Approximation of the Parallel Machine Scheduling Problem with Additional Unit Resources

Emmanuel Hebrard[a,*], Marie-José Huguet[a,*], Nicolas Jozefowiez[a], Adrien Maillard[b], Cédric Pralet[b,*], Gérard Verfaillie[b]

[a] *LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France*
[b] *ONERA – The French Aerospace Lab, F-31055, Toulouse, France*

Abstract

We consider the problem of minimizing the makespan of a schedule on m parallel machines of n jobs, where each job requires exactly one of s additional unit resources. This problem collapses to $P||C_{max}$ if every job requires a different resource. It is therefore NP-hard even if we fix the number of machines to 2 and strongly NP-hard in general.

Although very basic, its approximability is not known, and more general cases, such as *scheduling with conflicts*, are often not approximable. We give a $(2 - \frac{2}{m+1})$-approximation algorithm for this problem, and show that when the deviation in jobs processing times is bounded by a ratio ρ, the same algorithm approximates the problem within a tight factor $1 + \rho\frac{(m-1)}{n}$.

This problem appears in the design of download plans for Earth observation satellites, when scheduling the transfer of the acquired data to ground stations. Within this context, it may be required to process jobs by batches standing for the

*Corresponding author

Email addresses: hebrard@laas.fr (Emmanuel Hebrard), huguet@laas.fr (Marie-José Huguet), jozefowiez@laas.fr (Nicolas Jozefowiez), cedric.pralet@onera.fr (Cédric Pralet)

set of files related to a single observation. We show that there exists a $(2 - \frac{1}{m})$-approximation algorithm respecting such batch sequences. Moreover, provided that the ratio ρ, between maximum and minimum processing times, is bounded by $\lfloor \frac{s-1}{m-1} \rfloor$, we show that the proposed algorithm approximates the optimal schedule within a factor $1 + \frac{s-1}{n}$.

Keywords: Scheduling, Approximation, Additional resource

1. Introduction

We consider the problem of scheduling a set of n jobs on m parallel machines. Moreover, each job requires exactly one of s additional resources. These additional resources have a unit capacity, hence no two jobs requiring the same resource can be scheduled in parallel. The objective is to minimize the overall makespan of the schedule.

This problem occurs in numerous applications where an exclusive resource must be shared. Our particular motivation, however, comes from the problem of planning the download of acquisitions made by agile observation satellites (Pralet et al., 2014).

When a satellite makes an observation, the acquired data is compressed and stored onboard, waiting to be downloaded once a communication link with a ground station is established. The data of an acquisition corresponds to a set of exactly s files, each one corresponding to a frequency (observations are multi-frequency optic recordings). There are s memory banks, and every file of a given observation is stored on a different one.

When flying over a ground station, some of the data stored on the memory banks can be downloaded. The objective is to download as many files as possible,

possibly weighted by some priority function.

The satellite has a communication link with a ground station only for a short period called *download window*. The problem we consider is, given a subset of the files stored onboard, whether it is possible to download them all within the download window, or in other words, whether the optimal makespan is less than a given constant. Typically, this problem will be solved as a subproblem of a larger problem, including the selection of the subset of files to be downloaded.

Since there are m independent download channels, this problem is equivalent to parallel machine scheduling. Each file is a job, and we know exactly the duration of its download: it is linearly dependent on the size of the file (varying according to the type of observation and to the compression). Any file can be downloaded through any download channel, and on every channel, downloads must be sequential. We must therefore allocate the files to channels and sequence all the tasks on every channel.

Moreover, in a given memory bank, files can be read, and thus downloaded, only one at a time. We can therefore view the set of files stored on the same memory bank as a set of jobs sharing the same mutually exclusive resource. In other words, we can model our problem as scheduling n jobs (files) on m machines (download channels) subject to s resources (memory banks), where all demands and capacities are unit.

Example 1. Consider two acquisitions, I_a and I_b, composed of 5 files to be downloaded to the ground. For each acquisition, each file is stored in a given memory bank (resource), R_1, R_2, R_3, R_4 or R_5. There are then 10 jobs to be downloaded. For instance, job a_1 stands for the file of acquisition I_a stored on memory bank R_1. Job processing times and resource allocations are shown in Figure 1, and two

possible download plans using three download channels (machines), M_1, M_2, M_3 are shown in Figure 2.

	Memory bank:	R_1	R_2	R_3	R_4	R_5
Acq. I_a	jobs:	a_1	a_2	a_3	a_4	a_5
	processing time:	3	4	3	3	3
Acq. I_b	jobs:	b_1	b_2	b_3	b_4	b_5
	processing time:	3	3	3	2	4

Figure 1: An instance with two acquisitions using five memory banks (additional resources)

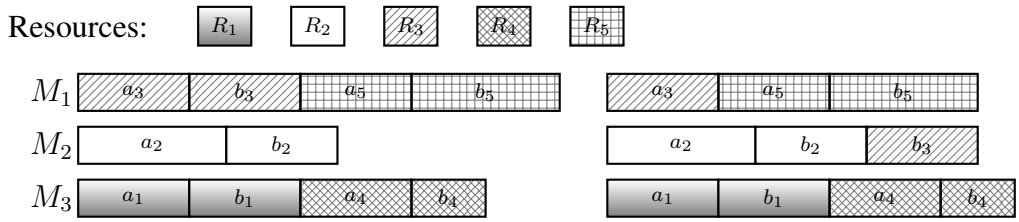

Figure 2: Two feasible solutions for the instance shown in Figure 1.

This problem is a generalization of $P||C_{max}$ which is strongly NP-hard (Blazewicz et al., 1983; Garey and Johnson, 1978). Moreover, the particular case of 2 parallel machines is also NP-hard since the problem $P2||C_{max}$ is NP-hard, though pseudopolynomial.

In a recent survey (Edis et al., 2013), the authors give an overview of parallel machine scheduling with additional resources. To this purpose, the field β of the classical three-field notation $\alpha|\beta|\gamma$ for scheduling problems was extended to take into account additional resources in parallel machine scheduling (Blazewicz et al., 1983). In particular, $\beta = res\lambda\sigma\delta$ indicates that each job requires δ units of up to

λ resources, each of capacity σ. A dot "." instead of one such value denotes that it is part of the input.

The problem we consider is a subproblem of $P|res.11|C_{max}$ where each job requires exactly one additional resource. Alternatively, one can see the s unit resources as s dedicated parallel machines, and the m parallel machines as a single cumulative resource of capacity m (jobs require exactly one unit of the cumulative resource in this case). This alternative is a subproblem of $PD|res1.1|C_{max}$ where again each job requires exactly one additional resource. There is no known approximation algorithm for either $P|res.11|C_{max}$ or $PD|res1.1|C_{max}$ (Edis et al., 2013) however we show one for the subproblem considered in this paper.

Notice that $PD2|res1..|C_{max}$ is polynomially solvable (Kellerer and Strusevich, 2003), which means that our problem has a polynomial time algorithm if there are no more than 2 additional resources. Moreover, the even more general problem $PD|res1..Int|C_{max}$[1] has a $(3+\epsilon)$-approximation algorithm (Grigoriev and Uetz, 2009).

Finally, it is also a particular case of the problem of *scheduling with conflicts* (SWC) (Even et al., 2009). In this problem, n jobs should be scheduled on m parallel machines such that the makespan is minimized. Moreover, as in our case, two conflicting jobs cannot be processed concurrently. However, whereas in our case two jobs are conflicting if and only if they require the same resource, in SWC the set of conflicts can be an arbitrary graph. In other words we consider the particular case of SWC where the conflict graph is a set of disjoint cliques. The positive results can therefore be inherited from SWC. For instance, it is polynomial for two

[1]The notation Int indicates that the processing time of a job depends on the number of units of resources it receives.

machines ($m = 2$) and processing times in $\{1, 2\}$ (Even et al., 2009), and there is a $\frac{4}{3}$-approximation algorithm for processing times in $\{1, 2, 3\}$. However, results are mainly negative: SWC is NP-hard when the processing times are in $\{1, 2, 3\}$ even for two machines (Bendraouche and Boudhar, 2012), and APX-hard when they are in $\{1, 2, 3, 4\}$ (Even et al., 2009).

The main contributions of the paper are as follows. Firstly, we give an algorithm that approximates the optimal solution for the problem described above within a tight factor $1 + \rho\frac{(m-1)}{n}$, where ρ is the ratio between maximum and minimum processing times. Secondly, we show that the same procedure is a $(2-\frac{2}{m+1})$-approximation algorithm in general, however, we show that this factor is tight only for $m \leq 2$. Finally, we consider the case where jobs must be inserted by batches involving jobs that require distinct additional resources. These batches correspond to all the files constituting a single acquisition in our observation satellite planning application. We show that if the ratio ρ between maximum and minimum processing times is less than or equal to $\lfloor\frac{s-1}{m-1}\rfloor$ then the optimal schedule can be approximated within a factor $1 + \frac{s-1}{n}$.

Throughout the paper, we use the following notations and conventions:

n:	the number of jobs, denoted $a_1, \ldots, a_n$
$res(a)$:	the resource required by job a
p_a:	the processing time of job a, whereas p_{min} and p_{max} stand for the minimum and maximum processing times, respectively
s_a and e_a:	the start and end times of job a in the schedule, respectively
m:	the number of machines, denoted $M_1, \ldots, M_m$
e_M:	the end time of the last job scheduled on machine M
e_{min} and e_{max}:	the completion time of the last job on the machine finishing first and last, respectively. That is, $e_{min} = \min(\{e_{M_j} \mid 1 \leq j \leq m\})$ and $e_{max} = \max(\{e_{M_j} \mid 1 \leq j \leq m\})$
s:	the number of additional resources, denoted $R_1, \ldots, R_s$.
L:	represents the cumulated processing time of the jobs $L = \sum_{i=1}^{n} p_{a_i}$
R:	denotes the set of jobs requiring this resource.
$L(R)$:	stands for the load of the resource R, $L(R) = \sum_{a \in R} p_a$

The rest of the paper is organized as follows. In Section 2, we give some general results on an enqueueing heuristic method that can be applied for both online and offline scheduling. This heuristic shall be used in all other algorithms. Then, in Section 3, we prove approximation results for offline scheduling problems. Last, in Section 4, we consider the case of batch scheduling.

2. Online Scheduling

To solve our problem, we consider greedy heuristics which iteratively insert a job a into a partial schedule $\mathcal{S}$. We can define these heuristics as an ordering on the jobs and an insertion procedure called ENQUEUE shown in Algorithm 1. This

procedure shall be reused in all subsequent heuristics.

The algorithm ENQUEUE considers a job a and a partial schedule $\mathcal{S}$. An invariant of this algorithm is that trailing jobs (i.e., the last jobs on their respective machines) require distinct resources. Therefore, the resource required by a is either required by a single trailing job, in which case a will follow that job on the same machine; or no trailing job requires the same resource, and it will be inserted at the back of one of the first available machines.

Algorithm 1: ENQUEUE($\texttt{schedule} : \mathcal{S}, \texttt{job} : a$)

if *there exists a machine* M_j *whose last job in* $\mathcal{S}$ *requires* $res(a)$ **then**
    insert a at the back of machine M_j in $\mathcal{S}$
else
    insert a at the back of a machine with mimimum completion time in $\mathcal{S}$

In the following lemma, we show that the schedules obtained by applying the insertion procedure ENQUEUE on every job of any set, and in any order, are feasible and dense. In other words, there is no idle time between the first and last job on any machine in such schedules. From this, we can deduce an approximation factor of $(2 - \frac{1}{m})$ inherited by all algorithms introduced in this paper. Moreover, the density property will be useful in subsequent proofs. In particular, it entails an upper bound on the makespan:

Lemma 1. *Let σ be the makespan of a dense solution, then $\sigma \leq L - (m-1)e_{min}$.*

PROOF. Immediate from the fact that one machine finishes at time σ, and $m - 1$ machines finish at or after time e_{min}. □

Lemma 2. *A schedule obtained by a sequence of calls to* ENQUEUE *is feasible and dense, and for every machine M, all jobs a processed on M with $e_a \geq e_{min}$ require the same resources.*

PROOF. We prove this proposition by induction on the number of jobs n. For 1 job, it trivially holds. Now, we suppose that it holds for n jobs, and show that it also holds for $n + 1$ jobs.

Let R_M be the unique resource such that every job scheduled on machine M and finishing later than e_{min} requires R_M (or $\emptyset$ if $e_M = e_{min}$). Observe that since the schedule is feasible, there are no two machines M_1, M_2 such that $R_{M_1} = R_{M_2}$ unless $e_{M_1} = e_{M_2} = e_{min}$ (hence for any job a, there is at most one machine M_j such that $R_{M_j} = res(a)$). When inserting the $n + 1$-th job a, there are two cases (illustrated in Figure 3):

Case 1: $\nexists j \in [1, \ldots, m]$ *s.t.,* $res(a) = R_{M_j}$.
Job a is processed on a machine M whose last job finishes first. It can start immediately after this last, i.e., at time e_{min}, since there is no resource conflict with $res(a)$ on the interval $[e_{min}, e_{max}]$. Now, e_{min} and maybe e_{max} can increase because of this insertion. In all machines but M, this does not invalidate the induction hypothesis. On M, there is now at most a single job on the interval $[e_{min}, e_{max}]$, hence the induction hypothesis is also verified.

Case 2: $\exists j \in [1, \ldots, m]$ *s.t.,* $res(a) = R_{M_j}$.
Job a is processed on the machine M_j such that $res(a) = R_{M_j}$. It can start immediately after the last job of this machine, since $e_{M_j} \geq e_{min}$, and there is no other machine M_i such that $e_{M_j} \leq e_{M_i}$ and $R_{M_j} = R_{M_i}$. Observe that in this case R_{M_k} remains unchanged for every machine M_k, hence the induction hypothesis is also verified. □

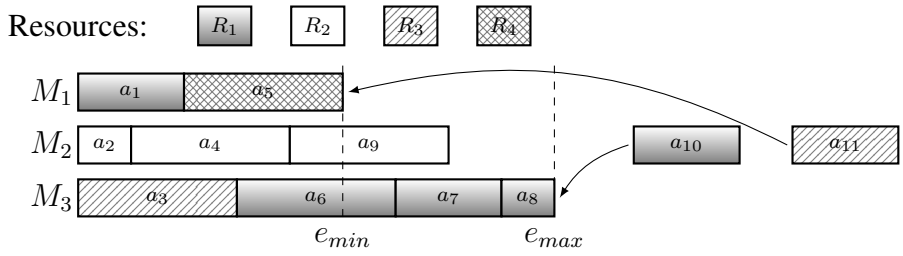

Figure 3: Illustration of the proof of Lemma 2. Job a_{10} falls into the second insertion case whilst job a_{11} falls into the first case.

Theorem 1. *If σ is the makespan of a schedule obtained by calling the procedure* ENQUEUE *on every job, then there is a resource R such that $\sigma \leq \frac{L}{m}+(1-\frac{1}{m})L(R)$.*

PROOF. The schedule is dense thus $\sigma+e_{min}(m-1) \leq L$ by Lemma 1. Moreover, let R be the resource associated with a job finishing at time σ and let M be the machine used for this job. By Lemma 2, all jobs on interval $[e_{min}, \sigma]$ for machine M require resource R, and hence $L(R) \geq \sigma - e_{min}$. Therefore, we have $\sigma + (\sigma - L(R))(m-1) \leq L$. □

Corollary 1. *A succession of calls to the procedure* ENQUEUE*, one for every job, in any order, is a $(2-\frac{1}{m})$-approximation algorithm.*

PROOF. Let σ be the makespan of a schedule obtained by a succession of calls to ENQUEUE and σ^* be the optimal makespan. Since $\sigma^* \geq \frac{L}{m}$ and for every resource R, $\sigma^* \geq L(R)$ we have by Theorem 1 $\sigma \leq \sigma^* + \frac{m-1}{m}\sigma^*$. □

Notice that this Theorem holds for the on-line version of the problem since no assumption is made about the order in which jobs are given to Algorithm 1.

Moreover, the procedure ENQUEUE is optimal if $s \leq m$ since in that case two jobs requiring different resources would never be processed on the same machine.

In Example 2, we show that the given ratio is tight for any value of m.

Example 2. The n jobs to be scheduled on the set of m parallel machines consist of $m-1$ jobs with processing time $m-1$, $m-1$ jobs with processing time 1 and 1 job with processing time m, all requiring a different resource (that is $s = n$).

If we use ENQUEUE in input order, we obtain a makespan $\sigma = 2m - 1$. However, it is possible to obtain the optimal makespan $\sigma^* = m$ by processing the longest job alone on a machine, and on each $m-1$ remaining machine, exactly one job with processing time $m-1$ and one job with processing time 1. We illustrate this example for $n = 7$ jobs, $m = 4$ parallel machines and $s = 7$ resources in Figure 4.

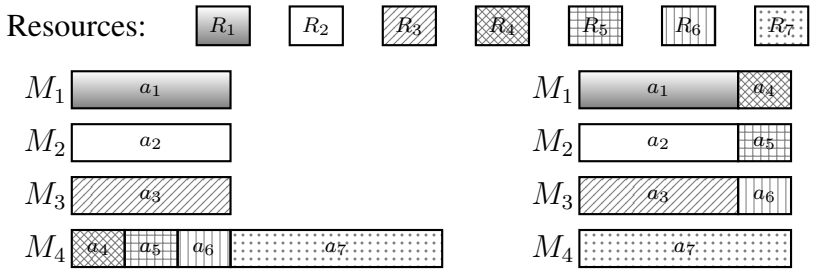

Figure 4: Illustration of Example 2, for $m = 4$ machines. The schedule on the left is obtained by calling ENQUEUE on a_1 up to a_7 in lexicographic order. The schedule on the right is optimal.

3. Offline Scheduling

In this section, we study the greedy heuristic called MAXLOAD, shown in Algorithm 2. This heuristic gives a better approximation ratio than ENQUEUE

thanks to the order in which it explores the jobs. It makes its choice based on the *current* load of a resource, that is the sum of the processing time of jobs requiring this resource that are not yet scheduled. At each step, it selects the resource R with the maximum current load. Then, the job a requiring R and having the maximum processing time is selected and is added to the partial schedule using the insertion procedure ENQUEUE. We show that the greedy heuristic MAXLOAD is a $(2 - \frac{2}{m+1})$-approximation algorithm for the parallel machine scheduling problem with additional unit resources. Moreover, the approximation factor tends to 1 when the ratio between maximum and minimum processing time of any job is bounded by a constant.

Algorithm 2: MAXLOAD(`set of jobs` : $\mathcal{T}$)

$\mathcal{S} \leftarrow \emptyset$

while $\mathcal{T} \neq \emptyset$ **do**

  let R be a resource with maximum load $\sum_{a \in R \cap \mathcal{T}} p_a$

  pick a job $a \in R \cap \mathcal{T}$ with maximum p_a, and remove it from $\mathcal{T}$

  ENQUEUE($\mathcal{S}, a$)

return $\mathcal{S}$

In order to prove the approximation result, we first introduce some notations for sequences of jobs requiring the same resource.

Let $\overrightarrow{R}(a)$ denote the sequence of all jobs requiring R in the order they are selected by MAXLOAD, i.e., by non-increasing processing time, starting from a (included). We can extend the notion of load to such sequences: $L(\overrightarrow{R}(a)) = \sum_{a_i \in \overrightarrow{R}(a)} p_{a_i}$.

Moreover, let $[s, e]$ be an interval. We say that $[s, e]$ is a *stretch* of the resource

R if at any time in $[s, e]$, there is a job requiring R in process.

Lemma 3. *Let a_1, a_2 be two jobs requiring the resources R_1 and R_2, respectively, and let s_a, e_a be the start and end times of a job a. If $L(\overrightarrow{R_1}(a_1)) > L(\overrightarrow{R_2}(a_2))$ and if $s_{a_1} \leq e_{a_2} \leq e_{a_1}$, then $[s_{a_2}, e_{a_1}]$ is a stretch of R_1.*

PROOF. First, observe that $L(\overrightarrow{R_1}(a_1)) > L(\overrightarrow{R_2}(a_2))$ implies that one of the two following propositions holds:

1. a_1 starts at least as early as a_2.
2. a_1 follows directly another job of R_1 on the same machine.

Indeed, a_1 must be inserted before a_2. It can either go on the earliest finishing machine, which guarantees that proposition 1 will be satisfied; or on the machine that ends with a job of R_1, if it exists, which satisfies proposition 2.

Now, suppose that the claim is false, i.e., there is a window in $[s_{a_2}, e_{a_1}]$ with no job requiring R_1 in process, and let a_1' be the job of R_1 directly following that window. It precedes (or is) a_1, therefore $L(\overrightarrow{R_1}(a_1')) > L(\overrightarrow{R_2}(a_2))$. However, this is a contradiction since a_1' satisfies neither of the two propositions above. □

Now, we can prove an intermediate theorem bounding the gap between the end e_{min} of the dense part of the schedule and the total makespan σ.

Lemma 4. *If p_{max} is the maximum processing time of a job, and if the schedule returned by* MAXLOAD *is not optimal, then the completion times of the last jobs on any pair of machines are less than p_{max} apart.*

PROOF. We assume that the conclusion does not hold, that is, $e_{max} - e_{min} > p_{max}$, and we show that it contradicts the premise.

Without loss of generality, we assume $e_{M_1} \leq \ldots \leq e_{M_m}$. By Lemma 2, we know that $[e_{min}, e_{max}]$ is a stretch of a single resource that we shall call R_m on the machine M_m. Let a_m be the first job requiring R_m finishing in this stretch (i.e., such that $e_{a_m} \geq e_{min}$). Moreover, let a_1 be the last job on machine M_1 and let R_1 be its resource. Since $\overrightarrow{R_1}(a_1)$ involves only one job a_1 we have $L(\overrightarrow{R_1}(a_1)) \leq p_{max}$ and since $e_{max} - e_{min} > p_{max}$ and $[e_{min}, e_{max}]$ is a stretch of R_m, we have $L(\overrightarrow{R_1}(a_1)) < L(\overrightarrow{R_m}(a_m))$. Therefore by Lemma 3, we know that $[s_{a_1}, e_{max}]$ is a stretch of R_m. By the same argument we can extend this stretch over all jobs directly preceding a_1 and requiring the same resource R_1. Let a'_m and a'_1 be the first jobs of these extended stretches, for R_m and R_1, respectively. We have $L(\overrightarrow{R_1}(a'_1)) + p_{max} < L(\overrightarrow{R_m}(a'_m))$, because the stretch on R_m is longer than the one on R_1 by at least $e_{max} - e_{min} > p_{max}$.

Now, let a_2 be the last job of M_1 that does not require R_1 and let R_2 be its resource (a_2 directly precedes a'_1). Since we switched from that job to a job of R_1, we know that $L(\overrightarrow{R_2}(a_2)) - p_{a_2} \leq L(\overrightarrow{R_1}(a'_1))$. Therefore we have $L(\overrightarrow{R_m}(a'_m)) > L(\overrightarrow{R_2}(a_2))$. Now, we can use Lemma 3 again to stretch on R_m over that stretch on R_2. This argument can be applied to extend the stretch on R_m over every preceding job on M_1. Since the schedule is dense by Lemma 2, the stretch on R_m can be extended to time 0 and hence we have shown that there is a job requiring the resource R_m in process at all times. It follows that if $e_{max} - e_{min} > p_{max}$, then the schedule is optimal. □

Theorem 2. *If σ is the makespan of a schedule obtained by the procedure* MAXLOAD*, then either σ is optimal or $\sigma \leq \frac{L}{m} + (1 - \frac{1}{m})p_{max}$.*

PROOF. The schedule is dense thus $\sigma + e_{min}(m-1) \leq L$ by Lemma 1. Moreover,

by Lemma 4, either σ is optimal or $\sigma - e_{min} \leq p_{max}$, and hence $\sigma + (\sigma - p_{max})(m - 1) \leq L$. □

Corollary 2. *If* $\frac{p_{max}}{p_{min}} \leq \rho$ *then* MAXLOAD *approximates the optimal schedule within a factor* $1 + \rho\frac{m-1}{n}$.

PROOF. By Theorem 2, if σ is the makespan of a schedule returned by MAXLOAD and σ^* is the optimal makespan, we have: $\sigma \leq \sigma^* + (1 - \frac{1}{m})p_{max}$.

However, we have $\frac{L}{n} \geq p_{min}$ hence $\rho\frac{L}{n} \geq p_{max}$, and since $\sigma^* \geq \frac{L}{m}$ then $\rho\frac{m\sigma^*}{n} \geq p_{max}$. We inject this into the first inequality to obtain: $\sigma \leq \sigma^* + \rho\frac{m-1}{n}\sigma^*$. □

Lemma 5. *Let* a_j *be a job starting strictly after* $t \geq 0$ *in a schedule returned by* MAXLOAD. *Either there exists a job processed at time* t *and requiring the same resource as* a_j*, or every job in process at time* t *has been inserted before* a_j.

PROOF. Suppose that there is no job processed at time t and requiring the same resource as a_j, and consider a job a_i processed at t. If they are processed on the same machine, it follows immediately that a_i had been inserted before a_j.

Otherwise, suppose that a_j has been inserted before a_i, and let e'_{min} and e'_{max} be the earliest and latest machine completion times just before the insertion of a_i. Since a_i starts at t or before we have $e'_{min} \leq t$. Now since a_j was inserted before a_i, it finishes before or at e'_{max}, and since it starts after t, it is included in the interval $[e'_{min}, e'_{max}]$ and we have $e'_{min} \leq t < e'_{max}$.

However, the job processed at time t on the same machine as a_j does not require the same resource as a_j, which contradicts Lemma 2. □

Theorem 3. MAXLOAD *is a* $(2 - \frac{2}{m+1})$*-approximation algorithm.*

PROOF. Suppose that the schedule returned by MAXLOAD has a makespan $\sigma > (2 - \frac{2}{m+1})\sigma^*$. It entails $s > m > 1$ and $K = \sigma - e_{min} > 0$ since otherwise MAXLOAD is optimal. By Lemma 2 we know that $[e_{min}, \sigma]$ is a stretch on some resource R_{m+1} on a machine M_1. Therefore, we have $L(R_{m+1}) \geq K$. This stretch must have started at a time t greater than 0, otherwise the schedule would be optimal. So there exists a time point in $[0, t)$ such that none of the jobs in process at that time require R_{m+1}. Moreover, there are m such jobs, all with distinct resources, since the schedule is dense up until $e_{min} > t$. By Lemma 5, we know that these jobs have all been inserted before the first job a_{m+1} of the stretch of R_{m+1}. It follows that each one of these m resources has a load of at least K.

The total load ($L = \sum_{i=1}^{n} p_{a_i}$) is therefore at least $(m+1)K$, and thus $\sigma^* \geq \frac{m+1}{m}(\sigma - e_{min})$ or $e_{min} \geq \sigma - \frac{m}{m+1}\sigma^*$. The claim now follows from Lemma 1 and from $\sigma^* \geq \frac{L}{m}$. □

In Example 3, we show that this ratio is tight only for $m \leq 2$ (for $m = 2$ we have a tight ratio of $\frac{4}{3}$). However, it tends to 2, and is not tight, when m grows.

Example 3. Consider a problem with m parallel machines, $m + 1$ resources and the following jobs:

resource	jobs	processing time
R_1	$a_1, \ldots, a_{m-1}$	
R_2	$a_m, \ldots, a_{2m-2}$	$\forall 1 \leq i \leq m(m-1),\ p_{a_i} = p$
$\ldots$	$\ldots$	
R_m	$a_{(m-1)^2}, \ldots, a_{m(m-1)}$	
R_{m+1}	$a_{m(m-1)+1}, \ldots, a_{m^2}$	$\forall m(m-1)+1 \leq i \leq m^2,\ p_{a_i} = \frac{p}{m}$

MAXLOAD might enqueue all jobs of resources R_1 to R_m before enqueueing any job requiring R_{m+1} since the load of R_{m+1} is equal to p. The makespan

obtained with this insertion order is equal to mp. Indeed, on each machine there are $m-1$ jobs of any one resource in R_1 to R_m. Moreover, every job requiring the last resource R_{m+1} will be scheduled on the same machine, which completion time is thus mp.

However, it is possible to obtain a makespan of $(m-1)p + \frac{p}{m}$ by interleaving the jobs requiring resource R_{m+1} with jobs requiring other resources. More precisely, job $a_{m(m-1)+i}$ should be the i^{th} job on machine M_i, with all other jobs on that machine being those requiring resource R_i. We illustrate this example in Figure 5. The difference between σ and σ^* is thus:

$$mp - ((m-1)p + \frac{p}{m}) = (1 - \frac{1}{m})p$$

which shows that Theorem 2 provides a tight bound for any value of m. However, the ratio between σ and σ^* is:

$$\frac{mp}{(m-1)p + \frac{p}{m}} = 1 + \frac{m-1}{m(m-1)+1}$$

This lower bound is equal to the upper bound given by Theorem 3 for $m \leq 2$, however, it is strictly smaller for $m > 2$.

4. Batch Scheduling

In our acquisitions download planning application presented in Section 1, jobs are grouped by *batches* of s jobs, each one requiring a different resource. In this application, jobs represent files and a batch is a set of files linked to an acquisition, resources correspond to memory banks and parallel machines to telecommunication channels. Indeed, it is preferred to group the download of all the files of an acquisition for at least three reasons. First, if for some reason the download is

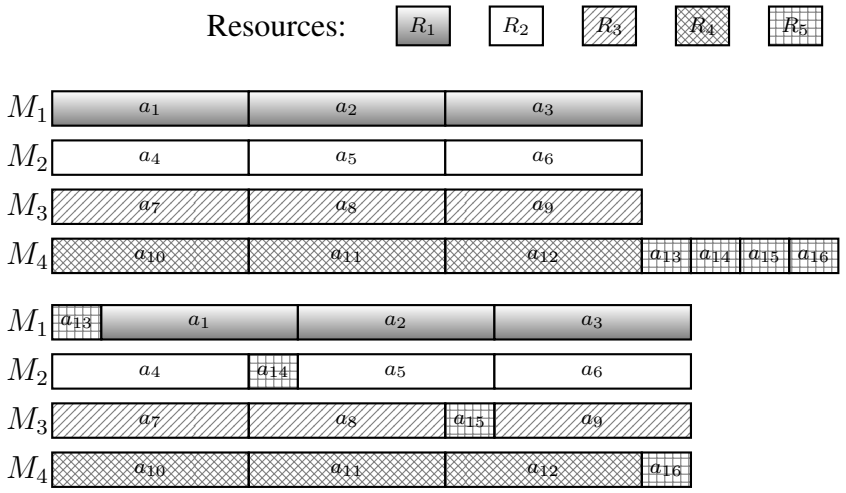

Figure 5: Illustration of Example 3, for $m = 4$ machines. The schedule on the top is obtained by calling MAXLOAD. The schedule on the bottom is optimal.

interrupted, any acquisition for which a single file is not yet downloaded is lost. By grouping all the files of an acquisition, we reduce the probability of these occurrences. Second, some acquisitions have higher priority and we might want to download them first if possible. Third, all files of a single acquisition must be downloaded during the same fly-by of the chosen ground station. We therefore consider a "pseudo" on-line version of the problem where jobs arrive by batches: we must completely schedule a batch before moving to the next one.

Example 4. Consider again Example 1. Now we assume that jobs are given in two batches, containing the 5 files of acquisitions I_a and I_b, respectively. Figure 6 is a feasible solution for this setting.

It is therefore important, in our application, to have an efficient method, albeit following a predefined batch order. By Corollary 1, we know that using the insertion procedure ENQUEUE in any order is a $(2 - \frac{1}{m})$-approximation.

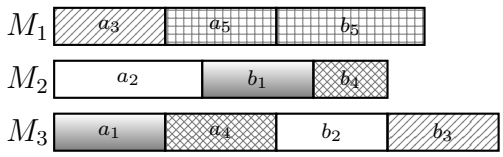

Figure 6: A batch solution of the instance shown in Figure 1.

The greedy heuristic ENQUEUEBATCH, given in Algorithm 3, follows the given batch order, and uses the ENQUEUE in any order within a batch.

Algorithm 3: ENQUEUEBATCH(`set of batches` : $\mathcal{B}$)

```
while B ≠ ∅ do
    remove β from B
    while β ≠ ∅ do
        Pick and remove a job a from β
        ENQUEUE(S, a)
return S
```

We assume that we have $n = \alpha s$ jobs each requiring one of s resources, and each resource supplies for exactly α jobs. In other words, we have α batches of s jobs. Under these assumptions, we show that if all jobs have relatively similar processing times, i.e., if $\rho = \frac{p_{max}}{p_{min}} \leq \lfloor \frac{s-1}{m-1} \rfloor$, then ENQUEUEBATCH approximates the optimal schedule within a factor $1 + \frac{s-1}{n}$.

Theorem 4. *If $p_{max} \leq \lfloor \frac{s-1}{m-1} \rfloor p_{min}$ and if σ is the makespan of a schedule obtained by procedure* ENQUEUEBATCH*, then either σ is optimal, or $\sigma \leq \frac{L}{m} + (1 - \frac{1}{m}) p_{max}$.*

PROOF. We first prove the following proposition by induction on α (this proof is illustrated in Figure 7):

If there are α batches of s jobs, the schedule returned by ENQUEUEBATCH is such that $e_{max} - e_{min} \leq p_{max}$.

If there is no job at all, this property trivially holds.

Now we suppose that it is true for α batches and we show that it is also true for $\alpha + 1$. Let e_j^α be the end time e_{M_j} of machine M_j after inserting the α^{th} batch. Without loss of generality, we suppose $e_1^{\alpha+1} \leq \ldots \leq e_m^{\alpha+1}$, and we assume that the induction hypothesis is falsified at step $\alpha + 1$, i.e., $e_m^{\alpha+1} - e_1^{\alpha+1} > p_{max}$. Since $e_m^{\alpha+1} - e_1^{\alpha+1} > p_{max}$ and since no job is larger than p_{max}, it means that the penultimate job of machine M_m finishes later than the last job of machine M_1. Therefore, the penultimate and last jobs of machine M_m must require the same resource, hence belong to a different batch.

It follows that only one job of batch $\alpha + 1$ is on machine M_m. Hence $e_m^{\alpha+1} - p_{max} \leq e_m^\alpha$. Therefore, since by induction hypothesis $e_{\max}^\alpha - e_{\min}^\alpha \leq p_{max}$, the following inequalities hold:

$$\forall 1 \leq i < m,\ e_i^\alpha \geq e_m^{\alpha+1} - 2p_{max} \tag{1}$$

Moreover, since a single job of batch $\alpha + 1$ is allocated to machine M_m, the $s - 1$ other jobs must be distributed amongst machines M_1 to M_{m-1}.

If $\lfloor \frac{s-1}{m-1} \rfloor$ jobs or more have been allocated to machine M_1, we have $e_1^{\alpha+1} \geq e_1^\alpha + \lfloor \frac{s-1}{m-1} \rfloor p_{min}$. By applying inequality 1, we have $e_1^{\alpha+1} \geq e_m^{\alpha+1} - 2p_{max} + \lfloor \frac{s-1}{m-1} \rfloor p_{min}$, hence $e_1^{\alpha+1} \geq e_m^{\alpha+1} - p_{max}$, which would contradict the induction hypothesis. Therefore, at most $\lfloor \frac{s-1}{m-1} \rfloor - 1$ jobs have been allocated to machine M_1. In other words, among the s jobs of the batch, one has been assigned to

machine m and at most $\lfloor \frac{s-1}{m-1} \rfloor - 1$ to machine M_1, so there are $s - \lfloor \frac{s-1}{m-1} \rfloor$ jobs to distribute over the $m-2$ other machines. It follows that at least one machine will be allocated strictly more than $\lfloor \frac{s-1}{m-1} \rfloor$ jobs, because the following relation is a tautology:

$$s - \lfloor \tfrac{s-1}{m-1} \rfloor > \lfloor \tfrac{s-1}{m-1} \rfloor (m-2) \tag{2}$$

$$\Leftrightarrow \quad s > \lfloor \tfrac{s-1}{m-1} \rfloor (m-1) \tag{3}$$

$$\Leftarrow \quad s > s-1 \tag{4}$$

Now, let M_j be the machine that was allocated at least $\lfloor \frac{s-1}{m-1} \rfloor + 1$ jobs ($1 < j < m$). Consider the schedule when adding the $(\lfloor \frac{s-1}{m-1} \rfloor + 1)^{\text{th}}$ job of batch $\alpha + 1$ on machine M_j. Since this is not the first job of the batch to be allocated to machine M_j, it cannot require the same resource as the previous one. Therefore machine M_1 must finish later than the start of this $(\lfloor \frac{s-1}{m-1} \rfloor + 1)^{\text{th}}$ job on machine M_j. In other words, we have $e_1^{\alpha+1} \geq e_j^{\alpha} + \lfloor \frac{s-1}{m-1} \rfloor p_{min}$. By applying inequality 1 as above, we have $e_1^{\alpha+1} \geq e_m^{\alpha+1} - p_{max}$. Therefore, the assumption that the induction hypothesis does not lift to $\alpha + 1$ is contradicted.

Now, by Lemma 2 and the induction above, we have that the schedule returned by EnqueueBatch is dense and such that $e_{max} - e_{min} \leq p_{max}$. We can therefore apply Lemma 1 (with $\sigma = e_{max}$) and we have:

$$\sigma \leq \sigma^* + (1 - \frac{1}{m}) p_{max} \tag{5}$$

□

Corollary 3. *If* $p_{max} \leq \lfloor \frac{s-1}{m-1} \rfloor p_{min}$ *then* EnqueueBatch *approximates the optimal schedule with a ratio* $1 + \frac{s-1}{n}$.

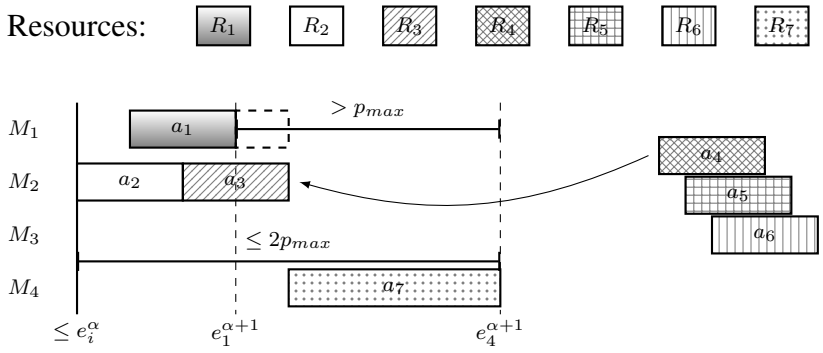

Figure 7: Illustration of the proof of Theorem 4 with $s = 7$ and $m = 4$ and $\frac{p_{max}}{p_{min}} \leq \lfloor \frac{s-1}{m-1} \rfloor = 2$. We start from the hypothesis that $e_4^{\alpha+1} - e_1^{\alpha+1} > p_{max}$. There is a single job of batch $\alpha + 1$ allocated to machine M_4, and at most $\lfloor \frac{s-1}{m-1} \rfloor - 1 = 1$ job allocated to machine M_1. The 5 remaining jobs must be allocated to 2 machines, hence one machine (say M_2) receives at least 3 jobs (all requiring distinct resources). When enqueing the last of these jobs, the machine M_1 must have been in process because it was not chosen. Therefore, $e_4^{\alpha+1} - e_1^{\alpha+1} \leq p_{max}$.

PROOF. If the makespan of the schedule produced by ENQUEUEBATCH is not optimal, then we can write from Theorem 4 and from $L \leq m\sigma^*$:

$$\sigma \leq \sigma^* + (1 - \frac{1}{m})p_{max} \tag{6}$$

Moreover, we have $\frac{L}{n} \geq p_{min}$ hence $\lfloor \frac{s-1}{m-1} \rfloor \frac{L}{n} \geq p_{max}$, that is $\lfloor \frac{s-1}{m-1} \rfloor \frac{m\sigma^*}{n} \geq p_{max}$ and therefore $\frac{m(s-1)}{(m-1)n}\sigma^* \geq p_{max}$. We inject this into inequality 6 to obtain: $\sigma \leq \sigma^* + \frac{(s-1)(m-1)\sigma^*}{n(m-1)} = (1 + \frac{s-1}{n})\sigma^*$, which concludes our proof.

□

If the ratio $\rho = \frac{p_{max}}{p_{min}}$ is strictly greater than $\lfloor \frac{s-1}{m-1} \rfloor$ then the gap between the makespan of a schedule returned by Algorithm 3 and the optimal makespan cannot be bounded by a function of p_{max}. We show that however small the gap between the ratio $\frac{p_{max}}{p_{min}}$ and $\lfloor \frac{s-1}{m-1} \rfloor$ is, there exist instances for which this gap is arbitrarily

large. Given an instance $\mathcal{I}$, that is, a pair (m, s) and a sequence of batches of jobs, let $\sigma_{\mathcal{I}}$ and $\sigma^*_{\mathcal{I}}$ denote, respectively, the makespan of a schedule returned by Algorithm 3 and the optimal makespan.

Theorem 5. *$\forall \epsilon, \gamma > 0$ and for any $m > 1$, there exists an instance $\mathcal{I}$ such that $\frac{p_{max}}{p_{min}} \leq \lfloor \frac{s-1}{m-1} \rfloor + \epsilon$ and $\sigma_{\mathcal{I}} > \sigma^*_{\mathcal{I}} + \gamma$.*

Proof. Let x be an integer such that $x(m-1) > \gamma$, and let $k = \lceil \frac{1}{\epsilon} \rceil$. We build an example (see Figure 8) with $2m-1$ resources and $(1+k)xm$ batches of $2m-1$ jobs. In the first xm batches, all jobs have a processing time 1. On each of the kxm subsequent batches, the jobs of the resources $R_2, \ldots, R_{2m-1}$ also have processing time 1, but the job requiring resource R_1 has processing time $2 + \frac{1}{k}$.

After inserting the first xm batches, the schedule is dense by Lemma 2. Moreover, since all processing times are equal and since m divides xm, the completion time on every machine is exactly $x(2m-1)$. Now, in the last kxm batches, the sum of the processing times of the jobs requiring R_1 is $kxm(2+\frac{1}{k})$. Therefore, the schedule returned by Algorithm 3 has a total makespan of at least $3xm-x+2kxm$.

However, we can build an optimal schedule as follows: on a machine M, we put only the jobs requiring the resource R_1. The makespan on this machine will thus be $xm+kxm(2+\frac{1}{k}) = 2xm+2kxm$. Then, on the $m-1$ remaining machines, we schedule the $xm(2m-2)+kxm(2m-2)$ remaining jobs. Since they all have a processing time of 1, it is easy to see that they can be scheduled within a makespan of $2xm + 2kxm$, which is therefore the overall optimal makespan σ^*.

We can verify that we indeed have $\sigma - \sigma^* = (3xm - x + 2kxm) - (2xm + 2kxm) = x(m-1) > \gamma$. □

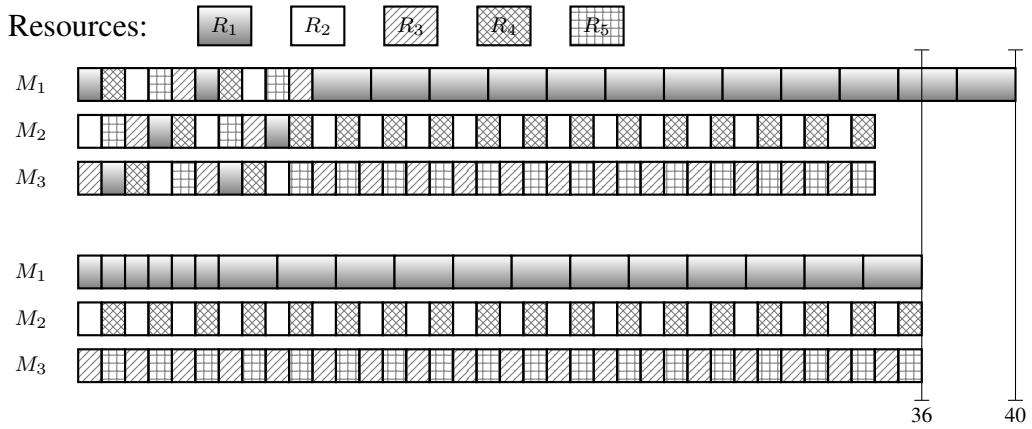

Figure 8: Illustration of the proof of Theorem 5, with $m = 3, \epsilon = \frac{1}{2}$ and $\gamma = 4$. There are 6 batches of 5 jobs with unit processing time, then 12 batches in which the job requiring resource R_1 is of length 2.5. The schedule returned by Algorithm 3 (top) has a makespan of 40, whereas the optimal schedule (bottom) has a makespan of 36.

In our application, there are 3 download channels ($m = 3$) and 5 memory banks ($s = 5$). It follows that Algorithm 3 is a $1 + \frac{4}{n}$-approximation if no file is more than twice larger than any other. Unfortunately, this condition is not guaranteed to hold in our application. Indeed, the hypothesis of our industrial partner was a maximum ratio of $\frac{1}{4}$ between minimum and maximum image sizes. However, if the compression rates have a normal distribution, the procedure ENQUEUEBATCH should often be efficient. Indeed if the standard deviation is of $\frac{5}{6}$ or less, we can expect that most of the images will have a size in the range $[\frac{5}{3}, \frac{10}{3}]$, which satisfies the condition of Corollary 3.

Algorithm	Condition	Absolute properties	Approx. ratios
ENQUEUE	none	$\sigma \leq \frac{L}{m}+(1-\frac{1}{m})\max_R L(R)$	$2-\frac{1}{m}$
MAXLOAD	none	$\sigma \leq \frac{L}{m}+(1-\frac{1}{m})p_{max}$ or $\sigma = \sigma^*$	$2-\frac{2}{m+1}$ $1+\frac{p_{max}}{p_{min}}\frac{m-1}{n}$
ENQUEUEBATCH	$\frac{p_{max}}{p_{min}} \leq \lfloor \frac{s-1}{m-1} \rfloor$	$\sigma \leq \frac{L}{m}+(1-\frac{1}{m})p_{max}$ or $\sigma = \sigma^*$	$1+\frac{s-1}{n}$

Table 1: Summary of the results obtained (note that for the results given in the "absolute properties" column, we always have $\frac{L}{m} \leq \sigma^*$).

5. Conclusion

We have introduced a heuristic insertion algorithm for parallel machine scheduling problems with additional unit resources which is a $(2-\frac{2}{m+1})$-approximation. Second, we have shown that if the maximum processing time p_{max} and minimum processing time p_{min} of a job are such that $p_{max} \leq \rho p_{min}$ for a given value of ρ, then the approximation factor tends toward 1 when the size of the problem grows.

Next, we have shown that a slightly different heuristic restricted to proceed along a predefined batch order is a $(2-\frac{1}{m})$-approximation. Moreover, if the ratio between maximum and minimum processing time ρ is bounded by $\lfloor \frac{s-1}{m-1} \rfloor$, then the approximation factor becomes $1+\frac{s-1}{n}$. All the results obtained are summarized in Table 1. They are important in a planning application for observation satellites.

Acknowledgments

The motivation for this work originates from a study supported through a grant from CNES and Astrium (contract 4500166027 "Planification Flexible"). We would especially like to thank Jean Jaubert (CNES), Pierre Blanc-Paques (Astrium) and Thierry Desmousceaux (Astrium).

References

Mohamed Bendraouche and Mourad Boudhar. Scheduling jobs on identical machines with agreement graph. *Computers & Operations Research*, 39(2):382 – 390, 2012.

Jacek Blazewicz, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

Emrah B. Edis, Ceyda Oguz, and Irem Ozkarahan. Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, 230:449–463, 2013.

Guy Even, Magnús M. Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2): 199–224, 2009.

Michael R. Garey and David S. Johnson. "Strong" NP-Completeness Results: Motivation, Examples and Implications. *J. ACM*, 25(3):499–508, 1978.

Alexander Grigoriev and Marc Uetz. Scheduling jobs with time-resource tradeoff via nonlinear programming. *Discrete Optimization*, 6(4):414–419, 2009.

Hans Kellerer and Vitaly A. Strusevich. Scheduling problems for parallel dedicated machines under multiple resource constraints. *Discrete Appl. Math.*, 133 (1-3):45–68, November 2003.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.

Cédric Pralet, Gérard Verfaillie, Adrien Maillard, Emmanuel Hebrard, Nicolas Jozefowiez, Marie-José Huguet, Thierry Desmousceaux, Pierre Blanc-Paques, and Jean Jaubert. Satellite Data Download Management with Uncertainty about the Generated Volumes. In *ICAPS (application track)*, 2014.