Computers & Operations Research 139 (2022) 105639

Contents lists available at ScienceDirect

Computers and Operations Research

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

Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine

Alessandro Druetto, Andrea Grosso *

Dipartimento di Informatica, Università di Torino, Via Pessinetto 12, 10149 Torino, Italy

ARTICLE INFO

Keywords:
Parallel batch scheduling
Column generation
Weighted completion time

ABSTRACT

This paper deals with the single-machine, parallel batching, total weighted completion time scheduling problem. A new graph-based formulation of the problem is proposed, where such graph has an exponential number of nodes and arcs. The problem is modeled as an integer linear program on this graph, combining features of a minimum cost flow problem and features of a set-partition problem as well. The continuous relaxation of this integer problem is solved via column generation, providing a very tight lower bound. Two different flavors of column generation are tested, leading to two models with identical performances in terms of bound tightness but in practice very different in terms of running time. A simple and effective rounding strategy applied to the faster model allows to generate heuristic solutions with values within a few percentage points from the optimum. Two different variants of the heuristic rounding procedure are tested; one allows to certify the optimality gap, while the other trades the ability to certify the gap with a greater computational speed.

1. Introduction

We consider the single-machine scheduling problem of minimizing the total weighted completion time in a parallel-batching environment. In this problem a set of jobs $N = \{1, 2, \dots, n\}$ is to be partitioned into batches and processed on a single machine. Each job $j \in N$ has a given processing time p_j, a weight w_j and a *size* s_j; jobs are partitioned and processed without interruptions in a batch sequence $S = (B_1, B_2, \dots, B_t)$, and each batch B_k in S must satisfy $\sum_{j \in B} s_j \leq b$, where b is a given parameter called the machine *capacity* (Potts and Kovalyov, 2000). The jobs in a same batch B_k are processed simultaneously, with the longest job determining the processing time for the whole batch, i.e. $p_{B_k} = \max_{j \in B_k} \{p_j\}$. All the jobs in the same batch B_k share the same completion time, that is $C_j = C_{B_k} = \sum_{r=1}^{k} p_{B_r}$. The considered problem calls for finding S that minimizes:

$$f(S) = \sum_{j=1}^{n} w_j C_j$$

This problem can be denoted as $1|p\text{-batch}, s_j| \sum_j w_j C_j$ in the classical three-fields notation (Graham et al., 1979). At the time of writing, and to the authors' knowledge, the broadest survey available for problems with batching, from an algorithmic point of view, is still (Potts and Kovalyov, 2000), although followed by Mathirajan and Sivakumar (2006) and Mönch et al. (2011). This absolutely does not mean that there is no recent literature about batching problems; on the contrary the big picture about batching problems seems to have become extremely varied and complex, where each work focuses on very specific technological constraints, that often change the mathematics of the models. The reader can consider for example (Kovalyov and Šešok, 2019; Kong et al., 2020a,b; Liao et al., 2020; Pei et al., 2020, 2021), to have an idea of how fragmented the recent literature is. As a result, also the basic $1|p\text{-batch}, s_j| \sum_j w_j C_j$ lies somehow in a niche. Although the latter is an immediate generalization of the unweighted $1|p\text{-batch}, s_j| \sum_j C_j$ problem, very few works can be found in the literature about the version with general weights. Among others, Baptiste (2000) deals with the special case where $p_i = p$ for all jobs i, proving that this case can be solved in polynomial time. In Fang and Lu (2016), another special case involving *online scheduling* is addressed. An exact approach to the $1|p\text{-batch}, s_j| \sum_j w_j C_j$ problem is presented by Azizoglu and Webster (2000), generalizing results and techniques from Uzsoy's seminal work (Uzsoy, 1994; Uzsoy and Yang, 1997). Heuristics for the problem that considers also incompatible jobs families are given by Dobson and Nambimadom (2001). A polynomial time approximation scheme is given for the case of multiple parallel machines in Li et al. (2006). Before our work, the bounds in Azizoglu and Webster (2000) were the current state-of-the-art for the total weighted completion time case of the problem, considering a single parallel batching

* Corresponding author.
E-mail addresses: alessandro.druetto@unito.it (A. Druetto), andrea.grosso@unito.it (A. Grosso).

https://doi.org/10.1016/j.cor.2021.105639
Received 19 March 2021; Received in revised form 24 September 2021; Accepted 12 November 2021
Available online 27 November 2021

machine with non-unitary capacity and non-identical job sizes and processing times.

Several works analyze different objective functions. For example in Hulett et al. (2017) the total weighted tardiness for testing of electrical circuits is addressed, in Mönch and Unbehaun (2007) the earliness-tardiness for heat-treating ovens, in Ozturk et al. (2012) the makespan for medical device sterilization, in Ozturk et al. (2017) the makespan for unit-size jobs. The majority of works found in literature analyze especially the minimization of makespan (Li, 2017; Muter, 2020) as it seems to be the most common objective function. Another recurrent objective function is the maximum lateness: in Malapert et al. (2012) a constraint programming model is developed, and in Emde et al. (2020) the problem is addressed using Benders decomposition. Other works, for example: Takamatsu et al. (1979), Mönch et al. (2012) and Liu et al. (2016), analyze several types of objective functions and job constraints considering only the unweighted version of the problem. In Ozturk (2020) a column generation approach is developed to minimize the total completion time, with release dates and multiple parallel machines, but again on the unweighted version of the problem.

Recently, new interest has grown about batching problems, since additive manufacturing often requires a batch production to optimize the chamber space (Zhang et al., 2020). However, processing times depend on different factors than those of conventional production and there are typically more size constraints to be addressed.

The contribution of this paper is twofold.

- Working along the lines of Alfieri et al. (2019) and Alfieri et al. (2021), a strong lower bound for the $1|p\text{-batch}, s_j|\sum_j w_j C_j$ problem is developed. The lower bound is based on the continuous relaxation of a very large integer linear program solved by means of column generation techniques. The linear program does not rely on time-indexing, instead it uses a graph-based formulation based on peculiarities of the $\sum w_j C_j$ objective. The lower bound given by this formulation is stronger than any other currently available bound, to the authors' knowledge.
- The lower bound delivered by column generation is sharp but still too computationally heavy for supporting an exact branch and price procedure. Anyway, simple but effective rounding heuristics applied to the relaxed problem allow to quickly derive feasible solutions whose cost is within a few percentage points from the optimum.

We develop models based on column generation for the $1|p\text{-batch}, s_j|\sum_j w_j C_j$ problem in Section 2, together with some useful properties, in order to develop a lower bound. Heuristic procedures for generating feasible solutions are then illustrated in Section 3, whereas computational results are discussed in 4.

2. Column generation models

2.1. A graph-based model

We consider two distinct (but equivalent) column generation models for the $1|p\text{-batch}, s_j|\sum_j w_j C_j$ problem. Both are based on a very large (multi)graph specified as follows. The graph $G(V, A)$ is made of a set of vertices

$$V = \{1, 2, \dots, W+1\}, \quad \text{where } W = \sum_{j=1}^{n} w_j$$

and a set of arcs

$$A = \left\{(i,k,B) : 1 \le i < k \le W+1, B \subseteq N, \sum_{j\in B} w_j = k-i, \sum_{j\in B} s_j \le b\right\}$$

Every arc $(i,k,B) \in A$ connects two nodes i,k and is associated to a possible batch B. Each arc is given a cost $c_{ikB} = (W-i+1)p(B)$. We call a path P from node 1 to node $W+1$ a *partition path* if the sets $\{B : (i,k,B) \in P\}$ form a partition of N. Feasible batch sequences are mapped onto partition path and vice versa, as established by the following property.

Property 1. *$S = (B_1, B_2, \dots, B_t)$ is a feasible batch sequence iff*

$$P = [(i_1,k_1,B_1),(i_2.k_2,B_2),\dots,(i_t,k_t,B_t)] \qquad \textit{(with } i_1 = 1, k_t = W+1\textit{)}$$

is a partition path $1 \to W+1$ in G, and $f(S) = \sum_{(i,k,B)\in P} c_{ikB}$.

Proof. The one-to-one mapping of batch sequences to paths is easily established. Given a batch sequence $S = (B_1, B_2, \dots, B_t)$, the path

$$P = \left[(i_1,k_1,B_1),(i_2,k_2,B_2),\dots,(i_t,k_t,B_t)\right] \tag{1}$$

can be built from arcs of G, choosing the arcs from the arc set of G as follows:

$$i_1 = 1, \qquad k_1 = i_1 + w_{B_1}$$
$$i_\ell = k_{\ell-1}, \quad k_\ell = i_\ell + w_{B_\ell} \qquad \ell = 2, \dots, t$$

Note that $k_t - i_1 = \sum_{\ell=1}^{t} w_{B_\ell} = W$, hence $k_t = W+1$, and P is a path $1 \to W+1$ in G; the job set N is guaranteed to be partitioned over the arcs of P because it is partitioned in the batches of S by hypothesis.

Vice versa, given a path P from node 1 to node $W+1$ of G, such that the set $\{B : (i,k,B) \in P\}$ forms a partition of N, it can be easily seen that the arcs of P satisfy the connectivity of the path, and every $\sum_{j\in B} s_j \le b$ for all $(i,k,B) \in P$ by definition of the arc set A. Hence, $S = (B_1, B_2, \dots, B_t)$ with the $B_1, \dots, B_t$ defined by the arc-batches of P is a feasible batch sequence.

It remains to prove that $f(S) = \sum_{(i,k,B)\in P} c_{ikB}$. The objective function for the batch sequence is:

$$f(S) = \sum_{j=1}^{n} w_j C_j = \sum_{\ell=1}^{t} C_{B_\ell} w_{B_\ell} = \sum_{\ell=1}^{t} \left[\left(\sum_{k=1}^{\ell} p_{B_k}\right) w_{B_\ell}\right]$$

$$\begin{aligned} &= w_{B_1}p_{B_1} \quad + \\ &\quad w_{B_2}p_{B_1} \quad + \quad w_{B_2}p_{B_2} \quad + \\ &\quad w_{B_3}p_{B_1} \quad + \quad w_{B_3}p_{B_2} \quad + \quad w_{B_3}p_{B_3} \quad + \\ &\qquad\qquad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ &\quad w_{B_t}p_{B_1} \quad + \quad w_{B_t}p_{B_2} \quad + \quad w_{B_t}p_{B_3} \quad + \quad \cdots \quad + \quad w_{B_t}p_{B_t} \end{aligned}$$

Adding by column, it can be rewritten as:

$$\begin{aligned} f(S) =& p_{B_1}\sum_{\ell=1}^{t} w_{B_\ell} + p_{B_2}\sum_{\ell=2}^{t} w_{B_\ell} + p_{B_3}\sum_{\ell=3}^{t} w_{B_\ell} + \cdots + p_{B_t} w_{B_t} \\ =& \sum_{q=1}^{t}\left[p_{B_q}\sum_{\ell=1}^{t} w_{B_\ell}\right] \end{aligned}$$

Now, in every path P like (1) it holds that, for an arc (i_q, k_q, B_q) in position q on P, the total weight of jobs scheduled from B_q to B_t is

$$\sum_{\ell=q}^{t} w_{B_\ell} = \sum_{\ell=q}^{t}(k_\ell - i_\ell) = \sum_{\ell=q}^{t-1} i_{\ell+1} + (W+1) - \sum_{\ell=q}^{t} i_\ell = (W+1-i_q)$$

hence

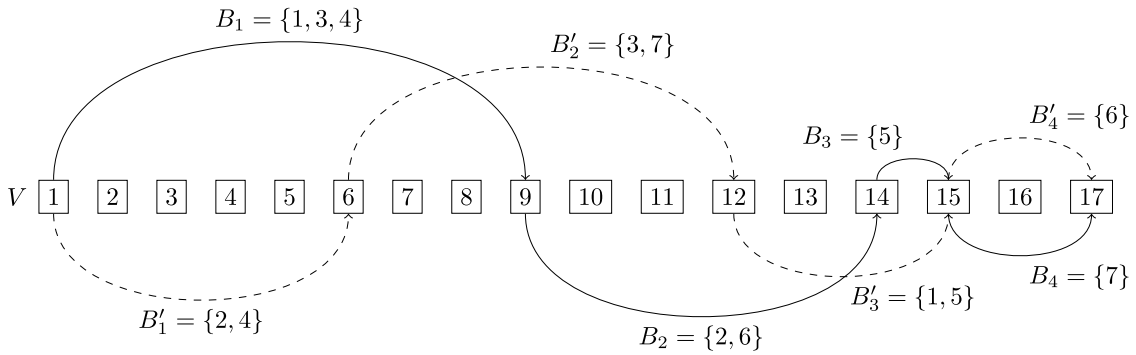
$$f(S) = \sum_{q=1}^{t} p_{B_q}(W+1-i_q) = \sum_{(i,k,B)\in P} c_{ikB}. \quad \square$$

As an example for Property 1, consider a 7-jobs instance with job set $N = \{1,2,\dots,7\}$, machine capacity $b = 10$ and:

$$\begin{aligned} p_1,\dots,p_7 =& 12, 10, 8, 8, 6, 4, 3, \\ s_1,\dots,s_7 =& 3, 3, 3, 4, 4, 7, 7, \\ w_1,\dots,w_7 =& 2, 3, 4, 2, 1, 2, 2 \qquad (W = 16) \end{aligned}$$

The graph $G(V, A)$ is made on node $V = \{1, 2, \dots, 17\}$ — see Fig. 1. It is not practical to show all the arcs even for this small instance. For example between node 1 and 5 all the arcs $(1,5,\{3\})$, $(1,5,\{1,4\})$, $(1,5,\{1,6\})$, $(1,5,\{2,5\})$, etc. — all batches B with $\sum_{j\in B} w_j = 5-1 = 4$ — should appear; the same arcs should appear between node 2 and node 6, and so on.

In Fig. 1 two examples of partition paths P, P' for this example are sketched. Path P corresponds to the batch sequence $S = (B_1, B_2, B_3, B_4)$;



$$c_{(1,9,B_1)} = p_{B_1}(W+1-1) = 12 \cdot 16 = 192$$
$$c_{(9,14,B_2)} = p_{B_2}(W+1-9) = 10 \cdot 8 = 80$$
$$c_{(14,15,B_3)} = p_{B_3}(W+1-14) = 6 \cdot 3 = 18$$
$$c_{(15,17,B_4)} = p_{B_4}(W+1-15) = 3 \cdot 2 = 6$$

$$c_{(1,6,B'_1)} = p_{B'_1}(W+1-1) = 10 \cdot 16 = 160$$
$$c_{(6,12,B'_2)} = p_{B'_2}(W+1-6) = 8 \cdot 11 = 88$$
$$c_{(12,15,B'_3)} = p_{B'_3}(W+1-12) = 12 \cdot 5 = 60$$
$$c_{(15,17,B'_4)} = p_{B'_4}(W+1-15) = 4 \cdot 2 = 8$$

Fig. 1. Example of partition paths. Batch sequence $S = (B_1, B_2, B_3, B_4)$ corresponds to partition path $P = [(1,9,B_1),(9,14,B_2),(14,15,B_3),(15,17,B_4)]$ (continuous lines). Batch sequence $S' = (B'_1, B'_2, B'_3, B'_4)$ corresponds to path $P' = [(1,6,B'_1),(6,12,B'_2),(12,15,B'_3),(15,17,B'_4)]$ (dashed lines).

note how B_1, B_2, B_3, B_4 form a partition of the job set N. Obviously $C_{B_1} = p_{B_1} = p_1 = 12$, $C_{B_2} = p_{B_1} + p_{B_2} = p_1 + p_2 = 22$, $C_{B_3} = p_{B_1} + p_{B_2} + p_{B_3} = p_1 + p_2 + p_5 = 28$, $C_{B_4} = p_{B_1} + p_{B_2} + p_{B_3} + p_{B_4} = p_1 + p_2 + p_5 + p_7 = 31$, and $w_{B_1} = w_1 + w_4 + w_4 = 8$, $w_{B_2} = w_2 + w_6 = 5$, $w_{B_3} = w_5 = 1$, $w_{B_4} = w_7 = 2$. The objective function computed for S is

$$\begin{aligned} f(S) &= w_{B_1} C_{B_1} + w_{B_2} C_{B_2} + w_{B_3} C_{B_3} + w_{B_4} C_{B_4} \\ &= 8 \cdot 12 + 5 \cdot 22 + 1 \cdot 28 + 2 \cdot 31 = 296, \end{aligned}$$

but also

$$\begin{aligned} f(S) &= p_{B_1} w_{B_1} + \\ &\quad p_{B_1} w_{B_2} + p_{B_2} w_{B_2} + \\ &\quad p_{B_1} w_{B_3} + p_{B_2} w_{B_3} + p_{B_3} w_{B_3} + \\ &\quad p_{B_1} w_{B_4} + p_{B_2} w_{B_4} + p_{B_3} w_{B_4} + p_{B_4} w_{B_4} \\ &= p_{B_1} \cdot 16 + p_{B_2} \cdot 8 + p_{B_3} \cdot 3 + p_{B_4} \cdot 2 \\ &= c_{(1,9,B_1)} + c_{(9,14,B_2)} + c_{(14,15,B_3)} + c_{(15,17,B_4)} \\ &= c(P) = 192 + 80 + 18 + 6 = 296 \end{aligned}$$

Similarly, the reader can verify that for S' and P', $f(S') = w_{B'_1} C_{B'_1} + w_{B'_2} C_{B'_2} + w_{B'_3} C_{B'_3} + w_{B'_4} C_{B'_4} = 316$ and $c(P') = 316$. □

Determining an optimal batch sequence amounts to computing a minimum cost $1 \to W+1$ partition path on G. The huge number of arcs involved in such problem can be handled by means of column generation techniques. In general, column generation techniques apply the simplex method to very large linear programs like

$$\min \left\{ \sum_{i \in \mathcal{S}} c_i x_i : \boldsymbol{a}_i x_i = \boldsymbol{b},\ x_i \geq 0,\ i \in \mathcal{S} \right\} \qquad (\boldsymbol{a}_i \in \mathbb{R}^m) \tag{2}$$

where the number of variables x_i and columns $\boldsymbol{a}_i$, $i \in \mathcal{S}$ is so large that the whole program cannot be kept in memory. In the main iteration of a column generation procedure, problem (2) — called *master problem* — is solved on a restricted set $\mathcal{S}' \subset \mathcal{S}$. The solution of such restricted master problem is then proved (or disproved) optimal for (2) by identifying columns having minimum reduced cost; this is done using the simplex multipliers/dual variables $\boldsymbol{\eta} \in \mathbb{R}^m$ associated with the optimal basis of the restricted master problem, computing

$$r^* = \min\{r_i = c_i - \boldsymbol{\eta}^{\mathsf{T}} \boldsymbol{a}_i : i \in \mathcal{S}\}. \tag{3}$$

If $r^* < 0$, one or more columns with reduced cost $r_i < 0$ are found and they are added to the restricted set $\mathcal{S}'$ for a new iteration, otherwise optimality of the current solution is proved. The whole column generation procedure can usually be made computationally efficient if the *pricing problem* (3) presents a suitable combinatorial structure. A broad presentation of column generation can be found for example in Desrosiers and Lübbecke (2005).

Two "natural" procedures arise for solving the (continuous relaxation of the) problem of computing a minimum-cost partition path by means of column generation techniques, leading to different master programs and pricing problems. The first one relies on a master problem which combines characteristics of an arc-based minimum cost flow problem formulation with additional set-partitioning constraints. The second one relies on a path-based minimum cost flow formulation with additional set-partitioning constraints. In both cases, a crucial step for the pricing procedure consists in solving a cardinality-constrained knapsack problem that is handled by dynamic programming.

2.2. An arc-based flow model

Master problem

The problem of finding a minimum-cost $1 \to W+1$ partition path can be represented by a very large binary linear program. For a batch B, we denote by $\boldsymbol{a}_B \in \{0,1\}^n$ the incidence vector of set B, where each of the n components $a_{B,j} = 1$ iff $j \in B$; also, let us introduce a vector constant $\mathbf{1} = (1, 1, \ldots, 1)^{\mathsf{T}}$. As control variables we define binaries $x_{ikB} = 1$ iff arc $(i,k,B) \in A$ is used in the minimum cost $1 \to W+1$ path. The linear program for finding the best $1 \to W+1$ partition path writes out as follows.

$$\text{minimize} \sum_{(i,k,B) \in A} c_{ikB} x_{ikB} \tag{4}$$

$$\text{subject to} \sum_{(i,k,B) \in A} x_{ikB} - \sum_{(k,i,B) \in A} x_{kiB} = \begin{cases} 1 & \text{for } i = 1 \\ -1 & \text{for } i = W+1 \\ 0 & \text{for } i = 2, \ldots, W \end{cases} \tag{5}$$

$$\sum_{(i,k,B) \in A} \boldsymbol{a}_B x_{ikB} = \mathbf{1} \tag{6}$$

$$x_{i,k,B} \in \{0,1\} \quad (i,k,B) \in A. \tag{7}$$

The objective function (4) and constraints (5) require a unit of integer flow to be pushed along a $1 \to W+1$ path at minimum total cost: constraints (5) are flow conservation constraints written for nodes $i = 1, \ldots, W+1$, with source 1 and sink $W+1$. This a well known linear programming formulation of a shortest path problem (Ahuja et al., 1993). The vector constraint (6) subsumes n scalar constraints requiring that the path is actually a partition path, i.e. the job set $N = \{1, 2, \ldots, n\}$

is exactly partitioned over the arcs (i,k,B) with $x_{ikB} = 1$. A lower bound for the integer program (4)–(7) is immediately obtained by replacing (7) with

$$x_{ikB} \geq 0 \quad (i,k,B) \in A. \tag{7'}$$

The dual of (4)–(7′) is the following, with dual variables $u_1, \dots, u_{W+1}$ and $v_1, \dots, v_n$.

$$\text{maximize } u_1 - u_{W+1} + \sum_{j \in N} v_j \tag{8}$$

$$\text{subject to } u_i - u_k + \sum_{j \in B} v_j \leq c_{ikB} \qquad (i,k,B) \in A \tag{9}$$

The number of constraints in program (4)–(7′) is $n + W + 1$, which can be quite large but is still manageable on most instances, up to a reasonable number of jobs — also, the columns of constraints (5)–(6) are rather sparse. On the other hand, the number of variables x_{ikB} is too high, and requires a column generation approach. We consider a subset $A' \subset A$ of arcs/variables and formulate a restricted master program like (4)–(7′) over the arcs in A' only. Next we illustrate how new arcs can be added to the restricted master program.

Pricing

The dual of the restricted master problem is (8)–(9) with constraints restricted to the set A'. Once the restricted program is solved, the optimal dual variables for the restricted dual are also available: $u_1, \dots, u_{W+1}$ from the simplex multipliers of constraints (5) and $v_1, \dots, v_n$ from the simplex multipliers of constraints (6). Such values are used to formulate a pricing problem. For a fixed pair of node indices $1 \leq i < k \leq W+1$, an arc with minimum reduced cost $\bar{c}_{ikB^*}$ must satisfy

$$\begin{aligned} \bar{c}_{ikB^*} &= \min_B \left\{ c_{ikB} - (u_i - u_k) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b,\ w_B = k - i \right\} \\ &= \min_B \left\{ p_B(W - i + 1) - \sum_{j \in B} v_j : \sum_{j \in B} s_j \leq b,\ w_B = k - i \right\} \\ &\quad - (u_i - u_k) \end{aligned} \tag{10}$$

Note that fixing i, k determines the required weight for the batch B^*. Finding the batch B^* that minimizes this equation, for each given pair of indices $1 \leq i < k \leq W+1$ and given batch processing time p_{B^*}, can be done by exploiting the dynamic programming state space of a family of knapsack problems, where items correspond to jobs and an additional constraint must be enforced on the total weight packed into the batch. Assume that the jobs are indexed by Longest Processing Time (LPT) order, so that $p_1 \geq p_2 \geq \cdots \geq p_n$.

We define, for $r = 1, \dots, n$, the following problem:

$$g_r(\tau, \ell) = \max \left\{ \sum_{j=r}^{n} v_j y_j : \sum_{j=r}^{n} s_j y_j \leq \tau,\ \sum_{j=r}^{n} w_j y_j = \ell,\ y_j \in \{0,1\} \right\} \tag{11}$$

In this, $g_r(\tau, \ell)$ is the optimal value of a knapsack with profits v_j and sizes s_j, limited to jobs $r, r+1, \dots, n$, total size $\leq \tau$ and total weight $= \ell$. Variable y_j is set to 1, i.e., $y_j = 1$, iff job j is included in the solution.

Optimal values for $g_r(\tau, \ell)$ can be recursively computed as

$$g_r(\tau, \ell) = \max \begin{cases} g_{r+1}(\tau - s_r, \ell - w_r) + v_r & (y_r = 1) \\ g_{r+1}(\tau, \ell) & (y_r = 0) \end{cases} \tag{12}$$

with boundary conditions

$$g_r(\tau, w_r) = \begin{cases} v_r & \text{if } s_r \leq \tau\ (y_r = 1) \\ 0 & \text{otherwise } (y_r = 0) \end{cases} \qquad r = 1, \dots, n,\ \tau = 0, \dots, b \tag{13}$$

$$g_r(\tau, 0) = 0 \qquad r = 1, \dots, n,\ \tau = 0, \dots, b \tag{14}$$

$$g_r(\tau, \ell) = -\infty \qquad \text{if } \ell > \sum_{i=r}^{n} w_i \text{ or } \ell < 0 \text{ or } \tau < 0 \tag{15}$$

The corresponding optimal job sets are denoted by $B_r(\tau, \ell)$; such sets can be retrieved by backtracking. The following property establishes that the state space $g_r(\tau, \ell)$ for $r \in \hat{N}$ is sufficient for pricing all the relevant arcs.

Property 2. *Consider the subset of jobs $\hat{N} = \{1\} \cup \{j > 1 : p_j < p_{j-1}\}$ which holds one job, with index as small as possible, for each processing time represented in the problem. For any given pair of indices $i < k$, an arc with minimum reduced cost (i,k,B^*) is one of*

$$(i, k, B_r(b, k-i)) \qquad r \in \hat{N}.$$

Proof. Every arc (i,k,B) can be shown to have a reduced cost not less than some of the arcs in the aforementioned set. Let $\bar{c}_{ikB} = p_B(W - i + 1) - (u_i - u_k) - \sum_{j \in B} v_j$ be the reduced cost of an arc (i,k,B). Recall that $w_B = k - i$, and the jobs are numbered in non-increasing order of processing times.

Choose r as the smallest job index such that $p_r = p_B$. Note that $B \subseteq \{r, r+1, \dots, n\}$ and $r \in \hat{N}$. Consider knapsack $g_r(b, k-i)$ and the associated optimal subset $B_r^* = B_r(b, k-i)$. The batch B is a feasible solution for knapsack $g_r(b, k-i)$, hence $\sum_{j \in B} v_j \leq g_r(b, k-i)$; also, because of the choice of r, $p_{B_r^*} \leq p_r = p_B$. Thus:

$$\begin{aligned} \bar{c}_{ikB} &= p_B(W - i + 1) - (u_i - u_k) - \sum_{j \in B} v_j \\ &\geq p_{B_r^*}(W - i + 1) - (u_i - u_k) - g_r(b, k-i) = \bar{c}_{ikB_r^*} \end{aligned} \tag{16}$$

□

Given the optimal multipliers from the restricted master problem, Algorithm 1 (NewCols) generates a set of arcs with negative reduced cost, if some exist, or certifies optimality for (4)–(7′) if none is found. The size of the state space required for the pricing is bounded by $\mathcal{O}(nWb)$. A *memoized* dynamic programming table is used, so that the execution of the top-down recursion for computing an entry $g_r(\tau, \ell)$ is deferred until the first time the value is queried. Then, the value is kept in storage and accessed in $\mathcal{O}(1)$ time if it is queried again. NewCols exhibits three nested loops that account for a $\mathcal{O}(nW^2)$ complexity; taking into account the filling (memoized or not) of the dynamic programming table, the running time of the pricing procedure is bounded from above by $\mathcal{O}(\max(nW^2, nWb))$.

Algorithm 1 Pricing procedure for arc-based model.

```
1: function NewCols(N, b, u, v)        ▷ u, v = vectors of multipliers
2:     Sort and renumber jobs in N such that p_1 ≥ p_2 ··· ≥ p_n;
3:     n ← |N|;
4:     W ← Σ_{j=1}^{n} w_j;
5:     H ← ∅;
6:     for ℓ = 1, ..., W do                          ▷ for each weight
7:         for r = 1, ..., n do                      ▷ for each job index
8:             Retrieve g_r(b, ℓ) and B_r(b, ℓ);
9:             B ← B_r(b, ℓ);
10:            for i = 1, ..., W − ℓ + 1 do          ▷ for each position
11:                k ← i + ℓ;
12:                c̄_ikB ← p_B(W − i + 1) − (u_i − u_k) − g_r(b, ℓ);
13:                if c̄_ikB < 0 then                 ▷ if the reduced cost is negative...
14:                    H ← H ∪ {(i, k, B)};          ▷ ...add the corresponding arc
15:                end if
16:            end for
17:        end for
18:    end for
19:    return H;
20: end function
```

A minimal subset of starting columns is required to evaluate the first optimal multipliers and start generating new columns. To populate in a meaningful way the restricted master problem, we order all jobs j in a non-decreasing order with regard to the processing time p_j

and generate all possible batches, made by grouping subsequent jobs together, starting from every job in the sequence. These batches B are then added at the problem as new columns in every possible position (i, k, B) where $k - i = \sum_{j \in B} w_j$ as shown in Algorithm 2 (InitCols).

Algorithm 2 Generation of initial columns for arc-based model.

```
1: function InitCols(N, b)
2:     Sort and renumber jobs in N such that p_1 ≤ p_2 ··· ≤ p_n;
3:     n ← |N|, W ← Σ_{j=1}^{n} w_j;
4:     H ← ∅;
5:     for i = 1, ..., W do                          ▷ for each starting position
6:         for ℓ = 1, ..., n do                      ▷ for each job index
7:             B ← ∅;
8:             for j = ℓ, ..., 0 do                  ▷ for every previous job
9:                 if s_B + s_j ≤ b then
10:                    B ← B ∪ {j};                  ▷ add this job only if it fits
11:                end if
12:            end for
13:            k ← i + w_B;
14:            H ← {(i, k, B)};
15:        end for
16:    end for
17:    return H;
18: end function
```

These InitCols and NewCols algorithms can then be combined in a procedure for solving program (4)–(7′) by column generation, as can be seen in Algorithm 3, that will be called CG–LB from now on.

Algorithm 3 Column generation for solving the arc-based relaxation

```
1: function CG-LB(N, b)
2:     Set A′ := ∅;
3:     Set H := InitCols(N, b);
4:     while H ≠ ∅ do
5:         Set A′ := A′ ∪ H;
6:         x, u, v ← solution of (4)–(7′) on G(V, A′); ▷ get solution and duals
7:         Set H := NewCols(N, b, u, v);
8:     end while
9:     return (x_{ikB} : (i, k, B) ∈ A′);            ▷ generated batches
10: end function
```

2.3. A path-based model

Master problem

Program (4)–(7) is basically made of a "flow" section (constraints (5)) and a "cover" section (constraints (6)). The flow section carries along with it some features of flow models, among them the presence of heavily degenerate bases in cases (like this) with a single source and a single sink. This can (possibly, but not certainly) lead to stalling/slow convergence behaviors. In (4)–(7) the flow moves along arcs in the graph. Moving the flow on whole paths can be seen as a workaround for the problems stemming from degenerate bases. We consider the set $\mathcal{P}$ of all possible paths $1 \to W+1$ and define binary variables x_P for each $P \in \mathcal{P}$. Let us introduce a vector $\boldsymbol{a_P} \in \mathbb{N}^n$ where each component $a_{P,j}$ counts the number of times that job j appears in the batches B of the arcs $(i, k, B) \in P$ — we stress that here generally P is not necessarily a partition-path, so more than a batch on the path might contain the same job j, or some job j can even fail to appear on the path P. The search for a minimum cost partition path is then captured by the following program, where $c_P = \sum_{(ikB) \in P} c_{ikB}$.

$$\text{minimize} \quad \sum_{P \in \mathcal{P}} c_P x_P \tag{17}$$

$$\text{subject to} \quad \sum_{P \in \mathcal{P}} x_P = 1 \tag{18}$$

$$\sum_{P \in \mathcal{P}} \boldsymbol{a_P} x_P = \mathbf{1} \tag{19}$$

$$x_P \in \{0, 1\} \quad P \in \mathcal{P}. \tag{20}$$

We replace (20) with

$$x_P \geq 0 \qquad P \in \mathcal{P} \tag{20′}$$

when considering the continuous relaxation. Constraint (18) forces the integer flow to move along a single path. The vector constraint (19) forces the selected path to be a partition-path: actually (19) subsumes n scalar constraints enforcing that each job $j \in \{1, 2, \dots, n\}$ is counted exactly once in the selected path. If the continuous relaxation is considered, constraints (19) requires that summing up the number of times that each job is used over all the paths P with $x_P > 0$ must exactly add up to 1, for each job. We remind the reader that while only one path can take $x_P = 1$ in the solution of the binary program, the continuous relaxation can in general have several $x_P \in (0, 1)$.

The dual of (17)–(20′) is the following, with dual variables μ for constraint (18) and $\lambda_1, \dots, \lambda_n$ for the n scalar constraints from (19).

$$\text{maximize} \quad \mu + \sum_{j \in N} \lambda_j \tag{21}$$

$$\text{subject to} \quad \mu + \sum_{j=1}^{n} a_{P,j} \lambda_j \leq c_P \qquad P \in \mathcal{P} \tag{22}$$

We formulate (17)–(20′) as a restricted master problem on a subset of paths $\mathcal{P}' \subset \mathcal{P}$.

Pricing

Given optimal multipliers/dual variables $\mu, \lambda_1, \dots, \lambda_n$, a path P^* with minimum reduced cost must satisfy

$$\bar{c}_{P^*} = \min_{P \in \mathcal{P}} \left\{ c_P - \mu - \sum_{j=1}^{n} a_{P,j} \lambda_j \right\} = \min_{P \in \mathcal{P}} \left\{ \sum_{(ikB) \in P} \left[c_{ikB} - \sum_{j \in B} \lambda_j \right] \right\} - \mu \tag{23}$$

The problem in Eq. (23) amounts to determine a minimum cost path $1 \to W+1$ on the very large graph G, using arcs with modified costs

$$\hat{c}_{ikB} = p_B(W - i + 1) - \sum_{j \in B} \lambda_j$$

Graph G is naturally layered — only arcs (i, k, B) with $i < k$ are present — and a minimum cost path $1 \to W+1$ on G can be computed basically by an implementation of Gondran and Minoux's shortest path algorithm (Gondran et al., 1984). Given a node k and a predecessor $i < k$, the cheapest arc from i to k must be associated with a batch B such that

$$\hat{c}_{ikB} = \min_{B} \left\{ p_B(W - i + 1) - \sum_{j \in B} \lambda_j : \sum_{j \in B} s_j \leq b,\ w_B = k - i \right\} \tag{24}$$

Eq. (24) is completely analogous to (10), and can be solved by relying on a dynamic programming table similar to (12):

$$\hat{g}_r(\tau, \ell) = \max \left\{ \sum_{j=r}^{n} \lambda_j y_j : \sum_{j=r}^{n} s_j y_j \leq \tau,\ \sum_{j=r}^{n} w_j y_j = \ell,\ y_j \in \{0, 1\}. \right\} \tag{25}$$

Given such table, whose size is limited by $\mathcal{O}(nWb)$, Algorithm 4 computes a path with minimum reduced cost in a running time bounded by $\mathcal{O}(nW^2)$.

Again, a minimal subset of starting columns (paths in this case) is required to evaluate the first optimal multipliers and start generating new columns. To populate in a meaningful way the restricted master problem, we first add to the problem the path corresponding to Azizoglu and Webster (2000) heuristic. Then, we order all jobs j in a non-decreasing order with regard to the processing time p_j and partition them in batches, creating a new batch when the previous is full (i.e. adding the next job in the previous batch would invalidate max capacity constraint), adding the resulting path to the restricted master problem. Finally, we generate a number of random sequences and partition them in batches again as we do with the ordered sequence.

Algorithm 4 Pricing for the path-based model.

```
function MinimumCostPath(μ, λ_1, ..., λ_n)
    Set Π_1 = 0, Π_k = ∞ for k = 2, ..., W + 1;
    for i = 0, ..., W do
        for k = i + 1, ..., W + 1 do
            for r ∈ N̂ do
                Set Π_k = min{Π_k, Π_i + p_r(W − i + 1) − ĝ_r(b, k − i)}
            end for
        end for
    end for
    return Π_{W+1} − μ;
end function
```

3. Upper bounding: heuristics

This section outlines simple but effective rounding strategies for generating integer solutions from the fractional optimal solution of the relaxed model. The focus is kept on the arc-based model, because the arc-based model is especially well-suited for a strategy where one arc at a time is rounded to an integral flow value in order to build a partition path. Rounding a single (or few) arc variables in program (4)–(7′) is easily managed in the LP solver, while partially rounding a path-variable in a program like (17)–(20′) would require a trickier handling of the residual problem. Also, despite of the attractive feature of having a reduced number of constraints in the master problem, the path-based model turned out to be computationally heavier than the arc-based model (see Section 4 ahead).

Once program (4)–(7′) has been solved, a lower bound is available; a minimal-effort strategy for getting an upper bound could be setting all variables back to the binary type and solve the integer version of the restricted master problem by branch and bound, possibly truncating the process when a limit on computation time and/or processed branch nodes is reached. This approach — also called "price and branch" as opposed to the exact branch and price approach — was pursued in Alfieri et al. (2019, 2021) for the special case of the problem where $w_j = 1$ for all $j \in N$. For the general $1|p\text{-batch}, s_j|\sum_j w_j C_j$ problem anyway, the large size of the involved linear programs involved makes using a branch and bound impractical, especially for large instances.

We investigated simple strategies based on rounding fractional variables in order to generate feasible solutions within shorter computation times. This approach is inspired by Mourgaya and Vanderbeck (2007) where rounding heuristics are applied to vehicle routing problems.

3.1. Variable rounding upper bound

In the basic rounding approach, we build a partition path by rounding flow values, starting from the source and moving towards the sink node. Given the optimal fractional flow on the final restricted master, we search for the arc (i, k, B) with maximal nonzero flow that lies as near as possible to the source, and round its flow value to 1. If the previous value of the flow on (i, k, B) was less than 1, the restricted master is reoptimized, possibly adding other columns in the reoptimization process.

Note that rounding a fractional variable to 1 could in principle make the restricted master unfeasible; such an event cannot happen in our implementation, because of how the initial set of columns is generated by InitCols — note that a partition path made of single-job arcs is always available in order to build a (possibly bad) feasible solution for the residual problem.

We then iterate this rounding step until an arc reaching the sink node $W + 1$ is fixed, thus completing a feasible solution. The value of such solution is called VR–UB in the following, and Algorithm 5 sketches the procedure.

Algorithm 5 Variable Rounding Upper Bound.

```
x ← SolveArcModel(N, b);
Let x = (x_{ikB} : (i, k, B) ∈ A′) the set of variable of the restricted master;
i ← 1;
while i < W + 1 do
    Set (i, k, B) := arg max{x_{i,k′,B′} : (i, k′, B′) ∈ A′}; ▷ i-outgoing arc with maximum flow
    if x_{ikB} = 1 then
        fix x_{ikB} = 1;
    else
        fix x_{ikB} = 1;
        Reoptimize the restricted master, adding columns as necessary;
    end if
    i ← k;
end while
VR–UB ← optimum of the restricted master;
```

3.2. Early rounding upper bound

The CG–LB plus VR–UB procedure delivers in a *relatively* short time both a lower and an upper bound for the $1|p\text{-batch}, s_j|\sum_j w_j C_j$ problem, but for larger instances (e.g. more than 50 jobs) and some size distributions the execution could go on for a considerable amount of time (more than 3 min for 60 jobs, more than 30 min for 100 jobs).

In order to speed up the construction of a partition path without losing too much of the quality of our bounds, a possible strategy consists of trusting the information gathered in the solution of the continuous relaxation before (sometimes even *well* before) convergence to the relaxed optimum is achieved, and proceeding to round variables anyway. This strategy is called *early rounding* in what follows. After a certain number of "unfruitful" iterations of column generation exhibiting only a small improvement in objective function, an arc variable (i, k, B) with i as small as possible and maximum flow value is fixed to 1, by a logic analogous to the one used in Algorithm 5.

This obviously prevents the delivery of a valid lower bound at the end, offering in exchange a faster shrinking of the residual problems to be optimized in the next iterations. This tradeoff turns out to be practically advantageous, allowing to consistently reduce computation times without giving up much of the solution quality. The procedure is summarized in Algorithm 6, where *threshold* and *steps* are parameters determining respectively the minimum relative improvement in objective function below which the iteration is considered unfruitful, and the maximum number of unfruitful iterations to be tolerated before a rounding is performed. The value of the resulting solution is called ER–UB.

4. Computational results

4.1. Testing environment

All the tests ran in a Linux environment with `Intel Core i7-6500U CPU @ 2.50GHz` processor; all algorithms have been implemented in D (using a language subset that practically guarantees C-like performances); the LP solver used was CPLEX 12.8.

A number of test instances were generated following the de-facto standard for this type of parallel batching problems.

- The processing times p_j were randomly drawn from the uniform discrete distribution $[1, 100]$.
- The machine capacity was fixed to $b = 10$.

Algorithm 6 Early Rounding Upper Bound.

$A' \leftarrow$ InitCols(N, b);
$W \leftarrow \sum_{j=1}^{n} w_j$, $k \leftarrow 1$, $s \leftarrow 0$;
$z, \mathbf{u}, \mathbf{v} \leftarrow$ optimum of $G(V, A')$; ▷ evaluating to obtain duals
while $i < W + 1$ **do** ▷ $i = W + 1 \implies$ partition path complete
  $A' \leftarrow A' \cup$ NewCols$(N, b, \mathbf{u}, \mathbf{v})$; ▷ generating new columns
  $z', \mathbf{u}, \mathbf{v} \leftarrow$ optimum of $G(V, A')$; ▷ evaluating new optimum
  if $(z - z')/z < threshold$ **then** ▷ if difference is less than threshold...
    $s \leftarrow s + 1$; ▷ ...count an "unfruitful" iteration
  else
    $s \leftarrow 0$;
  end if
  if $s \geq steps$ **then**
    Set $(i, k, B) := \arg\max\{x_{i,k',B'} : (i, k', B') \in A'\}$; ▷ i-outgoing arc with maximum flow
    `fix` $x_{ikB} = 1$;
    $i \leftarrow k$; ▷ new starting position is old ending position
    $s \leftarrow 0$; ▷ reset counter
    $z, \mathbf{u}, \mathbf{v} \leftarrow$ optimum of $G(V, A')$; ▷ evaluating to obtain duals
  end if
end while
ER–UB $\leftarrow z$; ▷ is an integral solution

Table 1
CPU times and restricted master size for arc-based model vs. path-based model.

Param		Arc-based		Path-based	
n	σ	Time (s)	Columns	Time (s)	Columns
10	σ_1	0.05	2 096	1.26	76
	σ_2	0.05	2 017	0.89	85
	σ_3	0.03	1 955	0.75	79
	σ_4	0.07	2 344	1.55	90
20	σ_1	1.33	9 215	118.24	459
	σ_2	0.98	8 686	53.37	367
	σ_3	0.59	8 487	92.74	667
	σ_4	3.30	13 592	112.50	295
30	σ_1	9.90	23 944	912.98	965
	σ_2	5.67	22 907	664.42	1013
	σ_3	2.70	21 858	605.11	1322
	σ_4	24.30	33 242	1670.77	828

- The job sizes s_j were randomly drawn from four possible uniform discrete distributions, labeled by $\sigma \in \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$:

  $\sigma_1 : s_j \in [1, 10]$ $\sigma_3 : s_j \in [3, 10]$
  $\sigma_2 : s_j \in [2, 8]$ $\sigma_4 : s_j \in [1, 5]$.

- The job weights were drawn from the uniform discrete distribution $[1, 50]$.

A batch with ten instances for each σ class was generated, plus a larger batch with twenty-five examples for each σ for more extensive testing; the latter is named "Extended Set" in the following.

4.2. Performance of basic models

The first tests compared the performances of the arc-based model against the path-based model. Both models were run on the same set of instances in order to compute the continuous lower bound. The path-based model proved to be unable to handle instances with more than 30 jobs within reasonable computation times; detailed results are given in Table 1. The lower bound is (as expected by the theory) the same for both models, even if the computation times are heavily different.

Profiling the algorithms allows to observe that the solution processes for the two models have different bottlenecks.

- For the arc-based model, most of the CPU time is spent in the solution of the master problem; the typical basis in the master problem is quite degenerate, often exhibiting more than 80% of basic variables fixed to zero. On the other hand, a very short time is spent in the pricing routine: despite of the pseudopolynomial worst-case time complexity, thanks to the memoization approach usually a small portion of the dynamic programming state space actually need to be exploited, and time measures (omitted in table for conciseness) show that usually about 15% of the CPU time is spent generating new columns.
- For the path-based model, the basis is still heavily degenerate (apparently only slightly less than in the arc-based model); the size of the master problem (both in rows and columns) is considerably smaller than for the arc-based model, and the time spent solving the master is proportionally smaller. Anyway, a more efficient method for generating columns is required in order to make the path-based model competitive. The overall complexity of the pricing procedure in Algorithm 4 is comparable with that of Algorithm 1 used for the arc-based model, but whereas Algorithm 1 can generate a set of columns to be added to the restricted master problem, only one column at a time will emerge from the application of Algorithm 4. This usually results in a snap reoptimization of the heavily degenerate restricted master with only a small (or null) decrease in the objective value and a new call to the pricing phase. The number of calls to the pricing procedure for the path-based model is so high that more than 90% of the computation time is spent in the pricing procedure, with a much higher number of iterations required to reach convergence.

In view of this, the use of arc-based models has been pursued in this work. The path-based model still remains interesting because of the smaller size of the restricted master problem, but some more research effort has to be devoted to enhancing the pricing phase.

4.3. Generating feasible solutions by rounding

From now on, the arc-based continuous relaxation (4)–(7′) is considered. Results with the basic variable rounding heuristic are presented first. In this approach, once convergence has been reached for the continuous relaxation, Algorithm 5 is applied, iteratively selecting an arc of graph G carrying a fractional flow and rounding it to 1.

Very few heuristics and/or relaxations are available in literature for $1|p\text{-batch}, s_j|\sum_j w_j C_j$, among them are both a relaxation and a greedy heuristic developed by (Azizoglu and Webster, 2000) in the context of a branch and bound algorithm for $1|p\text{-batch}, s_j|\sum_j w_j C_j$. Such procedures are labeled AW–LB and AW–UB respectively.

All gaps in the following tables are evaluated by the relative difference between upper bound and lower bound, using the formula

$$\text{Gap } (\%) = \frac{UB - LB}{UB} \times 100$$

where LB = CG–LB in all cases, and UB = VR–UB for Table 2. In Table 3, Tables 4 and 5 both VR–UB and ER–UB are considered as UB, depending on the column sets. Average, highest and lowest gap values are presented. In Table 3, Tables 4 and 5 the values highlighted in **bold** refer to the best performing algorithm, VR–UB or ER–UB, considering execution time and gap quality.

Lower Bound and Upper Bound columns in Table 2 give a comparison between our lower/upper bounds and the lower/upper bounds by (Azizoglu and Webster, 2000). In Lower Bound, the higher is the value the better is the result for CG–LB; in Upper Bound, the lower is the value the better is the result for VR–UB. Finally, the opt column in Table 2 shows how many optima (amongst the 10 instances of the test set) the combination of CG–LB and VR–UB have certified; this happens when CG–LB = VR–UB and the lower bound has zero variables with fractional value.

Table 2
Results for CG–LB and VR–UB on the Weighted Parallel Batching model. The third-to-last and second-to-last columns compare. CG–LB and VR–UB with Azizoglu & Webster's lower and upper bounds (AW–LB, AW–UB resp).

Param		Times (s)		Gap (%)			Lower bound	Upper bound	
n	σ	CG–LB	VR–UB	avg	worst	best	CG–LB/AW–LB	VR–UB/AW–UB	opt
10	σ_1	0.05	0.07	0.33	3.33	0.00	1.39	0.83	9
	σ_2	0.05	0.06	0.04	0.40	0.00	1.31	0.94	9
	σ_3	0.03	0.04	0.08	0.80	0.00	1.27	0.97	9
	σ_4	0.07	0.08	0.02	0.18	0.00	1.41	0.82	9
20	σ_1	1.33	1.55	0.46	3.84	0.00	1.29	0.87	3
	σ_2	0.98	1.18	0.39	2.11	0.00	1.25	0.86	6
	σ_3	0.59	0.82	0.31	2.24	0.00	1.21	0.94	7
	σ_4	3.30	4.05	1.09	2.53	0.00	1.35	0.78	2
30	σ_1	9.90	11.25	0.72	1.73	0.00	1.18	0.81	3
	σ_2	5.67	7.62	0.69	3.63	0.00	1.21	0.86	1
	σ_3	2.70	4.01	0.48	2.25	0.00	1.18	0.95	5
	σ_4	24.30	31.10	1.25	4.03	0.00	1.27	0.73	2
40	σ_1	28.09	41.38	0.74	2.38	0.00	1.20	0.76	3
	σ_2	33.26	46.61	0.42	1.19	0.00	1.17	0.83	3
	σ_3	17.50	24.18	0.80	2.13	0.00	1.18	0.87	4
	σ_4	64.58	107.58	1.30	2.14	0.13	1.21	0.78	0
50	σ_1	43.05	62.84	0.30	0.87	0.00	1.15	0.73	1
	σ_2	40.43	55.88	0.41	1.37	0.00	1.16	0.84	2
	σ_3	19.55	25.37	0.28	1.50	0.00	1.19	0.94	6
	σ_4	128.90	203.24	1.10	2.25	0.00	1.18	0.76	1
60	σ_1	127.81	205.69	0.61	1.54	0.00	1.17	0.72	1
	σ_2	117.85	202.06	1.04	1.86	0.22	1.13	0.79	0
	σ_3	59.00	92.81	0.70	1.57	0.01	1.17	0.88	0
	σ_4	234.28	418.74	1.42	2.68	0.49	1.18	0.77	0
80	σ_1	244.65	478.08	0.95	2.02	0.25	1.14	0.71	0
	σ_2	289.18	570.80	0.48	1.43	0.07	1.11	0.78	0
	σ_3	197.93	278.11	0.34	1.04	0.00	1.15	0.85	2
	σ_4	563.20	1249.07	1.29	2.55	0.47	1.15	0.75	0
100	σ_1	625.20	1363.70	0.44	1.34	0.06	1.11	0.76	0
	σ_2	499.86	938.33	0.53	1.33	0.00	1.13	0.83	1
	σ_3	291.80	408.78	0.22	0.46	0.00	1.16	0.93	1
	σ_4	1245.92	2916.67	1.26	2.02	0.61	1.11	0.73	0

Table 2 points to the performance comparison between CG–LB/VR–UB and AW–LB/AW–UB. The "Times" columns report the average CPU times for computing the column generation based lower bound (CG–LB) and the time needed to reach a completely integral solution by the rounding/fixing procedure (VR–UB), including of the time for computing CG–LB. Instances from classes σ_1, σ_4 turn out to be computationally harder than the easy σ_3 instances, with σ_2 somehow in the middle; this conforms to what is reported in literature about similar problems. From the point of view of solution quality, both CG–LB and VR–UB strongly dominate AW–LB and AW–UB: CG–LB is 10% to 40% higher than AW–LB, whereas VR–UB improves up to 25% on VR–UB. This does not come as a surprise, since AW–LB and AW–UB are cheap, quick-and-dirty components for a more complex branch and bound method. What really points to the added value of CG–LB + VR–UB is the very narrow optimality gap that can be certified for instances up to $n = 100$ jobs; such gap in no case raised above 5%, usually being much smaller. On the other hand the computation time required by such procedures is probably not acceptable for a heuristic algorithm on large (say $n > 50$ jobs) instances.

Algorithm 6 (ER–UB) is specifically considered in order to overcome this performance issue; as explained in Section 3.2 the price to pay is that of no longer having a valid lower bound at the end of the computation. The experimental results show that solutions delivered by ER–UB retain most of the high-quality features of the solutions delivered by VR–UB, while leading to drastic savings in computation times. This means that the (relaxation of the) restricted master problem can provide valuable information for building a good feasible solution from the very first stages of the optimization. In the tests, the algorithm's parameters were set to $steps = 3$ and $threshold = 0.1\%$ after a few sample trials, without attempting a fine calibration. In Table 3, where the gaps are evaluated against CG–LB, is shown that such gaps for ER–UB are generally not unacceptably higher than those of VR–UB, with much smaller CPU times.

Table 3
Comparison between VR–UB and ER–UB on the Weighted Parallel Batching model. Reference lower bound for gap evaluation is CG–LB in both cases.

Param		Times (s)		VR–UB Gap (%)			ER–UB Gap (%)		
n	σ	VR–UB	ER–UB	avg	worst	best	avg	worst	best
20	σ_1	1.55	**0.90**	**0.46**	3.84	**0.00**	0.82	**3.70**	**0.00**
	σ_2	1.18	**0.83**	**0.39**	**2.11**	**0.00**	1.29	3.49	**0.00**
	σ_3	0.82	**0.60**	**0.31**	**2.24**	**0.00**	0.42	2.52	**0.00**
	σ_4	4.05	**1.30**	1.09	2.53	**0.00**	**1.04**	**2.50**	**0.00**
40	σ_1	41.38	**14.67**	**0.74**	**2.38**	**0.00**	1.79	3.79	0.20
	σ_2	46.61	**13.75**	**0.42**	**1.19**	**0.00**	1.62	2.85	0.62
	σ_3	24.18	**12.40**	**0.80**	**2.13**	**0.00**	0.88	2.58	0.08
	σ_4	107.58	**15.84**	**1.30**	**2.14**	**0.13**	1.81	3.44	0.94
60	σ_1	205.69	**48.54**	**0.61**	**1.54**	**0.00**	1.21	3.01	0.31
	σ_2	202.06	**47.67**	**1.04**	**1.86**	**0.22**	1.59	3.13	0.35
	σ_3	92.81	**45.19**	**0.70**	**1.57**	**0.01**	0.84	2.45	0.10
	σ_4	418.74	**56.35**	**1.42**	**2.68**	**0.49**	2.34	3.82	1.62
80	σ_1	478.08	**129.63**	**0.95**	2.02	**0.25**	1.02	**1.93**	0.35
	σ_2	570.80	**116.23**	**0.48**	**1.43**	**0.07**	1.31	2.17	0.33
	σ_3	278.11	**112.99**	**0.34**	**1.04**	**0.00**	0.53	1.12	0.12
	σ_4	1249.07	**148.11**	**1.29**	**2.55**	**0.47**	2.50	3.05	1.21
100	σ_1	1363.70	**282.04**	**0.44**	**1.34**	**0.06**	0.89	2.24	0.14
	σ_2	938.33	**280.70**	**0.53**	1.33	**0.00**	0.59	**1.15**	0.13
	σ_3	408.78	**260.11**	0.22	**0.46**	**0.00**	**0.21**	0.54	0.02
	σ_4	2916.67	**302.78**	**1.26**	**2.02**	**0.61**	1.89	3.07	1.27

Results for VR–UB and ER–UB on an Extended Set of instances (25 per each σ class instead of 10) are reported in Table 4. This further analysis certifies, on a bigger instance pool, what already happened in

Table 4
Comparison between VR–UB and ER–UB on the Weighted Parallel Batching model on the Extended Set. Reference lower bound for gap evaluation is CG–LB in both cases.

Param		Times (s)		VR–UB Gap (%)			ER–UB Gap (%)		
n	σ	VR–UB	ER–UB	avg	worst	best	avg	worst	best
20	σ_1	1.81	**1.23**	**0.49**	**2.61**	**0.00**	2.02	9.61	**0.00**
	σ_2	2.60	**1.61**	**0.79**	**5.82**	**0.00**	1.96	6.19	**0.00**
	σ_3	1.59	**1.17**	**0.56**	**4.61**	**0.00**	0.90	4.79	**0.00**
	σ_4	5.42	**1.77**	**1.07**	**4.67**	**0.00**	1.64	6.52	**0.00**
40	σ_1	37.44	**12.81**	**0.89**	**2.82**	**0.00**	1.37	4.03	0.20
	σ_2	48.32	**15.03**	**0.77**	**3.54**	**0.00**	1.55	3.69	0.22
	σ_3	17.88	**12.04**	**0.55**	**2.85**	**0.00**	0.83	3.27	**0.00**
	σ_4	86.01	**15.34**	**1.52**	**3.51**	**0.01**	2.62	6.81	0.54
60	σ_1	176.55	**52.82**	**0.80**	**2.26**	**0.00**	1.04	2.36	0.34
	σ_2	163.18	**50.68**	**0.56**	**1.35**	**0.00**	1.21	3.57	0.30
	σ_3	98.46	**50.03**	**0.60**	2.16	**0.00**	0.66	**1.74**	**0.00**
	σ_4	407.89	**60.13**	**1.58**	4.11	**0.67**	2.34	**3.90**	1.25
80	σ_1	627.62	**147.53**	**0.72**	**2.11**	**0.10**	1.02	2.57	0.13
	σ_2	522.87	**145.79**	**0.65**	2.31	**0.00**	0.85	**1.58**	0.23
	σ_3	269.40	**128.21**	**0.43**	**1.51**	**0.00**	0.53	1.53	0.01
	σ_4	1127.97	**155.38**	**1.21**	**2.55**	**0.17**	2.21	4.18	0.75
100	σ_1	1536.97	**354.50**	**0.54**	**1.17**	**0.17**	0.79	1.47	0.27
	σ_2	1128.45	**280.00**	**0.52**	**1.12**	**0.02**	0.81	1.76	0.21
	σ_3	680.02	**277.80**	0.37	1.27	**0.00**	**0.35**	**1.10**	**0.00**
	σ_4	2399.67	**465.23**	**1.20**	**2.21**	**0.37**	1.92	3.15	0.91

Table 5
Comparison between VR–UB and ER–UB on the Weighted Parallel Batching model on the Extended Set, considering job size distribution $\sigma_1 : s_j \in [1, 10]$ with batch capacity $b = 10$ and job distribution $\sigma_5 : s_j \in [1, 50]$ with batch capacity $b = 50$. Reference lower bound for gap evaluation is CG–LB in both cases.

Param			Times (s)		VR–UB Gap (%)			ER–UB Gap (%)		
n	b	σ	VR–UB	ER–UB	avg	worst	best	avg	worst	best
20	10	σ_1	1.81	**1.23**	**0.49**	**2.61**	**0.00**	2.02	9.61	**0.00**
	50	σ_5	2.98	**1.62**	**0.54**	**4.70**	**0.00**	1.41	7.28	**0.00**
40	10	σ_1	37.44	**12.81**	**0.89**	**2.82**	**0.00**	1.37	4.03	0.20
	50	σ_5	52.64	**18.03**	**0.72**	**2.86**	**0.00**	1.47	3.26	0.22
60	10	σ_1	176.55	**52.82**	**0.80**	**2.26**	**0.00**	1.04	2.36	0.34
	50	σ_5	321.99	**70.51**	**0.94**	**2.15**	**0.00**	1.29	3.80	0.23
80	10	σ_1	627.62	**147.53**	**0.72**	**2.11**	**0.10**	1.02	2.57	0.13
	50	σ_5	817.38	**184.49**	**0.77**	2.74	**0.00**	0.95	**1.88**	0.20
100	10	σ_1	1536.97	**354.50**	**0.54**	**1.17**	**0.17**	0.79	1.47	0.27
	50	σ_5	1684.46	**372.23**	**0.71**	**1.88**	**0.09**	1.04	2.43	0.33

Table 3: algorithm ER–UB runs in significantly smaller CPU times, while algorithm VR–UB gives slightly more accurate gaps. It is worth noting that, especially with the increase in number of jobs n, the percentage difference in gaps between VR–UB and ER–UB is very small.

In Table 5 a new job size distribution, $\sigma_5 : s_j \in [1, 50]$, and a new batch capacity, $b = 50$, are introduced. The $(\sigma_5, b = 50)$ combination is very similar to the $(\sigma_1, b = 10)$ one, because it can be seen as a "rescaling" of values over two different ranges (remember that $\sigma_1 : s_j \in [1, 10]$). Obviously this increase in job size granularity and in max capacity has an impact over time/space complexity of the algorithm, but it is not as high as one could have feared. For example in the case $n = 100$ the average time increase of VR–UB is 9.6% and of ER–UB is 5.0%. On the other hand, the gap quality of the two algorithms seems to not suffer much from the change in values of σ and b, especially with increasing number n of jobs. This is probably due to the fact that the two combinations are very similar except a "rescaling" of values.

5. Conclusions

The $1|p\text{-batch}, s_j|\sum_j w_j C_j$ problem can be effectively tackled by means of column generation techniques. The CG–LB lower bound obtained by solving the relaxed arc-based model (4)–(7′) is, to the writers' knowledge, the sharpest bound currently available for this problem.

The proposed bounding procedure generates arcs (columns) to be added to a restricted master problem via dynamic programming. The resulting restricted master problems are large but still manageable by an LP solver like CPLEX. The same bound could be obtained using a path-based model for the problem, like (17)–(20′), resulting in more compact restricted master problems; anyway some more efficient pricing procedure is needed in order to speed up the optimization for such model.

The computation time required for computing CG–LB is still too high for using it into a branch and price exact procedure, but a simple rounding strategy turns out to be quite effective in building heuristic solutions whose value are within a few percentage from the optimum, with optimality gap certified by CG–LB. In order to speed up the heuristic procedure, variable rounding can be performed well before complete convergence to the relaxed optimum of the restricted master problem. This looses the availability of a valid lower bound at the end of the heuristic procedure, but computational experience shows that in practice the quality of the heuristic solution is not severely affected by this early rounding.

Whereas the arc-based columns generation model allows to generate good heuristic solutions in reasonable computation times, it seems unlikely that, with the current performances, it can be embedded into an exact branch and price procedure: computing CG–LB is still computationally heavy for that kind of application. Some directions for future research are currently under consideration. Among others, the InitCols procedure for initializing the column set in the restricted master is not very sophisticated; significant speedups could be obtained being more aggressive in generating "good" columns in this startup phase. Also, an enhanced (exact or heuristic) pricing procedure for the path-based model could pave the way for interesting developments.

CRediT authorship contribution statement

Alessandro Druetto: Conceptualization, Methodology, Software, Writing. **Andrea Grosso:** Conceptualization, Methodology, Software, Writing.

References

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice hall.

Alfieri, A., Druetto, A., Grosso, A., Salassa, F., 2019. Column generation for minimizing total completion time on a single machine with parallel batching. IFAC-PapersOnLine 52 (13), 969–974, 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

Alfieri, A., Druetto, A., Grosso, A., Salassa, F., 2021. Column generation for minimizing total completion time in a parallel-batching environment. J. Sched. 24 (6), 569–588.

Azizoglu, M., Webster, S., 2000. Scheduling a batch processing machine with non-identical job sizes. Int. J. Prod. Res. 38 (10), 2173–2184.

Baptiste, P., 2000. Batching identical jobs. Math. Methods Oper. Res. (ZOR) 52 (3), 355–367.

Desrosiers, J., Lübbecke, M., 2005. A primer in column generation. In: Desaulniers, G., Desrosiers, J., Solomon, M. (Eds.), Column Generation. Springer, Boston, MA.

Dobson, G., Nambimadom, R.S., 2001. The batch loading and scheduling problem. Oper. Res. 49 (1), 52–65.

Emde, S., Polten, L., Gendreau, M., 2020. Logic-based benders decomposition for scheduling a batching machine. Comput. Oper. Res. 113, 104777.

Fang, Y., Lu, X., 2016. Online parallel-batch scheduling to minimize total weighted completion time on single unbounded machine. Inform. Process. Lett. 116 (8), 526–531.

Gondran, M., Minoux, M., Vajda, S., 1984. Graphs and Algorithms. John Wiley & Sons, Inc., USA.

Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A., 1979. Optimization and approximation in deterministic sequencing and scheduling : a survey. Ann. Discrete Math. 5, 287–326.

Hulett, M., Damodaran, P., Amouie, M., 2017. Scheduling non-identical parallel batch processing machines to minimize total weighted tardiness using particle swarm optimization. Comput. Ind. Eng. 113, 425–436.

Kong, M., Liu, X., Pei, J., Pardalos, P.M., Mladenovic, N., 2020a. Parallel-batching scheduling with nonlinear processing times on a single and unrelated parallel machines. J. Global Optim. 78 (4), 693–715.

Kong, M., Liu, X., Pei, J., Zhou, Z., Pardalos, P.M., 2020b. Parallel-batching scheduling of deteriorating jobs with non-identical sizes and rejection on a single machine. Optim. Lett. 14 (4), 857–871.

Kovalyov, M.Y., Šešok, D., 2019. Two-agent scheduling with deteriorating jobs on a single parallel-batching machine: refining computational complexity. J. Sched. 22 (5), 603–606.

Li, S., 2017. Approximation algorithms for scheduling jobs with release times and arbitrary sizes on batch machines with non-identical capacities. European J. Oper. Res. 263 (3), 815–826.

Li, S., Li, G., Qi, X., 2006. Minimizing total weighted completion time on identical parallel batch machines. Internat. J. Found Comput. Sci. 17 (06), 1441–1453.

Liao, B., Song, Q., Pei, J., Yang, S., Pardalos, P.M., 2020. Parallel-machine group scheduling with inclusive processing set restrictions, outsourcing option and serial-batching under the effect of step-deterioration. J. Global Optim. 78 (4), 717–742.

Liu, J., Li, Z., Chen, Q., Mao, N., 2016. Controlling delivery and energy performance of parallel batch processors in dynamic mould manufacturing. Comput. Oper. Res. 66, 116–129.

Malapert, A., Guéret, C., Rousseau, L.-M., 2012. A constraint programming approach for a batch processing problem with non-identical job sizes. European J. Oper. Res. 221 (3), 533–545.

Mathirajan, M., Sivakumar, A.I., 2006. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor. Int. J. Adv. Manuf. Technol. 29 (9), 990–1001.

Mönch, L., Fowler, J.W., Dauzère-Pérès, S., Mason, S.J., Rose, O., 2011. A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. J. Sched. 14 (6), 583–599.

Mönch, L., Fowler, J.W., Mason, S.J., 2012. Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems, Vol. 52. Springer Science & Business Media.

Mönch, L., Unbehaun, R., 2007. Decomposition heuristics for minimizing earliness - tardiness on parallel burn-in ovens with a common due date. Comput. Oper. Res. 34 (11), 3380–3396.

Mourgaya, M., Vanderbeck, F., 2007. Column generation based heuristic for tactical planning in multi-period vehicle routing. European J. Oper. Res. 183 (3), 1028–1041.

Muter, İ., 2020. Exact algorithms to minimize makespan on single and parallel batch processing machines. European J. Oper. Res. 285 (2), 470–483.

Ozturk, O., 2020. A truncated column generation algorithm for the parallel batch scheduling problem to minimize total flow time. European J. Oper. Res. 286 (2), 432–443.

Ozturk, O., Begen, M.A., Zaric, G.S., 2017. A branch and bound algorithm for scheduling unit size jobs on parallel batching machines to minimize makespan. Int. J. Prod. Res. 55 (6), 1815–1831.

Ozturk, O., Espinouse, M.-L., Mascolo, M.D., Gouin, A., 2012. Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates. Int. J. Prod. Res. 50 (20), 6022–6035.

Pei, J., Song, Q., Liao, B., Liu, X., Pardalos, P.M., 2021. Parallel-machine serial-batching scheduling with release times under the effects of position-dependent learning and time-dependent deterioration. Ann. Oper. Res. 298 (1–2), 407–444.

Pei, J., Wei, J., Liao, B., Liu, X., Pardalos, P.M., 2020. Two-agent scheduling on bounded parallel-batching machines with an aging effect of job-position-dependent. Ann. Oper. Res. 294 (1–2), 191–223.

Potts, C.N., Kovalyov, M.Y., 2000. Scheduling with batching: A review. European J. Oper. Res. 120 (2), 228–249.

Takamatsu, T., Hashimoto, I., Hasebe, S., 1979. Optimal scheduling and minimum storage tank capacities in a process system with parallel batch units. Comput. Chem. Eng. 3 (1–4), 185–195.

Uzsoy, R., 1994. Scheduling a single batch processing machine with non-identical job sizes. Int. J. Prod. Res. 32 (7), 1615–1635.

Uzsoy, R., Yang, Y., 1997. Minimizing total weighted completion time on a single batch processing machine. Prod. Oper. Manage. 6, 57–73.

Zhang, J., Yao, X., Li, Y., 2020. Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing. Int. J. Prod. Res. 58 (8), 2263–2282.