Discrete Optimization 10 (2013) 241–250

Contents lists available at ScienceDirect

Discrete Optimization

journal homepage: www.elsevier.com/locate/disopt

A polynomial time algorithm for makespan minimization on one machine with forbidden start and completion times

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

[a] Université de Lorraine, LGIPM, île du Saulcy, 57045 Metz, France

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

ARTICLE INFO

Article history:
Received 5 July 2010
Received in revised form 24 June 2013
Accepted 18 July 2013
Available online 9 August 2013

Keywords:
Scheduling
Makespan criterion
Availability constraints

ABSTRACT

We consider the problem of scheduling independent jobs on a single resource under a special unavailability constraint: a set of *forbidden* instants is given, where no job is allowed to start or 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 quite a fast algorithm to find such a schedule, based on an hybridization between a list algorithm and local exchange. As a corollary minimizing the makespan for a fixed number of forbidden instants is polynomial.

1. Introduction

We consider in this article a scheduling problem on one machine with a special type of unavailability constraints. More precisely we call a *forbidden start & end instant* a point in time where no job is allowed to start or to complete. Notice that contrary to a classical unavailability constraint, the machine can be processing a job during a forbidden start & end instant (FSE for short, or simply *forbidden*), as long as it started its execution before the forbidden instant and will complete after it. Such forbidden instants may arise when the jobs need some additional resources at launch and at completion, and these additional resources are not continuously available, for instance if they are shared with other yet planed activities. For example, 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 set-up and at completion. While the device is continuously available, the operators have some days off and holidays. This creates some forbidden days when the jobs can be performed by the device, but none can start or complete. We encountered this problem in the chemical industry [1] through an industrial collaboration with the *Institut Français du Pétrole* (IFP), a large research centre in the fields of energy and transportation. The jobs consisted of chemical experiments whose durations typically last between 3 days and 3 weeks. The intervention of a chemist is required at the start and completion: at the start, the chemist basically fills up the device and launches the process. On completion, he has to stop the chemical reactions for the analysis of the experimental results. Each intervention can be performed within an hour, but requires the presence of the chemist in the laboratory. The objective is then to find a schedule of the experiments on the device such that an experiment neither starts nor completes on a day when the chemist is off.

The additional resource can also be for instance a special handling tool, expensive enough for the company not to own it but to call a subcontractor. For very large products like turbines for hydropower plants, a crane is needed to put the product in, and get it out of, 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 incurred to rent

* Corresponding author.
E-mail addresses: christophe.rapine@univ-lorraine.fr (C. Rapine), nadia.brauner@g-scop.inpg.fr (N. Brauner).

1572-5286/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.disopt.2013.07.003

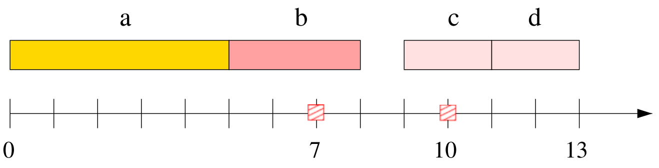

Fig. 1. The Gantt chart of a feasible schedule for $1|\text{FSE}|C_{\max}$ following the sequence (a, b, c, d). Forbidden instants $\mathcal{F} = \{7, 10\}$ are represented on the time axis by dashed rectangles. The schedule completes at time 13.

the crane on weekends. The objective is then to find a schedule of minimal duration for the different products such that ideally no overcost is paid.

These overcost instants are a generalization of the FSE instants, that we can consider as instants of infinite cost. One can imagine other additional resources such as energy (heating or cooling operations), water, etc., with consumption restriction or overcost during some periods. In this article we consider the scheduling problem of a set of jobs to be sequenced on a single resource taking into account FSE instants. The objective is to minimize the completion time of the last job, also called the *makespan* of the schedule.

The remainder 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 fixed number of forbidden instants. Finally, in Section 5, we study the extension of the problem to overcost instants.

2. Notations and preliminary definitions

We consider a set of n independent jobs, with processing times $p_1, p_2, \dots, p_n$, to be sequenced on a single resource in the presence of FSE instants. We denote by N the set $\{1, 2, \dots, n\}$ of job indices. The set $\mathcal{F}$ of forbidden instants is constituted of k distinct instants $\{\gamma_1, \gamma_2, \dots, \gamma_k\}$, indexed in increasing order. A schedule is feasible if no job starts or 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 $1|\text{FSE}|C_{\max}$. Fig. 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 appear, at time $\gamma_1 = 7$ and $\gamma_2 = 10$. We consider that the jobs are scheduled according to the sequence (a, b, c, d): due to the forbidden instant 10, the 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, the next section will assert the existence of such an idle-free schedule.

In scheduling theory, machine non-availability problems have been largely investigated, see Lee [2] for a survey. Machine non-availabilities correspond to periods where the machine cannot process any job, typically due to preventive maintenance. In contrast, an FSE instant only prevents the start or the completion of a job: the machine can go on processing its current job. Brauner et al. [3] and Rapine et al. [4] 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 or 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 $\{s+1, s+2, \dots, s+t-1\}$. However we do not consider in this article any special condition on the distribution of the forbidden instants.

Some previous works have also considered additional resources for task set-up or ending operations. Cheng et al. [5] study a 2-machine flowshop scheduling problem where the same operator performs set-up and dismounting operations on both machines following a cyclic movement pattern. This problem is connected with the server model (see for instance Hall et al. [6] and Abdekhodaee et al. [7]), where a server has to do some set-up operations before the processing of a job starts on a machine, or is required to unload the machine; see Ou et al. [8] where an example is given from logistics. Notice that a server model deals with the problem of sharing an additional resource among several machines, creating machine interferences as in Koulamas [9], while we consider in this paper unavailability constraints on the additional resource. In this context, our problem would 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 set-up time is one unit, the fixed schedule of the server is seen by this machine precisely as forbidden instants. 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. [6], while Kravchenko and Werner [10] propose a pseudo-polynomial time algorithm. More precisely they show that an optimal schedule for $P2 \parallel C_{\max}$ can be converted into an optimal schedule for $P2, S1|s_j = 1|C_{\max}$. This procedure runs in time $O(n \log n)$ and is based on local job exchanges. We use quite similar techniques in this paper, except that the forbidden instants are fixed.

The most relevant work to our problem is certainly Billaut and Sourd [11]. 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

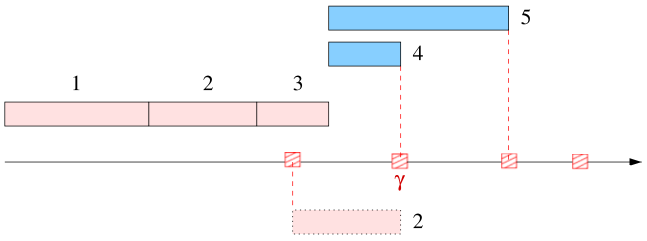

Fig. 2. An example of a blocked valid partition with $S = \{1, 2, 3\}$. Set $U = \{4, 5\}$ has been represented above the Gantt chart to visualize the corresponding forbidden instants. Under the time axis job 2 is represented to visualize that it is not exchangeable with job 4.

such an instant an Fs instant (for *forbidden start*). They prove that minimizing the makespan is polynomially solvable if the number of forbidden start times is fixed, and $\mathcal{NP}$-hard in the strong sense if this number is part of the input. Their algorithm runs in time complexity $O(n^{2k^2+2k-1})$, 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 appear in the instance. For the special case of 2 forbidden start instants, they prove that having 3 distinct processing times is a sufficient condition to assert 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 a sufficient condition 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 fixed number of forbidden instants is reduced, as a consequence of our result, to $O(n^k)$. In addition the proofs used new ideas and are far shorter.

Before presenting the concept of valid partition and exchangeable jobs, which are the main ingredients of our approach, we extend to FSE instants the theorem from Billaut and Sourd [11]:

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 and Sourd [11]. 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 $\mathcal{P} \neq \mathcal{NP}$. □

The number of distinct processing times will play a central role in our analysis. We say that two jobs are of the same *type* if and only if they have the same processing time. We denote by $\langle S\rangle$ the set of (distinct) types appearing in a subset S of jobs. The processing time of a type is defined as the processing time of a job of this type. For the sake of simplicity, we denote by p_t the processing time of a type t. We also extend p to any subset S of jobs (or of types) by letting $p(S) = \sum_{i\in S} p_i$. For short, for a given subset S, we define $p_{\min}(S)$ and $p_{\max}(S)$ as the minimal and maximal values of p_i over S, respectively.

We will prove, in the next section, that if the number $|\langle N\rangle|$ of types is greater than the number k of forbidden instants, then there exists a schedule without idle time. One originality of the proof is to focus on partitions of the jobs rather than on partial sequences. Basically, though we are interested in establishing the existence of an idle-free schedule, we will forget about the order of the jobs:

Definition 1. We call a partition $S \cup U$ of the jobs a *valid partition* if and only if there exists a schedule for S starting at time 0 and without idle time, that is, there exists a feasible schedule for the jobs of S ending at time $p(S)$.

A valid partition has to be thought of as the set S of the scheduled jobs and the set U of the unscheduled jobs at some step of a constructive algorithm. Notice that each idle-free partial sequence defines a unique valid partition, while a valid partition may correspond to several sequences. Fig. 2 shows an example of a partial schedule, defining the valid partition $S = \{1, 2, 3\}$ and $U = \{4, 5\}$. We say that a valid partition S is *blocked* if none of the unscheduled jobs can be appended to S without violating the forbidden instants, that is $p(S) + p_i \in \mathcal{F}$ for all $i \in U$. In Fig. 2, the remaining jobs to schedule are represented above the Gantt chart to stress the fact that the partition is blocked. Observe that a blocked partition S is maximal in the sense that there cannot exist a valid partition $S' \supseteq S$ of cardinality $|S| + 1$. Exchanges between jobs of S and U will be needed to "unblock" the partial schedule. Basically an exchange consists in swapping a job $s \in S$ with a job $u \in U$, such that the cardinality of the partition is unchanged. In this paper we consider only exchanges with a job $u \in U$ of minimal processing time. For short we denote by $\underline{u}$ a job of minimum duration over U, that is of processing time $p_{\min}(U)$. We introduce the following definition:

Definition 2. We call a job $s \in S$ *exchangeable* if $S \cup \{\underline{u}\} \setminus \{s\}$ defines a valid partition.

In Fig. 2 we have represented job 2 under the time axis to visualize that it is not exchangeable with job $\underline{u} = 4$: an idle-free schedule of set $\{1, 3, 4\}$ would complete on the first forbidden instant. By definition, if a job $s \in S$ is exchangeable, then any job $s' \in S$ of the same type is also exchangeable. For this reason, we extend below the notion of exchangeable job s to exchangeable type t, imposing the additional condition that set U contains no job of type t:

Definition 3. We say that a type e is exchangeable if all the jobs of type e belong to S and are exchangeable. We denote by $E(S) \subseteq \langle S\rangle$ the subset of exchangeable types.

Clearly a necessary condition for a type e to belong to $E(S)$ is that $p(S) - p_e + p_{\underline{u}}$ does not coincide with a forbidden instant. In next section Property 1 proves that it is in fact a sufficient condition in some circumstances. In Fig. 2, as job 3 is of the same type as job 4, set $E(S)$ is reduced to singleton $\langle\{1\}\rangle$.

For t a type of $\langle S\rangle$, we denote by $S_t = S \setminus \{s\} \cup \{\underline{u}\}$ the partition obtained by exchanging a job $s \in S$ of type t with the job $\underline{u}$. By definition, for any type e in $E(S)$, the set S_e defines a valid partition that differs from S only by a job of type e and $\underline{u}$. Of course their respective idle-free schedules may be totally different.

3. Existence of an idle-free schedule

We now state the main result of this paper. It describes a necessary condition for the existence of an idle-free schedule. A trivial 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 that instant 0 is not a forbidden instant. For a subset S of jobs, let $\mathcal{F}(S) = \{\gamma \in \mathcal{F} | \gamma \le p(S)\}$ be the set of forbidden instants appearing before time $p(S)$. Clearly we have $|\mathcal{F}(N)| \le k$. The remaining of this section is devoted to prove Theorem 2:

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

Theorem 2 can be rephrased as $N \cup \emptyset$ is a valid partition if $|\langle N\rangle| > |\mathcal{F}(N)|$ and neither instants 0 nor $p(N)$ are forbidden. We prove this result by induction on $|\mathcal{F}(N)|$. If $|\mathcal{F}(N)| = 0$, certainly Theorem 2 holds. Now consider that $|\mathcal{F}(N)| > 0$ and assume that the result is true for any set S of jobs such that $|\mathcal{F}(S)| < |\mathcal{F}(N)|$. That is, if $|\langle S\rangle| > |\mathcal{F}(S)|$ and $0, p(S) \notin \mathcal{F}$, then $S \cup (N \setminus S)$ is a valid partition. First, we use the induction hypothesis to prove the following property. It shows that the set of exchangeable types can be simply characterized as the types e such that the length of the set S_e does not coincide with a forbidden instant.

Property 1 (*Characterization of $E(S)$*)**.** *Consider a valid partition $S \cup U$ such that S is blocked. Then we have $E(S) = \{e \notin \langle U\rangle \mid p(S_e) \notin \mathcal{F}\}$.*

Proof. Recall that $S_e = S \setminus \{s\} \cup \{\underline{u}\}$, where s is a job of type e and $\underline{u}$ is a job of U with the smallest execution time. If $e \in E(S)$, then $p(S_e)$ cannot coincide with an instant of $\mathcal{F}$ by the definition of a valid partition. Hence we clearly have $E(S) \subseteq \{e \notin \langle U\rangle \mid p(S_e) \notin \mathcal{F}\}$.

To establish the reverse inclusion, consider a type $e \notin \langle U\rangle$ such that $p(S_e) \notin \mathcal{F}$. Let $\underline{\gamma}$ be the instant $p(S) + p_{\underline{u}}$. Notice that $p(S_e)$ is equal to $p(S) + p_{\underline{u}} - p_e$, and thus is lower than $\underline{\gamma}$. Since S is blocked, there exist at least $|\langle U\rangle|$ forbidden instants in interval $[\underline{\gamma}, p(N)]$, as $p(S) + p_u \in \mathcal{F}$ for all $u \in \langle U\rangle$. Thus we have $|\mathcal{F}(S_e)| \le |\mathcal{F}(N) \cap [0, \underline{\gamma})| \le |\mathcal{F}(N)| - |\langle U\rangle|$. In particular we have $|\mathcal{F}(S_e)| < |\mathcal{F}(N)|$. To use the induction hypothesis on S_e, it remains to establish that $|\langle S_e\rangle| > |\mathcal{F}(S_e)|$. Clearly we have $S_e \cup U \supseteq N \setminus \{s\}$ and $S_e \cap U \supseteq \{\underline{u}\}$. As for any sets A and B, we can write that $|\langle A \cup B\rangle| = |\langle A\rangle| + |\langle B\rangle| - |\langle A \cap B\rangle|$, it results in $|\langle S_e \cup U\rangle| = |\langle S_e\rangle| + |\langle U\rangle| - |\langle S_e \cap U\rangle|$. Hence we have $|\langle S_e\rangle| \ge |\langle N \setminus \{s\}\rangle| - |\langle U\rangle| + 1 \ge |\langle N\rangle| - |\langle U\rangle|$. It results in $|\langle S_e\rangle| > |\mathcal{F}(S_e)|$. Since $|\mathcal{F}(S_e)| < |\mathcal{F}(N)|$ and $p(S_e) \notin \mathcal{F}$ by our choice of e, the induction hypothesis shows that S_e is a valid partition, which concludes the proof. □

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, namely that the largest exchangeable type should be smaller than the smallest unscheduled type.

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

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

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

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 $p(S) + p_{\overline{u}}$, respectively, we have $\varphi(T(S)) \subseteq [0, \underline{\gamma})$ and $\varphi(\langle U\rangle) \subseteq [\underline{\gamma}, \overline{\gamma}]$. Thus φ is injective on $\langle U\rangle \cup T(S)$. The fact that S is blocked along with Property 1 shows that the image of the set $\langle U\rangle \cup T(S)$ under φ is included in $\mathcal{F}(N)$. As a consequence we have $|\langle U\rangle \cup T(S)| \le |\mathcal{F}(N)|$. Thus condition $|\langle N\rangle| > |\mathcal{F}(N)|$ necessarily implies that $\langle U\rangle \cup T(S) \subset \langle N\rangle$. In other words the set $E(S)$ cannot be empty, which proves the first assertion of Lemma 1.

Now consider that the condition $p_{\max}(E(S)) < p_{\min}(U)$ holds. Let us denote by $\overline{e}$ a job of maximal duration among the jobs of exchangeable type, $E(S)$. Recall that $\underline{u}$ denotes a job of U of minimal duration. Due to the condition, it results that for any type $e \in E(S)$ we have $p(S_e) \ge p(S) - p_{\overline{e}} + p_{\underline{u}} > p(S)$. As a consequence $\varphi(E(S)) \subseteq (\overline{\gamma}, p(N)]$ and thus φ is an injection on $\langle N\rangle$. Since $|\langle N\rangle| > |\mathcal{F}(N)|$, there necessarily exists a type e such that $\varphi(e) \notin \mathcal{F}(N)$. As already observed, we have the inclusions $\varphi(\langle U\rangle) \subseteq \mathcal{F}(N)$ and $\varphi(T(S)) \subseteq \mathcal{F}(N)$. Thus type e must be an exchangeable type. We claim that the valid partition S_e is not blocked. First, observe that set U cannot be reduced to a singleton $\{\underline{u}\}$: otherwise we have $p(S)+p_{\underline{u}} = p(N)$, and thus the fact that S is blocked contradicts our assumption that $p(N)$ does not coincide with a forbidden instant. It can be readily checked that if U contains at least 2 jobs, then the set $U_e = N \setminus S_e$ obtained after the exchange still contains a job of type $\overline{u}$. Writing the definition of $\varphi(e)$ and using the fact that $\varphi(e) \notin \mathcal{F}(N)$, we get $p(S_e) + p_{\overline{u}} \notin \mathcal{F}(N)$. It implies that a job $\overline{u} \in U_e$ can be appended to S_e without violating a forbidden instant, which shows that S_e is not blocked. □

Now to conclude the proof of Theorem 2, assume for the sake of contradiction that $N \cup \emptyset$ is not a valid partition. Among all the valid partitions (at least $\emptyset \cup N$ is one), we choose $S \cup U$ maximizing $|S|$, and of minimal length $p(S)$ among all partitions of maximal cardinality. The maximality of $|S|$ implies that S is blocked. We claim that necessarily $p_{\max}(E(S)) < p_{\min}(U)$: indeed for any type $e \in E(S)$, set S_e defines a valid partition of cardinality $|S|$ and of length $p(S_e) = p(S) + p_{\underline{u}} - p_e$; the minimality of $p(S)$ among the partitions of the same cardinality imposes that $p_e \leq p_{\underline{u}}$. However, by the definition of an exchangeable type, $e \notin \langle U\rangle$, and in particular $p_e \neq p_{\underline{u}}$, which leads to the inequality $p_e < p_{\underline{u}}$. Thus we are in the conditions stated by Lemma 1. It implies that there exists e such that $S_e \cup \{\overline{u}\}$ defines a valid partition, with $\overline{u}$ a job of maximal duration in U (see the proof of Lemma 1). This contradicts the maximality of $|S|$, and achieves the proof of Theorem 2.

Note that the assumptions of Theorem 2 cannot be further relaxed to ensure the existence of an idle-free schedule. Indeed if $|\langle N\rangle| = |\mathcal{F}(N)|$, for some instances, idle times will be forced in any schedule: consider for example the set $\mathcal{F} = \{p_i | i \in \langle N\rangle\}$. We have $|\langle N\rangle| = |\mathcal{F}(N)|$ and necessarily an idle time appears in any feasible schedule at time 0, although it is not a forbidden instant.

We stated Theorem 2 with the general condition $|\langle N\rangle| > |\mathcal{F}(N)|$. Of course if the number of types is greater than the number of forbidden instants, this condition is always fulfilled. This motivates the following definition:

Definition 4. An instance is said to be of *large diversity* if $|\langle N\rangle| > |\mathcal{F}|$, of *small diversity* otherwise.

4. Polynomial time algorithm

Theorem 2 proves the existence of a schedule without idle time for large diversity instances. However the proof, based on valid partitions, is not constructive. In this part, we derive an algorithm to construct such a schedule. The algorithm is a simple hybridization between a list scheduling algorithm and a 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 permits obtaining a partial schedule that is no more blocked. The basic idea is to exchange at each step the smallest unscheduled job $\underline{u}$ with a (scheduled) job of the largest exchangeable type $\overline{e}$. We call such an exchange a *min–max* exchange. Using the vocabulary of the previous section, if (S, U) is a blocked partition, a *min–max* exchange consists then in exchanging a job $\underline{u}$ of processing time $p_{\min}(U)$ with a job of type $\overline{e}$ of processing time $p_{\max}(E(S))$. These exchanges are performed as long as S remains blocked and that $p_{\max}(E(S)) \geq p_{\min}(U)$. If S is blocked but $p_{\max}(E(S)) < p_{\min}(U)$, we then select a job of type $e \in E(S)$ such that S_e is not blocked. We call this latter exchange an *unblocking* exchange. Notice that Lemma 1 ensures that, if S is blocked, there does exist exchangeable jobs. It also ensures that if $p_{\max}(E(S)) < p_{\min}(U)$, an *unblocking* exchange is always possible. Next lemma bounds the number of successive exchanges. Recall that k is the number of forbidden instants.

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

Proof. Let $(S_1, U_1), \dots, (S_l, U_l), (S_{l+1}, U_{l+1})$ 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 $p_{\min}(U_i) < p_{\max}(E(S_i))$. 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 in N of processing times smaller than or equal to $p_{\min}(U_i)$, that is β_i is the rank of the type of processing time $p_{\min}(U_i)$ following a non-decreasing order. Notice that α_i and β_i are non-decreasing sequences. More precisely for all index $i = 0, \dots, l - 1$:

(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 the partition (S_i, U_i) into (S_{i+1}, U_{i+1}) is a *min–max* exchange. Let $\overline{e}_i$ be the type of the job, of duration $p_{\max}(E(S_i))$, selected to exchange with $\underline{u}_i$ of duration $p_{\min}(U_i)$. By the definition of $E(S_i)$, type $\overline{e}_i$ does not belong to $\langle U_i\rangle$. In addition $\overline{e}_i$ 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 $\overline{e}_i$, *i.e.* we are in situation (1). Otherwise $\underline{u}_i$ is the only job of duration $p_{\min}(U_i)$ in U_i. Then clearly we have $p_{\min}(U_{i+1}) > p_{\max}(U_i)$, which corresponds to situation (2).

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

To decrease the number of exchanges from $|\langle N\rangle|$ to $k + 1$, we can restrict our attention to the set $\mathcal{M}$ of the first $(k + 1)$ types. More precisely in each *min–max* exchange, we select a job e of the greatest processing time in the set $E(S) \cap \mathcal{M}$. If a partition is blocked, this set cannot be empty. Indeed, one can readily follow the proof of Lemma 1, substituting the set $\langle N\rangle$ by any subset of cardinality greater than k. If the partition is blocked, the function φ restricted to the set $\mathcal{M}$ clearly remains injective, and the condition $|\mathcal{M}| > k$ permits to conclude in the same way that $E(S) \cap \mathcal{M} \neq \emptyset$. In addition, if $\mathcal{M}$ does not contain a type of processing time $p_{\min}(U)$, an *unblocking* exchange is then performed by the algorithm. Otherwise *min–max* exchanges will swap only jobs of type in $\mathcal{M}$. By the previous argument the number of exchanges is then bounded by $|\mathcal{M}| = k + 1$. □

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, say of type e, and $\underline{u}$. Indeed if Theorem 2 ensures that S_e 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 to decompose the problem:

Definition 5. Given a list L, a subset $N' \subset N$ defines a *L-partition* if:

(1) set N' defines a valid partition,
(2) if N' is not empty, then we have $|\mathcal{F}(N')| < |\mathcal{F}(N)|$ and $|\mathcal{F}(N')| < |\langle N'\rangle|$. That is, set N' together with the forbidden instants $\mathcal{F}(N')$ define a large diversity instance.
(3) the remaining jobs of $N \setminus N'$ can be scheduled in time interval $[p(N'), p(N)]$ by the list scheduling algorithm of list L, without idle time.

We represent such an 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, in finding an L-partition (N', π). Considering the conditions of Definition 5, the existence of an L-partition is not guaranteed. We will prove in Lemma 3 that for any large diversity instance, an L-partition exists and can be found quite efficiently. First, we want to make apparent here why the L-partition problem constitutes a helpful way to find an idle-free schedule for large diversity instances. Assume that we have an algorithm $\mathcal{A}$ to solve the L-partition problem on large diversity instances. If we call $\mathcal{A}$ on the set N of jobs with the set $\mathcal{F}$ of forbidden instants, we obtain a first L-partition (N_1, π_1). By definition, the sequence π_1 describes an idle-free schedule for the jobs of $N \setminus N_1$, starting at time $p(N_1)$. Thus the problem reduces to finding an idle-free schedule for the instance composed of the set N_1 of jobs, and restricted to the set $\mathcal{F}(N_1)$ of forbidden instants. If the set N_1 is empty, we can directly return π_1 as an idle-free schedule for N. Otherwise, as the restricted instance is still of large diversity due to condition (2), we call the L-partition algorithm $\mathcal{A}$ on the set N_1 to obtain an L-partition (N_2, π_2). This way, we iteratively call $\mathcal{A}$ on the current subset N_l of jobs till N_l is empty. At this point we can return the concatenation $\pi_l\pi_{l-1}\cdots\pi_1$ of the sequences as an idle-free schedule for N. Basically this procedure constructs an idle-free schedule from the end, ensuring that at least one more forbidden instant is covered by the scheduled tasks at each call of the algorithm $\mathcal{A}$. Formally, due to condition (2) in Definition 5 of a L-partition, we can assert that after at most $(k + 1)$ iterations we terminate on an empty set N_l. Indeed, as the number of forbidden instants $|\mathcal{F}(N_j)|$ decreases by at least one at each iteration, if the set N_k is not empty we necessarily have $|\mathcal{F}(N_k)| = 0$. That is, there is no forbidden instant on the time interval $[0, p(N_k)]$. At the next iteration the algorithm $\mathcal{A}$ must return an empty set N_{k+1} to fulfil condition (2) of Definition 5. By the way, any list scheduling algorithm can easily schedule the set N_k without idle time since no instant is forbidden in the restricted instance.

As a consequence, if one can design an algorithm for the L-partition problem, one can find an idle-free schedule for large diversity instances. This is summarized in the following remark:

Remark 1. For large diversity instances, 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 $O(kt(k, n))$.

Thus we can restrict our attention to developing an algorithm for the L-partition problem. This algorithm, formally described by Algorithm 1, mixes greedy allocation with local exchanges of jobs. That is, each time a partial schedule cannot be extended without creating an idle time, *min–max* exchanges are performed until an *unblocking* exchange takes place. To deliver an 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 Fig. 1, which is an instance of large diversity. Using list $L = (a, b, c, d)$, jobs a and b are successively scheduled. We are then in situation of Fig. 3 with a blocked partition $S = \{a, b\}$, $U = \{c, d\}$.

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 the situation of Fig. 4 with a new valid partition $S' = \{b, c\}$.

Partition S' is still blocked. As jobs 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 Fig. 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 $p(S'')$.

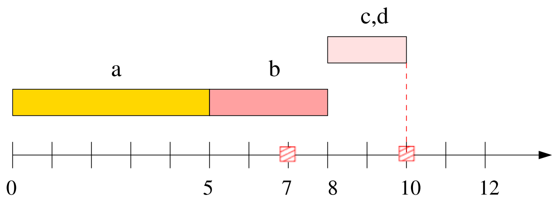

Fig. 3. The partial schedule obtained for list $L = (a, b, c, d)$ on the instance of Fig. 1.

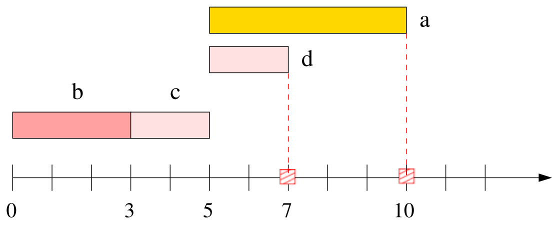

Fig. 4. The partial schedule obtained after exchanging a and c.

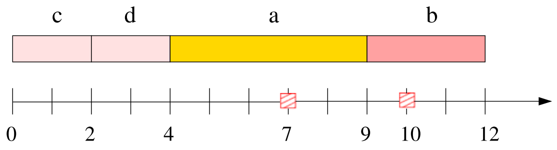

Fig. 5. The final schedule. The corresponding L-partition is $(\{c, d\}, ab)$.

Note that, by definition of *min–max* exchanges, the length $p(S)$ of the current valid partition decreases between 2 exchanges. Hence we cannot assert that if S is a blocked partition, after a series of exchanges the resulting non-blocked partition S' will obey $|\mathcal{F}(S')| > |\mathcal{F}(S)|$. Such a property would have bounded the total number of exchanges to obtain an L-partition by $O(k^2)$. However we can assert that this number is at most $O(kn)$, as a new job is scheduled after each series of exchanges. We have the following lemma:

Lemma 3. *For any large diversity instance verifying conditions of Theorem 2, an L-partition exists and can be found in time $O(k^2n)$.*

Proof. We first show that, if Algorithm 1 terminates, it returns an L-partition. If no exchange occurred, by definition $N' = \emptyset$ defines an 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 of some type $e \in E(S)$ is exchanged with $\underline{u}$. Notice that after the exchange, by construction, S_e 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. Thus set S_e fulfils conditions (1) and (3) of Definition 5 of a L-partition. Finally, in Property 1 characterizing the exchangeable types, we have established that for any type $e' \in E(S)$, $|\langle S_{e'}\rangle| > |\mathcal{F}(S_{e'})|$ and $|\mathcal{F}(S_{e'})| < |\mathcal{F}(N)|$. In particular this is true for the type e: as a consequence S_e also fulfils the second condition, and thus is an 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 $O(kn)$. To obtain the time complexity of the algorithm, we show that both the allocation and the exchange step can be done in time $O(k)$. Note that L can be described as a vector of integers of dimension $|\langle N\rangle|$. More generally each subset can be represented by a vector of types. For an allocation step 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 $O(k)$. For an exchange step, we scan only the $k + 1$ smallest types of set S as explained in Lemma 2 to determine the set $E(S) \cap \mathcal{M}$. Checking if a type is exchangeable can be done in constant time due to Property 1. Thus an exchange step can also be achieved in time $O(k)$. □

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

Proof. If neither instant 0 nor $p(N)$ belongs to $\mathcal{F}$, Theorem 2 proves that an idle free schedule exists. This schedule can be found in time $O(k^3n)$ by Algorithm L-partition due to Lemma 3 and Remark 1. Notice that conditions (1) and (2) in Definition 5 ensure that the set N' obtained from the L-partition of N verifies in its turn the conditions of Theorem 2, and thus each iterative call to Algorithm 1 can be achieved in time $O(k^2n)$.

Otherwise, either instant 0 or $p(N)$ belongs to $\mathcal{F}$, and some idle times necessarily occur in any schedule. Let t_1 be the first integer instant which is not forbidden, and let t_2 be the first integer instant greater than $p(N) + t_1$ which is also

Algorithm 1 L-partition Algorithm

```
Require: a set $N$ of jobs and a set $\mathcal{F}$ of forbidden instants verifying conditions of Theorem 2
Ensure: an $L$-partition $(N',\pi)$
  mark all jobs unscheduled // $U = N$
  set $S = \emptyset$ ; $N' = \emptyset$ ; $\pi = \emptyset$ // initialization
  while an unscheduled job remains do // $U \neq \emptyset$
    // perform local exchanges while $S$ is blocked
    while $S$ is blocked do
      if $p_{\max}(E(S)) > p_{\min}(U)$ 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 $S := S_x$ ; $N' := S_x$ ; $\pi := \emptyset$ ; $U := N \backslash S$
    end while
    // greedy list allocation
    select the first unscheduled job $u$ in $L$ such that $p(S) + p_u \notin \mathcal{F}$
    mark $u$ scheduled
    set $S := S \cup \{u\}$; $\pi := \pi\{u\}$; $U := U \backslash \{u\}$
  end while
  return $(N', \pi)$
```

not forbidden. Obviously no feasible schedule can complete before time t_2. Let $\tau = t_2 - t_1 - p(N)$. We start the schedule at time t_1 adding to N a dummy job of duration τ. Thus we are in conditions of Theorem 2: we can find an idle free schedule completing at time t_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 $1|\text{FSE}|C_{\max}$ is $\mathcal{NP}$-hard, we cannot hope to obtain a polynomial time algorithm, not even a constant approximation algorithm, if one believes that $\mathcal{P} \neq \mathcal{NP}$. However, the 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 slot. Due to the greedy allocation of the algorithm, necessarily either t or $t + p_l$ belongs to $\mathcal{F}$, otherwise l should have been scheduled at time t. Hence the set of idle slots is included in $\{\gamma_j, \gamma_j - p_l \mid j \in 1, \ldots, k\}$, which is of cardinality at most $2k$. $\square$

The bound of Lemma 4 is tight: consider an instance constituted of a single job of duration 1 and with 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 fixed, an optimal schedule can be found in polynomial time.*

Proof. Consider an instance $(N, \mathcal{F})$ of problem $1|k-\text{FSE}|C_{\max}$. If $|\langle N\rangle| > |\mathcal{F}|$, Theorem 3 proves that we can find an optimal schedule in linear time $O(n)$. Otherwise, in the 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 $(s_1, \ldots, s_q)$, where s_i is the number of jobs of type i present in S. As in Billaut and Sourd [11], we use dynamic programming to compute predicate $\mathcal{P}(S, K)$ which is 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 $K \in \{0, \ldots, 2k - 1\}$ (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 $(n_1, \ldots, n_q)$ is the multiplicity vector of N. Each state $\mathcal{P}(S, K)$ can be computed in time $O(q)$ by considering which type (or idle slot) can be scheduled in last position. Hence the running time of the dynamic program is in $O(2kq(n/q + 1)^q)$. For k a constant, this time complexity is bounded by $O(n^k)$, which is a polynomial in n. $\square$

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

5. Extension to overcost instants

Finally, we consider what we name the *subcontractor problem*. We still have n independent jobs to schedule on a single resource. All the jobs are due to complete before a common due date D. This may correspond to the situation where the jobs represent different products of the same order to be delivered at time D. If the last job completes after time D, a penalty b is charged for each day the schedule is beyond this due date.

In addition to this tardiness penalty, at each integer instant t an overcost is eventually incurred if a job starts or completes. More precisely we denote by c_t^- and c_t^+ the overcost paid to complete, respectively to start, a job at time t. Tardiness penalty b and resource overcosts c_t^- and c_t^+ are assumed to be positive or null. The objective is to find a schedule of minimal cost, accounting for the tardiness penalty and the resource overcosts. We denote by $\mathcal{G}$ the set of integer instants with a positive overcost, that is $\mathcal{G} = \{t \in \mathbb{Z}_+ \mid c_t^- + c_t^+ > 0\}$. The instants in $\mathcal{G}$ are said to be the overcost instants. To use more standard scheduling notations, let s_i and C_i be the starting time and the completion time of job i in a given schedule, and let $T_{\max}$ denote the maximum tardiness of a job, that is $\max\{0, C_{\max} - D\}$. The cost of a schedule can then be written as $\sum_i c^+(s_i) + \sum_i c^-(C_i) + bT_{\max}$.

We can alternatively consider a joint overcost c_t paid whatever we start, complete or start and complete a job at time t. This joint cost would correspond to the ability to use an additional resource during instant t. Notice that the subcontractor problem captures the different variants of the makespan minimization problems with forbidden instants: for instance by letting $D = 0$, $b = 1$ and $c_t^+ = +\infty$ if $t \in \mathcal{G}$, 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 results for $1|\text{FSE}|C_{\max}$ can easily be extended to the subcontractor problem, as stated in Theorem 5 below. Since forbidden instants are a special case of overcost instants, we extend our definition of large diversity instances to the subcontractor problem: An instance is said of large diversity if the number of overcost instants, $k = |\mathcal{G}|$, is smaller than the number of types.

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

Proof. For small diversity instances, we simply compute the function $f(S, K)$ instead of the predicate $\mathcal{P}(S, K)$, defined as the minimal cost of a schedule for the subset S of jobs with a makespan of $p(S) + K$. Due to Lemma 4, the makespan of an optimal schedule is at most $p(N) + 2k$. Hence the complexity remains in $O(n^k)$. 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, due to Theorem 2, if neither instant 0 nor $p(N)$ are overcost instants, there exists a schedule completing at time $p(N)$ without paying any overcost. This schedule is clearly optimal and is found in time $O(k^3 n)$ by Algorithm 1. More generally, let $\tilde{s}$ be the first instant such that $c_{\tilde{s}}^+ = 0$. Notice that $\tilde{s} \le k + 1$. It may be advantageous (if the tardiness penalty is high) to pay an overcost to start the schedule earlier than instant $\tilde{s}$. Let s^* and t^* be respectively the starting time and completion time of an optimal schedule. Necessarily we have $t^* \ge p(N) + s^*$ and $t^* \le p(N) + s^* + k$, since there is no incentive to delay the end of the last job if it can complete at a non-overcost instant. The cost of the schedule is then $c_{s^*}^+ + c_{t^*}^- + b\max\{t^* - D, 0\}$, that is, 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 $O(k^2)$ by inspection, considering all possible couples (s, t) in $[0, \tilde{s}] \times [p(N), p(N) + 2k]$. However this time complexity can be easily reduced to $O(k)$ in the following way. Let us denote by $\text{OPT}(s)$ the minimal cost among all the schedules starting at time s. Let us also introduce $\xi(u) = \min\{c_t^- + b\max\{t - D, 0\} \mid u \le t \le p(N) + 2k\}$. From what precedes we have $\text{OPT}(s) = c_s^+ + \xi(s + p(N))$. Since all values of $\xi(u)$ for $u = p(N), \dots, p(N) + 2k$ can be computed by accumulation in time $O(k)$, determining s^* minimizing $\text{OPT}(s)$ can be achieved in time $O(k)$. □

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

6. Conclusion

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 $k+1$, and neither 0 nor $p(N)$ is a forbidden instant.

We proposed an efficient $O(k^3 n)$ algorithm to solve any large diversity instance, and we established, as a consequence of Theorem 2, that problem $1|k - \text{FSE}|C_{\max}$ can be solved in time $O(n^k)$. This time complexity is polynomial for any fixed value k. In this case, recall that an instance with $(k+1)$ types can be solved in linear time, while an instance with k types requires precisely a time complexity of $O(n^k)$. This gap motivates us to design more efficient algorithms for small diversity instances, certainly using the strong result on the existence of an 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 logarithmically bounded by n. This would capture practical situations when the number of forbidden instances to consider grows (slowly) with the number of jobs to schedule. However, since the problem on small diversity instances is $\mathcal{NP}$-hard for arbitrary values of k, it would be also 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.

In this paper, the types of the jobs play a central part in the design of the algorithms. We may consider a different encoding of the instances, describing for each type the processing time and the number of jobs of this type—the multiplicity. One then falls in the field of high-multiplicity scheduling problems, see e.g. [12] for a seminal paper and [13] for more recent references. A High Multiplicity encoding has a size linear in the number of types, but only logarithmic in the number of jobs. Thus the algorithm presented in this paper, and any algorithm allocating jobs one by one, is exponential in the size of this encoding. It would be of particular interest to study the complexity status of $1|k - \textsc{Fse}|C_{\max}$ under High Multiplicity encoding. The question whether the problem remains polynomial under such a compact encoding is left open.

References

[1] V. Lebacque, N. Brauner, B. Celse, G. Finke, C. Rapine, Planification d'expériences dans l'industrie chimique, in: J.-F. Boujut, D. Llerena, D. Brissaud (Eds.), Les systèmes de Production: Applications Interdisciplinaires et Mutations, Hermès-Lavoisier, ISBN: 978-2-7462-1819-2, 2007, pp. 21–32.
[2] C.-Y. Lee, Machine scheduling with availability constraints, in: J.Y.-T. Leung (Ed.), Handbook of Scheduling: Algorithms, Models and Performance Analysis, Chapman & Hall/CRC, London, 2004, chapter 22.
[3] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Rapine, H. Kellerer, C. Potts, V. Strusevich, Operator non-availability periods, 4OR: A Quarterly Journal of Operations Research 7 (3) (2008) 239–253.
[4] C. Rapine, N. Brauner, G. Finke, V. Lebacque, Single machine scheduling with small operator-non-availability periods, Journal of Scheduling 15 (2012) 127–139.
[5] T.C.E. Cheng, G. Wang, C. Sriskandarajah, One-operator-two-machine flowshop scheduling with setup and dismounting times, Computers & Operations Research 26 (1999) 715–730.
[6] N.G. Hall, C. Potts, C. Sriskandarajah, Parallel machine scheduling with a common server, Discrete Applied Mathematics 102 (2000) 223–243.
[7] A.H. Abdekhodaee, A. Wirth, H.S. Gan, Scheduling two parallel machines with a single server: the general case, Computers & Operations Research 33 (2006) 994–1009.
[8] J. Ou, X. Qi, C.-Y. Lee, Parallel machine scheduling with multiple unloading servers, Journal of Scheduling 13 (3) (2009) 213–226.
[9] C.P. Koulamas, Scheduling two parallel semiautomatic machines to minimize machine interference, Computers & Operations Research 23 (10) (1996) 945–956.
[10] S.A. Kravchenko, F. Werner, Parallel machine scheduling problems with a single server, Mathematical and Computer Modelling 26 (12) (1997) 1–11.
[11] J.-C. Billaut, F. Sourd, Single machine scheduling with forbidden start times, 4OR: A Quarterly Journal of Operations Research 7 (2009) 37–50.
[12] D.S. Hochbaum, R. Shamir, Strongly polynomial algorithms for the high multiplicity scheduling problem, Operations Research 39 (4) (1991) 648–653.
[13] N. Brauner, Y. Crama, A. Grigoriev, J. Van De Klundert, A framework for the complexity of high-multiplicity scheduling problems, Journal of Combinatorial Optimization 9 (2005) 313–323.