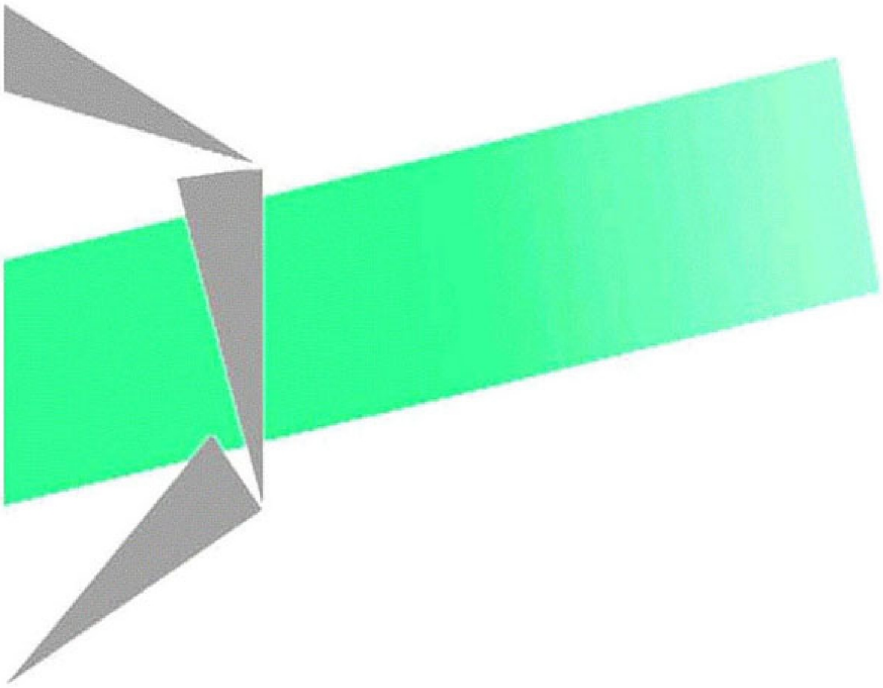
Les cahiers
Leibniz



Single Machine Scheduling with Small Operator-Non-Availability Periods

C. Rapine, N. Brauner, G. Finke, V. Lebacque

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

n° 178

Mars 2009

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

Single Machine Scheduling with Small Operator-Non-Availability Periods

C. Rapine, N. Brauner, G. Finke, V. Lebacque

Laboratoire G-SCOP, Grenoble University, France

Abstract

Through an industrial application, we were confronted with the planning of experiments where human intervention of a chemists is required to handle the starting and termination. This gives rise to a new type of scheduling problems, namely problems of finding schedules with time periods when the tasks can neither start nor finish. We consider in this paper the natural case of *small* periods where the duration of the periods is smaller than any processing time. This assumption corresponds to experiments lasting several days whereas the operator unavailability periods are the week-ends. These problems are analyzed on a single machine with the makespan as criterion.

We first prove that, contrary to the case of machine unavailability, the problem with one small operator non-availability period can be solved in polynomial time. We then derive approximation and inapproximability results for the general case of k small unavailability periods. We finally focus on the practical case of periodic and equal small unavailability periods. We prove that this problem is not in FPTAS for more than 3 periods and we derive an FPTAS for the two-period case.

keywords: One machine scheduling, unavailability constraints, forbidden start and end, approximation

1 Introduction

Machine unavailability periods is a well known notion in scheduling theory. During such a period the machine is not available to process jobs, typically due to preventive maintenance. Depending on the model (*resumable* or *non-resumable*), the processing of a job may or may not be interrupted during an unavailability period, and resumed partially afterwards. We consider in this paper a new availability constraint defining a new type of period : in contrast with machine unavailability period, a job can be processed, but can neither start nor complete during the period. We were confronted with such special unavailability periods through an industrial collaboration with the *Institut Français du Pétrole* (IFP), a large research center in fields of energy and transport. Their problem consists in the planning of a large set of chemical experiments. Each experiment is performed by an automatic device (a robot), during a specified amount of time, but the intervention of a chemist is required to handle its start and termination. At the beginning, the chemist basically prepares the chemical, fills up the device and launches the process. The termination phase corresponds to an analysis of the experimental results, which is to be carried out in a no-wait fashion to stop chemical reactions (see [14] for a detailed description). While the automatic device is operational and available seven days a week, the chemists may be absent from the laboratory due to weekends, vacations or other planned activities. This creates time intervals when experiments can be performed by the device, but none can start nor complete. It leads us to define the following notion in scheduling theory:

Definition 1 *An operator non-availability (*Ona*) period corresponds to an open time interval in which no task can start nor end.*

Although we named it *operator* non-availability in contrast to *machine* non-availability, such periods of forbidden start and completion may be encountered each time an additional resource (for instance a special tool for handling) is required to start and to complete a task. If this additional resource is not continuously available and its time of use is negligible compared to the processing time of a task, it then results into an

Ona scheduling problem. We discuss in this paper the complexity and approximability of scheduling problems where Ona periods occur. We focus on a one machine environment with the makespan as criterion. Problem Onas is formally defined in the following way:

Problem Operator Non-Availability Scheduling (Onas)
Instance: A set of n tasks, of durations $p_1, \ldots, p_n$, together with a list of k intervals $(s_j, s_j + L_j)$
Solution: A schedule π, such that no task ends nor starts in any open time interval $(s_j, s_j + L_j)$.
Measure: The makespan of π.

We call $k-$Onas the version of the problem where k is not part of the instance and thus is a constant.

To our best knowledge, the scheduling model with Ona periods was first introduced in [14] and studied in [2]. The authors analyze the complexity of the general problem and establish that any list scheduling algorithm has a performance ratio of $2(k-1)$ for $k \geq 4$, this bound being tight. Other works have yet considered an additional resource for task set-up: in the single server model [9, 3], a server has to do some set-up before the processing of a job starts on a machine. Compared to a server problem, we neglect the set-up time of the operator while no article have considered unavailability periods for the server. The most relevant work to our problem is certainly [1] where a set of time slots is forbidden for starting the processing of a job on the machine. They prove that the problem is polynomially solvable if the number of forbidden start times is a constant, and $\mathcal{NP}$-hard in the strong sense if this number of instants is part of the input. For a general introduction to classical machine non-availability periods, we refer the reader to Lee [15].

The paper is organized as follows. We show in Section 3 that minimizing the makespan is already $\mathcal{NP}$-hard for one Ona period, but becomes polynomially solvable if we restrict to one unavailability period whose duration is smaller than any processing time. This assumption is quite natural, and at least is verified in our industrial context where experiments last several days and Ona periods generally correspond to weekends of two days. The remaining of the article focuses on *small* Onas where all unavailability periods are smaller than any processing time. Then Section 4 presents complexity and inapproximability results for the general small Onas problem. In section 5 we analyze the performances of list scheduling algorithms and derive a PTAS for $k-$Onas. Section 6 is devoted to the special case of periodic unavailability periods, which is quite relevant in practice if one thinks of Ona periods as weekends. Finally section 7 proposes a FPTAS for the 2-Onas periodic case.

2 Notations and basic properties

Throughout this paper we consider a set of n independent tasks to schedule on a single resource in presence of k Ona periods. The following notations will be used:

- p_i: duration of task i
- k: number of Ona periods.
- s_j: beginning of the jth Ona period
- L_j: duration of the jth Ona period
- $L \equiv \max_j L_j$: the maximum duration
- $\delta_i = p_i - L$: margin of task i
- $\delta \equiv \max_i \delta_i$: maximum margin

We denote by N the set $\{1, \ldots, n\}$ of task indices, and for a subset $X \subseteq N$, we note for short $p(X) = \sum_{i \in X} p_i$. For convenience we define $s_0 = L_0 = 0$ and $s_{k+1} = +\infty$. To avoid trivial cases, we assume that $n \geq 2$ and $p(N) > s_1$, *i.e.* we have at least 2 tasks to schedule, and not all the tasks can fit before the first Ona period.

For a schedule π, we denote by $C_{\max}(\pi)$ its makespan, defined as the largest completion time of a task. The quantity $OPT(x)$ refers to the minimum value of $C_{\max}(\pi)$ over all feasible schedules for a given instance x. Recall that a schedule π is feasible if no task starts nor ends in any time interval $(s_j, s_j + L_j)$. Preemption is not allowed and tasks are non-resumable. A task that starts before time s_j for some j and completes after time $s_j + L_j$ is said to *cover* the jth Ona period. Notice that an Ona period is covered by at most one task, while a task may cover several periods. In our notations δ_i represents the margin left to task i if it covers a period of duration L to be moved "backward" or "forward" in the schedule. Figure 1 presents an example of a feasible schedule for an instance of 3-Onas.

Our aim in this article is to determine the approximability status of the Onas problem. In our context, an algorithm is said to have a *performance guarantee*

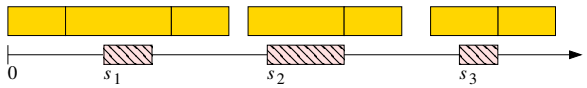


Figure 1: An example of a Gantt chart for 3-ONAS. Each ONA period is represented on the time axis by a dashed rectangle. In the schedule depicted, all the ONA periods are covered. Two idle times occur, before the second and the third ONA periods

of ρ (or to be a ρ-approximation) if, for any instance x, it delivers a schedule whose makespan is not larger than $\rho OPT(x)$.

We now give some basic dominance properties. First, it can easily be noticed that semi-active schedules are dominant for ONAS. Such a schedule is entirely defined as a permutation of N. Moreover the order of execution in the time interval between 2 successive ONA periods is meaningless. We denote by S_j the set of tasks that start after (or exactly at) time $s_{j-1} + L_{j-1}$ and complete before (or exactly at) time s_j in a given schedule π. We call *OA-packing* of a schedule the partition $S_1 \cup \ldots \cup S_{k+1}$ of the tasks not covering an ONA period.

For a schedule π, we call *last* ONA index the largest integer K such that $s_K + L_K \leq C_{\max}(\pi)$. We can make the following remark:

Remark 1 *Let K be the last* ONA *index of a semi-active schedule π. If $L_K \leq \max\{p_x \mid x \in S_{K+1}\}$ then period K is covered in π.*

As a consequence if $L \leq \min p_i$, the last period of a semi-active schedule is always covered. Finally the following property states that it is dominant for a task covering an ONA period to be locally the largest one:

Property 1 *There exists an optimal schedule such that, for $j = 1, \ldots, k$, if the jth* ONA *period is covered by a task y, then $p_y \geq \max\{p_x \mid x \in S_j \cup S_{j+1}\}$.*

Proof. We use a simple interchange argument. Consider an optimal schedule π, and let j be the first ONA period not verifying the property. It means that j is covered by a task y whose duration is smaller than the duration of the largest task, say z, of $S_j \cup S_{j+1}$. Consider for instance that $z \in S_j$. Without loss of generality we can assume that z is the task of S_j scheduled last in π. Let C_y and C_z be the completion time of y and z, respectively. We then interchange tasks y and z to create a new (semi-active) schedule π'. Notice that y finishes in π' before time $C_z \leq s_j$ while task z completes at time $\max\{C_z + p_y, s_j + L_j\}$, which is at most the completion time of y in π. Hence π' is feasible and also optimal. The case where $z \in S_{j+1}$ is similar. Repeating step by step this interchange to all other ONA periods we get an optimal schedule verifying the property. □

3 Single ONA period (1−ONAS)

In this section, we consider a single operator non-availability period of length L, located at $(s, s + L)$. Clearly if $L > \max_i\{p_i\}$, no task can overlap the ONA period in any feasible schedule. Hence we have a classical non-resumable machine unavailability period scheduling problem, known to be $\mathcal{NP}$-hard. One may wonder if the problem remains hard if some processing times may be smaller than the duration of the period. For completeness we give the complexity proof of theorem 1 from [2]. In this paper the authors also derive a FPTAS for 1-ONAS.

Theorem 1 *If L is greater than some processing time, then problem* 1−ONAS *is $\mathcal{NP}$-hard.*

Proof. The reduction uses the SUBSET SUM decision problem:
INSTANCE: $n + 1$ integers $a_1, \ldots, a_n$ and B.
QUESTION: Does there exist a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = B$?

We encode an instance of SUBSET SUM into an instance of the 1−ONAS associated decision problem as follows: we consider $n + 1$ tasks, with $p_i = a_i$ for $i = 1, \ldots, n$, and $p_{n+1} = \sum_i a_i + 1$. The ONA period starts at time B and its duration L is equal to p_{n+1}. We are thus in the situation where $\min_i\{p_i\} < L \leq \max_i\{p_i\}$. We ask whether there exists a schedule with makespan at most $C = \sum_{i=1}^{n+1} p_i$. Clearly if such a schedule exists, its makespan must be exactly C and the machine is continuously occupied till the end of the schedule. It implies that task $n + 1$ covers exactly the ONA period. Hence such a schedule exists if and only if the subset S of the tasks scheduled before the ONA period verifies $\sum_{i \in S} p_i = B$. □

In our industrial application, experiments are to be processed during several days (typically between 3 and

21) while the ONA periods are usually 2 days (weekends) or a single day (day off). We will show that in the case where the processing times are all greater than the length of the ONA period, minimizing the makespan can be done in time $\mathcal{O}(n \log n)$. This contrasts with the case of non-resumable machine unavailability periods which is $\mathcal{NP}$-hard even with only one period of arbitrarily small length. In the remaining of this section, we index the tasks in non-decreasing order of their processing times. We then have the following simple dominance property:

Property 2 *If L is smaller than any processing time, then there exists an optimal schedule where task n covers the* ONA *period.*

Proof. Consequence of Remark 1 and Property 1. □

We now introduce the following decision problem, which we call SUBSET SUM WITH TOLERANCE. It is a variant of SUBSET SUM where we are asked if there exists a subset summing up to a target B, with a tolerance from below of Δ:

Problem SUBSET SUM WITH TOLERANCE
INSTANCE: $m+2$ integers $a_1, \ldots, a_m$, B and Δ.
QUESTION: Does there exist $S \subseteq \{1, \ldots, m\}$ such that $\sum_{i \in S} a_i \in [B - \Delta, B]$?

Its optimization version, MAXSSWT, corresponds to finding a subset S whose sum of the elements, $a(S)$, is lower than a given bound B. The criterion to maximize is $\min\{a(S), B - \Delta\}$. It means that we are looking for the largest subset S not exceeding B, except that all subsets larger than $B - \Delta$ are optimal for our criterion. If tolerance Δ is set to 0, we have a classical SUBSET SUM problem, known to be $\mathcal{NP}$-hard. Clearly for large values of Δ the problem becomes polynomial. We have the following result:

Proposition 1 *If $\Delta \geq \max_{i,j} |a_i - a_j|$, problem* MAXSSWT *can be solved in time $\mathcal{O}(m \log m)$.*

Proof. Consider Algorithm 1. It is simply a 2−OPT procedure that repeatedly exchanges an element of S with a larger element not in S. The clever point is to start with an initial feasible set S of maximal cardinality. For short, for a subset X, we denote by $a(X) \equiv \sum_{i \in X} a_i$. We claim that Algorithm 1 solves optimally the problem.

Let $S_0 = \{1, \ldots, \chi\}, S_1, \ldots, S_q$ be the successive sets considered by the algorithm. By construction the $a(S_l)$ form an increasing sequence. In addition notice that between two steps, the sets S_l can not increase by more than Δ, since $a(S_{l+1}) = a(S_l) + a_j - a_i \leq a(S_l) + \Delta$ for any pair of indices i and j. Hence $a(S_l) \leq B - \Delta$ implies that $a(S_{l+1}) \leq B$.

Algorithm 1 Algorithm 2−OPT MAXSSWT

Inputs: $m+2$ integers a_i, B and Δ
Output: a subset S, if it exists, such that $a(S) \in [B - \Delta, B]$, otherwise a subset S with largest $a(S)$ smaller than $B - \Delta$.

```
Sort the a_i's in non-decreasing order.
Let χ be the largest index verifying a_1 + ... + a_χ ≤ B.
Initialize S := {1, ..., χ}
while a(S) < B − Δ and min{a_i | i ∈ S} < max{a_j | j ∉ S} do
    Chose arbitrarily i ∈ S and j ∉ S such that a_i < a_j
    S := S\{i} ∪ {j}
end while
return S
```

Now assume that the algorithm fails to find a subset in $[B - \Delta, B]$. Since $a(S_0) < B - \Delta$, it implies from what precedes that $a(S_q)$ is also smaller than $B - \Delta$. By construction, set S_q contains in this case the χ largest tasks when the algorithm terminates. But the choice of χ ensures that there is no subset with $\chi + 1$ tasks of size less than B. Hence S_q is precisely the largest subset smaller than B.

A possible implementation of the algorithm is to interchange at each step the smallest element of S with the largest element outside S, bounding by $\chi \leq m$ the number of steps. Pointers to these elements can be maintained in $\mathcal{O}(1)$ times since the a_i's are initially sorted. Thus the overall complexity of the algorithm is dominated by the sorting step, in $\mathcal{O}(m \log m)$. □

Theorem 2 *The scheduling problem with one* ONA *period can be solved in $\mathcal{O}(n \log n)$ time if all tasks are greater than (or equal to) the unavailability period.*

Proof. Property 2 ensures that an optimal schedule π^* exists with the largest task n covering the ONA period. We have 2 cases to distinguish:

1. No idle time occurs in the optimal schedule.

2. An idle time occurs before the ONA period.

Let (S_1^*, S_2^*) be the OA-packing of π^*. Recall that $\delta = p_n - L$ represents the difference between the duration of the largest task and the duration of the Ona period. The makespan of the optimal schedule is then given by $C_{\max}(\pi^*) = p(N) + \max\{0, s - \delta - p(S_1^*)\}$. Clearly in the second case, S_1^* must be the largest subset of tasks smaller than $s - \delta$. In the first case, the makespan is obviously equal to $p(N)$, and hence we have $p(S_1^*) \in [s - \delta, s]$. Notice that any subset S of $\{1, \ldots, n-1\}$ such that $p(S)$ belongs to the interval defines an optimal schedule by sequencing the tasks of S in any order, followed by n, and then the remaining tasks, once again in any order. Thus solving a 1−Onas instance corresponds to a MaxSSwT problem on entry $(p_1, p_2, \ldots, p_{n-1}, s, \delta)$. Since $\max_{i,j} |p_i - p_j| \leq p_n - L = \delta$, Algorithm 1 is solving the instance in time $\mathcal{O}(n \log n)$. □

4 Complexity and inapproximability results

The previous section motivates to introduce the following definition:

Definition 2 *A period is said to be small if it is smaller than (or equal to) any processing time. We refer to problem small* Onas *to mean that all periods are small.*

In this section we prove that small k−Onas is $\mathcal{NP}$-hard in the ordinary sense for any constant number $k \geq 2$ of periods, and becomes strongly $\mathcal{NP}$-hard when k is part of the instance. In addition we provide inapproximability results, establishing that small k−Onas does not belong to FPTAS, while small Onas does not belong to APX, unless $\mathcal{P} = \mathcal{NP}$. Notice that complexity and inapproximability results for small Onas are also obviously valid for the general Onas problem, *i.e.* without the small assumption.

4.1 Constant number of Ona periods

One natural question, considering Theorem 2, is to determine if the problem remains polynomial if we have 2, 3, ..., k periods, all small. The answer is negative:

Theorem 3 *Problem* 2-Onas *is NP-hard even if both periods are equal and small.*

Proof. Consider the Equal Partition problem:
Instance: m integers $a_1 \ldots a_m$.
Question: Does there exist a partition $A \cup B$ of $\{1, \ldots, m\}$ such that $|A| = |B|$ and $\sum_{i \in A} a_i = \sum_{i \in B} a_i$?

This problem is known to be $\mathcal{NP}$-complete [7]. To avoid trivial cases we assume that both m and $\sum_i a_i$ are even, and at least 2 tasks are not of zero duration. We associate to an instance $\mathcal{I}$ the following instance $f(\mathcal{I})$ (see Figure 2) of 2-Onas:

- the duration of both periods is set to $L = \sum_i a_i$
- we consider m tasks to schedule, with processing time $p_i = a_i + L$
- the starting time s_2 of the second Ona period is $\sum_i p_i/2$
- the starting time s_1 of the first Ona period is $s_2 - L$. Hence the second Ona period occurs immediately after the first one.

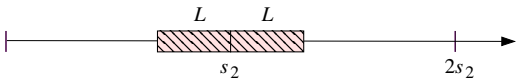


Figure 2: Ona periods in instance $f(\mathcal{I})$

This transformation f is clearly polynomial. Notice that the periods are small, since $2L > p_i \geq L$ by construction. We now establish that an instance $\mathcal{I}$ of Equal Partition is positive if and only if there exists a schedule for the instance $f(\mathcal{I})$ of makespan at most $\sum_i p_i = (m+1)L$.

Assume first that there exists a partition $A \cup B$ such that $\sum_{i \in A} a_i = \sum_{i \in B} a_i$ and $|A| = |B|$. We have $p(A) = \sum_{i \in A} a_i + |A|L = (\sum_i a_i + mL)/2 = s_2$. Since L is small, we can schedule set A of tasks in any order before the first Ona period, completing exactly at time s_2, and then schedule the tasks of B. The resulting schedule has a makespan of $\sum_i p_i$.

Conversely assume that such a schedule exists. Since all processing times are smaller than $2L$, both Ona periods must be covered by a different task. Since no idle time can occur, it implies that there exists a partition $A \cup B$ such that $p(A) = s_2$. We have $p(A) = \sum_{i \in A} a_i + |A|L$ and by construction $s_2 = \sum_i p_i/2 \equiv (m+1)L/2$. As m is even, it implies that

$\sum_{i\in A} a_i = L/2 \mod L$. But $\sum_{i\in A} a_i \leq L$ by definition of L, thus we get $\sum_{i\in A} a_i = L/2$ and $|A| = m/2$. Which proves that $\mathcal{I}$ is positive. □

Recall that a problem belongs to FPTAS if and only if for any $\varepsilon > 0$ there exists an approximation algorithm A_ε with guarantee $(1+\varepsilon)$ running in time polynomial in the instance size *and* $1/\varepsilon$. It implies in particular that problems of FPTAS can be approximated in polynomial time in $|x|$ with guarantee $1 + 1/P(|x|)$ for any instance x and any given polynomial P. As a consequence of Theorem 3, we have the following corollary on the inapproximability of $k-$ONAS:

Corollary 1 *Problem small $k-$*ONAS *does not belong to FPTAS for $k \geq 3$, unless $\mathcal{P} = \mathcal{NP}$.*

Proof. We prove in fact a slightly stronger result. It states that if problem $k-$ONAS is $\mathcal{NP}$-hard for some constant k, then problem $(k+1)$-ONAS does not belong to FPTAS, under the usual hypothesis that $\mathcal{P} \neq \mathcal{NP}$. Consider an $\mathcal{NP}$-complete language $\mathcal{L}$ over an alphabet Σ. Assume that there exists a real function C together with a polynomial transformation g that maps words of Σ^* onto instances of $k-$ONAS, for some constant k, such that a word x belongs to $\mathcal{L}$ if and only if $OPT(g(x)) \leq C(|x|)L$. We claim that if $C(|x|)$ is polynomially bounded in $|x|$, then $(k+1)$-ONAS does not belong to FPTAS. Indeed consider the transformation g' that simply adds to instance $g(x)$ a $(k+1)$-th unavailability period of duration L, occurring at time $s_{k+1} = C(|x|)L$. We have immediately that:

$$\begin{array}{rcl} x \in \mathcal{L} & \Rightarrow & OPT(g'(x)) \leq s_{k+1} \\ x \notin \mathcal{L} & \Rightarrow & OPT(g'(x)) \geq s_{k+1} + L \end{array}$$

This gap reduction implies that no better approximation ratio than $1 + 1/C(|x|)$ can be guaranteed by a polynomial time algorithm unless $\mathcal{P} = \mathcal{NP}$. Since $C(|x|) \leq P(|x|)$ for some polynomial P, it proves that $(k+1)$-ONAS can not belong to FPTAS.

To establish Corollary 1, simply consider the transformation f used to prove Theorem 3. We have established that $\mathcal{I}$ is a positive instance of EQUAL PARTITION if and only if $f(\mathcal{I})$ admits a schedule of makespan $\sum_i p_i$. Since $\sum_i p_i = (n+1)L$, 3-ONAS can not belong to FPTAS, unless $\mathcal{P} = \mathcal{NP}$. □

We show later that simple algorithms can be very efficient in practice, especially for large instances. First we establish, in the next section, complexity results if the number of periods is part of the instance.

4.2 Problem small ONAS

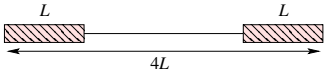
Recall that problem ONAS refers to the scheduling problem where the number of periods is not a constant but is part of the instance. This problem is of course at least as difficult as k-ONAS and thus $\mathcal{NP}$-hard. The next theorem shows that ONAS is hard in the strong sense.

Proposition 2 *Problem* ONAS *is $\mathcal{NP}$-hard in the strong sense, even if all periods are equal and small.*

Proof. Consider the classical 3-PARTITION Problem. INSTANCE: $3m+1$ integers $a_1, \ldots, a_{3m}$ and B verifying $\sum_i a_i = mB$ and $B/4 < a_i < B/2 \;\; \forall i = 1, \ldots, 3m$. QUESTION: Does there exist a partition $A_1 \cup \ldots \cup A_m$ of $\{1, \ldots, 3m\}$ such that $\sum_{i\in A_j} a_i = B \;\; \forall j = 1, \ldots, m$?

Without loss of generality we can assume that $B = 0 \bmod 4$ (otherwise multiply every integers by 4). We encode an instance $\mathcal{I}$ of 3-PARTITION into an instance $f(\mathcal{I})$ of the ONAS associated decision problem as follows:

- We have $k = 2m$ periods of duration $L = B/4$. They reproduce the following pattern every $4L$ time units:



- We have $n = 3m$ tasks, of duration $p_i = a_i$. Notice that $L < p_i < 2L$ for all tasks.
- QUESTION: Does there exist a schedule of makespan at most $C = \sum_i p_i$?

The transformation is a reduction. Consider $\mathcal{I}$ in language 3-PARTITION. To prove that $f(\mathcal{I})$ is a positive instance, it is sufficient to show that the tasks associated with one partition set A_j can be scheduled exactly in a pattern. Indeed since the periods are small, any sequencing of the 3 tasks is valid.

Conversely assume that $f(\mathcal{I})$ is positive. A schedule of makespan C has clearly no idle time. In addition, each task is scheduled inside a pattern (starts and ends in the same pattern) since at the frontier of the patterns, we have two adjacent periods of total duration $2L$. It implies that the tasks scheduled inside each pattern have a sum of durations exactly $4L = B$. The schedule defines a valid partition for $\mathcal{I}$.

The reduction is polynomial. In fact it is pseudo-polynomial, as $|\mathcal{I}| \leq |f(\mathcal{I})|$ and $\textsc{Max}(f(\mathcal{I})) = C = mB \leq |\mathcal{I}|\textsc{Max}(\mathcal{I})$. Since 3-Partition is NP-hard in the strong sense, it implies that Onas is also NP-hard in the strong sense. □

The previous reduction can be used in a straightforward way to establish non-approximability results for Onas, as in Corollary 1.

Proposition 3 *If $P \neq \mathcal{NP}$, problem* Onas *is not in* Apx *even if all periods are equal and small. Moreover approximating* Onas *within a factor $k^{1-\varepsilon}$ is NP-hard for any constant $\varepsilon > 0$.*

Proof. Let $\varepsilon > 0$ be a constant and set $\alpha = \lceil 1/\varepsilon - 1 \rceil$. The gap reduction is quite immediate using the idea of Corollary 1. Consider again the 3-Partition problem and modify reduction f by appending $(4m)^{\alpha+1} - 4m$ Ona periods of length L after the last pattern (this is the gadget, see Figure 3). The resulting instance $g(\mathcal{I})$ of Onas has a total of $k = (4m)^{\alpha+1} - 2m$ periods.

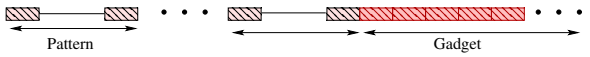


Figure 3: Ona periods in reduction g of Proposition 3

The transformation g is polynomial in $|\mathcal{I}|$, greater than $\max\{m, \log L\}$. Since f was a reduction, we clearly have:

$$\begin{array}{rcl} \mathcal{I} \in \text{3-Partition} & \Rightarrow & OPT(g(\mathcal{I})) = 4mL \\ \mathcal{I} \notin \text{3-Partition} & \Rightarrow & OPT(g(\mathcal{I})) \geq (4m)^{\alpha+1}L \end{array}$$

Indeed if $\mathcal{I}$ is a negative instance, a valid schedule for $g(\mathcal{I})$ must finish after time $4mL$. Hence at this time at least one task is not completed. Since only unavailability periods occur then, and tasks have duration in $(L, 2L)$, this task can not be scheduled before the last period. This gap reduction implies that approximating Onas within $(4m)^{\alpha} > k^{\alpha/(\alpha+1)} \geq k^{1-\varepsilon}$ is NP-hard. □

We summarize the complexity results of this section in Table 1. As one can notice, the remaining open question is to determine if small 2-Onas belongs to FPTAS or not.

	Complexity	Inapproximability
1-Onas	polynomial ($n \log n$)	
2-Onas	NP-hard	?
$k-$Onas, $k \geq 3$	NP-hard	not in FPTAS
Onas	strongly NP-hard	not in APX

Table 1: Complexity results for small Onas

5 Approximation algorithms for small Onas

The previous section has established that problems small $k-$Onas and small Onas are both $\mathcal{NP}$-hard. It is then natural to explore heuristic approaches. In scheduling theory, the most famous ones are *list scheduling* algorithms, introduced by Graham [8]. We establish in this section the performances we may hope from list scheduling algorithms and how they can be improved.

5.1 List scheduling algorithms

A list scheduling algorithm is based on the principle to forbid the resource from being idle if a task is ready for processing. Hence a list scheduling algorithm uses basically a greedy allocation of tasks to resources to prevent (locally) idleness. A list, defining a total ordering of the tasks, is used to break the ties between tasks concurrently available. The property implied by the greedy allocation is that, if the resource is idle at a time t in the schedule, then no task is available at this date.

This section is devoted to the analysis of the performances of list scheduling algorithms. Since an efficient polynomial algorithm exists for 1-Onas, we consider that $k \geq 2$ throughout this section. We establish that any list scheduling algorithm is essentially a $(k+1)/2$ approximation. We start by giving some upper bounds on the idle time in a schedule built by a greedy allocation.

Bounding the idle time

We consider a schedule π of makespan $C_{\max}(\pi)$ obtained by a list scheduling algorithm. We denote by $\tilde{I}_j$ the idle time occurring between the $(j-1)$-th and the j-th Ona period, and by I_j the total time of in-

activity in interval $[s_{j-1} + L_{j-1}, s_j + L_j]$. Notice that $I_j = \tilde{I}_j$ if the j-th ONA period is covered, $I_j = \tilde{I}_j + L_j$ otherwise. We will focus on the last task scheduled by the algorithm, say l. We denote by K the last period of the schedule, *i.e.* the largest integer such that $s_K + L_K \leq C_{\max}(\pi)$. Due to Remark 1, period K is necessarily covered in π since it is a semi-active schedule. Our first remark bounds the idle time occurring between two ONA periods. This upper bound is valid even if the periods are not small. Recall that L denotes the largest duration of an ONA period.

Remark 2 *In schedule π, for all $j \leq K$, we have $\tilde{I}_j \leq L$.*

Proof. Consider a non-zero idle interval $\tilde{I}_j$, starting at some instant t. At this time the algorithm fails to schedule any remaining task, which implies in particular that we must have $t + p_l \in (s_m, s_m + L_m)$ for some index $m \geq j$. This clearly holds for any instant t' in the idle interval. Hence $\tilde{I}_j \leq L_m \leq L$. □

Remark 2 implies that $I_j \leq 2L$ for all j. If the j-th ONA period is not covered we have another upper bound for small ONAS, considering two consecutive periods.

Remark 3 *If the j-th ONA period is not covered in π, then $I_j + \tilde{I}_{j+1} \leq p_l$.*

Proof. As previously, consider the first instant t of the idle interval I_j. Interval $\tilde{I}_j$ is then $[t, s_j]$. Notice that $\tilde{I}_j$ may be reduced eventually to a singleton $\{s_j\}$ but is not empty. Due to the greedy allocation, we necessarily have $t + p_l \in (s_m, s_m + L_m)$ for some index $m \geq j$. It suffices to show that $m \geq j+1$. For the sake of contradiction assume that $m = j$. Then consider instant $e = s_j + L_j - p_l$. The fact that L_j is small implies that $e \leq s_j$. But since we assume that $t + p_l < s_j + L_j$, we have $e \geq t$. It implies that p_l can be scheduled at time $e \in \tilde{I}_j$, which contradicts the fact that the list scheduling algorithm fails to start any task in $\tilde{I}_j$. □

Using the two previous remarks and the fact that the periods are small, we can notice that the idle time I_j of any period j is smaller than p_l, whenever the period is covered or not. As another consequence of Remark 3, for any two consecutive periods, we have $I_j + I_{j+1} \leq p_l + L$. Indeed if period j is uncovered, this is a direct consequence of the remark, since $L_{j+1} \leq L$. Otherwise $I_j \leq L$ due to Remark 2, while, as we have noticed, $I_{j+1} \leq p_l$. For the last two periods, the next remark gives a stronger upper bound:

Remark 4 *For the last two periods of the schedule we have $I_{K-1} + I_K \leq L + L_K$.*

Proof. Let $I = I_{K-1} + I_K$ be the idle time on the two intervals. Since period K is covered, using the argument of Remark 2, we have $I_K \leq L_K$. If period $K-1$ is also covered, we get directly the required upper bound. Otherwise consider the first inactivity instant t of I_{K-1}. As $K-1$ is not covered and small, we must have $t + p_l \in (s_K, s_K + L_K)$. Let e be the instant $s_K + L_K - p_l$. By definition we have $e - t \leq L_K$, and since $K-1$ is not covered, $e > s_{K-1}$. Now consider the instant $t' = \max\{e, s_{K-1} + L_{K-1}\}$. Clearly task p_l can be scheduled at any time after t'. Thus the greedy allocation implies that no idle time occurs after t', and hence $I \leq t' - t$. Now simply write that $t' - t = \max\{-L_{K-1}, s_{K-1} - e\} + L_{K-1} + e - t$. The first term is negative, while the last difference is smaller than L_K, which provides the upper bound. □

As $I_j \leq 2L$ for all $j = 1, \ldots, K-2$, the previous remark gives a first upper bound of the total idle time of the schedule:

Corollary 2 *The total idle time of a list schedule is bounded by $2(K-1)L$ for $K \geq 2$.*

Combining Remarks 2 and 3, we can derive another upper bound of the total idle time of a list schedule. The next lemma, quite simple, is the keystone of the analysis.

Lemma 1 *For any index $m \leq K$,*

$$I_m + \ldots + I_K \leq (K - m + 1)\frac{p_l + L}{2} - \frac{p_l - L}{2}$$

Proof. Let $q = K - m + 1$ be the number of periods considered in Lemma 1 and $I = I_m + \ldots + I_K$ be the total idle time. The idea is to group periods 2 by 2 to use the upper bound of Remark 3, to isolate the last or last two periods of the schedule, and apply to them the upper bound of Remark 2 or 4. Recall that for any two consecutive periods j and $j+1$ we have $I_j + I_{j+1} \leq p_l + L$. If q is even, it results that $I \leq (q-2)(p_l + L)/2 + I_{K-1} + I_K \leq q(p_l + L)/2 - (p_l - L)$. Hence we have a stronger upper bound than the one of the lemma. If q is odd, we group the periods 2 by 2 till

period $K-1$. We then get $I \leq (q-1)(p_l+L)/2 + I_K$. As period K is covered, Remark 2 implies than $I_K \leq L$. Thus $I \leq q(p_l+L)/2-(p_l-L)/2$, which is the required upper bound. □

Performance guarantee

Now consider an optimal schedule, with makespan OPT. If the instance contains a unique task, any list schedule is clearly optimal. Hence we assume that $n \geq 2$, and thus $OPT \geq \sum_i p_i \geq 2L$. We can also assume w.l.o.g. that the first task of the list schedule starts at time 0, since no task can start earlier in any feasible schedule due to the greedy allocation. We state a first result which corresponds in fact to a particular case of the proof of Proposition 5. We emphasize it in Proposition 4 since it shows that any list scheduling algorithm has a guarantee $2-1/k$ when OPT is sufficiently large. In particular the condition is fulfilled if $\sum_i p_i > s_k$.

Proposition 4 *For problem small* ONAS *with* $k \geq 2$*, if* π *is a list schedule such that its last* ONA *period* K *verifies* $OPT \geq s_K + L_K$*, then* $C_{\max}(\pi) \leq (2 - 1/k)OPT$.

Proof. Let I be the idle time occurring in the list schedule π, and let Q be the amount of work processed in π before time OPT. From Corollary 2, the idle time I satisfies $I \leq 2(K-1)L$. Since K is the last ONA period of the schedule, the idle time can only occur before time OPT. Thus we also have $I = OPT - Q$.

If $2L \leq Q$, combining those inequalities, we get directly $I \leq (1-1/k)OPT$. Using $C_{\max}(\pi) = \sum_i p_i + I$ implies the required result.

Thus assume that $Q < 2L$. Let x be the task scheduled at time 0, and y the one covering period K. Notice that $x = y$ implies $I = 0$ and $C_{\max}(\pi) = OPT$. Thus, we assume that $x \neq y$. Let a be the amount of work of y finished by time OPT. $Q < 2L$ implies that no other task can be processed in time interval $[0, OPT]$, and hence we have $Q = p_x + a \geq L + a$. It also implies that task y can not be completed by time OPT.

Let Z be the set of tasks finishing after time OPT. From what precedes we have simply $Z = N \backslash \{x\}$. On one hand, since no idle time occurs after time OPT, we can write that $C_{\max}(\pi) = OPT - a + p(Z)$. On the other hand, work conservation implies $OPT \geq p(Z) + p_x$. Thus

$$\frac{C_{\max}(\pi)}{OPT} \leq 1 + \frac{p(Z) - a}{p(Z) + p_x}$$

The right term is an increasing function of $p(Z)$. To upper bound this quantity, first observe that, due to the greedy allocation, no task of Z can start in time interval $[p_x, OPT - a)$. Since $a < L$, in fact all tasks of Z must start in time interval $[0, p_x)$ in any optimal schedule. Hence we have $p(Z) \leq p_x + \max\{p(z) | z \in Z\}$.

Because y finishes strictly after time $s_K + L_K$, certainly y waits to start its execution for either the end of an ONA period or the completion of x. This latter case would result in an optimal schedule, thus we can assume that y starts exactly at the end of an uncovered ONA period. Once again the greedy allocation asserts that at the beginning of this ONA period no task of Z can be scheduled. It directly implies that $p_z < a + L$ for all $z \in Z$. Thus $p(Z) < p_x + a + L$. Putting all together, we have :

$$\frac{C_{\max}(\pi)}{OPT} < 1 + \frac{p_x + L}{2p_x + a + L}$$

Similarly, this term is a decreasing function of a and p_x. Hence the ratio is maximum for $a = 0$ and $p_x = L$ which leads to $C_{\max}(\pi) \leq (1 + \frac{2}{3})OPT$.

This inequality proves Proposition 4 for $k \geq 3$. It remains the case $k = 2$ to consider. Notice that our previous analysis remains valid, in particular we have the inequality $p_y < a + L < 2L$. If $N = \{x, y\}$, we have, by definition of a, $C_{\max}(\pi) = OPT + p_y - a \leq OPT + L$. As $OPT \geq p_x + p_y \geq 2L$, we obtain $C_{\max}(\pi) \leq 1.5OPT$. Otherwise $p(N) \geq p_y + 2L$. As interval $[p_x, OPT - a]$ contains at most one ONA period, the greedy allocation ensures that $I \leq p_y$ (see Remark 3). We get

$$\frac{C_{\max}(\pi)}{OPT} \leq 1 + \frac{I}{OPT} \leq 1 + \frac{p_y}{p_y + 2L} \leq \frac{3}{2}$$

Indeed, the fraction is an increasing function of p_y, smaller than $2L$. □

We finally give the general guarantee of any list scheduling for ONAS:

Proposition 5 *For problem small* ONAS *with* $k \geq 2$*, any list scheduling algorithm has a performance guarantee of* $(k+1)/2$.

Proof. Consider a schedule π obtained by a list scheduling algorithm. If $OPT \geq s_K + L_K$, Proposition 4 establishes the ratio, since $2 - 1/k \leq (k+1)/2$. Otherwise, let m be the smallest index such that $OPT \leq s_m$. We have $m \leq K \leq k$. We then write down the equation of work conservation at time OPT for

schedule π. If Q denotes the amount of work achieved at this time in π, we have:

$$C_{\max}(\pi) \leq OPT + \sum_{j=m}^{K} I_j + \sum_{i=1}^{n} p_i - Q$$

Using Lemma 1 for periods $m, m+1, \ldots, K$, we get the bound:

$$C_{\max}(\pi) \leq 2OPT + \frac{K-m+1}{2}(p_l + L) - \frac{1}{2}(p_l - L) - Q$$

We will consider 2 cases, depending on the optimal value. First assume that $m \geq 3$, *i.e.* the optimal schedule finishes after the second ONA period. Notice that at time OPT, schedule π has completed at least its first task. It implies in particular that $Q \geq L$. Since we assume that $n \geq 2$, we also have the inequality $OPT \geq p_l + L$. Finally we get from the work conservation:

$$\begin{array}{rcl} C_{\max}(\pi) & \leq & 2OPT + \frac{K-2}{2}(p_l + L) - \frac{1}{2}(p_l - L) - L \\ & \leq & 2OPT + \frac{K-3}{2}(p_l + L) \\ & \leq & \frac{K+1}{2} OPT \end{array}$$

Now assume that the optimum schedule completes before the second ONA period. An optimal schedule can then be computed in polynomial time, see Theorem 2, but for completeness we analyze what happens to list scheduling algorithms. In this situation, we get a looser upper bound on the idle time of π, as one more term $(p_l + L)/2$ is added. Thus we need to improve our lower bounding of Q. More precisely we claim that at least $L + p_l$ quantity of work is completed in π by time OPT. Let t_I be the first instant of inactivity in π. Now observe the tasks that remain to schedule in π at this point in time. Since the machine is continuously occupied on $[0, t_I[$, for any such task h, we certainly have $t_I + p_h \leq \sum_i p_i \leq OPT \leq s_2$. And since task h is not scheduled at time t_I, it implies in fact that $t_I + p_h \in (s_1, s_1 + L_1)$, *i.e.* an idle time occurs before the first ONA period (unless π is optimal). Hence any remaining task h has an earliest date $s_1 + L_1 - p_h$ when it can be scheduled, overlapping the first ONA period. Due to the greedy allocation, the algorithm schedules the largest remaining task, say b, to overlap the first ONA period, completing exactly at time $s_1 + L_1$. Thus at time OPT at least the first task (say a, starting at time 0) and b have been completed in π. We get $Q \geq p_a + p_b \geq L + p_l$. The equation of work conservation at time OPT gives:

$$\begin{array}{rcl} C_{\max}(\pi) & \leq & 2OPT + \frac{K-1}{2}(p_l + L) - (p_l + L) \\ & \leq & \frac{K+1}{2} OPT \end{array}$$

which concludes the proof. □

The guarantee of list scheduling algorithms given in Proposition 5 is not so bad, knowing the inapproximability result on ONAS. In particular we have a guarantee of 3/2 for $k = 2$ and 2 for $k = 3$, which can be achieve in linear time $\mathcal{O}(n)$ using an arbitrary priority list. But the guarantee $(k+1)/2$ becomes quickly not satisfactory for larger k. However, surprisingly, any list scheduling has roughly speaking the same performance guarantee:

Proposition 6 *Even if all periods are equal and small, no list scheduling algorithm can have a better performance guarantee for $k-$*ONAS *than $k/2+1/6$ for k odd, and $k/2+1/3$ for k even.*

Proof. Consider the following instance of $k-$ONAS for k even:

- ONA periods are grouped by 2. Each one has duration L
- time between 2 ONA blocks is $L - 2\epsilon$, except for the first block that starts at time L. Hence the last ONA period ends at time $3Lk/2 + \mathcal{O}(\varepsilon)$
- we have 2 tasks to schedule, $p_1 = L$ and $p_2 = 2L - \epsilon$, that is $2L - 2\epsilon < p_2 < 2L$.

If task 1 is scheduled first, at time 0, it is not possible to schedule task 2 before the last ONA period, see Figure 4. Hence the makespan is equal to $3Lk/2+L+\mathcal{O}(\varepsilon)$. But an optimal schedule can first schedule task 2, starting at time ε, and then task 1 which overlaps exactly the second ONA period. The makespan is $3L$.

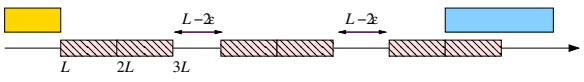


Figure 4: The schedule produced by any list scheduling algorithm

The ratio between the two makespans tends to $k/2 + 1/3$ for ε small. Notice that any list scheduling algorithm starts by scheduling task 1 to avoid the ε idle time at the beginning. This proves the result for k even. If k is odd, we use the same construction, except that the last block is replaced by a single ONA period. The makespan of any list scheduling algorithm

is then $3L(k-1)/2+2L+\mathcal{O}(\varepsilon) = 3Lk/2+L/2+\mathcal{O}(\varepsilon)$. □

It shows that for ONAS *any* list scheduling algorithm has a guarantee that lies between $k/2+1/6$ and $k/2+1/2$. Hence it is of little use to search for a clever list on this problem. We show in the next section how it is possible to take advantage of list scheduling algorithms to obtain better performances.

5.2 Better approximations

The previous proof of Proposition 6 incites to think that list scheduling algorithms perform the worst when only a few tasks are to be scheduled. Proposition 4 already states that ONAS can be approximated within the ratio $2-1/k$ when $\sum_i p_i$ is larger that s_k. Here we go a step further. Notice that, using Corollary 2, we can write for any list schedule π:

$$C_{\max}(\pi) = \sum_i p_i + \sum_j I_j \leq OPT + 2(k-1)L$$

Thus we have the following property:

Proposition 7 *List scheduling algorithms are asymptotically optimal for small $k-$ONAS when $n \to +\infty$.*

Proof. As $\sum_i p_i \geq nL$, we have $C_{\max}(\pi) \leq (1 + 2(k-1)/n)OPT$. □

Hence large instances are in fact the easy ones, for which any list scheduling algorithm will perform optimally. It may appear as a paradox, but for example BINPACKING is well known to act the same (when its optimal value becomes large). If Proposition 7 is of practical importance, it also permits to derive the following theoretical result:

Proposition 8 *Problem small $k-$ONAS belongs to PTAS.*

Proof. Let $\varepsilon > 0$ be a constant. Consider the algorithm A_ε described in Algorithm 2.

First notice that algorithm A_ε is polynomial for any fixed constant ε. Indeed, since semi-active schedules are dominant, it is sufficient to enumerate all task sequences to solve the problem optimally. For $n < N_\varepsilon$ we have only a constant number of tasks, and thus a constant number of sequences to evaluate.

Secondly, we claim that algorithm A_ε has a guarantee $1+\varepsilon$. Clearly if $n < N_\varepsilon$, the algorithm is optimal. Otherwise its makespan is bounded by $(1+2(k-1)/n)OPT \leq (1+\varepsilon)OPT$. □

Algorithm 2 PTAS for $k-$ONAS

```
set N_ε = ⌈2(k − 1)/ε⌉
if n < N_ε then
  solve the instance optimally
else
  apply any list scheduling algorithm
end if
```

As small $k-$ONAS does not belongs to FPTAS unless $\mathcal{P} = \mathcal{NP}$, for $k \geq 3$, we have determined exactly its approximability class. To approximate small ONAS problem, we can use the same technique, leading to the following result:

Proposition 9 *Problem small ONAS can be approximated with a guarantee $\mathcal{O}(k \ln\ln k / \ln k)$.*

Proof. Simply set $N = 2\ln k / \ln\ln k$, and as before either solve the problem optimally if $n \leq N$, or apply any list scheduling algorithm. The guarantee of a list scheduling algorithm is then bounded by $(1+2k/n)OPT$, smaller than $(1+k\ln\ln k/\ln k)OPT$. Hence we only have to prove that instances of at most N tasks can be solved in polynomial time. We can use Stirling's formula, or simply write $\ln(N!) \leq \ln 1 + \ln 2 + \ldots + \ln N < N \ln N$. But $\ln N \leq 1 + \ln\ln k - \ln\ln\ln k$. Hence $N \ln N \leq N + 2\ln k$. For k sufficiently large ($k \geq e^{e^2}$), N is smaller than $\ln k$. Thus the number of possible permutations of the tasks is at most $\exp(3\ln k) = k^3$, which is polynomially bounded in the instance size. Since each sequence can be evaluated in time $\mathcal{O}(n+k)$, an exhaustive search can be done in polynomial time. □

Since problem small ONAS can not be approximated within $k^{1-\varepsilon}$ for any constant $\varepsilon > 0$, Proposition 9 gives slightly the best approximation guarantee we may hope to obtain in polynomial time. Table 2 synthesizes our approximation results.

	Approximation	Inapproximability
$k-$ONAS, $k \geq 3$	PTAS	not in FPTAS
ONAS	$\mathcal{O}(k\ln\ln k/\ln k)$	$k^{1-\varepsilon}$

Table 2: Approximability results for small ONAS

While PTAS 2 has a good theoretical complexity of $\mathcal{O}(\max(2k/\varepsilon)!, n)$, *i.e.* linear in n, clearly it is very inefficient in practice. To improve the resolution of the problem with a constant number of tasks $n \leq N_\varepsilon$, notice that solving the problem optimally can be done in time k^n by deciding the partition of the tasks that start in the OA-periods. Since the order of the tasks inside an OA-period does not have any impact, one just has to end in each period with the largest task. This can be further improved grouping the periods two by two and applying each time Algorithm 1, just being careful of the tasks to be put on the ONA periods.

6 Periodic ONAS

We consider in this section periodic unavailability periods, *i.e.* we assume that unavailability periods occur every s time units. In addition we restrict to instances with equal and small periods. In this setting we have $s_1 = s$ and $s_j = s_{j-1} + L + s$ for $j = 2, 3, \ldots, k$, see Figure 5. The periodic case is clearly of practical interest, since week-ends typically induce 2 days of unavailability every 5 days in many industries. In addition one may hope better approximation algorithms due to the regular structure of the problem. In the following a subset S of tasks is said to *fit* before a period j if $p(S) \leq s$. If ONA period j is covered by task i then S *matches* the period if $s - \delta_i \leq p(S) \leq s$.

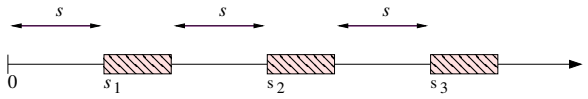


Figure 5: An instance of periodic 3-ONAS

For the periodic problem with machine unavailability periods and non-resumable jobs, Ji, He and Cheng [11] show that minimizing the makespan can not be approximated within a factor 2. In addition they establish that LPT list schedule has precisely a performance guarantee of 2. As noticed in their analysis, the problem is related to bin packing (even if the objective functions differ) where each interval of machine availability can be seen as a bin of capacity s and each task as an item of size p_i. When considering ONA periods, this analogy to packing does not hold any more since the subsets of tasks scheduled during an availability period can be larger than the duration of the time interval, if one of the tasks covers the following ONA period. This can be related to the so called *open-end bin packing* introduced by Leung, Dror and Young [16]: in this problem the last item of a bin is allowed to go beyond the bin capacity. They prove open-end bin packing to be strongly $\mathcal{NP}$-hard and give an (asymptotic) FPTAS. However periodic ONAS is intuitively an harder problem due to the interdependence between successive available intervals. It this sense ONAS is a "real" scheduling problem with temporal dependences, and not a variation on packing.

Another closely related problem is the MULTIPLE SUBSET-SUM problems (MSS), for which PTAS has been recently developed [4, 6, 10]. For classical machine unavailability periods, a $(2+\epsilon)$-approximation can be easily derived in a straightforward way from previous PTAS's for MSS even if the instance is not periodic but verifies that each availability periods is sufficiently large. On the opposite, using a PTAS for MULTIPLE SUBSET-SUM or MULTIPLE KNAPSACK problems to solve periodic ONAS is nothing but obvious. Due to the fact that some tasks will cover some periods, the size of the knapsacks are not known in advance. Notice that Caprara, Kellerer and Pferschy [4] proved that MULTIPLE SUBSET-SUM does not admit a FPTAS even for only 2 knapsacks. In contrast we will show that, under the small assumption, problem periodic 2−ONAS admits a FPTAS.

Notice that the proof of $\mathcal{NP}$-completeness for 2-ONAS considers 2 adjacent unavailability periods in the reduction. Hence one may hope the periodic version of the problem to be polynomial. It happens that even under these strong restrictions, the problem remains $\mathcal{NP}$-hard even for 2 periods:

Theorem 4 *Problem periodic $k-$ONAS with all periods equal and small is $\mathcal{NP}$-hard for $k \geq 2$.*

Proof. Again we consider the EQUAL PARTITION problem:
INSTANCE: n integers $a_1, \ldots, a_n$.
QUESTION: Does there exists a partition $A \cup B$ of $\{1, \ldots, n\}$ such that $|A| = |B|$ and $\sum_{i \in A} a_i = \sum_{i \in B} a_i$?

To avoid trivial instances, we assume that both n and $\sum_i a_i$ are even, and no task is greater than $\sum_i a_i/2$. For short we denote by $m = n/2$ and $\alpha = \sum_i a_i/2$. The idea of the reduction is to construct an instance of 2-ONAS such that any subset of $m-1$ tasks is significantly smaller than s, and thus fits before an ONA period. On the contrary, a subset of m tasks smaller than s, has

to contain only "small" tasks, associated with the a_i's of EQUAL PARTITION. The transformation from an instance $\mathcal{I}$ of EQUAL PARTITION into an instance $f(\mathcal{I})$ of 2-ONAS is the following:

- we consider 2 periods, with $L = m\alpha$ and $s = mL + \alpha$,
- we have n tasks with processing time $p_i = a_i + L$, $i = 1, \ldots, n$,
- In addition we have $m + 1$ tasks with processing time $p_{n+i} = L + \alpha$, $i = 1, \ldots, m + 1$.

By construction, the instance $f(\mathcal{I})$ is small and periodic, and f is clearly polynomial. We will prove that $\mathcal{I}$ is positive if and only if it exists a schedule for $f(\mathcal{I})$ of makespan $\sum_i p_i$, *i.e.* a schedule without idle time. Before, we make some preliminary remarks on the structure of the instance $f(\mathcal{I})$. Let us denote by N the set $\{1, \ldots, n\}$ of tasks associated with the a_i's and by $M = \{n + 1, \ldots, n + m + 1\}$ the set of "large" tasks of processing time $L + \alpha$. Recall that δ_i represents the difference $p_i - L$. Notice that we have $0 < \delta_i \leq \alpha$, with $\delta_i = \alpha$ if and only if $i \in M$. Consider now a subset S of tasks. Computing its duration we get:

$$|S|L < p(S) = |S|L + \sum_{i \in S} \delta_i \leq |S|(L + \alpha)$$

It implies the following properties:

(1) if $|S| > m$ then $p(S) > s$

(2) if $|S| < m$ then $p(S) \leq s - 2\alpha$

(3) if $|S| = m$ then $p(S) \leq s$ implies $S \subseteq N$

The first 2 properties are direct consequences of the previous inequalities. Indeed simply notice that $(m + 1)L > s$ and $(m - 1)(L + \alpha) = s - 2\alpha$, since $L = m\alpha$. Consider now a subset S of m tasks. Then $p(S) = s - \alpha + \delta(S)$. As we assume that $m > 1$, if S contains a task of M, then $\delta(S) > \alpha$, implying $p(S) > s$. This establishes the third property. Notice that for any task $i \in N$, we simply have $\delta_i = a_i$. Thus the time duration of a subset $S \subseteq N$ of m tasks is equal to $p(S) = s - \alpha + a(S)$.

We now prove that $\mathcal{I}$ is positive if and only if $f(\mathcal{I})$ admits a schedule of makespan $\sum_i p_i$. First assume that $\mathcal{I}$ is a positive instance, and consider a valid partition $A \cup B$ of N. We construct for $f(\mathcal{I})$ the following sequence π_A: schedule first the tasks of A in any order, then all the tasks of M, then the tasks of B in any order. We claim that this schedule is valid for $f(\mathcal{I})$ and without idleness. Indeed the time duration of A is equal to $p(A) = s - \alpha + a(A) = s$. Thus set A can be entirely scheduled before the first ONA period, completing exactly at time s. If we compute the duration of M, we get $p(M) = (m+1)(L+\alpha) = (mL+\alpha)+L+m\alpha = s+2L$, which corresponds exactly to the duration between the beginning of the first ONA period and the end of the second ONA period. Since the instance is small, tasks of M can be scheduled without idleness to complete at time $2s + 2L$, the first and last tasks overlapping the ONA periods.

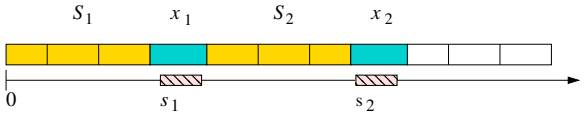


Figure 6: Structure of the schedule π for $f(\mathcal{I})$

Conversely consider that $f(\mathcal{I})$ admits a schedule π of makespan $\sum_i p_i$. Let (S_1, S_2, S_3) be the OA-packing of π. Since π is without idleness, two additional tasks, say x_1 and x_2, cover the two ONA periods, see Figure 6. It also implies that S_1 must fit its interval, *i.e.* that $s - \alpha \leq p(S_1) \leq s$. Our properties show that S_1 is a subset of N of cardinality m. Hence it only remains to prove that $a(S_1) = \alpha$. For this, consider the subset S_2 of tasks starting at time $s_1 + L$ or later and completing in π before time s_2.

By definition we have $p(S_2) \leq s$, which implies that $|S_2| \leq m$ due to property (1). Now suppose that $|S_2| = m$. It implies from property (3) that $S_2 \subseteq N$. Thus $S_1 \cup S_2$ if in fact a partition of N, and x_1 may only belong to set M. If we consider all the tasks completing before time s_2, their processing time is then exactly $p(N) + (L + \alpha) = 2s + L + \alpha > s_2$. This contradicts the definition of S_2. Hence S_2 contains at most $m - 1$ tasks. Let $S = S_1 \cup \{x_1\} \cup S_2 \cup \{x_2\}$, representing all the tasks starting before time s_2 in π. We then have $p(S) \leq p(S_1) + 2(L + \alpha) + p(S_2)$. Since no idle time occurs, we certainly have $p(S) \geq s_2 + L = 2s + 2L$. It implies that

$$p(S_2) \geq 2s - 2\alpha - (s - \alpha + a(S_1)) = s - \alpha - a(S_1)$$

As $|S_2| < m$, property (2) proves that $p(S_2) \leq s - 2\alpha$.

It results that $a(S_1) \geq \alpha$. In addition $p(S_1) = s - \alpha + a(S_1)$ must be smaller than s, which implies that $a(S_1) = \alpha$. We can conclude that set S_1 defines a valid partition of N. □

Using the argument of Corollary 1, this complexity result implies immediately the following inapproximability corollary:

Corollary 3 *Problem periodic $k-$*Onas *does not belong to FPTAS for $k \geq 3$, unless $\mathcal{P} = \mathcal{NP}$*

Proof. The reduction establishes that it is $\mathcal{NP}$-complete to decide the existence of a schedule of makespan $\sum_i p_i = (3m+2)L + 3\alpha = (3m^2+2m+3)\alpha$. Hence the gap reduction technique we use asserts that Onas can not be approximated within factor $1 + m/(3m^2+2m+3) \leq 1 + 1/3m$ in polynomial time. □

Even if the problem remains hard, list scheduling algorithms perform substantially better on periodic Onas. The next theorem shows in particular that periodic Onas belongs to APX:

Proposition 10 *For periodic* Onas *with all periods equal and small, any list scheduling has a performance guarantee of* 2.

Proof. Consider a periodic instance $\mathcal{I}$ with k unavailability periods. We call *unschedulable* a task that can not be scheduled in the infinite periodic version of the instance. Clearly in $\mathcal{I}$ an *unschedulable* task can not complete before time $s_k + L + s$ in any feasible schedule. Since the makespan of any list schedule is at most $s_k + L + \sum_i p_i$, the ratio 2 holds.

Hence assume that there is no *unschedulable* task in the instance. It implies that for each task it exists a point in time interval $[0, s]$ (and in all other intervals by translation of $s + L$) when it can be scheduled. Now consider a list schedule with an inactivity interval I starting at time t. We denote by j the index such that $t \in (s_{j-1}, s_j]$. Let i be the first task scheduled after time t. We will show that the duration of I is necessarily smaller that p_i. First, notice that since i is not scheduled at time t, we must have $t + p_i \in]s_l, s_l + L[$ for some index $l \geq j$. This implies that $s_l - t \leq p_i$. Consider the two following cases to conclude:

1. If $l = j$, then task i can be scheduled at time $s_j + L - p_i$ due to the small assumption. And certainly the list scheduling algorithm does so. Hence we have $|I| \leq s_j - t \leq p_i$.
2. Otherwise $l \geq j + 1$. But since i is not *unschedulable*, we know that there is a time in $[s_j + L, s_{j+1}]$ when the algorithm can schedule it. Hence $|I| \leq s_{j+1} - t \leq p_i$.

We proved that the duration of any idle interval is smaller than the duration of the task scheduled immediately after it. It results that the sum of the total idle time of the schedule is bounded by the sum of the processing times, proving the ratio 2. □

7 An FPTAS for periodic 2−Onas

We consider the periodic 2−Onas problem with small periods. For sake of simplicity we denote by s the starting time of the first period and L its duration. The second period starts then at time $2s+L$. Before presenting our analysis, let us rephrase the result on SubsetSum with Tolerance in a more general setting.

Proposition 11 *Let $a_1 \dots a_n$ be a list of integers indexed by increasing value, and let $\Delta = \max_{i,j} |a_i - a_j|$. For an integer B, if there exists 2 subsets X and Y of $\{1, \dots, n\}$ such that :*

$$a(X) \leq B \leq a(Y) \quad \text{and} \quad |Y| \leq |X|$$

then a subset $Z \subseteq X \cup Y$ can be found in linear time such that $a(Z) \in [B - \Delta, B]$.

Proof. Apply Algorithm 1 on instance $X \cup Y$ with X as initial set. Since the $|X|$ largest tasks do not fit in $[0, B]$, necessarily the algorithm stops on a subset Z such that $a(Z) \in [B - \Delta, B]$. □

In the next section we assume that an optimal solution exists with the first period covered by task a and the second period covered by task b. We show how to build a near optimal schedule with makespan at most at a factor $(1 + \varepsilon)$ for any $\varepsilon > 0$ in time polynomial both in n and $1/\varepsilon$.

Since SubsetSum is in FPTAS, one may think in solving 2 subset sum problems (at a precision ε) to find subsets S_1 and S_2 to schedule in interval $[0, s]$ and $[s + L, 2s + L]$, respectively. However these 2 problems are clearly not independent and can not be solved

sequentially, as selecting one task x for S_1 may be an optimal local decision for S_1, but leading to a poor subset S_2. In fact it is well-known [5] that such iterative filling for BIN PACKING has a quite poor asymptotic worst-case ratio of $\simeq 1.6$. On the opposite, guessing the set $S^* = S_1^* \cup S_2^*$ of tasks scheduled in an optimal solution before the second ONA period, we can build an optimal solution simply by applying Algorithm 1 on S^* to get subset S_1^*.

To get some clue about set S^*, very classically we solve a relaxation of the problem considering only the second ONA period. From an optimal solution S of the relaxation, we then apply the polynomial algorithm for 1−ONAS to find a subset $S_1 \subseteq S$ to be scheduled before the first ONA period. Clearly the subset $S_2 = S \backslash S_1$ may not fit before the second ONA period, resulting in an infeasible solution. We show then that a feasible (near optimal) solution can be rebuild by eventually solving a SUBSETSUM problem.

7.1 Relaxation

Consider an instance $\mathcal{I}$ of the 2−ONAS problem together with an optimal schedule π^* verifying Property 1. We denote by OPT the makespan $C_{\max}(\pi^*)$. We first restrict our attention to the case where the two ONA periods are covered by two different tasks. The remaining cases will be detailed in Section 7.4. We denote by a and b the tasks covering in π^* the first and the second ONA period, respectively. Let $S_1^* \cup S_2^* \cup S_3^*$ be the OA-packing defined by π^*, see Figure 7. We denote by N_x the set $\{i \neq x \mid p_i \leq p_x\}$ of tasks whose processing time is smaller or equal to x, x excluded. Property 1 involves that $S_1^* \subseteq N_a$, $S_2^* \subseteq N_a \cap N_b$ and $S_3^* \subseteq N_b$. We can make the following immediate remark:

Remark 5 *Let $M = \{i \neq a \mid p_b < p_i \leq p_a\}$. We have $M \subseteq S_1^*$.*

Due to our choice of an optimal dominant schedule π^*, we can assume that M fits before the first ONA period. Notice that Property 1 also involves that either a or b is the largest task of N, that we index by n. If $b = n$, then set $N_b = N \backslash \{n\}$ and set $M = \emptyset$. Otherwise if $a = n$ then we have the partition $N_a = N_b \cup \{b\} \cup M$. In any case we have by definition $M \subseteq N_a$.

We denote by $S^* = S_1^* \cup S_2^*$ the set of tasks scheduled before the second ONA period. Notice that from what precedes we have $M \subseteq S^* \subseteq N_a$, and by definition

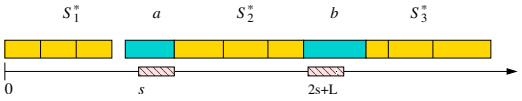


Figure 7: Structure of an optimal solution for $\mathcal{I}$

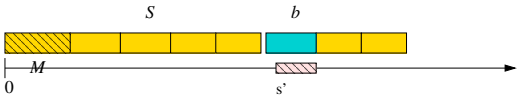


Figure 8: Structure of the relaxation for $\underline{\mathcal{I}}$. Subset M is represented with dashed line and is reduced to one single task on the example.

of M, $S^* \backslash M \subseteq N_b$. Before presenting the relaxation, we first derive a trivial upper bound on the cardinality of set S^*. We denote by χ the maximum number of tasks that can fit in time interval $[0, s]$. Indexing tasks by non-decreasing value of their processing time, quantity χ can be easily computed in linear time as $\chi = \max\{i \mid p([1, i]) \leq s\}$, see the first step of Algorithm 1. By definition both $|S_1^*|$ and $|S_2^*|$ are smaller than χ. It results that $|S^*| \leq 2\chi$.

We relax the problem by considering only the second ONA period. However, we do not disregard totally the first ONA period : as task a is to cover it, we discard its processing time from interval $[0, 2s+L]$. It corresponds to solve an instance of 1-ONAS problem with the ONA period starting at time $s' = 2s - \delta_a$ considering only the tasks of $N_a \backslash \{b\}$, see Figure 8. Equivalently we can formulate this problem as a MAXSSWT problem on instance $N_a \backslash \{b\}$, with a bound s' and a tolerance δ_b. We impose two additional constraints on the subset S we are looking for : we require that $M \subseteq S$ and $|S| \leq 2\chi$. Hence we have the following optimization problem $\underline{\mathcal{I}}$ to solve:

$$\begin{array}{ll} \max_Z & \min\{p(Z), s' - \delta_b\} \\ \text{s.t.} & p(Z) \leq s' \\ & |Z| \leq 2\chi \\ & M \subseteq Z \subseteq N_a \backslash \{b\} \end{array}$$

Problem $\underline{\mathcal{I}}$ has been constructed such that S^* is a feasible solution for it. Indeed from what precedes we have $M \subseteq S^*$ and $|S^*| \leq 2\chi$, and by definition $p(S^*) \leq 2s + L - p_a = s'$. We now establish that $\underline{\mathcal{I}}$ can be solved easily, providing an upper bound on the cardinality of S^*:

Proposition 12 *An optimal set S for $\underline{\mathcal{I}}$ can be found in linear time. In addition $|S|$ is an upper bound of $|S^*|$.*

Proof. Algorithm 1 can be easily adapted to solve $\underline{\mathcal{I}}$ in linear time, assuming that tasks are indexed by non-decreasing order of their processing times. To fulfill the two additional conditions, it suffices to choose an appropriate initial set S_0 to initialize the algorithm. We construct set S_0 including all the tasks of M, that we complete with the smallest tasks of $N_a \backslash \{b\}$, up to bound s' or up to obtain 2χ tasks. Let S be the solution returned by the algorithm on initial set S_0. Notice that by construction $|S_0| \leq 2\chi$ and $|S_0|$ is an upper bound of any feasible solution of $\underline{\mathcal{I}}$ and thus of $|S^*|$. As the number of tasks in the solution does not change through iterations, so is $|S|$. In addition set M, if not empty, contains the largest tasks of N_a and thus none of its tasks will be exchanged by the algorithm, which asserts that $M \subseteq S$. Finally to assert that S is optimal, we have to check that the difference $|p_i - p_j|$ between two tasks swapped by the algorithm is bounded by tolerance δ_b. This is immediate, writing that $N_a \backslash \{b\} \backslash M \subseteq N_b$. □

Relaxation $\underline{\mathcal{I}}$ also provides a lower bound of the optimal makespan for $\mathcal{I}$:

Proposition 13 *Let S be the optimal solution of $\underline{\mathcal{I}}$ found in proposition 12. Then $\underline{\omega} \equiv p(N) + (s' - \delta_b - p(S))^+$ is a lower bound of OPT, where $(x)^+$ stands for $\max(0, x)$.*

Proof. For any subset Z, let $\omega(Z) = p(N) + (s' - \delta_b - p(Z))^+$. Quantity $\omega(Z)$ represents the makespan of a solution packing set Z before the Ona period in the relaxation. The makespan of π^* is equal to the total work plus the total idleness in the schedule, *i.e.* $OPT = p(N) + I_1 + I_2$, with I_1 and I_2 the idleness occurring before the first Ona and the second Ona period, respectively. Before the second Ona period, schedule π^* completes at time $p(S^*) + p_a + I_1$. We have $I_2 = (2s + L - \delta_b - (p(S^*) + p_a + I_1))^+ = (s' - \delta_b - p(S^*) - I_1)^+$. Hence $OPT = \max\{\omega(S^*), p(N) + I_1\} \geq \omega(S^*)$. Since S^* is a feasible solution for $\underline{\mathcal{I}}$, optimality of S implies that $\omega(S) \leq \omega(S^*)$, which permits to conclude. □

Set S is our guess for S^*. To derive a partition (S_1, S_2) of S analogue to (S_1^*, S_2^*), we solve the 1−Onas problem restricted to set S and with only the first Ona period (covered by task a), see Figure 9.

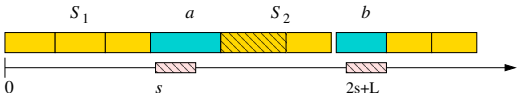


Figure 9: Structure of the relaxation (S_1, S_2) constructed from Figure 8

Definition 3 *We denote by S_1 the optimal solution of* MaxSSwT *on set S, with bound s and tolerance δ_a.*

Since $S \subseteq N_a$, this MaxSSwT problem can be solved in linear time by Algorithm 1. We choose for S_1 the solution returned by the algorithm, and we let $S_2 = S \backslash S_1$. By definition set S_1 fits before the first Ona period. We denote for clarity (S_1, S_2) the (eventually infeasible) schedule for $\mathcal{I}$ sequencing tasks of S_1 in the first interval $[0, s]$ and tasks of S_2 in time interval $[s + L, 2s + L]$, with task a covering the first Ona period. We prove in the next section that relaxation (S_1, S_2) can lead to an optimal solution for $\mathcal{I}$ in some cases.

7.2 Polynomial optimal cases

In this section we exhibit the cases were the relaxed solution (S_1, S_2) can be transformed in polynomial time into a feasible and optimal solution for instance $\mathcal{I}$. We start with the following immediate remark, which states that if S_1 matches its interval, then the relaxation gives an optimal schedule. This is the situation represented in Figure 9:

Property 3 *If $p(S_1) \geq s - \delta_a$, then schedule (S_1, S_2) is optimal for $\mathcal{I}$.*

Proof. Since $p(S_1) \leq s$ by definition, it means that S_1 matches the first Ona period. As $p(S) \leq 2s - \delta_a$, we can schedule without idle time S_1, a, S_2 before the second Ona period, *i.e.* (S_1, S_2) is feasible. The makespan clearly equals the lower bound $\underline{\omega}$ of the relaxation, which establishes the optimality. □

Hence we assume from now that $p(S_1) < s - \delta_a$. It implies that S_1 contains the largest tasks of S, and by consequence S_2 the smallest tasks of S. Also notice that tasks of S_2 starts at time $s + L$. A second polynomial case corresponds to the antisymmetric situation for S_2, *i.e.* if it does not match its interval.

Property 4 *If $p(S_2) \leq s - \delta_b$, then schedule (S_1, S_2) is optimal for $\mathcal{I}$.*

Proof. Schedule (S_1, S_2) is clearly feasible. In this schedule task a completes at the end of the first ONA period, and task b at the end of the second period. Thus the makespan clearly equals the lower bound $\underline{\omega}$. □

Thus we assume both $p(S_1) < s - \delta_a$ and $p(S_2) > s - \delta_b$. The last polynomial case is less obvious, and some transformations are needed to obtain an optimal schedule from relaxation (S_1, S_2). It corresponds to the case where set S_1 has at least as many tasks as set S_2:

Property 5 *If $|S_1| \geq |S_2|$, then an optimal schedule for $\mathcal{I}$ can be found in linear time.*

Proof. Since S_1 contains the largest tasks of S, it involves that $p(S_2) \leq p(S_1) \leq s - \delta_a$. We will distinguish two cases: $|S_1| = \chi$ and $|S_1| < \chi$.

case $|S_1| = \chi$. Let S_1' be the subset of tasks delivered by Algorithm 1 on the MAXSSWT instance composed of set S_1 union the set of the $|S_1|$ largest tasks of $N_a \backslash \{b\}$, with a bound s and a tolerance δ_a. By construction we have $S_1' \cap S_2 = \emptyset$. Thus (S_1', S_2) is a feasible schedule for $\mathcal{I}$, since $p(S_2) \leq s - \delta_a$. We claim that it is optimal:

- If no idle appears, this is certainly true,
- otherwise an idle time can only appear before the first ONA period, since $p(S_2) \geq s - \delta_b$. This idle time is hence equal to $s - p(S_1')$. Algorithm 1 asserts that S_1' then contains the χ largest tasks of N_a. In addition it is not possible to schedule more than $|S_1| = \chi$ tasks before the first ONA period, by definition of χ. It results that any feasible schedule contains an idle time greater or equal to $s - p(S_1')$.

case $|S_1| < \chi$. Algorithm 1 asserts that any subset of S of $|S_1| + 1$ tasks does not fit before s. On the contrary the set of the χ smallest tasks of N fits, by definition of χ. Applying Proposition 11, we can find in linear time a subset $S_1' \subseteq N_a \backslash \{b\}$ of cardinality $|S_1| + 1$ matching time interval $[s - \delta_a, s]$. Notice that S_1' contains at most $|S_1|$ tasks of S. We choose arbitrarily a subset S_2' of $|S_2|$ tasks among $S \backslash S_1'$. Since S_2 is composed of the smallest tasks of S, certainly we have $p(S_2') \geq p(S_2) > s - \delta_b$. Similarly we have $p(S_2') \leq p(S_1) < s - \delta_a$. Thus (S_1', S_2') is a valid schedule, without idle time. □

7.3 The $\mathcal{NP}$-hard case

As periodic 2−ONAS is $\mathcal{NP}$-hard, it must happen that S can not be polynomially transformed into an optimal schedule. Considering results of the previous section, we are in the situation where $p(S_1) < s - \delta_a$ and $p(S_2) > s - \delta_b$ and $|S_1| < |S_2|$. Actually the next property establishes that S_2 contains exactly one more task than S_1, and does not fit into its time interval.

Remark 6 *Under the previous assumptions, we have $\chi \geq |S_2| = |S_1| + 1$ and $p(S_2) > s$.*

Proof. Algorithm 1 ensures that any subset of S of cardinality $|S_1| + 1$ does not fit in time interval $[0, s]$. As $p(S) \leq 2s$, it involves that $|S| \leq 2|S_1| + 1$. Writing that $2\chi \geq |S| = |S_1| + |S_2|$ and $|S_2| \geq |S_1| + 1$, we get the first inequality. For the second one, notice that $p(S_2) \leq s$ would contradict that any subset of $|S_1| + 1$ tasks of S does not fit. □

In light of the previous remark, we clearly need to modify set S to get a feasible solution, since no subset of S with $|S_1| + 1$ tasks can fit in $[0, s]$ and S contains $2|S_1| + 1$ tasks. In fact we will swap set S_1 and S_2 in the schedule, and replace set S_2, too large, by the optimum solution of a SUBSETSUM problem. First we make the following helpful remark:

Remark 7 *We can assert that set S_1 contains the largest tasks of $N_a \backslash \{b\}$.*

Proof. Since S_1 contains the largest tasks of S, we need to prove that when solving relaxation $\underline{\mathcal{I}}$, set S contains the largest tasks of $N \backslash \{a, b\}$. Recall that set M is already composed of the largest tasks of $N \backslash \{a, b\}$. To ensure that S contains the $|S_1| - |M|$ largest following tasks, we impose at each step of Algorithm 1 to exchange the largest task $y \notin S$ with the largest task $x \in S$ with a processing time smaller than y. Notice that this version of Algorithm 1 remains linear using a merge procedure, as a task may enter at most once in set S. After l steps, as M belongs to the initial set, solution S contains the $|M| + l$ largest tasks of $N \backslash \{a, b\}$. For the sake of contradiction assume that $l + |M| < |S_1|$. It implies that set S_2 contains only tasks of the initial set, M excluded, that is the smallest tasks of N. Remark 6 implies that set S_2 should then fit in time interval $[0, s]$, since its cardinality is at most χ. But the same remark states that $p(S_2) > s$, which is a contradiction. □

Now consider the following optimization problem:

$$\max\{\ p(Z) \mid p(Z) \leq s,\ Z \subseteq N_a \backslash S_1\ \} \qquad (\mathcal{Q})$$

Let Z^* be an optimal subset for $\mathcal{Q}$. Clearly (Z^*, S_1) is a feasible schedule for $\mathcal{I}$, since Z^* fits in $[0, s]$ by construction, and $p(S_1) \leq s - \delta_a$. We claim that it is an optimal schedule:

Proposition 14 *Let Z^* be an optimal solution of $\mathcal{Q}$. Then schedule (Z^*, S_1) is optimal.*

Proof. If no idle time occurs, this is obviously true. Thus assume that some idle periods are present. We first establish that an idle time can occur only before the second Ona period. To see this consider set $\mathcal{S}_2$ constituted of the $|S_2|$ smallest tasks of the instance. Due to Remark 7 none of these tasks belongs to S_1. On one hand, since $|S_2| \leq \chi$, set $\mathcal{S}_2$ certainly fits in time interval $[0, s]$. On the other hand subset S_2 does not fit due to Remark 6. Applying Proposition 11 there exists a set $Z \subset \mathcal{S}_2 \cup S_2 \subseteq N_a \backslash S_1$ that matches the first Ona period. By definition $p(Z^*) \geq p(Z) \geq s - \delta_a$, *i.e.* set Z^* also matches the first Ona period.

As a consequence task b completes at time $2s + 2L$ in (Z^*, S_1). For the sake of contradiction assume that (Z^*, S_1) is not optimal. Since b can not complete earlier than $2s + 2L$ in π^*, we must have $p(S_1^* \cup S_2^*) > p(Z^* \cup S_1)$. Let $u = \min\{|S_1^*|, |S_2^*|\}$ and $v = \max\{|S_1^*|, |S_2^*|\}$. By definition we have $v \leq \chi$. In addition, since $|S|$ is an upper bound of $|S^*|$, Remark 6 implies that $u \leq |S_1|$. Now consider the partition of S^* into (X, Y) such that X contains the $|S_1|$ largest tasks of S^*. Since $S^* \subseteq N_a$, Remark 7 tells us that $p(X) \leq p(S_1)$. It also implies that $Y \cap S_1 = \emptyset$. As $u \leq |S_1|$, we certainly have $|Y| \leq v$. By construction of our partition it results that $p(Y) \leq \max\{p(S_1^*), p(S_2^*)\}$, *i.e.* Y is a feasible solution of $\mathcal{Q}$. The optimality of Z^* asserts that $p(Z^*) \geq p(Y)$. In conclusion we have $p(S^*) = p(X) + p(Y) \leq p(S_1) + p(Z^*)$, which is a contradiction. □

Problem $\mathcal{Q}$ is a SubsetSum problem, and thus may require an expensive computational effort to solve to optimality. However requesting only for a near optimal solution Z instead of Z^*, we obtain a feasible schedule of makespan bounded by $[p(Z^*)/p(Z)]OPT$. Since SubsetSum belongs to FPTAS, it results that $\forall \varepsilon > 0$, a schedule of makespan at most $(1 + \varepsilon)OPT$ can be found in polynomial time in n and $1/\varepsilon$.

7.4 The overall algorithm

In the previous section we have investigated the case where a dominant optimal schedule covers the two Ona periods. The remaining cases are fortunately more immediate. First, the second Ona period is necessarily covered, except if $OPT \leq 2s + L$. This latter situation reduces immediately to a 1−Onas problem, dropping the second Ona period, and thus can be solved optimally in linear time by Algorithm 1. Similarly if the first Ona period is not covered, we can assume that no task is scheduled before s (otherwise it can be shifted to complete at time $s + L$). Thus the problem reduces once again to a 1−Onas problem, starting at time $s + L$. Finally consider the case of a single task a covering the two Ona periods. Without loss of generality we can assume that a is the largest task of the instance. Notice that task a may have very little freedom in the schedule ; for instance if its processing time is $s+2L$, we can not shift neither forward nor backward the beginning of the task. Hence we can not use Algorithm 1 to solve the MaxSSwTproblem constituted of the $n - 1$ other tasks, with a bound s and a tolerance $\Delta = p_a - (s + 2L)$. But we can ignore the tolerance and solve the corresponding SubsetSum problem at a precision ε. We have the following theorem:

Theorem 5 *If $\mathcal{I}$ is an instance of small periodic* 2−Onas*, for any $\varepsilon > 0$, a feasible schedule of makespan at most $(1 + \varepsilon)OPT$ can be found in time $\mathcal{O}(n^2 + n/\varepsilon^2 \log 1/\varepsilon)$.*

Proof. Algorithm 1 is linear if tasks are sorted in non-decreasing order of their processing times, which can be done in an initial step in time $\mathcal{O}(n \log n)$. For a given couple of tasks a and b, determining a near optimal schedule can be done in $\mathcal{O}(\min(n/\varepsilon, n + 1/\varepsilon^2 \log(1/\varepsilon))$ using Lawler's FPTAS for SubsetSum [13], or the one of Kellerer et al [12]. We have $2n$ couples of tasks to consider, since the largest task is either a or b in an optimal dominant schedule. For the other cases, the complexity is at most the complexity of the FPTAS for SubsetSum. □

8 Conclusion and extensions

The concept of operator non availability periods is a new concept in the theory of scheduling arising from a practical industrial problem: the scheduling of chemical experiments taking into account the absence of the

chemists. We focused on the case of small ONA periods, which reflected in fact our industrial context. A new polynomial algorithm was found for a single ONA period that is related to a variant of the well-known SUBSET-SUM problem. For small $k-$ONAS and ONAS problems, we have precisely determined their complexity and inapproximability status. The performance of list scheduling algorithms has also been analyzed. The remaining approximability open question is whether there exists a FPTAS for the small 2-ONAS problem. It would also be interesting to develop a more efficient PTAS for small $k-$ONAS, for instance by solving more efficiently constant size problems to optimality. In the same idea, it could be investigated if good approximation ratio, smaller than 3/2 for instance, can be obtained by a list schedule or any fast algorithm on the periodic version.

Finally one can wonder if positive and negative approximability results for ONA extend to relaxed version of the problem: for instance if tasks are only forbidden to start, like in [1], or to complete during the periods. For short we call FS-schedule (for *forbidden start*) a schedule where no task starts inside a period, and FE-schedule (for *forbidden end*) a schedule where no task completes. An ONA-schedule is then both a FS and a FE schedule. Consider the following language:

$$\mathcal{L}_k^Y = \{ \quad x \text{ an instance of } k-\text{ONAS} \mid \text{there exists a } Y\text{-schedule for } x \text{ of makespan at most } \sum_i p_i \}$$

where Y is either ONA, FE or FS environment. It means that $\mathcal{L}_k^Y$ represents all the instances for which a Y-schedule exists without idle time. Clearly as an ONA -schedule is also a FS-schedule and a FE-schedule, we have $\mathcal{L}_k^{\text{ONA}} \subseteq \mathcal{L}_k^{\text{FS}}, \mathcal{L}_k^{\text{FE}}$. But in a schedule without idle time, any task must start immediately after the completion of its predecessor. Hence we also have the reverse inclusion, proving that the 3 languages $\mathcal{L}_k^{\text{ONA}}$, $\mathcal{L}_k^{\text{FS}}$ and $\mathcal{L}_k^{\text{FE}}$ are in fact identical. Since proofs of Theorems 3 and 4 establish that $\mathcal{L}_2^{\text{ONA}}$ is $\mathcal{NP}$-complete, both FS and FE scheduling problems are $\mathcal{NP}$-hard for more than 2 periods, even in the case small and periodic.

If the complexity status is the same, inapproximability of the problems differ. Indeed this is not difficult to see that any list scheduling algorithm has a performance guarantee of 2 for FS or FE with small unavailability periods. Recall that in contrast problem ONAS does not belong to APX if $\mathcal{P} \neq \mathcal{NP}$. When the number of periods is fixed, we have proved that k-ONAS do not belong to FPTAS for more than $k \geq 3$ periods. Is it still true for FE or FS relaxations ? For FE scheduling, we can use the same gap reduction as in Corollary 1 adding to an instance x a $(k+1)$th period at time $C = \sum_i p_i$. The new instance $g(x)$ has $k+1$ periods, and:

$$\begin{array}{rcl} x \in \mathcal{L}_k^{\text{FE}} & \Rightarrow & OPT(g(x)) = C \\ x \notin \mathcal{L}_k^{\text{FE}} & \Rightarrow & OPT(g(x)) \geq C + L \end{array}$$

proving that k-FE scheduling does not belong to FPTAS for $k \geq 3$. For FS scheduling, we need to restrict to instances of $\mathcal{L}_k^{\text{FS}}$ verifying $L \leq p_i < 3/2L$ for all tasks. Proofs of Theorems 3 and 4 again show that this language remains $\mathcal{NP}$-complete. In the gap reduction g we add to such an instance x a new task l of duration $3/2L$ and again a $(k+1)$th period at time $C = \sum_i p_i$. Then we have

$$\begin{array}{rcl} x \in \mathcal{L}_k^{\text{FS}} & \Rightarrow & OPT(x') = C + 3/2L \\ x \notin \mathcal{L}_k^{\text{FS}} & \Rightarrow & OPT(x') \geq C + 2L \end{array}$$

since if $x \notin \mathcal{L}_k^{\text{FS}}$, the last task of the schedule can not start before time $C + L$. Hence once again k-FS scheduling does not belong to FPTAS for $k \geq 3$, even for equal and periodic unavailability periods.

Finally note that, maybe surprisingly, algorithm 1 for $1-$ONA remains optimal in both environments: we can always shift tasks in an FE or FS schedule to convert it into an ONA schedule of same makespan, except for trivial cases.

To conclude we can cite some interesting extensions of the model. The first natural one that arises from the previous paragraph is to mix the 3 models, the tasks being from one of them (forbidden start, forbidden end or both), *i.e.* some tasks may required the additionnal resource only for set-up, or only for terminaison, or for both of them. Mauguire, Billaut and Bouquard [17] study such mix of tasks for machine unaivalability periods, where some tasks are resumable and some others are not. They also mix different types of periods: a second extension for ONAS could consider several additional resources with their own availability constraints. Some resources may be required only for start-up, while others only for terminaison. The case of two additionnal ressources, one needed for start-up and one needed for terminaison, seems particularly interesting. Finally our model could be compared to a server that handles zero duration set-ups and terminations of the tasks. One could consider a classical

server model with non-zero set-up times, but with unavailability constraints.

References

[1] 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*, to appear. DOI 10.1007/s10288-007-0061-5.

[2] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Rapine, 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.

[3] P. Brucker, C. Dhaenens-Flipo, S. Knust, S.A.Kravchenko, and F. Werner. Complexity results for parallel machine problems with a single server. *Journal of Scheduling*, 5:429–457, 2002.

[4] A. Caprara, H. Kellerer, and U. Pferschy. The multiple subset sum problem. *SIAM Journal on Optimization*, 11(2):308–319, 2000.

[5] A. Caprara and U. Pferschy. Worst-case analysis of the subset sum algorithm for bin packing. *Operations Research Letters*, 32:159–166, 2004.

[6] C. Chekuri and S. Khanna. A PTAS for the multiple knapsack problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 213–222, 2000.

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

[8] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17, 1969.

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

[10] K. Jansen. Parametrized approximation scheme for the multiple knapsack problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 665–674, 2009.

[11] Min Ji, Yong He, and T.C.E. Cheng. Single-machine scheduling with periodic maintenance to minimize makespan. *Computers & Operation Research*, 34:1764–1770, 2007.

[12] H. Kellerer, R. Mansini, U. Pferschy, and M.G. Speranza. An efficient fully polynomial approximation scheme for the subset-sum problem. *Journal of Computer and System Sciences*, 66(2):349–370, 2003.

[13] E.L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4:339–356, 1979.

[14] 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.

[15] 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.

[16] J.Y.-T. Leung, M. Dror, and G.H. Young. A note on an open-end bin packing problem. *Journal of Scheduling*, 4:201–207, 2001.

[17] P. Mauguire, J.-C. Billaut, and J.-L. Bouquard. New single machine and job-shop scheduling problems with availability constraints. *Journal of Scheduling*, 8:211–231, 2005.

Les cahiers Leibniz ont pour vocation la diffusion des rapports de recherche, des séminaires ou des projets de publication sur des problèmes liés au mathématiques discrètes.

Pour soummettre un articles dans les cahiers,
`http://www.g-scop.inpg.fr/CahiersLeibniz/`

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