Mathematical Methods of Operations Research manuscript No.
(will be inserted by the editor)

Batching Identical Jobs

Philippe Baptiste

CNRS, UMR 6599 HeuDiaSyC, Univ. de Tech. de Compiègne, F-60205 Compiègne
(Philippe.Baptiste@hds.utc.fr)

The date of receipt and acceptance will be inserted by the editor

Abstract We study the problems of scheduling jobs, with different release dates and equal processing times, on two types of batching machines. All jobs of the same batch start and are completed simultaneously. On a serial batching machine, the length of a batch equals the sum of the processing times of its jobs and, when a new batch starts, a constant setup time s occurs. On a parallel batching machine, there are at most b jobs per batch and the length of a batch is the largest processing time of its jobs. We show that in both environments, for a large class of so called "ordered" objective functions, the problems are polynomially solvable by dynamic programming. This allows us to derive that the problems where the objective is to minimize the weighted number of late jobs, or the weighted flow time, or the total tardiness, or the maximal tardiness are polynomial. In other words, we show that $1|p\text{-}batch, b < n, r_i, p_i = p|F$ and $1|s\text{-}batch, r_i, p_i = p|F$, are polynomial for $F \in \{\sum w_i U_i, \sum w_i C_i, \sum T_i, T_{\max}\}$. The complexity status of these problems was unknown before.
Keywords: Scheduling Theory, Complexity, Dynamic Programming

1 Introduction

In this paper, we study the situation where n jobs $\{J_1, \ldots, J_n\}$ have to be scheduled on a batching machine. Each job J_i is described by a processing time p_i, a release date r_i, and eventually by a due date d_i and a weight w_i (all numerical values used in this paper being integer). Jobs cannot start before their release dates and all jobs of the same batch start and are completed simultaneously, *i.e.*, at the starting time (respectively at the completion time) of the batch. Two types of batching machines are studied.

- On a serial batching machine, the length of a batch equals the sum of the processing times of its jobs. When a new batch starts, a constant setup time s occurs.
- On a parallel batching machine, there are at most b jobs per batch and the length of a batch is the largest processing time of its jobs. Two situations are often distinguished. The bounded case with $b < n$ and the unbounded case with $b = n$. In this environment, no setup is assumed. However, a constant setup time s could be easily taken into account by increasing of s the processing time of each job.

Following the notation of [7], these problems are denoted respectively by $1|s\text{-}batch, r_i|F$, $1|p\text{-}batch, b < n, r_i|F$ (bounded case) and $1|p\text{-}batch, r_i|F$ (unbounded case). We refer to [1], [6], [7], [14], [16], [17], [19] for extended reviews on pure batch scheduling problems and on extensions (*e.g.* scheduling group of jobs with group-dependent setup times, jobs requiring several machines throughout their execution, *etc.*). Complexity results for problems with identical release dates are summarized in Table 1. The problems $1|r_i|F$ are NP-Hard for $L_{\max}, \sum C_i, \sum T_i$, hence the corresponding batching problems $1|s\text{-}batch, r_i|F$ and $1|p\text{-}batch, r_i|F$ are also NP-Hard.

Problem	Complexity	References
$1\|p\text{-}batch, b < n\|C_{\max}$	$O(n \log n)$	[6]
$1\|s\text{-}batch\|L_{\max}$	$O(n^2)$	[19]
$1\|p\text{-}batch\|L_{\max}$	$O(n^2)$	[6]
$1\|p\text{-}batch, b < n\|L_{\max}$	Unary NP-Hard	[6]
$1\|s\text{-}batch\|\sum U_i$	$O(n^3)$	[8]
$1\|s\text{-}batch\|\sum w_i U_i$	binary NP-Hard	[8]
$1\|s\text{-}batch, p_i = p\|\sum w_i U_i$	$O(n^4)$	[15]
$1\|p\text{-}batch\|\sum U_i$	$O(n^3)$	[6]
$1\|p\text{-}batch\|\sum w_i U_i$	binary NP-Hard	[6]
$1\|p\text{-}batch, b < n\|\sum U_i$	Unary NP-Hard	[6]
$1\|s\text{-}batch\|\sum C_i$	$O(n \log n)$	[11]
$1\|s\text{-}batch, p_i = p\|\sum C_i$	polynomial in $\log p, \log s, \log n$	[18]
$1\|s\text{-}batch\|\sum w_i C_i$	unary NP-Hard	[1]
$1\|s\text{-}batch, p_i = p\|\sum w_i C_i$	$O(n \log n)$	[1]
$1\|p\text{-}batch\|\sum w_i C_i$	$O(n \log n)$	[6]
$1\|p\text{-}batch, b < n\|\sum C_i$	$O(n^{b(b-1)})$	[6]
$1\|s\text{-}batch\|\sum T_i$	binary NP-Hard	[13]
$1\|p\text{-}batch\|\sum w_i T_i$	binary NP-Hard	[6]
$1\|p\text{-}batch, b < n\|\sum T_i$	unary NP-Hard	[6]

Table 1 Overview of the complexity results

These results leave open the status of most of the problems with arbitrary release dates and equal processing times. In this paper, we show that serial and parallel batching problems can be solved polynomially for a large class of so called **ordered objective functions**.

A strict subclass of ordered objective functions has been studied in [4], where it has been shown that scheduling identical jobs on a fixed number of identical machines can be done in polynomial time for $\sum w_i C_i$ and $\sum T_i$. The class of ordered objective functions provides a unified framework to study a larger variety of functions.

Definition 1 *F is an ordered objective function if and only if :*

1. *F is a sum function, i.e., $F = \sum f_i(C_i)$,*
2. *F is regular, i.e., $\forall i, f_i$ is non-decreasing,*
3. *f_i is constant after a time point $\boldsymbol{\delta_i}$, i.e., $\forall t \geq \delta_i, f_i(t) = f_i(\delta_i)$,*
4. *$\forall i < j$, $\delta_i \leq \delta_j$ and $t \mapsto (f_i - f_j)(t)$ is non-decreasing over $[0, \delta_i]$.*

The last condition of the definition ensures that the function has a "Monge"-like property. For such functions, it is known (*e.g.* [5]) that many unit execution time scheduling problems are polynomially solvable.

It is easy to verify that the weighted number of late jobs, $\sum w_i U_i$, is an ordered objective function. On the contrary, $\sum w_i C_i$ and $\sum T_i$ are not ordered objective functions. However, conditions 1 and 2 of Definition 1 hold for these functions and jobs can be renumbered to meet condition 4.

We show how a function like $\sum w_i C_i$ or $\sum T_i$ can be modified, without changing the optimum value, to become an ordered objective function: Consider a large time point T and alter the functions f_i after T so that $\forall t \geq T, f_i(t) = M$, where M is another large value. If T and M are large enough, the optimum of the modified problem is also the optimum of the original one. Moreover, the modified functions are ordered objective functions.

¿From now on, we restrict our study to ordered objective functions. Jobs can be renumbered so that $\boldsymbol{\delta_1 \leq \cdots \leq \delta_n}$ and $\boldsymbol{\forall i \leq j, t \mapsto (f_i - f_j)(t)}$ **is non-decreasing** over $[0, \delta_i]$. By analogy with due date scheduling, we say that a job is late when it is completed after δ_i and that it is on-time otherwise. Notice that a late job can be scheduled arbitrary late.

The paper is organized as follows. Several dominance properties are stated in §2. The dynamic programming algorithms for serial and parallel problems are described in §3 and §4. Finally in §5 we show that our approach can be extended to handle $T_{\max}$ and we draw some conclusions.

2 Dominance Properties

We first define two sets of time points at which batches start on active schedules. We then study a dominance related to ordered objective functions and finally we show that both dominances can be combined.

2.1 Starting Times

As shown in [3], [4], [9], [10], [12], one of the reasons why it is easy to schedule equal length jobs is that there are few possible starting times. Indeed, active

schedules are dominant and thus starting times are equal to a release date modulo p. This idea can be extended to batching problems.

Serial Batching Ordered objective functions are regular, hence active schedules are dominant. Consequently, each batch starts either at the release date of one of its jobs or immediately after the completion time of another batch plus the setup time s. Hence we can assume that batches start and end in S, where

$$\mathsf{S} = \{r_\lambda + \mu p + \nu s, \lambda \in \{1, \cdots, n\}, \mu \in \{0, \cdots, n\}, \nu \in \{0, \cdots, n\}\} \quad (1)$$

Parallel Batching Because jobs have the same processing time p we can assume that the length of a batch is p. On top of that, active schedules are dominant so, we can also assume that batches start and end in P, where

$$\mathsf{P} = \{r_\lambda + \mu p, \lambda \in \{1, \cdots, n\}, \mu \in \{0, \cdots, n\}\} \quad (2)$$

Notice that $|\mathsf{S}| = O(n^3)$ and $|\mathsf{P}| = O(n^2)$.

2.2 Ordered Objective Functions

The following proposition holds both for serial and parallel problems.

Proposition 1 *Any feasible schedule can be transformed in a "better" one where for any pair of on-time jobs $J_u, J_v (u < v)$, being executed in batches starting respectively at t_u and t_v, either $t_u \leq t_v$ or $t_v < r_u$.*

Proof Sketch. Let (u, v) be the smallest vector, according to the lexicographical order, such that $t_u > t_v$ and $t_v \geq r_u$. The jobs J_u and J_v may be exchanged and the value of the objective function is not increased since it is an ordered objective function. Moreover, it is easy to see that the smallest vector (u', v') that does not satisfy the condition on the new schedule is such that $(u', v') > (u, v)$. Repeated exchanges can be used to replace any schedule by a schedule which satisfies the condition for all u, v (the vector (u, v) increases at each step, hence the number of exchanges is finite).

2.3 Combining All Dominance Properties

Now consider an optimal and active schedule and apply the exchanges of Proposition 1. The resulting schedule is still optimal and the set of starting times is kept the same. Hence we can combine both dominance criteria, *i.e.*, schedules such that

- batches start and are completed either in S for serial problems, or in P for parallel problems
- and such that for any pair of jobs $J_u, J_v (u < v)$, being executed in batches starting respectively at t_u and t_v, either $t_u \leq t_v$ or $t_v < r_u$,

are dominant. In the following such schedules will be referred to as **Serial-Dominant** or **Parallel-Dominant** schedules.

3 Dynamic Programming for the Serial Problem

The dynamic program relies on the notion of **partial** batch. Usually, once the starting time t and the completion time $t + p\nu$ of a batch is known, it is easy to compute that $(t + p\nu - t)/p = \nu$ jobs are in the batch (because the batch is full). In a partial batch, there may be some hole and we only know that **at most** $(t + p\nu - t)/p$ jobs are in the batch. The batch is therefore "partial" since some additional jobs could be *a priori* added. In the following, the maximal number of jobs in a partial batch is denoted by the symbol ν, while μ denotes the number of jobs actually scheduled.

Before defining the variables of the Dynamic Program, let us introduce the set of jobs $U_k(t_l, t_r)$. For any integer $k \leq n$ and for any time points $t_l \leq t_r$, $U_k(t_l, t_r)$ is the set of jobs whose index is lower than or equal to k and whose release date is in the interval $(t_l, t_r]$.

$$U_k(t_l, t_r) = \{J_i | i \leq k, r_i \in (t_l, t_r]\} \tag{3}$$

3.1 Variables of the Dynamic Program

The dynamic search is controlled by 5 parameters $k, t_l, t_r, \nu_l, \nu_r$ and μ_r. Each combination of these parameters defines a sub-problem involving the jobs in $U_k(t_l, t_r)$. As explained below, the objective of the sub-problem is to minimize $\sum_{J_i \in U_k(t_l,t_r)} f_i(C_i)$ under some constraints.

Definition 2 *A schedule $\mathcal{K}$ is Serial-Dominant for $(k, t_l, t_r, \nu_l, \nu_r, \mu_r)$ iff*

- *$\mathcal{K}$ is Serial-Dominant,*
- *jobs in $U_k(t_l, t_r)$ are either late or are completed before or at $t_r + p\nu_r$,*
- *a partial batch, containing 0 job of $U_k(t_l, t_r)$, starts at t_l and is completed at $t_l + p\nu_l$,*
- *a partial batch, containing μ_r jobs of $U_k(t_l, t_r)$, starts at t_r and is completed at $t_r + p\nu_r$.*

Now we can define the variables of the dynamic program.

Definition 3 *$S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$ is the minimal value taken by the function*

$$\sum_{J_i \in U_k(t_l,t_r)} f_i(C_i) \tag{4}$$

over the Serial-Dominant schedules for $(t_l, t_r, \nu_l, \nu_r, \mu_r)$. If there is no such schedule, $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) = +\infty$.

Given this definition $S_0(t_l, t_r, \nu_l, \nu_r, \mu_r)$ equals 0 if $t_l + p\nu_l + s \leq t_r$ and $+\infty$ otherwise.

To get an intuitive picture of the decomposition scheme, assume that $J_k \in U_k(t_l, t_r)$ is on-time (if it is not, take into account the "late" cost $f_k(\delta_k)$ and set k to $k-1$) and consider the time point t at which the batch containing J_k starts on $\mathcal{K}$ (*cf.* Figure 1).

- If $t = t_r$, *i.e.*, if J_k is put in the right partial batch, then we take into account the cost $f_k(t_r + p\nu_r)$ associated to J_k and we solve the sub-problem defined by $(k-1, t_l, t_r, \nu_l, \nu_r, \mu_r - 1)$.
- If $t < t_r$, then J_k is scheduled in an in-between batch starting at t and containing ν jobs. Thanks to Proposition 1, on-time jobs of $U_{k-1}(t_l, t)$ are all completed before or at the completion time of this batch. On-time jobs of $U_{k-1}(t, t_r)$ are completed after this batch. Hence, we have a left sub-problem defined by $(k-1, t_l, t, \nu_l, \nu, \nu - 1)$ and a right one defined by $(k-1, t, t_r, \nu, \nu_r, \mu_r)$

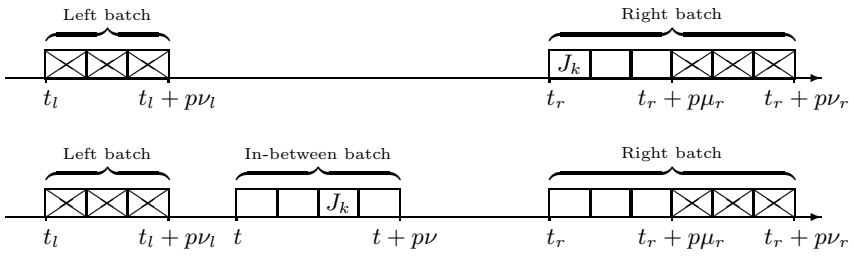

Fig. 1 Two ways to schedule J_k: Either in the right partial batch or in an in-between batch (J_k can also be late). Ticked boxes in a batch indicate that it is only partially available for the jobs of $U_k(t_l, t_r)$.

3.2 Fundamental Recursion Formula

We define three values L, R, I. We will see that they are the costs of optimal schedules where J_k is in the **L**eft, in the **R**ight or in the **I**n-between batch.

Definition 4 *L equals $S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r) + f_k(\delta_k)$*

Definition 5 *R equals $f_k(t_r + p\nu_r) + S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r - 1)$ if $\mu_r > 0$ and $+\infty$ otherwise.*

Definition 6 *I is the minimum of*

$$S_{k-1}(t_l, t, \nu_l, \nu, \nu - 1) + f_k(t + p\nu) + S_{k-1}(t, t_r, \nu, \nu_r, \mu_r) \tag{5}$$

under the constraints

$$\begin{cases} \nu \in \{1, \cdots, n\} \\ t \in \mathsf{S} \\ r_k \leq t \leq \delta_k - p\nu \\ t_l + p\nu_l + s \leq t \leq t_r - p\nu - s \end{cases} \tag{6}$$

If there are no such t and ν, $I = +\infty$.

The following propositions lead to Theorem 1 that provides the fundamental recursion formula of the dynamic program.

Proposition 2 *If* $J_k \in U_k(t_l, t_r)$ *then* $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \leq L$.

Proof Assume that L takes a finite value and let $\mathcal{R}$ be the schedule that realizes it. $\mathcal{R}$ is Serial-Dominant for $(k-1, t_l, t_r, \nu_l, \nu_r, \mu_r)$ and also for $(k, t_l, t_r, \nu_l, \nu_r, \mu_r)$ (J_k is late). The additional cost is exactly $f_k(\delta_k)$.

Proposition 3 *If* $J_k \in U_k(t_l, t_r)$ *then* $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \leq R$.

Proof Assume that R takes a finite value and let $\mathcal{Q}$ be the schedule that realizes $S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r - 1)$. Let $\mathcal{Q}'$ be the schedule obtained by adding the job J_k in the right batch (it can be added because it is released before the starting time of the batch since $J_k \in U_k(t_l, t_r) \Rightarrow r_k \leq t_r$). On the new schedule, at most $\mu_r - 1 + 1 = \mu_r$ jobs of $U_k(t_l, t_r)$ are scheduled in the right batch. It is easy to verify that $\mathcal{Q}'$ is a Serial-Dominant schedule for $(k, t_l, t_r, \nu_l, \nu_r, \mu_r)$. The additional cost is $f_k(t_r + p\nu_r)$. Hence $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \leq R$

Proposition 4 *If* $J_k \in U_k(t_l, t_r)$ *then* $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \leq I$.

Proof Assume that I takes a finite value and let t and ν be the values that realize 5 and meet the constraints 6. Let $\mathcal{K}_l$ and $\mathcal{K}_r$ be the schedules that realize respectively $S_{k-1}(t_l, t, \nu_l, \nu, \nu - 1)$ and $S_{k-1}(t, t_r, \nu, \nu_r, \mu_r)$. We build a schedule $\mathcal{K}'$ by adding the schedules $\mathcal{K}_l$ and $\mathcal{K}_r$ and by adding J_k in the right batch of $\mathcal{K}_l$. Since the set $J_k(t_r, t_l)$ is the direct sum of $J_{k-1}(t_r, t)$ plus $J_{k-1}(t, t_l)$ plus $\{J_k\}$, all jobs of $J_k(t_r, t_l)$ are scheduled exactly once on $\mathcal{K}'$. J_k can be added to the right batch because no more than $\nu - 1$ jobs are scheduled in this batch on $\mathcal{K}_l$. At least s time units elapse between batches (because $t_l + p\nu_l + s \leq t \leq t_r - p\nu - s$) and it is easy to verify that the in-between batch starts after the release dates of its jobs and before their due dates. We have proven that $\mathcal{K}'$ is feasible.
We claim that $\mathcal{K}'$ is a Serial-Dominant schedule for $(k, t_l, t_r, \nu_l, \nu_r, \mu_r)$. We only prove that "for any pair of jobs $J_u, J_v (u < v)$, being executed in batches starting respectively at t_u and t_v, either $t_u \leq t_v$ or $t_v < r_u$". The verification of all other conditions is easy and is left to the reader. If $v = k$ and if $t_v < t_u$ then J_u belongs to the sub-schedule $\mathcal{K}_r$ and thus, $J_u \in U_{k-1}(t, t_r)$, which leads to $t = t_v < r_u$ and the condition holds. Now assume that $v < k$. If both jobs belong to the same sub-schedule either $\mathcal{K}_r$ or $\mathcal{K}_l$, the condition holds because they are Serial-Dominant. Now assume that it is not the case and that $t_v < t_u$. We know that $t \in [t_v, t_u)$ and since $J_u \in U_{k-1}(t, t_r)$, $t_v \leq t < r_u$.
On $\mathcal{K}'$, the batch of J_k is completed at $t + p\nu$, hence the total cost of $\mathcal{K}'$ is $S_{k-1}(t_l, t, \nu_l, \nu, \nu - 1) + f_k(t + p\nu) + S_{k-1}(t, t_r, \nu, \nu_r, \mu_r) = I$. As a consequence, $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \leq I$.

Proposition 5 *If* $J_k \in U_k(t_l, t_r)$ *then* $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \geq \min(I, R, L)$.

Proof We can assume that $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$ takes a finite value otherwise the proposition obviously holds. Let $\mathcal{W}$ be the schedule that realizes $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$. If J_k is not scheduled on $\mathcal{W}$, *i.e.*, it is late, then $\mathcal{W}$ is also Serial-Dominant for $(k-1, t_l, t_r, \nu_l, \nu_r, \mu_r)$ and the cost is decreased of $f_k(\delta_k)$ (because J_k is not taken into account). Hence, $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \geq L$. Now assume that J_k is scheduled on-time on $\mathcal{W}$ and let t be the time point at which the batch containing J_k starts and let ν be the size of this batch.

- If J_k is scheduled in the last batch, *i.e.*, if $t = t_l$, then remove it. It is easy to verify that the resulting schedule is Serial-Dominant for $(k-1, t_l, t_r, \nu_l, \nu_r, \mu_r - 1)$ and its cost is $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) - f_k(t_r + p\nu_r)$. Hence, $S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r - 1) \leq S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) - f_k(t_r + p\nu_r)$. Consequently, $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \geq R$.
- If J_k is not scheduled in the last batch then, because of the setup time constraint, it must start after or at $t_l + p\nu_l + s$ and it must be completed before or at $t_r - s$. Hence, $t \in [t_l + p\nu_l + s, t_r - p\nu - s]$. Moreover, the batch starts before the release date of J_k and is completed before its due date. Finally, the batch starts at a time point of S because $\mathcal{W}$ is Serial-Dominant. Hence, constraints 6 are met. Let now $\mathcal{W}_l$ be the sub-schedule obtained from $\mathcal{W}$ by removing the jobs that are scheduled in batches starting strictly after t and by removing J_k. This sub-schedule is Serial-Dominant for $(k-1, t_l, t, \nu_l, \nu, \nu-1)$ and thus its cost is greater than or equal to $S_{k-1}(t_l, t, \nu_l, \nu, 1)$. Similarly, let $\mathcal{W}_r$ be the sub-schedule obtained from $\mathcal{W}$ by removing the jobs that are scheduled in a batch starting before or at t. $\mathcal{W}_r$ is Serial-Dominant for $(k-1, t, t_r, \nu, \nu_r, \mu_r)$ and its cost is greater than or equal to $S_{k-1}(t, t_r, \nu, \nu_r, \mu_r)$. Finally, notice that the cost of $\mathcal{W}$, $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$, is the sum of the costs of $\mathcal{W}_l$ and $\mathcal{W}_r$ plus the cost of scheduling J_k in the batch starting at t, *i.e.*, $f_k(t + p\nu)$. Altogether, this leads to $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \geq I$.

Hence, $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$ is greater than or equal to either I, R or L.

Theorem 1 *If* $J_k \in U_k(t_l, t_r)$ *then* $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) = \min(I, R, L)$. *Otherwise,* $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) = S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r)$.

Proof If $J_k \notin U_k(t_l, t_r)$ then $U_k(t_l, t_r) = U_{k-1}(t_l, t_r)$ and thus the theorem obviously holds. If $J_k \in U_k(t_l, t_r)$ then the result comes immediately from Propositions 3, 4, 2 and 5.

3.3 An $O(n^{14})$ Algorithm

There is an optimal schedule that is serial-dominant and it comes directly from the definition of $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$ that the optimum is met for

$$\begin{cases} k & \leftarrow n \\ t_l & \leftarrow \min(\mathsf{S}) - s - 1 \\ \nu_l & \leftarrow 0 \\ t_r & \leftarrow \max(\mathsf{S}) + s \\ \nu_r & \leftarrow 0 \\ \mu_r & \leftarrow 0 \end{cases} \tag{7}$$

Notice that if $s > 0$ then t_l can be set to $\min(\mathsf{S}) - s$ instead of $\min(\mathsf{S}) - s - 1$ because $U_n(\min(\mathsf{S}) - s, \max(\mathsf{S}) + s) = \{J_1, \cdots, J_n\}$; which is not the case when $s = 0$.

Algorithm 1 Computation of the values $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$

for $k \leftarrow 1$ to n **do**
  for $\nu_r \leftarrow 0$ to n **do**
    for $\nu_l \leftarrow 0$ to n **do**
      for $\mu_r \leftarrow 0$ to ν_r **do**
        for $t_r \in \mathsf{S}$ taken in increasing order **do**
          for $t_l \in \mathsf{S}$ $(t_l \leq t_r)$ taken in increasing order **do**
            $R \leftarrow +\infty, I \leftarrow +\infty, L \leftarrow +\infty$
            if $\mu_r > 0$ **then**
              $R \leftarrow f_k(t_r + p\nu_r) + S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r - 1)$
            end if
            for $\nu \leftarrow 1$ to n **do**
              for $t \in \mathsf{S} \cap [t_l + p\nu_l + s, t_r - p\nu - s] \cap [r_k, \delta_k - p\nu]$ **do**
                $I \leftarrow \min(I, S_{k-1}(t_l, t, \nu_l, \nu, \nu - 1) + f_k(t + p\nu) + S_{k-1}(t, t_r, \nu, \nu_r, \mu_r))$
              end for
            end for
            $L \leftarrow S_{k-1}(t_l, t_r, \nu_l, \nu_r, \mu_r) + f_k(\delta_k)$
            $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r) \leftarrow \min(I, R, L)$
          end for
        end for
      end for
    end for
  end for
end for

Thanks to Theorem 1, we have a straight dynamic programming algorithm to reach the optimum. The relevant values for t_l and t_r are those in S plus the special values $\min(\mathsf{S}) - s - 1$ and $\max(\mathsf{S}) + s$ that are useful to start the dynamic search (to simplify the pseudo-code, these special values have been omitted in Algorithm 1). The relevant values for k, ν_l, ν_r and μ_r are

$\{0, \cdots, n\}$. Finally, the values of $S_k(t_l, t_r, \nu_l, \nu_r, \mu_r)$ are stored in a multi-dimensional array of size $O(n^{10})$ (n possible values for k, ν_l, ν_r, μ_r, and n^3 possible values both for t_l and t_r).

In the initialization phase, $S_0(t_l, t_r, \nu_l, \nu_r, \mu_r)$ is set to 0 if $t_l + p\nu_l + s \leq t_r$ and to a very large value otherwise. The initialization phase runs in $O(n^9)$ (n possible values for ν_l, ν_r, μ_r, n^3 possible values both for t_l and t_r).

We then iterate on all possible values of the parameters to reach the optimum (*cf.* Algorithm 1). Each time, a minimum over $O(n^4)$ terms ($O(n^3)$ for t and $O(n)$ for ν) is computed. This leads to an overall time complexity of $O(n^{14})$. A rough analysis of the space complexity leads to an $O(n^{10})$ bound but since, at each step of the outer loop on k, one only needs the values of S computed at the previous step $(k-1)$, the algorithm can be implemented with 2 arrays of $O(n^9)$ size: one for the current values of S and one for the previous values of S. (To build the optimal schedule, all values of S have to be kept; hence the initial $O(n^{10})$ bound applies.)

4 Dynamic Programming for Parallel Batching

4.1 Variables of the Dynamic Program

The dynamic search is controlled by 4 parameters k, t_l, t_r and μ_r. Each combination of these parameters defines a sub-problem involving the jobs in $U_k(t_l, t_r)$. The objective is to minimize $\sum_{J_i \in U_k(t_l, t_r)} f_i(C_i)$ under some constraints.

Definition 7 *A schedule $\mathcal{K}$ is Parallel-Dominant for (k, t_l, t_r, μ_r) iff*

- *$\mathcal{K}$ is Parallel-Dominant,*
- *jobs in $U_k(t_l, t_r)$ do not start before $t_l + p$,*
- *jobs in $U_k(t_l, t_r)$ are either late or are completed before or at $t_r + p$,*
- *a partial batch, containing μ_r jobs of $U_k(t_l, t_r)$, starts at t_r.*

Now we can define the variables of the dynamic program.

Definition 8 *$P_k(t_l, t_r, \mu_r)$ is the minimal value taken by the function*

$$\sum_{J_i \in U_k(t_l, t_r)} f_i(C_i) \tag{8}$$

over the Parallel-Dominant schedules of $U_k(t_l, t_r)$. , onek

- If $t = t_r$, *i.e.*, if J_k is put in the right batch, we have to solve the same problem with $k \leftarrow k-1, t_l \leftarrow t_l, t_r \leftarrow t_r$ and $\mu_r \leftarrow \mu_r - 1$.
- If $t < t_r$, then an in-between batch is created. Thanks to Proposition 1 jobs of $U_{k-1}(t_l, t)$ are either late or are completed before or at the completion time of this batch. Jobs in $U_{k-1}(t, t_r)$ are completed after this batch. Hence, we have two sub-problems. A left one with $(k-1, t_l, t, b-1)$ and a right one with $(k-1, t, t_r, \mu_r)$.

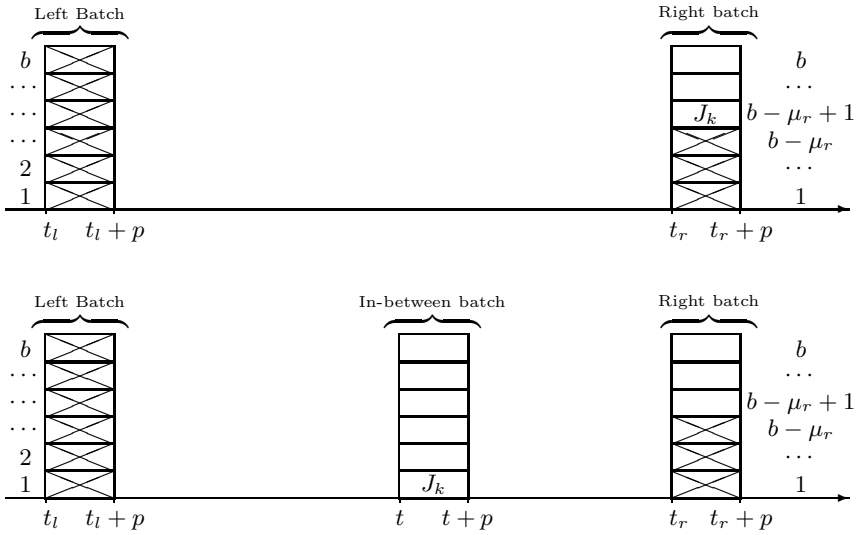

Fig. 2 Two ways to schedule J_k: Either in the right batch or in an in-between batch (J_k can also be late). Ticked boxes in a batch indicate that it is only partially available for the jobs of $U_k(t_l, t_r)$.

4.2 Fundamental Recursion Formula

Let us redefine the values of R, I, L.

Definition 9 *L equals $P_{k-1}(t_l, t_r, \mu_r) + f_k(\delta_k)$*

Definition 10 *R equals $f_k(t_r + p) + P_{k-1}(t_l, t_r, \mu_r - 1)$ if $\mu_r > 0$ and $+\infty$ otherwise.*

Definition 11 *I is the minimum of*

$$P_{k-1}(t_l, t, b-1) + f_k(t+p) + P_{k-1}(t, t_r, \mu_r) \tag{9}$$

under the constraints

$$\begin{cases} t \in \mathsf{P} \\ r_k \leq t \leq \delta_k - p \\ t_l + p \leq t \leq t_r - p \end{cases} \tag{10}$$

If there is no such $t, I = +\infty$.

We are ready to formulate the fundamental theorem.

Theorem 2 *If* $J_k \in U_k(t_l, t_r)$ *then* $P_k(t_l, t_r, \mu_r) = \min(I, R, L)$. *Otherwise,* $P_k(t_l, t_r, \mu_r) = P_{k-1}(t_l, t_r, \mu_r)$.

Proof See proof of Theorem 1.

Algorithm 2 Computation of the values $P_k(t_l, t_r, \mu_r)$

```
for k ← 1 to n do
  for μ_r ← 0 to b do
    for t_r ∈ P taken in increasing order do
      for t_l ∈ P (t_l ≤ t_r) taken in increasing order do
        R ← +∞, I ← +∞, L ← +∞
        if μ_r > 0 then
          R ← f_k(t_r + p) + P_{k-1}(t_l, t_r, μ_r − 1)
        end if
        for t ∈ P ∩ [t_l + p, t_r − p] ∩ [r_k, δ_k − p] do
          I ← min(I, P_{k-1}(t_l, t, b − 1) + f_k(t + p) + P_{k-1}(t, t_r, μ_r))
        end for
        L ← P_{k-1}(t_l, t_r, μ_r) + f_k(δ_k)
        P_k(t_l, t_r, μ_r) ← min(I, R, L)
      end for
    end for
  end for
end for
```

4.3 An $O(n^8)$ Algorithm

There is an optimal Parallel-Dominant schedule and it comes directly from the definition of $P_k(t_l, t_r, \mu_r)$ that the optimum is met for

$$\begin{cases} k & \leftarrow n \\ t_l & \leftarrow \min(\mathsf{P}) - p \\ t_r & \leftarrow \max(\mathsf{P}) + p \\ \mu_r & \leftarrow 0 \end{cases} \tag{11}$$

Thanks to Theorem 2, we have a straight dynamic programming algorithm to reach the optimum. The relevant values for t_l and t_r are those in P plus the special values $\min(\mathsf{P}) - p$ and $\max(\mathsf{P}) + p$. The relevant values for k

and μ_r are in $\{0, \cdots, n\}$. The values of $P_k(t_l, t_r, \mu_r)$ are stored in a multi-dimensional array of size $O(n^6)$.

In the initialization phase, $P_0(t_l, t_r, \mu_r)$ is set to 0 if $t_l + p \leq t_r$ and to a very large value otherwise. It runs in $O(n^5)$. We then iterate on all possible values of the parameters to reach the optimum (*cf.* Algorithm 2). For each value of the parameters, a minimum over t ($O(n^2)$ terms) is computed. This leads to an overall time complexity of $O(n^8)$. A rough analysis of the space complexity leads to an $O(n^6)$ bound but, as for the serial problem, the algorithm can be implemented with 2 arrays of $O(n^5)$ size.

5 Conclusion

We have introduced the class of ordered objective function, a class of functions for which batching identical jobs is easy. $\sum w_i U_i$, $\sum w_i C_i$ or $\sum T_i$ are ordered objective functions or can be transformed into ordered objective functions. Hence, we have shown that the problems

- $1|s\text{-}batch, r_i, p_i = p| \sum w_i U_i$,
- $1|s\text{-}batch, r_i, p_i = p| \sum w_i C_i$,
- $1|s\text{-}batch, r_i, p_i = p| \sum T_i$,
- $1|p\text{-}batch, b < n, r_i, p_i = p| \sum w_i U_i$,
- $1|p\text{-}batch, b < n, r_i, p_i = p| \sum w_i C_i$,
- $1|p\text{-}batch, b < n, r_i, p_i = p| \sum T_i$,

are solvable in polynomial time by dynamic programming. These problems were open. Of course, unbounded parallel batching problems are also solvable by the same dynamic programming scheme.

It is also interesting to notice that both problems remain solvable for $C_{\max}, L_{\max}$ and $T_{\max}$. We give the underlying idea for $T_{\max}$ only: Left-shifted schedules are dominant for this regular criteria. Thus the set of possible completion times is bounded. Since the optimum T^* equals either 0 or a value in the set of possible completion time minus a due date, the number of possible values for T^* is $|\mathsf{S}| * n = O(n^4)$ for the serial problem or $|\mathsf{P}| * n = O(n^3)$ for the parallel problem. We try for each possible value T of T^* to solve a derived problem where the due date of J_i is set to $d_i + T$ and where the objective is to minimize the number of late jobs. If the optimum is greater than 0 then $T < T^*$ otherwise $T \geq T^*$. A dichotomic search on the set of possible values for T^* is performed . This leads to an overall time complexity of $O(n^{14} \log n^4) = O(n^{14} \log n)$ for $1|s\text{-}batch, r_i, p_i = p|T_{\max}$ and of $O(n^8 \log n)$ for $1|p\text{-}batch, b < n, r_i, p_i = p|T_{\max}$. A slightly better time complexity could be reached by some specific dynamic programming algorithm, this is however out of the scope of the paper.

To conclude we would like to point out that, for the criteria $\sum w_i T_i$, batching identical jobs is still an open problem. The fact that even the simple single machine problem $1|r_i, p_i = p| \sum w_i T_i$ is open, while all other ones are closed, makes us think that this batching problem is at the very close border line of NP-hardness.

Acknowledgments

The author would like to thank Jacques Carlier and the anonymous referees for their numerous comments which hopefully led to significant improvements of this paper. Special thanks to Sigrid Knust for insightful discussions on batching problems.

References

1. S. Albers and P. Brucker, The Complexity of One-Machine Batching Problems, Discrete Applied Mathematics 47(1993), pp. 87–107.
2. Ph. Baptiste, *A Theoretical and Experimental Study of Resource Constraint Propagation*, Thèse de doctorat, Université de Technologie de Compiègne (1998).
3. Ph. Baptiste, Polynomial Time Algorithms for Minimizing the Weighted Number of Late Jobs on a Single Machine when Processing Times are Equal, Journal of Scheduling 2(1999), pp. 245–252.
4. Ph. Baptiste, Scheduling Equal-Length Jobs on Identical Parallel Machines, Technical Report 98-159(1998) Université de Technologie de Compiègne, to appear in Discrete Applied Mathematics.
5. P. Brucker, *Scheduling Algorithms*, Springer Lehrbuch (1995).
6. P. Brucker, A. Gladky, H. Hoogeveen, M. Kovalyov, C. Potts, T. Tautenhahn and S. van de Velde, Scheduling a Batching Machine, Journal of Scheduling 1(1998), pp. 31–54.
7. P. Brucker and S. Knust, Complexity Results of Scheduling Problems, URL: www//mathematik.uni-osnabrueck.de/research/OR/class.
8. P. Brucker and M.Y. Kovalyov, Single machine batch scheduling to minimize the weighted number of late jobs, Mathematical Methods of Operations Research 43(1996), pp. 1–8.
9. J. Carlier, Problème à une machine et algorithmes polynômiaux, QUESTIO 5 (1981), pp. 219–228.
10. J. Carlier, *Problèmes d'ordonnancements à contraintes de Ressources : Algorithmes et Complexité*, Thèse d'Etat, Université Paris VI (1984).
11. E.G. Coffman, M. Yannakakis, M.J. Magazine and C. Santos, Batch sizing and sequencing on a single machine, Annals of Operations Research 26(1990), pp. 135–147.
12. M. I. Dessouky, B. J. Lageweg, J. Karel Lenstra and S. L. van de Velde, Scheduling identical jobs on uniform parallel machines, Statistica Neerlandica 44(1990), pp. 115–123.
13. J. Du and J. Y-T. Leung, Minimizing Total Tardiness on One Machine is NP-Hard, Mathematics of Operations Research 15(1990), pp. 483–495.
14. L. Dupont, Ordonnancements sur machines à traitement par batch (fournée), research report GILCO Institut National Polytechnique de Grenoble (1999).
15. D.S. Hochbaum and D. Landy, Scheduling with batching: minimizing the weighted number of tardy jobs, Operations Research Letters 16(1994), pp. 79–86.
16. C. L. Monma and C. N. Potts, On the Complexity of Scheduling with Batch Setup Times, Operations Research 37(1989), pp. 798–804.

17. C. N. Potts and M. Y. Kovalyov, Scheduling with batching: A review, European Journal of Operational Research 120(2000), pp. 228–249.
18. D. F. Shallcross, A polynomial algorithm for a one machine batching problem, OR Letters 11(1992), pp. 213–218.
19. S. Webster and K.R. Baker, Scheduling Groups of Jobs on a Single Machine, Operations Research 43(1995), pp. 692–703.