 1, \ldots, |\Gamma| + 1$ such that $t + p_{\langle l \rangle} \geq \gamma_1$ do
    if $t + p_{\langle l \rangle} \notin \Gamma$ then
      return $\pi(l, 1)$
    end if
  end for
  for all $l \in 2, \ldots, |\Gamma| + 1$ such that $t + p_{\langle l \rangle} < \gamma_1$ do
    if $t + p_{\langle l \rangle} + p_{\langle 1 \rangle} \notin \Gamma$ then
      return $\pi(l, 1)(1, 1)$
    end if
  end for
```

only one job of each type and only $|\Gamma| + 1$ types, such that Algorithm 1 can be used efficiently. Initially one job of each of the $|\Gamma| + 1$ largest types is put aside in order to control the number of types. Let F be this set, and let us call *additional* jobs the set $N \backslash F$. We schedule then iteratively all the additional jobs of type 1, then all the additional jobs of type 2, ..., as long as they all fit before the first forbidden instant γ_1. When this process terminates, either only set F remains to schedule, or there is not enough room left before γ_1 to schedule all the additional jobs of the ith type. In the latter case the algorithm schedules as much as possible of jobs of type i before γ_1, and tries to cross forbidden instant γ_1. Our second idea is here to ensure that the schedule of each job of F permits to cross at least one forbidden instant in order to keep a large diversity instance. We claim that Algorithm 2 is correct, *i.e.* delivers an optimal prefix π. In addition if (N', Γ') is the remaining instance to schedule, then (N', Γ') is a large diversity instance and:

1. either $|\Gamma'| < |\Gamma|$, *i.e.* we have strictly less forbidden instants,
2. or $|N'| = |\langle N' \rangle| = |\Gamma| + 1$, *i.e.* all the remaining jobs have distinct processing times.

In the first case, we recursively call the prefix algorithm on instance (N', Γ'). The second case corresponds to the basis of the recursion: we simply solve instance (N', Γ') using the L-partition algorithm. Since N' contains at most $(k + 1)$ jobs, the running time of Algorithm 1 on this instance is in $\mathcal{O}(k^4)$. We prove our claim in the following theorem:

Theorem 5 *Problem $1|\text{FSE}|C_{\max}$ is polynomial under HM encoding for large diversity instances, and can be solved in time $\mathcal{O}(k|\langle N \rangle| + k^4)$*

Proof. **Correctness of Algorithm 2**. Let (N', Γ') be the instance remaining to schedule at the end of Algorithm 2. Recall that F denotes a set with exactly one job of the $|\Gamma| + 1$ largest types of N. Let $A = N \backslash Z$ be the additional jobs. If only set F remains to schedule at the end of the algorithm, we are clearly in the second case of our claim. Otherwise the algorithm has stopped the first loop on a type i such that all its additional jobs has not enough room left to fit before γ_1. At this point, there remains at least one unscheduled job of type i in A, and possibly another in F, if $i \leq k + 1$. Let $t < \gamma_1$ be the current completion time of the schedule, and consider the partition $F = \mathcal{S} \cup \mathcal{L}$ defined by $\mathcal{L} = \{j \in F \mid t + p_j \geq \gamma_1\}$. Notice that $\mathcal{L}$ is not empty as $t + p_{\langle i \rangle} \geq \gamma_1$; in particular a job of type 1 belongs to $\mathcal{L}$. By construction Algorithm 2 tries to extend π to complete after the first forbidden instant γ_1, which corresponds to the first case of our claim. We have to prove that it will always succeed, and that (N', Γ') is a large diversity instance. Basically we show in the following that if π completes after the lth forbidden instant, at most l jobs of F have been scheduled in π. Indeed we then have $|\langle N' \rangle| \geq |F| - l > |\Gamma| - l$ and $|\Gamma'| \leq |\Gamma| - l$, which ensures that (N', Γ') is a large diversity instance. Consider the last two loops of the algorithm. If one job of $\mathcal{L}$ can be scheduled, the property clearly holds as π completes after time γ_1. If this is not possible, instant $t + p_j$ is forbidden for all jobs j of $\mathcal{L}$. By construction any job of $\mathcal{S}$ can be scheduled before time γ_1. Therefore a simple counting argument ensures that there exists a job $s \in \mathcal{S}$ that can be scheduled at time t immediately followed by a job of type 1. If $t + p_s + p_{\langle 1 \rangle} \geq \gamma_2$, *i.e.* π completes after time γ_2, we are done. Otherwise, we have $t + p_{\langle 1 \rangle} < \gamma_2$. In this case $|\Gamma'| = |\Gamma| - 1$, while we apparently use 2 jobs of F. However since instant $t + p_{\langle 1 \rangle}$ is forbidden by construction, in fact we have $t + p_{\langle 1 \rangle} = \gamma_1$ and as a consequence $i = 1$. As we noticed, there is at least one unscheduled job of type i in A, i.e. we can use an additional job of type $i = 1$ to schedule after s. Hence we have $|\langle N' \rangle| \geq |\langle N \rangle| - 1$ which completes the proof of correctness of the algorithm.

Time complexity. We use the classic convention that basic operations on integers (addition, division,...) are performed in constant time. Then the time complexity of Algorithm 2 is in $\mathcal{O}(k + q)$, which is in $\mathcal{O}(q)$ for large diversity instances. To solve Problem $1|\text{FSE}|C_{\max}$, we call Algorithm 2 on the set of unscheduled jobs as long as there is still some forbidden instants in the future or that this set is not reduced to F. Thus we have at most k calls to Algorithm 2 as at least one forbidden instant is crossed each time, possibly followed by a call to Algorithm 1 on an instance containing at most $k + 1$ jobs. Therefore the overall complexity is in $\mathcal{O}(k|\langle N \rangle| + k^4)$. □

Notice that Theorem 5 provides a better time complexity that the one based exclusively on Algorithm 1 running in time $\mathcal{O}(k^3n)$, even for a traditionnal encoding of the instances.

6. Extensions and conclusion

Finally we consider what we called the *subcontractor problem.* We still have n independent jobs (the different products of the same order) to schedule on a single resource. All the jobs are due to complete before a given deadline D: a penalty d is charged for each day the schedule is beyond this due date. In addition we have a set $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$ of k instants where an overcost is incurred if a job starts or completes. We denote by c_j^- and c_j^+ the overcost paid to complete, respectively to start, a job at time γ_j. We can alternatively consider a joint overcost c_j paid whatever we start, complete or start and complete a job at time γ_j. This joint cost would correspond to the ability to use an additional resource during instant γ_j. Tardiness penalty d and resource overcosts c_j^- and c_j^+ are assumed to be positive. The objective is to find a schedule of minimal cost, counting tardiness penalties and resource overcosts. Notice that the subcontractor problem captures the different variants of the makespan minimization problems with forbidden instants: for instance by letting $D = 0$, $d = 1$ and $c_j^+ = +\infty$, the problem is equivalent to $1|\text{Fs }|C_{\max}$ as any finite cost solution corresponds to a feasible schedule with forbidden start instants, whose cost is clearly equal to the makespan. It happens that our result for $1|\text{Fse}|C_{\max}$ can easily be extended to the subcontractor problem:

Theorem 6 *Results of Theorems 3, 4 and 5 apply to the subcontractor cost minimization problem, i.e. large diversity instances can be solved in time complexity $\mathcal{O}(k^3n)$, and the general problem is polynomial if k is a given constant.*

Proof. For small diversity instances, we simply compute function $f(S, K)$ instead of predicate $\mathcal{P}(S, K)$, defined as the minimal cost schedule of makespan $p(S)+K$ for subset S of jobs. Due to Lemma 4, the makespan of an optimal cost schedule is at most $p(N) + 2k$. Hence the complexity remains in $\mathcal{O}(n^k)$. Note that for joint overcosts, we can add to states (S, K) a flag bit to indicate if the overcost is yet paid for instant $p(S) + K$.

Now consider a large diversity instance. Clearly if neither instant 0 nor $p(N)$ has overcost, there exists a schedule completing at time $p(N)$ without paying any overcost. This schedule is clearly optimal and is found in time $\mathcal{O}(k^3n)$ by Algorithm 1. More generally let k_1 be the first instant such that $c_{k_1}^+ = 0$. It may be advantageous (if tardiness penalty is high) to pay an overcost to start the schedule earlier than instant k_1. Let s^* and $t^* \geq p(N) + s$ be respectively the starting time and completion time of an optimal schedule. The cost of the schedule is then $c_{s^*}^+ + c_{t^*}^- + d\max\{t^* - D, 0\}$, *i.e.* no overcost is paid except possibly at instants s^* and t^*. Such a schedule is found by Algorithm 1 starting at time s^* and using a dummy job of duration $t^* - p(N) - s^*$. Notice that guessing s^* and t^* can be done in time $\mathcal{O}(k^2)$ by inspection considering all possible couples (s, t) in $[0, k_1] \times [p(N), p(N) + k]$. However this time complexity can be easily reduced to $\mathcal{O}(k)$. Let us denote by $OPT(s)$ the minimal cost among all schedules starting at time s. Let us also introduce $\xi(u) = \min\{c_t^- + d\max\{t - D, 0\} \mid u \leq t \leq p(N) + k\}$. From what precedes we have $OPT(s) = c_s^+ + \xi(s + p(N))$. Since all values of $\xi(u)$ for $u = p(N), \ldots, p(N) + k$ can be computed by accumulation in time $\mathcal{O}(k)$, determining s^* minimizing $OPT(s)$ can be achieve in time $\mathcal{O}(k)$. □

Note that without modifying the time complexity of our algorithm we could consider any tardiness penalty function $l(T)$ for the schedule instead of the linear function $l(T) = dT$, where $T = \max\{C_{\max} - D, 0\}$. We only require that $l(T)$ is positive and can be computed in constant time, possibly through an oracle. This allows to handle practical problems where the penalties grows faster than linearly, for instance in dT^2.

In this paper we have considered the scheduling of independent jobs on a single resource where k forbidden (or overcost) instants appear. We proved that an idle-free schedule always exists if the number of distinct processing times is at least $k+1$. For such so called large diversity instances we derive a strongly polynomial algorithm running in time $\mathcal{O}(k|\langle N\rangle| + k^4)$. Even if this complexity is quite low, a first perspective of this work would be to improve it. Notice that in Algorithm 2 the term in $\mathcal{O}(k^4)$ is due to the resolution (by Algorithm 1) of instances with $k+1$ distinct *jobs.* Thus one could investigate faster algorithms for large diversity instances with all distinct processing times. As a possible way to achieve this, the complexity analysis of L-partition algorithm can be refined, or improved considering a particular priority list.

As another consequence of Theorem 2, we showed that problem $1|k - \text{Fse}|C_{\max}$ where k is a fixed constant can be solved in time $\mathcal{O}(n^k)$. As we mentioned earlier, in this case an instance with $(k + 1)$ types can be solved in linear time while an instance with k types requires a time complexity of $\mathcal{O}(n^k)$. This gap motivates to design more efficient algorithms for small diversity instances, certainly using the strong result on the existence of idle free schedule for large diversity instances to develop a divide & conquer approach. Another related question would be to determine if the problem remains polynomial if we are less restrictive on the number of forbidden instants, say for instance if k is polylogarithmically bounded by n. Nevertheless, as the problem on small diversity instances is $\mathcal{NP}$-hard, it would also be of interest to develop fast algorithms to solve the problem with a small absolute error. We note that the absolute error of any list schedule is at most $2k$. We can wonder if a polynomial time algorithm can have an absolute error of 1 if for instance we have k different types. Finally it would be of particular interest to study the complexity status of $1|k - \textsc{Fse}|C_{\max}$ under a High Multiplicity encoding. The question is let open if the problem remains polynomial under a compact encoding.

References

[1] A.H. Abdekhodaee, A. Wirth, and H.S. Gan. Scheduling two parallel machines with a single server: the general case. *Computers & Operation Research*, 33:994–1009, 2006.

[2] J.-C. Billaut and F. Sourd. Single machine scheduling with forbidden start times. *4OR - Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 7:37–50, 2009.

[3] N. Brauner, Y. Crama, A. Grigoriev, and J. Van De Klundert. A framework for the complexity of high-multiplicity scheduling problems. *Journal of Combinatorial Optimization*, 9:313–323, 2005.

[4] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Rapine, H. Kellerer, C. Potts, and V. Strusevich. Operator non-availability periods. *4OR - Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, to appear. DOI 10.1007/s10288-008-0084-6.

[5] T.C.E. Cheng, G. Wang, and C. Sriskandarajah. One-operator-two-machine flowshop scheduling with setup and dismounting times. *Computers & Operation Research*, 26:715–730, 1999.

[6] J.J. Clifford and M. E. Posner. Parallel machine scheduling with high multiplicity. *Mathematical Programming*, 89(3):359–383, 2001.

[7] C. Filippi and A. Agnetis. An asymptotically exact algorithm for the high-multiplicity bin packing problem. *Mathematical Programming*, 104:21–37, 2005.

[8] N.G. Hall, C. Potts, and C. Sriskandarajah. Parallel machine scheduling with a common server. *Discrete Applied Mathematics*, 102:223–243, 2000.

[9] D.S. Hochbaum and R. Shamir. Strongly polynomial algorithms for the high multiplicity scheduling problem. *Operations Research*, 39(4):648–653, 1991.

[10] C.P. Koulamas. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers & Operation Research*, 23(10):945–956, 1996.

[11] S.A. Kravchenko and F. Werner. Parallel machine scheduling problems with a single server. *Mathematical & Computer Modelling*, 1997.

[12] V. Lebacque, N. Brauner, B. Celse, G. Finke, and C. Rapine. Planification d'expériences dans l'industrie chimique. In J.-F. Boujut, D. Llerena, and D. Brissaud, editors, *Les systèmes de production : applications interdisciplinaires et mutations*, pages 21–32. Hermès-Lavoisier, 2007. ISBN 978-2-7462-1819-2.

[13] C.-Y. Lee. Machine scheduling with availability constraints. In J.Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, pages 22–1 – 22–13. Chapman & Hall/CRC, London, 2004.

[14] J. Ou, X. Qi, and C.-Y. Lee. Parallel machine scheduling with multiple unloading servers. *Journal of Scheduling*, to appear. DOI 10.1007/s10951-009-0104-1.

[15] C. Rapine, N. Brauner, G. Finke, and V. Lehoux-Lebacque. Single machine scheduling with small operator non-availability periods. *submitted*.

Responsables de la publication : Nadia Brauner et András Sebő
ISSN : 1298-020X - © Laboratoire G-SCOP