Column generation for minimizing total completion time in a parallel-batching environment

Journal of Scheduling
https://doi.org/10.1007/s10951-021-00703-9

Column generation for minimizing total completion time in a parallel-batching environment

A. Alfieri[1] · A. Druetto[2] · A. Grosso[2] · F. Salassa[1]

Accepted: 10 August 2021

Abstract
This paper deals with the $1|p-\text{batch}, s_j \leq b|\sum C_j$ scheduling problem, where jobs are scheduled in batches on a single machine in order to minimize the total completion time. A *size* is given for each job, such that the total size of each batch cannot exceed a fixed capacity b. A graph-based model is proposed for computing a very effective lower bound based on linear programming; the model, with an exponential number of variables, is solved by column generation and embedded into both a heuristic *price and branch* algorithm and an exact branch and price algorithm. The same model is able to handle parallel-machine problems like $Pm|p-\text{batch}, s_j \leq b|\sum C_j$ very efficiently. Computational results show that the new lower bound strongly dominates the bounds currently available in the literature, and the proposed heuristic algorithm is able to achieve high-quality solutions on large problems in a reasonable computation time. For the single-machine case, the exact branch and price algorithm is able to solve all the tested instances with 30 jobs and a good amount of 40-job examples.

Keywords Price and branch · Column generation · Parallel batching · Scheduling

1 Introduction

In manufacturing system management, capacity is a key factor in matching supply with demand, i.e., having a system that is able to produce what is needed to satisfy customer demand.

Several factors negatively impact system capacity. The most frequently studied ones are those related to system balancing and to part batching when setup times are present, as severe bottlenecks and/or small batches can significantly reduce system capacity, thus leading to the inability of the manufacturing system to respond in a timely way to the market demand (Cachon and Terwiesch 2012).

Batches induced by setup times are called serial batches, and although they are very important in manufacturing systems, they are not the only type of batches that can be present on the shop floor. Transfer batches and parallel batches can also be found in manufacturing systems, the first being related to the capacity of the material handling resources, and the second, as the serial ones, to the capacity of the machines.

Although both serial and parallel batches are related to and affect machine capacity, their nature is very different. Serial batches are due to the presence of setup times, while parallel batches stem from the ability of machines to accommodate and manufacture several jobs at the same time. They are less studied than serial and transfer batches because they are less frequent; however, they are no less important.

Specifically, parallel batches can be found in many manufacturing processes where heating operations are necessary, such as in mold manufacturing (Liu et al. 2016) and the semiconductor industry (Mönch et al. 2012), or when there are sterilization phases (Ozturk et al. 2012), just to cite a few examples.

In all these cases, operations take quite a long time, and the machines are usually batch machines that can accommodate several parts and process them simultaneously and

✉ A. Druetto
alessandro.druetto@unito.it

A. Alfieri
arianna.alfieri@polito.it

A. Grosso
andrea.grosso@unito.it

F. Salassa
fabio.salassa@polito.it

[1] Department of Management and Production Engineering, Politecnico di Torino, Turin, Italy

[2] Department of Computer Science, Università di Torino, Turin, Italy

Published online: 01 October 2021

exactly to virtually share the long processing time among all the parts processed at the same time. Each part has an individual size, and batch machines (e.g., batch oven for heating treatments or autoclaves for sterilization operations) have a limited capacity; therefore, the number of parts that can be in a single batch is limited.

Due to the limited capacity of the batch machine, and thus to the limited number of parts that can be accommodated in it, when several jobs have to be processed on the batch machine, they have to be partitioned in several batches. When batches have been created, their processing has to be scheduled on the machine, and this decision is obviously intertwined with batch creation. Moreover, the two decisions (i.e., how to create batches and how to sequence them on the batch machines) strictly depend on the objective of the shop floor manager (minimizing the number of tardy jobs, minimizing the maximum delay, reducing the total flow time, maximizing the machine utilization, etc.).

In this paper, the above-described parallel-batch problem is considered. Specifically, given a set of jobs that are all available at the same time, the paper addresses how to partition them in batches and how to sequence batches on machines, with the objective of minimizing the total completion time.

With respect to the current literature, the problem addressed in the paper is the same problem as in Rafiee Parsa et al. (2016), with the main difference that it is extended to the parallel-machine case. Following the three-field notation of Graham et al. (1979), the problems studied in the paper are $1|p-\text{batch}, s_j \leq b|\sum C_j$ and $Pm|p-\text{batch}, s_j \leq b|\sum C_j$.

A fundamental work in the field of parallel-batch processor scheduling is that of Uzsoy (1994), where a single-batch processing machine problem is studied with regard to makespan and total flow time criteria. In particular, in this work, nonidentical job sizes are taken into account, and complexity results are also provided.

A large part of the literature on parallel batching is devoted to the minimization of the makespan criterion—e.g., Damodaran et al. (2006), Dupont and Dhaenens-Flipo (2002), Rafiee Parsa et al. (2010), Li (2017) and Muter (2020)—while the total flow time problems have been less studied (Jolai Ghazvini and Dupont 1998; Rafiee Parsa et al. 2016).

The work in Jolai Ghazvini and Dupont (1998) and a modified version of the genetic algorithm presented in Damodaran et al. (2006) have been used as benchmark procedures for the hybrid max-min ant system presented in Rafiee Parsa et al. (2016). Different objective functions dealing with tardiness and lateness have also been addressed (e.g., Wang 2011; Malapert et al. 2012; Kosch and Beck 2014; Cabo et al. 2015).

Other recent works on single- and parallel-machine batching problems are Beldar and Costa (2018), Jia et al. (2018) and Ozturk et al. (2017). In Ozturk et al. (2017), the authors address a problem with unit size jobs and maximum completion time objective. In Beldar and Costa (2018) and Jia et al. (2018), instead, the total completion time criterion is considered, although Jia et al. (2018) consider the weighted version and equal processing time for all jobs, while in Beldar and Costa (2018) only the single-machine case is tackled, with constraints on cardinality and size of job batches.

A very recent contribution based on column generation is Ozturk (2020), where the fundamental model is a time-indexed formulation, with a set-partition problem as master, followed by a constructive heuristic that combines parts of the relaxed formulation.

The contribution of this paper is threefold.

- A new graph-based model for $1|p-\text{batch}, s_j \leq b|\sum C_j$ is developed; such a model induces a very large linear program with an exponential number of variables, which can be handled by standard column generation techniques. The pricing step is efficiently solved by dynamic programming. The new model provides the strongest linear relaxation currently available in the literature for the studied problems.
- A heuristic procedure of the so-called price and branch type—following the terminology of Desrosiers and Lübbecke (2011)—relying on the graph model is developed. This procedure allows one to generate high-quality solutions (with certified optimality gaps) for fairly large instances in short computation times. The column generation procedure is also embedded in an exact branch and price procedure that is able to deliver optimal solutions for $1|p-\text{batch}, s_j \leq b|\sum C_j$ instances with up to 40 jobs—previous state-of-the-art results in Azizoglu and Webster (2000) were limited to 25 jobs.
- The graph-based model and the related column generation procedure and heuristic are also extended to multi-machine problems—$Pm|p-\text{batch}, s_j \leq b|\sum C_j$, $Rm|p-\text{batch}, s_j \leq b|\sum C_j$, etc. Computational experience shows that $Pm|p-\text{batch}, s_j \leq b|\sum C_j$ for up to five machines and 100 jobs is solvable by the proposed approach with no significant loss of solution quality with respect to the single-machine case.

The remainder of the paper is structured as follows. The column generation approach for the $1|p-\text{batch}, s_j \leq b|\sum C_j$ problem is developed in Sect. 2, while Sect. 3 presents the extension of the approach to the parallel-machine case, with special attention to the case with identical parallel machines. Computational results are reported in Sect. 4. Section 5 concludes the paper, discussing directions for future research.

2 Single-machine models

This section deals with the single-machine problem. The new graph-based model is established in Sect. 2.1; the column generation procedure that solves its continuous relaxation is developed in Sect. 2.2, while in Sect. 2.3 a "price and branch" heuristic is formulated. The same algorithmic elements allow us to formulate an exact branch and price method in Sect. 2.4.

In the remainder of the paper, the following notation is used. $N = \{1, 2, \ldots, n\}$ denotes the set of jobs to be scheduled; for each job $j \in N$, its processing time p_j and its size s_j, both integers, are given. The machine has a given integer capacity denoted by b. When a subset of jobs is packed in a batch B, $p_B = \max\{p_j : j \in B\}$ is used to indicate the batch processing time. Every batch B is required to have $\sum_{j\in B} s_j \le b$. The machine processes the jobs in a batch sequence $S = (B_1, B_2, \ldots, B_t)$, where each job j in the kth batch B_k shares the batch completion time: $C_j = C_{B_k} = \sum_{l=1}^{k} p_{B_l}\ \forall j \in B_k$. The $1|p-\text{batch}, s_j \le b|\sum C_j$ problem calls for creating the batches and sequencing them in order to minimize $f(S) = \sum_{j\in N} C_j$.

The problem is known to be NP-hard (Uzsoy 1994). A mixed-integer linear programming (MILP) model for this problem, as given by Rafiee Parsa et al. (2016), is as follows, where the variable $x_{jk} = 1$ if job j is scheduled in the kth batch; the model always arranges the jobs in n batches $(B_1, B_2, \ldots, B_n)$, some of which can be empty. Other variables of the model are the completion time C_{B_k} of the kth batch—note that the model allows for empty batches; the variable p_{B_k} that represents the processing time of the kth batch; and the variable C_j corresponding to the completion time of job j.

$$\text{minimize} \quad \sum_{j=1}^{n} C_j \tag{1}$$

$$\text{subject to} \sum_{k=1}^{n} x_{jk} = 1 \qquad j = 1, \ldots, n \tag{2}$$

$$\sum_{j=1}^{n} s_j x_{jk} \le b \qquad k = 1, \ldots, n \tag{3}$$

$$p_{B_k} \ge p_j x_{jk} \qquad j = 1, \ldots, n,\ k = 1, \ldots, n \tag{4}$$

$$C_{B_1} \ge p_{B_1} \tag{5}$$

$$C_{B_k} \ge C_{B_{k-1}} + p_{B_k} \qquad k = 2, \ldots, n \tag{6}$$

$$C_j \ge C_{B_k} - M\left(1 - x_{jk}\right) \qquad j = 1, \ldots, n,\ k = 1, \ldots, n \tag{7}$$

$$p_{B_k}, C_{B_k}, C_j \ge 0 \qquad j = 1, \ldots, n,\ k = 1, \ldots, n \tag{8}$$

$$x_{jk} \in \{0, 1\} \qquad j = 1, \ldots, n,\ k = 1, \ldots, n \tag{9}$$

The total completion time is expressed by (1). Constraint set (2) ensures that each job is assigned to exactly one batch, and since all the jobs assigned to a batch cannot exceed the batch capacity, constraint set (3) has to be defined. Constraint set (4) represents the fact that the processing time of a batch is the maximum processing time of all the contained jobs. The completion time for the first batch is simply its processing time, since it is the first to be processed by the machine, as stated in constraint (5). Constraint set (6), instead, ensures that the completion time for each of the other batches is evaluated as the sum of its processing time and the completion time of the previous batch. Constraint set (7) specifies that the completion time of a job must be the completion time of the corresponding batch (the constant M must be very large).

Model (1)–(9) is known to be very weak. A state-of-the-art solver like CPLEX can waste hours over 15-job instances, with optimality gaps of 100% at the root branching node.

2.1 A new problem formulation

This paper proposes a new model where a batch sequence is represented as a path on a graph. Let $\mathcal{B} = \{B \subseteq N : \sum_{j\in B} s_j \le b\}$ be the set of all possible batches. Define a (multi)graph $G(V, A)$ with vertex and arc sets as follows

$$V = \{1, 2, \ldots, n+1\},$$
$$A = \{(i, k, B) :\ i, k \in V; i < k; B \in \mathcal{B}; |B| = (k - i)\}.$$

Each arc in A is a triple (i, k, B) with head k and tail i and an associated batch B with $(k - i)$ jobs; it will represent the batch B scheduled in a batch sequence such that exactly $n-i+1$ jobs are scheduled from batch B up to the end of the sequence. For each arc (i, k, B), a cost is defined as $c_{ikB} = (n - i + 1)p_B$, with $p_B = \max\{p_j : j \in B\}$, and where the $(n - i + 1)$ factor precisely models the abovementioned $(n - i + 1)$ jobs to the end of the sequence. Note that in G, a path

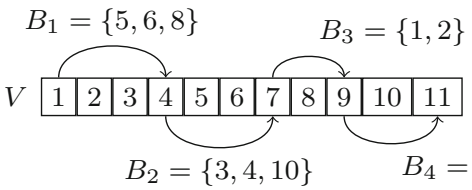
$$P = [(i_1, k_1, B_1), (i_2, k_2, B_2), \ldots, (i_r, k_r, B_r)]\ \ (i_\ell = k_{\ell-1})$$

connecting nodes from i_1 to k_r has the property

$$\sum_{\ell=1}^{r} |B_\ell| = \sum_{\ell=1}^{r} (k_\ell - i_\ell) = \sum_{\ell=1}^{r} k_\ell - \sum_{\ell=1}^{r} i_\ell$$
$$= \sum_{\ell=1}^{r} k_\ell - \sum_{\ell=2}^{r} k_{\ell-1} - i_1 = k_r - i_1, \tag{10}$$

Property 1 highlights the relationship between feasible batches and paths of the above-defined graph.

Property 1 *Each feasible batch sequence S corresponds to a path P in $G(V, A)$ from 1 to $n + 1$ such that the set*



$$N = \{1, 2, \ldots, 10\}$$

$$P = [(1, 4, B_1), (4, 7, B_2), (7, 9, B_3), (9, 11, B_4)] \iff S = (B_1, B_2, B_3, B_4)$$

$$\begin{aligned}\sum_{j=1}^{10} C_j(S) &= \sum_{\kappa=1}^{4} C_{B_\kappa}(S)|B_\kappa| \\ &= |B_1|p_{B_1} + \\ &\quad |B_2|(p_{B_1} + p_{B_2}) + \\ &\quad |B_3|(p_{B_1} + p_{B_2} + p_{B_3}) + \\ &\quad |B_4|(p_{B_1} + p_{B_2} + p_{B_3} + p_{B_4}) \\ &= 10p_{B_1} + 7p_{B_2} + 4p_{B_3} + 2p_{B_4} = \sum_{(i,k,B)\in P} (n - i + 1)p_B = \sum_{(i,k,B)\in P} c_{ikB}.\end{aligned}$$

Fig. 1 Batch sequence as a path on a graph. The job set N is partitioned over the arcs of P. $C_j(S)$ is the completion time of job j induced by sequence S. The path provides information about partitioning the jobs into batches B_1, B_2, B_3, B_4 and their ordered sequence

of jobs N is partitioned over the arcs in P, and $f(S) = \sum\{c_{ikB} : (i, k, B) \in P\}$.

Proof Refer to Fig. 1 for an illustration of the idea. The one-to-one mapping of batch sequences to paths is easily established. Given a batch sequence $S = (B_1, B_2, \ldots, B_t)$, the path

$$P = [(i_1, k_1, B_1), (i_2, k_2, B_2), \ldots, (i_t, k_t, B_t)] \tag{11}$$

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

$$\begin{aligned} &i_1 = 1, \qquad k_1 = i_1 + |B_1| \\ &i_\ell = k_{\ell-1}, \quad k_\ell = i_\ell + |B_\ell| \qquad \text{for } \ell = 2, \ldots, t \end{aligned} \tag{12}$$

Note that $k_t - i_1 = \sum_{\ell=1}^{t} |B_\ell| = n$, and hence $k_t = n+1$, and P is a path from 1 to $n + 1$ 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.

On the other hand, given a path P like (11) from node 1 to node $n + 1$ of G, such that $\{B : (i, k, B) \in P\}$ forms a partition of N, it can be easily seen that the arcs of P satisfy (12) by 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, $S = (B_1, B_2, \ldots, B_t)$ with the $B_1, \ldots, B_t$ defined by the arc-batches of P is a feasible batch sequence.

It remains to be proven that $f(S) = \sum_{(i,k,B)\in P} c_{ikB}$. For an arc (i_q, k_q, B_q) in position q on P, the number of jobs scheduled from B_q to B_t is $\sum_{\ell=q}^{t} |B_\ell| = k_t - i_q = (n + 1 - i_q)$. The objective function for the batch sequence is

$$\begin{aligned} f(S) &= \sum_{j=1}^{n} C_j = \sum_{\ell=1}^{t} C_{B_\ell}|B_\ell| = \sum_{\ell=1}^{t} \left[\left(\sum_{\kappa=1}^{\ell} p_{B_\kappa} \right) |B_\ell| \right] \\ &= |B_1|p_{B_1} + \\ &\quad |B_2|p_{B_1} + |B_2|p_{B_2} + \\ &\quad |B_3|p_{B_1} + |B_3|p_{B_2} + |B_3|p_{B_3} + \\ &\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &\quad |B_t|p_{B_1} + |B_t|p_{B_2} + |B_t|p_{B_3} + \ldots + |B_t|p_{B_t}. \end{aligned}$$

Adding by column, it can be rewritten as:

$$\begin{aligned} f(S) &= p_{B_1} \sum_{\ell=1}^{t} |B_\ell| + p_{B_2} \sum_{\ell=2}^{t} |B_\ell| + p_{B_3} \sum_{\ell=3}^{t} |B_\ell| \\ &\quad + \cdots + p_{B_t}|B_t| \\ &= \sum_{q=1}^{t} p_{B_q}(n - i_q + 1) = \sum_{(i,k,B)\in P} c_{ikB}. \end{aligned}$$

□

Example Figure 2 shows an example of a full graph with all possible batches. In this small four-job example, $p_1, \ldots, p_4 = 42, 37, 21, 16$, $s_1, \ldots, s_4 = 3, 4, 6, 8$ and $b = 10$, and batches can contain one or two jobs at most; note how the graph size is already quite large, even for this small example.

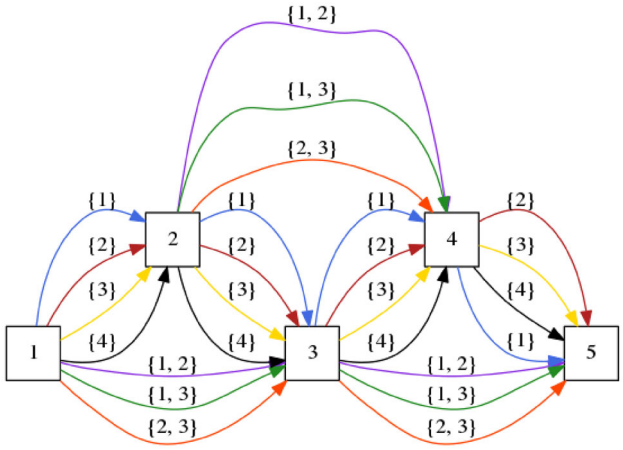


Fig. 2 Example of a full graph with $b = 10$ and four jobs. Job sizes are $s_1 = 3, s_2 = 4, s_3 = 6, s_4 = 8$

A path from node 1 to node 5 in this graph represents a feasible schedule if the job set $\{1, 2, 3, 4\}$ is partitioned over the arcs of the path. For example, the path $P = [(1, 2, \{4\}), (2, 4, \{1, 3\}), (4, 5, \{2\})]$ represents the feasible batch sequence $S = (\{4\}, \{1, 3\}, \{2\})$. In such sequence, the reader can easily compute that $C_4 = 16$, $C_1 = C_3 = 58$, $C_2 = 95$ and $C_1 + C_2 + C_3 + C_4 = 227$; in the path P, the arc costs are, by definition, $c_{(1,2,\{4\})} = 4p_4 = 64$, $c_{(2,4,\{1,3\})} = 3p_1 = 126$, $c_{(4,5,\{2\})} = p_2 = 37$, and the total cost of P is $c_{(1,2,\{4\})} + c_{(2,4,\{1,3\})} + c_{(4,5,\{2\})} = 227$.

On the other hand, a path like $P' = [(1, 2, \{3\}), (2, 4, \{1, 3\}), (4, 5, \{3\})]$ does not represent a feasible batch sequence since it fails to partition the job set $\{1, 2, 3, 4\}$ over its arcs.

By Property 1, then, an optimal solution for the $1|p - \text{batch}, s_j \le b|\sum C_j$ problem can be computed by identifying on the very large graph $G(V, A)$ a minimum-cost path from node 1 to node $n + 1$ such that the job set N is exactly partitioned over the "B" components of the arcs in that path. This problem can be modeled by a very large linear program that includes features of a shortest path/minimum cost flow model as well as partition constraints, as follows. Let $\boldsymbol{a_B} \in \{0, 1\}^n$ be the incidence column-vector of job set B, whose jth component $(\boldsymbol{a_B})_j = 1$ if $j \in B$, and $\mathbf{1} = (1, 1, \ldots, 1)^{\mathsf{T}}$ an all-ones column-vector with n components. Define binary decision variables x_{ikB} for each $(i, k, B) \in A$, so that $x_{ikB} = 1$ if arc (i, k, B) is on the optimal path from 1 to $(n + 1)$. The linear program is written as follows.

$$\text{minimize} \quad \sum_{(i,k,B)\in A} c_{ikB} x_{ikB} \tag{13}$$

$$\text{subject to} \sum_{\substack{(k,B):\\(i,k,B)\in A}} x_{ikB} - \sum_{\substack{(k,B):\\(k,i,B)\in A}} x_{kiB} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \ldots, n \\ -1 & i = n + 1 \end{cases} \tag{14}$$

$$\sum_{(i,k,B)\in A} \boldsymbol{a_B} x_{ikB} = \mathbf{1} \tag{15}$$

$$x_{ikB} \in \{0, 1\} \qquad (i, k, B) \in A \tag{16}$$

The objective function (13) together with the flow conservation constraints (14) is a classical formulation of the shortest path problem as a special case of a single-source single-sink minimum-cost flow linear program (see for example Ahuja et al. 1993). The vector expression of constraint (15) represents a group of n set-partitioning constraints on the job set $N = \{1, 2, \ldots, n\}$, enforcing the requirement that the job set is exactly partitioned over the arcs selected to be in the path—in scalar form, $\sum_{(i,k,B)\in A} (\boldsymbol{a_B})_j x_{ikB} = 1 \; \forall j \in N$.

2.2 Continuous relaxation for the new graph-based formulation: column generation

The continuous relaxation of (13)–(16), where the integrality constraints (16) are relaxed to

$$x_{ikB} \ge 0 \qquad (i, k, B) \in A, \tag{16'}$$

is solved by means of a column generation procedure. Model (13)–(16') is the master problem: a restricted master problem is made of a subset $A' \subset A$ of arcs. Introducing dual variables $u_1, u_2, \ldots, u_{n+1}$ for constraints (14) and $v_1, \ldots, v_n$ for constraints (15), the dual of (13)–(16') is

$$\text{maximize} \quad u_1 - u_{n+1} + \sum_{j=1}^{n} v_j \tag{17}$$

$$\text{subject to} u_i - u_k + \sum_{j\in B} v_j \le c_{ikB} \qquad (i, k, B) \in A. \tag{18}$$

Solving the restricted master problem leads to a basic feasible solution for the master problem and values for dual variables/simplex multipliers $\boldsymbol{u}, \boldsymbol{v}$. Pricing the arcs $(i, k, B) \in A$ corresponds to finding the most violated dual constraints (18). The strategy developed in this paper is to price the arcs separately for each pair of indices i, k with $i < k$, therefore determining minimum (possibly negative) reduced costs

$$\begin{aligned} \bar{c}_{ikB^*} &= \min_B \left\{ c_{ikB} - (u_i - u_k) - \sum_{j\in B} v_j : \sum_{j\in B} s_j \le b, \; |B| = (k - i) \right\} \\ &= \min_B \left\{ p_B(n - i + 1) - \sum_{j\in B} v_j : \sum_{j\in B} s_j \le b, \; |B| = (k - i) \right\} \\ &\quad - (u_i - u_k). \end{aligned} \tag{19}$$

For fixed indices i, k with $i < k$, the $(u_i - u_k)$ part of (19) is constant, and the cardinality of B is also fixed at $|B| = k - i$.

Finding the batch B that minimizes (19) for each given pair of indices i, k with $i < k$ and given batch processing time p_B can be done by exploiting the dynamic programming state space of a family of cardinality-constrained knapsack problems where items correspond to jobs. Assume that the jobs are indexed by longest processing time (LPT) order, so that

$$p_1 \geq p_2 \geq \cdots \geq p_n.$$

Define, for $r = 1, \ldots, n$,

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

where $g_r(\tau, \ell)$ is the optimal value of a knapsack with profits v_j and sizes s_j, limited to items/jobs $r, r+1, \ldots, n$, total size $\leq \tau$ and cardinality ℓ. Variable y_j is set to 1, i.e., $y_j = 1$, if item/job j is included in the solution.

Optimal values for $g_r(\tau, \ell)$ can be recursively computed (see Kellerer et al. 2004) as

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

with boundary conditions

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

$$g_r(\tau, 0) = 0 \qquad r = 1, \ldots, n, \ \tau = 0, \ldots, b$$

$$g_r(\tau, \ell) = -\infty \qquad \text{if } \ell > n - r + 1 \text{ or } \tau < 0.$$

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 $g_r(\tau, \ell)$ is sufficient for pricing all relevant arcs.

Property 2 *Let $L = \{1\} \cup \{j > 1 : p_j < p_{j-1}\}$. For any given pair of indices i, k with $i < k$, an arc with minimum reduced cost (i, k, B^*) is one of*

$$(i, k, B_r(b, k - i)) \qquad r \in L. \tag{20}$$

Proof Every arc (i, k, B) can be shown to have a reduced cost not less than some of the arcs in (20). Let $\bar{c}_{ikB} = (n - i + 1)p_B - \sum_{j \in B} v_j - (u_i - u_k)$ be the reduced cost of an arc (i, k, B). Recall that $|B| = k - i$, 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 $B \subseteq \{r, r+1, \ldots, n\}$ and $r \in L$. 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,

$$\bar{c}_{ikB} = (n - i + 1)p_B - \textstyle\sum_{j \in B} v_j - (u_i - u_k) \geq$$
$$\geq (n - i + 1)p_{B_r} - g_r(b, k - i) - (u_i - u_k) = \bar{c}_{ikB_r}.$$

□

Algorithm 1 Pricing procedure.

```
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:     Set H = ∅;                      ▷ Set of negative-reduced cost arcs
4:     for ℓ = 1, ..., n do
5:         Set r := 1, done := false;
6:         while not done do
7:             Retrieve g_r(b, ℓ) and B = B_r(b, ℓ);
8:             for i = 1, ..., n − ℓ + 1 do
9:                 Set k = i + ℓ;
10:                Compute c̄_ikB = p_B(n − i + 1) − (u_i − u_k) − g_r(b, ℓ);
11:                if c̄_ikB < 0 then
12:                    Set H := H ∪ {(i, k, B)};
13:                end if
14:            end for
15:            Set r := min{j : p_j < p_r};
16:            If no such index exists, set done := true;
17:        end while
18:    end for
19:    return H;
20: end function
```

All the relevant arcs with minimum reduced cost can be generated by the procedure reported in Algorithm 1 (NewCols).

The size of the state space required for the pricing is bounded by $\mathcal{O}(n^2 b)$, while the pricing procedure can have two bottlenecks:

(a) the $\mathcal{O}(n^3)$ effort due to the three nested loops on lines 4, 6, 8. The *while* on line 6 can be executed n times in the worst case;
(b) filling the state space $g_r(\tau, \ell)$, which requires at most $\mathcal{O}(n^2 b)$ arithmetic operations. A *memorized* 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.

Because of these two possible bottlenecks, the running time of NewCols is bounded from above by $\mathcal{O}(\max(n^3, n^2 b))$.

Algorithm 2 Generation of initial arcs

```
function InitCols(N, b)
    Sort jobs in N such that p_1 ≤ p_2 ≤ ··· ≤ p_n;
    Set H ← ∅                                   ▷ Set of initial arcs
    for j := 1, ..., n do
        Set B = {j};
        for h := j + 1, ..., n do
            if Σ_{j∈B} s_j + s_h ≤ b then
                Set B := B ∪ {h};
                Set H ← H ∪ {(i, i + |B|, B) : i = 1, ..., n − |B| + 1};
            end if
        end for
    end for
    return H;
end function
```

Algorithm 3 Price and branch procedure

```
1: A' ← InitCols(N, b);
2: G(V, A') ← restricted master problem
3: while true do
4:     z ← continuous optimum of G(V, A')
5:     u, v ← optimal multipliers
6:     H ← NewCols(N, b, u, v)
7:     if |H| = 0 then
8:         CG − LB ← z — continuous optimum, lower bound
9:         CG − UB ← integer solution computed over G(V, A') ▷ Use
   branch and bound
10:        break
11:    end if
12:    A' ← A' ∪ H
13: end while
```

2.3 Heuristic procedure: price and branch

The column generation described in the previous section is used to solve the continuous relaxation of the master problem. Once the relaxed optimum has been found, the resulting restricted master problem is taken, the variables are set to binary type, and the resulting MILP is solved using CPLEX in order to obtain a heuristic solution for the master. This is often called "price and branch," as opposed to the exact approach of branch and price.

In order to generate the initial column set, the jobs are sorted in order of shortest processing time (SPT), and all possible arcs with feasible batches composed of SPT-consecutive jobs are generated (Algorithm 2, InitCols).

The complete heuristic procedure is sketched in Algorithm 3.

2.4 Exact approach: branch and price

Given the strong relaxation from the column generation procedure, a natural step is trying to embed it in an exact algorithm—branch and price. The main issue in this step is being able to preserve the pricing problem structure at every node in the search tree. Trying to use the classical branching scheme from Foster and Ryan (1976), which leverages the partitioning constraints forcing pairs of jobs to always/never be batched together, would require handling disjunctive constraints in the pricing problem. This would make the latter a strongly NP-hard disjunctive knapsack problem, ruling out the possibility of using the dynamic programming procedure of Algorithm 1 (unless P=NP).

Still, branching can be performed on a compact formulation of the problem, building the batch sequence by scheduling one job at a time starting from the first position. The basic branching mechanism adopted here is the same as described in Uzsoy (1994) and Azizoglu and Webster (2000). Let $S = (B_1, B_2, \ldots, B_t)$ be a partial (possibly empty) batch sequence built at the current search node, and $\hat{N} = N \setminus (B_1 \cup B_2 \cup \cdots \cup B_t)$ the set of unscheduled jobs at such node. If the node is not fathomed by bound, two types of branching can take place.

(I) A new unscheduled job $j \in \hat{N}$ is added to B_t, provided that $\sum_{i \in B_t} s_i + s_j \leq b$.
(II) Batch B_t is closed, and a new one B_{t+1} is started, choosing its longest job among the $j \in \hat{N}$.

New jobs are added to the open batch in non-increasing order of processing time; hence, if a job j has been added to B_t, no other job j' with $p_{j'} > p_j$ will enter the same batch in successive branches. Also, batch B_t is closed only if it is maximal: as far jobs that can be added to B_t without exceeding the capacity b, this will prevent type II branches from the current node.

The column generation-based relaxation is solved at each node of the search tree; batches from the partial batch sequence $(B_1, B_2, \ldots, B_{t-1})$ correspond, in the relaxation, to arc-variables fixed at 1. At non-root nodes, the "open" batch B_t at the end of the partial sequence is handled by imposing constraints on the state space to be searched in the pricing step. The following can be observed:

- Only items corresponding to jobs in $j \in \hat{N}$ concur to form the state space.
- For pricing arcs (i, k, B) with $i = \sum_{\kappa=1}^{t-1} |B_\kappa|$, $B \supseteq B_t$—these are arcs extending the "open" batch at the tail of the partial sequence—the items in B_t are preloaded in the knapsack; hence, only states $g_r(\tau, \ell)$ with capacity $\tau \leq (b - \sum_{i \in B_t} s_i)$ and index $r > \max\{i : i \in B_t\}$ are solved.
- For all other arcs, a pricing with full capacity b is performed.

The nodes in the search tree are expanded in depth-first order. Feasible solutions are generated at the root node by running the price and branch procedure in Algorithm 3, and at the leaves of the search tree when the batch sequence is completed. Although Algorithm 3 implies running a branch

and bound itself, it is very fast for the tested problem sizes and allows us to achieve the highest-quality feasible solutions. Running quick and dirty heuristics at the intermediate nodes did not significantly improve the performance during preliminary testing.

3 Parallel-machine models

Model (13)–(16) is readily extended to parallel-machine cases. Consider the fairly general $Rm|p-\text{batch}, s_j \le b|\sum C_j$ problem with m parallel unrelated machines. Let p_{jh} be the processing time of job j on machine h. A special type of arc with empty batches is added to the graph developed for the single-machine case, using the arc set

$$A = \{(i,k,B):\ i,k \in V; i < k; B \in \mathcal{B}; |B| = (k-i)\} \cup \\ \cup \{(1,k,\emptyset): k = 2,\dots,n+1\}.$$

Arcs $(i,k,B) \in A$ are given machine-dependent costs $c^h_{ikB} = p_{Bh}(n-i+1)$, with $p_{Bh} = \max\{p_{jh}: j \in B\}$. Empty arcs $(1,k,\emptyset)$ are given costs $c^h_{1k\emptyset} = 0, k = 2,\dots,n+1, h = 1,\dots,m$.

Remark Now Eq. (10) holds, in G, only for paths of non-empty arcs connecting nodes from i_1 to k_r, i.e., for

$$P = [(i_1,k_1,B_1),(i_2,k_2,B_2),\dots,(i_r,k_r,B_r)] \\ \text{with } B_1 \neq 0$$

$\sum_{l=1}^{r} |B_l| = k_r - i_1$ holds. Note that at most the first arc in a path can be empty (if $i_1 = 1$ and $B_1 = \emptyset$).

Empty arcs are all added to the restricted master problem from the beginning, so they do not need to be considered in the dynamic programming pricing procedure. A feasible solution is composed of m batch sequences

$$S_h = (B^h_1,\dots,B^h_{t_h}), \qquad h = 1,\dots,m \tag{21}$$

processed by the m machines. Such batch sequences correspond to m paths (one path for each machine) from node 1 to $n+1$ on the non-empty arcs of which the set of jobs is exactly partitioned. Such paths will have an empty arc as first arc. Note that if (i,k,B) is on the hth path, this means that $n-i+1$ jobs will be scheduled from B to the end of the hth batch sequence.

Property 1 can be easily extended to the multi-machine case. Figure 3 shows a sketch of the proof with $m = 2$. The empty arcs act as placeholders.

The idea is formalized as follows.

Property 3 *Each feasible set of batch sequences $\{S_h\}_{h=1}^m$ corresponds to a set of paths $\{P_h\}_{h=1}^m$ such that the job set N is partitioned over the non-empty arcs of $\{P_h\}_{h=1}^m$, and $\sum_{j\in N} C_j(S_1,\dots,S_m) = \sum_{h=1}^m \sum_{(i,k,B)\in P_h} c^h_{ikB}$.*

Proof A one-to-one mapping between feasible solutions $\{S_h\}_{h=1}^m$ and collections of paths $\{P_h\}_{h=1}^m$ is quickly established.

Suppose that a feasible collection of batch sequences $S_1,\dots,S_m$ is given. Take a machine h and its batch sequence $S_h = (B_1,\dots,B_t)$—the dependence on h is dropped from the batch notation in order to keep it simple. The batch sequence S_h can be empty (i.e., the solution leaves machine h idle) or not.

If $S_h = \emptyset$, define $P_h = [(1,n+1,\emptyset)]$ with a single empty arc.

If $S_h \neq \emptyset$, by definition of the arc set A, the following arcs belong to the graph G:

$$\begin{array}{llll} (i_t,k_t,B_t) & k_t = n+1, & i_t = k_t - |B_t| & \\ (i_\ell,k_\ell,B_\ell) & k_\ell = i_{\ell+1}, & i_\ell = k_\ell - |B_\ell| & \ell = t-1,\dots,1 \\ (i_0,k_0,\emptyset) & k_0 = i_1, & i_0 = 1 & \text{if } i_1 > 1. \end{array}$$

If $i_1 = 1$, arc $(i_0,k_0,\emptyset)$ is omitted. The arcs are chosen so that $k_\ell = i_{\ell+1}, k_t = n+1$, and the first arc has a tail in node 1.

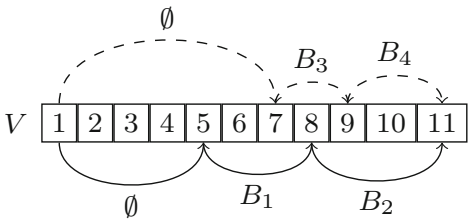
In both cases addressed above, P_h identifies a path from node 1 to node $n+1$. Repeat the construction for each machine $h = 1,\dots,m$ to obtain the collection of paths $P_1,\dots,P_m$; the partition of the job set over the non-empty chosen arcs is guaranteed by the fact that the batches in $\{S_h\}_{h=1}^m$ already form a partition of N by hypothesis.

On the other hand, given a collection of paths $P_1,\dots,P_m$, all connecting node 1 to node $n+1$, with the job set N partitioned over the non-empty arcs of such collection, take a machine index h, and let $P_h = [(i_0,k_0,B_0),(i_1,k_1,B_1),\dots,(i_t,k_t,B_t)]$. The batch sequence S_h is defined by $S_h = (B_1,\dots,B_t)$ if $B_0 = \emptyset$; otherwise $S_h = (B_0,\dots,B_t)$. Note that, by definition of the arc set A, at most B_0 can be empty. If $P_h = [(1,n+1,\emptyset)]$, $S_h = \emptyset$, and machine h is left idle. Repeat for each machine $h = 1,\dots,m$ in order to obtain a batch sequence for each machine. The feasibility of $S_1,\dots,S_m$ is guaranteed by the fact that by hypothesis, the job set N is partitioned over the set of non-empty arcs of $P_1,\dots,P_m$.

It remains to be proven that $\sum_{j=1}^n C_j(S_1,\dots,S_m) = \sum_{h=1}^m \sum_{(i,k,B)\in P_h} c^h_{ikB}$. To this end, it is sufficient to prove that on each machine h,

$$\sum_{B\in S_h}\sum_{j\in B} C_j = \sum_{(ikB)\in P_h} c_{ikB}$$

then, $\sum_{j\in N} C_j = \sum_{h=1}^m \sum_{B\in S_h}\sum_{j\in B} C_j$, and the result follows. Consider machine h and sequence S_h with the corresponding path $P_h = [(i_0,k_0,B_0),(i_1,k_1,B_1),\dots,(i_t,k_t,B_t)]$, with $i_0 = 1, k_t = n+1$. For the first arc (i_0,k_0,B_0), either $B_0 = \emptyset$ or $B_0 \neq \emptyset$.



$$
\begin{aligned}
N &= \{1, 2, \ldots, 10\} \\
S_1 &= (B_1, B_2) \\
S_2 &= (B_3, B_4) \\
B_1 &= \{5, 6, 8\} \\
B_2 &= \{3, 4, 10\} \\
B_3 &= \{1, 2\} \\
B_4 &= \{7, 9\}
\end{aligned}
$$

$$
\begin{aligned}
P_1 &= [(1, 5, \emptyset), (5, 8, B_1), (8, 11, B_2)] \\
P_2 &= [(1, 7, \emptyset), (7, 9, B_3), (9, 11, B_4)]
\end{aligned}
$$

$$
\begin{aligned}
\sum_{j=1}^{10} C_j &= \sum_{h=1}^{2} \sum_{B \in S_h} |B| C_B \\
&= |B_1| p_{B_1,1} + \\
&\quad |B_2|(p_{B_1,1} + p_{B_2,1}) + \\
&\quad |B_3| p_{B_3,2} + \\
&\quad |B_4|(p_{B_3,2} + p_{B_4,2}) \\
&= \underbrace{6p_{B_1,1} + 3p_{B_2,1}}_{c^1_{5,8,B_1} + c^1_{8,11,B_2}} + \underbrace{4p_{B_3,2} + 2p_{B_4,2}}_{c^2_{7,9,B_3} + c^2_{9,11,B_4}} = \sum_{(i,k,B) \in P_1} c^1_{ikB} + \sum_{(i,k,B) \in P_2} c^2_{ikB}.
\end{aligned}
$$

Fig. 3 Batch sequences on two machines as a collection of two paths on a graph. The job set N is partitioned into B_1, B_2, B_3, B_4. The two paths provide a partition into batches and sequencing information

Assume $B_0 = \emptyset$. Let $p_{B_\ell,h}$ be the processing time of batch B_ℓ on machine h. The algebraic manipulations proceed similarly to the proof of Property 1.

$$
\begin{aligned}
\sum_{B \in S_h} \sum_{j \in B} C_j &= \sum_{q=1}^{t} |B_q| C_{B_q} \\
&= |B_1| p_{B_1,h} + \\
&\quad |B_2| p_{B_1,h} + |B_2| p_{B_2,h} + \\
&\quad |B_3| p_{B_1,h} + |B_3| p_{B_2,h} + |B_3| p_{B_3,h} + \\
&\quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&\quad |B_t| p_{B_1,h} + |B_t| p_{B_2,h} + |B_t| p_{B_3,h} + \ldots + |B_t| p_{B_t,h} \\
&= p_{B_1,h} \sum_{\ell=1}^{t} |B_\ell| + p_{B_2,h} \sum_{\ell=2}^{t} |B_\ell| \\
&\quad + p_{B_3,h} \sum_{\ell=3}^{t} |B_\ell| + \cdots + p_{B_t,h} |B_t| \\
&= \sum_{q=1}^{t} p_{B_q,h}(n - i_q + 1) = \sum_{q=1}^{t} c^h_{i_q k_q B_q} = \sum_{(i,k,B) \in P_h} c^h_{ikB},
\end{aligned}
$$

where the last sum can be extended to all the arcs in P_h, since for the first arc $(1, k_0, \emptyset) \in P_h$, $c^h_{1,i_1,\emptyset} = 0$.

If $B_0 \neq \emptyset$, then, by Eq. (10), $\sum_{\ell=0}^{n} |B_\ell| = k_t - i_0 = n$; hence, all the jobs of the problem are scheduled on machine h while the other machines must be left idle. The analysis is reduced to a single-machine case, and Property 1 ensures that $\sum_{B \in S_h} \sum_{j \in B} C_j = \sum_{(i,k,B) \in P_h} c^h_{ikB}$. □

Model (13)–(16) can be extended to the parallel-machine case using multi-commodity flow constraints.

$$
\text{minimize} \quad \sum_{h=1}^{m} \sum_{(i,k,B) \in A} c^h_{ikB} x^h_{ikB} \tag{22}
$$

$$
\text{subject to} \quad \sum_{\substack{(k,B): \\ (i,k,B) \in A}} x^h_{ikB} - \sum_{\substack{(k,B): \\ (k,i,B) \in A}} x^h_{kiB} = \begin{cases} 1 & i = 1 \\ 0 & i = 2, \ldots, n \\ -1 & i = n + 1 \end{cases} \quad h = 1, \ldots, m \tag{23}
$$

$$
\sum_{h=1}^{m} \sum_{(i,k,B) \in A} \boldsymbol{a_B} x^h_{ikB} = \mathbf{1} \tag{24}
$$

$$
x^h_{ikB} \in \{0, 1\} \qquad (i, k, B) \in A,\ h = 1, \ldots, m \tag{25}
$$

Here, $x^h_{ikB} = 1$ if batch B is on the hth path. Flow conservation constraints (23) require that one unit of each commodity

is routed from node 1 to node $n+1$. Constraints (24) enforce the exact partitioning of the whole job set across the arcs belonging to the m paths.

In the column generation framework, with each restricted master optimum, constraint multipliers are computed:

$$\begin{aligned} &u_1^h, u_2^h, \ldots, u_{n+1}^h && \text{for constraints (23), } h = 1, \ldots, m, \\ &v_1, v_2, \ldots, v_n && \text{for constraints (24).} \end{aligned}$$

The reduced cost is then minimized separately for each combination of pairs of indices i, k with $i < k$ and machine h, searching for arcs (i, k, B) with reduced costs

$$\bar{c}^h_{ikB^*} = \min_B \left\{ p_{Bh}(n-i+1) - \sum_{j\in B} v_j : \sum_{j\in B} s_j \le b,\ |B| = k-i \right\} - (u_i^h - u_k^h).$$

This requires calling NewCols m times, once per machine, since the LPT ordering on each machine is different, and thus so is the state space $g_r(\tau, \ell)$. Hence, the running time for pricing increases to $\mathcal{O}(m \max(n^3, n^2 b))$.

A somewhat better situation arises in the case of *identical* parallel machines, with problem $Pm|p-\text{batch}, s_j \le b|\sum C_j$. Since each job j has the same processing time p_j on every machine, the state space $g_r(\tau, \ell)$ used for pricing is common to all the machines, and a slightly modified version of NewCols can do the entire pricing, still keeping the running time within $\mathcal{O}(\max(n^3, n^2 b))$. The procedure is reported in Algorithm 4. The key observation is that the p_B and $\sum_{j\in B} v_j$ components of the reduced costs c^h_{ikB} are machine-independent, and only the largest difference $\Delta u_{ik} = \max_h\{(u_i^h - u_k^h)\}$ is strictly needed in order to compute the minimum reduced costs. Such largest differences are precomputed in time $\mathcal{O}(mn^2)$ on line 3. For any (i, k, B), let r be the smallest index such that $p_r = p_B$, and let $B_r = B_r(b, k-i)$; then, similarly to what was proved in Property 2:

$$\begin{aligned} \bar{c}^h_{ikB} &= (n-i+1)p_B - \sum_{j\in B} v_j - (u_i^h - u_k^h) \ge \\ &\ge (n-i+1)p_{B_r} - g_r(b, k-i) - (u_i^h - u_k^h) \ge \\ &\ge (n-i+1)p_{B_r} - g_r(b, k-i) - \Delta u_{ik}. \end{aligned}$$

Finally, note that also taking into account different capacities for each machine, or even different job sizes on each machine, simply requires one to specify the knapsack family used. Details are omitted for the sake of concision.

Algorithm 4 Pricing procedure for identical parallel machines.

```
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:     Set Δu_ik = max_h{u_i^h − u_k^h} for 1 ≤ i < k ≤ n+1;   ▷ O(mn^2) time
4:     Set H = ∅;                      ▷ Set of negative-reduced cost arcs
5:     for ℓ = 1, ..., n do
6:         Set r := 1, done := false;
7:         while not done do
8:             Retrieve g_r(b, ℓ) and B = B_r(b, ℓ);
9:             for i = 1, ..., n − ℓ + 1 do
10:                Set k = i + ℓ;
11:                Compute c̄_ikB = p_B(n − i + 1) − Δu_ik − g_r(b, ℓ);
12:                if c̄_ikB < 0 then
13:                    Set H := H ∪ {(i, k, B)};
14:                end if
15:            end for
16:            Set r := min{j : p_j < p_r};
17:            If no such index exists, set done := true;
18:        end while
19:    end for
20:    return H;
21: end function
```

4 Computational results

The proposed algorithms discussed in the previous sections have been tested on randomly generated instances. For generating job data, the same approach as in Uzsoy (1994) and Rafiee Parsa et al. (2016) has been used. Specifically, all job processing times are drawn from a uniform distribution $p_j \in [1, 100]$, while job sizes s_j are drawn from four possible uniform distributions, labeled by $\sigma \in \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$:

$$\begin{aligned} &\sigma_1 : s_j \in [1, 10] && \sigma_3 : s_j \in [3, 10] \\ &\sigma_2 : s_j \in [2, 8] && \sigma_4 : s_j \in [1, 5]\,. \end{aligned}$$

In both Uzsoy (1994) and Rafiee Parsa et al. (2016), the machine capacity is fixed at $b = 10$.

Since the pricing procedure has a pseudo-polynomial running time, instances with $b = 30$ and $b = 50$ have also been generated in order to assess how the procedure behaves with a larger capacity. Single-machine instances have been generated with $n \in \{20, 40, 60, 80, 100\}$ and with all four σ size distributions. For each n, σ and b combinations, 10 random instances have been generated.

With the same job data, the corresponding instances of the parallel-machine problem $Pm|p-\text{batch}, s_j \le b|\sum C_j$ have been solved for $m = 2, 3, 5$ identical machines.

For testing the heuristics in the parallel-machine case and the branch and price exact approach for the single machine, only $b = 10$ instances have been used.

All the tests were run in a Linux environment with an `Intel Core i7-6500U CPU @ 2.50GHz` processor; `C++` language was used for coding the algorithms, and

CPLEX 12.8, called directly from C++ environment using CPLEX callable libraries, was used to solve relaxed and mixed-integer programs. The source code and instances are available.[1]

Results for the price and branch heuristic in both the single-machine and parallel-machine cases are discussed in Sect. 4.1, whereas Sect. 4.2 deals with the results of the exact branch and price procedure in the single-machine case.

4.1 Evaluation of the heuristic algorithms

Both the column generation-based lower bound $\mathtt{CG-LB}$ and the objective value of the heuristic solution $\mathtt{CG-UB}$ have been evaluated. As far as the quality of the lower bound is concerned, the continuous relaxation of model (1)–(9) is not a realistic competitor, zero being the typical value found by CPLEX at the root branching node. A more meaningful comparison can be performed against the combinatorial lower bound proposed by Uzsoy (1994). Such bound is based on a relaxation of $1|p-\text{batch}, s_j \leq b|\sum C_j$ to a preemptive problem on b parallel machines (refer to Uzsoy 1994 for details). This lower bound is referred to as PR in the following.

As far as the evaluation of $\mathtt{CG-UB}$ is concerned, it was difficult to compare the obtained results with the known literature, as neither the test instances nor the computer codes used by Uzsoy (1994) and Rafiee Parsa et al. (2016) have been made available. Hence, some comparisons have been made with the results of Rafiee Parsa et al. (2016), using instances *of the same type,*, but for this reason, the comparison has to be taken with some care. On the other hand, when CPLEX is fed with model (1)–(9) and given some time, its internal heuristics do generate a number of heuristic solutions, although it has no chance of certifying optimality. Hence, CPLEX has been run on some set of instances in order to obtain heuristic solutions with a time limit of 300 s.

The times required to compute $\mathtt{CG-LB}$ and $\mathtt{CG-UB}$ are reported separately. The gap between $\mathtt{CG-UB}$ and $\mathtt{CG-LB}$ is evaluated as

$$gap = \frac{\mathtt{CG-UB} - \mathtt{CG-LB}}{\mathtt{CG-UB}} \cdot 100\%.$$

Single machine

Tables 1, 2 and 3 show the results over an increasing number of jobs with batch capacity $b = 10$, 30 and 50, respectively; the $\mathtt{CG-UB}$ was computed using CPLEX with a time limit of 60 s. Values are shown as the average over each 10-instance group for the time, and as an average, maximum (worse) and minimum (best) over each 10-instance group for the gap. Column *opt* reports the number of instances (out of 10) in which the solution can be certified to be the optimum, i.e., in which $\mathtt{CG-UB} = \mathtt{CG-LB}$. The comparison between the $\mathtt{CG-LB}$ value and Uzsoy's PR lower bound is also reported, computing the average, maximum and minimum over each 10-instance group of the ratio $\mathtt{CG-LB/PR}$.

In Table 1, it can be seen that with $b = 10$, the computation of $\mathtt{CG-LB}$ is *fast,* with average CPU time less than 1 s in almost all cases (i.e., with any number of jobs). The σ_4 instances are the most time-demanding, with the only average computation time above 1 s. This is because a larger set of columns is usually generated on such instances. The computation of $\mathtt{CG-UB}$ is, as expected, the heaviest part of the procedure, with greater CPU time. However, the CPLEX time limit is reached only in the cases $n = 80, 100$ and $\sigma = \sigma_4$. Again, σ_4 instances were the most demanding of CPU time, because of the larger set of columns to be handled. The certified solution quality was very good, with an average optimality gap usually below 1.5%, and only one case ($n = 80$, $\sigma = \sigma_4$) above 5%.

From Table 1, it can be easily seen that $\mathtt{CG-LB}$ performance is much better than PR in every combination, ranging from an average 9% gain when $n = 100$ and $\sigma = \sigma_4$ to an average 29% when $n = 20$ and $\sigma = \sigma_4$. These values also suggest that PR performs better for large n; in fact, when a high number of batches are required in the feasible solutions, the usually weak parallel-machine relaxation of PR becomes slightly stronger.

From Table 2, it can be seen that the CPU time for $\mathtt{CG-LB}$ increases; this is expected, since a larger number of possible batches are generated with an increase in capacity. The larger reduced master problems obviously also affect the computation of $\mathtt{CG-UB}$, which reaches the time limit in all cases for $n = 80, 100$. The average optimality gaps worsen, but the largest increase is not found on σ_4 instances; instead, it affects σ_1 instances more heavily, especially for large n.

Overall, increasing the capacity also increases the distance between the two lower bounds $\mathtt{CG-LB}$ and PR; $\mathtt{CG-LB}$ performs better in every combination, ranging from an average 10% gain when $b = 30$, $n = 100$ and $\sigma = \sigma_3$, to an average of 81% when $b = 30$, $n = 20$ and $\sigma = \sigma_4$. This is reasonable, since PR is based on a preemptive relaxation to b parallel machines, and allowing the splitting of jobs on more machines weakens the relaxation.

Table 3 shows the results of the tests with capacity $b = 50$ that confirm the impact of b. The instances belonging to class σ_4 are still the most computationally demanding, both for lower bounding and heuristic solution. Instances with $\sigma = \sigma_1$ are the worst in terms of solution quality—with the exception of small 20-job instances—but, curiously, the gap decreases on $n = 80$ instances when passing from $b = 30$ to $b = 50$.

[1] https://drive.google.com/drive/folders/10g243gICweI_wNvqDh8vxL8G_OXg-Fk3?usp=sharing compilation and execution parameters are available in the makefile included.

Table 1 Results for CG − UB and CG − LB with $b = 10$

Param		Times (s)		Gap (%)			CG − LB/PR			
n	σ	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	σ_1	0.01	0.04	1.30	3.20	0.00	1.25	1.19	1.31	2
	σ_2	0.01	0.03	1.55	3.63	0.00	1.22	1.20	1.27	2
	σ_3	0.01	0.03	0.63	3.30	0.00	1.19	1.15	1.21	7
	σ_4	0.01	0.09	2.15	4.63	0.82	1.29	1.23	1.37	0
40	σ_1	0.03	0.58	1.30	2.34	0.20	1.20	1.16	1.25	0
	σ_2	0.03	0.39	1.17	2.14	0.24	1.16	1.13	1.19	0
	σ_3	0.02	0.18	0.89	1.87	0.00	1.18	1.13	1.27	1
	σ_4	0.07	1.89	2.61	4.03	0.45	1.19	1.15	1.22	0
60	σ_1	0.14	8.34	0.91	2.03	0.23	1.17	1.12	1.21	0
	σ_2	0.12	1.62	0.98	1.90	0.34	1.13	1.10	1.15	0
	σ_3	0.05	0.44	0.49	1.09	0.00	1.16	1.13	1.20	1
	σ_4	0.39	30.41	2.57	3.97	0.94	1.14	1.12	1.16	0
80	σ_1	0.41	7.96	0.74	1.88	0.28	1.14	1.11	1.17	0
	σ_2	0.36	24.06	0.82	1.40	0.07	1.11	1.09	1.13	0
	σ_3	0.19	1.89	0.47	0.85	0.17	1.15	1.12	1.19	0
	σ_4	0.88	limit	5.78	10.77	2.20	1.11	1.10	1.12	0
100	σ_1	0.73	8.76	0.46	0.82	0.06	1.13	1.11	1.14	0
	σ_2	0.45	4.03	0.41	0.75	0.14	1.14	1.11	1.16	0
	σ_3	0.32	0.81	0.17	0.68	0.00	1.15	1.12	1.20	2
	σ_4	1.61	limit	4.44	7.66	1.34	1.09	1.08	1.10	0

Table 2 Results for CG − UB and CG − LB with $b = 30$

Param		Times (s)		Gap (%)			CG − LB/PR			
n	σ	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	σ_1	0.02	0.07	1.03	3.69	0.00	1.46	1.36	1.66	3
	σ_2	0.02	0.08	1.28	3.77	0.00	1.39	1.29	1.49	5
	σ_3	0.01	0.05	1.10	5.46	0.00	1.35	1.30	1.40	4
	σ_4	0.02	0.09	0.00	0.00	0.00	1.81	1.62	2.11	10
40	σ_1	0.21	3.11	3.13	6.43	0.21	1.30	1.23	1.39	0
	σ_2	0.18	1.72	3.83	5.62	2.33	1.27	1.21	1.33	0
	σ_3	0.10	2.95	4.37	6.29	2.98	1.20	1.17	1.26	0
	σ_4	0.71	2.19	1.18	5.13	0.07	1.51	1.41	1.60	0
60	σ_1	0.77	limit	6.78	10.27	3.51	1.22	1.18	1.27	0
	σ_2	0.72	37.44	5.21	8.46	1.88	1.20	1.17	1.22	0
	σ_3	0.42	30.35	3.74	5.79	1.44	1.15	1.13	1.18	0
	σ_4	2.38	10.59	2.28	4.24	0.05	1.36	1.31	1.41	0
80	σ_1	1.68	limit	11.05	17.40	3.40	1.21	1.16	1.23	0
	σ_2	1.41	limit	11.68	41.32	2.65	1.16	1.13	1.20	0
	σ_3	0.88	limit	7.94	12.24	3.53	1.12	1.11	1.13	0
	σ_4	4.75	limit	7.01	18.67	2.41	1.29	1.27	1.32	0
100	σ_1	3.27	limit	15.43	18.54	12.12	1.16	1.13	1.19	0
	σ_2	2.87	limit	9.23	11.46	3.08	1.13	1.12	1.14	0
	σ_3	1.79	limit	11.66	15.45	8.87	1.10	1.09	1.11	0
	σ_4	8.85	limit	6.38	9.54	3.32	1.26	1.23	1.29	0

Table 3 Results for CG − UB and CG − LB with $b = 50$

Param		Times (s)		Gap (%)			CG − LB/PR			
n	σ	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	σ_1	0.02	0.09	0.00	0.00	0.00	1.70	1.48	1.94	10
	σ_2	0.02	0.09	0.10	0.38	0.00	1.65	1.53	1.77	7
	σ_3	0.02	0.08	0.28	2.66	0.00	1.52	1.35	1.62	7
	σ_4	0.02	0.10	0.00	0.00	0.00	2.13	1.90	2.39	10
40	σ_1	0.35	1.29	1.58	3.58	0.00	1.44	1.38	1.51	2
	σ_2	0.44	1.31	1.78	3.16	0.35	1.41	1.34	1.54	0
	σ_3	0.30	1.09	2.24	4.82	0.00	1.34	1.29	1.40	1
	σ_4	0.75	2.45	0.11	0.81	0.00	1.80	1.67	1.87	8
60	σ_1	1.39	23.59	4.47	6.64	1.24	1.38	1.30	1.44	0
	σ_2	1.45	8.38	2.54	5.54	0.08	1.32	1.29	1.37	0
	σ_3	0.76	13.37	4.30	6.65	1.81	1.23	1.20	1.27	0
	σ_4	6.72	12.56	1.25	2.73	0.00	1.52	1.43	1.64	3
80	σ_1	3.43	limit	7.40	10.26	3.17	1.32	1.27	1.38	0
	σ_2	3.71	limit	3.95	4.64	2.63	1.26	1.24	1.27	0
	σ_3	1.93	limit	4.94	10.32	2.00	1.19	1.17	1.20	0
	σ_4	18.34	limit	1.90	4.85	0.11	1.46	1.42	1.51	0
100	σ_1	7.09	limit	15.66	58.23	8.14	1.23	1.19	1.27	0
	σ_2	7.04	limit	7.19	10.33	4.27	1.21	1.19	1.23	0
	σ_3	3.50	limit	7.86	11.47	3.51	1.15	1.14	1.17	0
	σ_4	38.65	limit	3.26	5.70	0.84	1.39	1.35	1.47	0

The worst average gap is reached with $n = 100$ and $\sigma = \sigma_1$ (15.66%). Also, PR worsens considerably with respect to CG − LB.

Rafiee Parsa et al. (2016) provide a hybrid max-min ant system (HMMAS) that is, to the writers' knowledge, a state-of-the-art heuristic. A very recent paper from the same authors has been published on the same problem (Rafiee Parsa et al. 2019) where a new approach, a hybrid neural network (HNN), is proposed. In this new paper, a comparison with HMMAS is presented and, as in the previous case, only capacity $b = 10$ is considered; a statistical analysis is also presented. The quality of the results with the new procedure HNN is not better than with HMMAS; on the contrary, the average quality appears to be slightly worse. Neither the source code nor the tested instances appear to be currently available even though this new paper was recently published; hence, an attempt to compare CG − UB with HMMAS was made by testing CG − UB on generated instances of the same type and size as those used in Rafiee Parsa et al. (2016). Moreover, as in Rafiee Parsa et al. (2016), only the capacity $b = 10$ was investigated; the comparison performed was limited to such values. The reader being warned of the difficulty of such comparison, Table 4 points to the following situation: The results show that the performance of CG − UB, evaluated against Uzsoy's lower bound PR, seems to be very similar to that of HMMAS. Thus, it can be speculated that the two algorithms could give similar results for the upper bound when

Table 4 Comparison between HMMAS (values from Rafiee Parsa et al. 2016) and CG − UB algorithms

Param		Heuristic/PR	
n	σ	HMMAS	CG-UB
20	σ_1	1.25	1.27
	σ_2	1.25	1.24
	σ_3	1.21	1.20
	σ_4	1.28	1.31
40	σ_1	1.19	1.21
	σ_2	1.19	1.18
	σ_3	1.18	1.19
	σ_4	1.20	1.22
60	σ_1	1.17	1.18
	σ_2	1.16	1.14
	σ_3	1.18	1.17
	σ_4	1.18	1.17
80	σ_1	1.16	1.15
	σ_2	1.16	1.12
	σ_3	1.16	1.15
	σ_4	1.16	1.18
100	σ_1	1.16	1.13
	σ_2	1.15	1.14
	σ_3	1.15	1.16
	σ_4	1.15	1.14

Table 5 Comparison between CPLEX-UB (300 s) and CG − UB

Param		CPLEX-UB Gap (%)				CG − UB Gap (%)			
n	σ	Avg	Worst	Best	#Win	Avg	Worst	Best	#Win
20	σ_1	1.15	3.73	0.00	8	1.31	3.27	0.00	6
	σ_2	1.17	3.58	0.00	7	1.56	3.73	0.00	7
	σ_3	0.69	3.30	0.00	8	0.63	3.30	0.00	10
	σ_4	3.51	7.14	0.82	4	2.15	4.63	0.82	9
40	σ_1	9.60	14.63	5.97	0	1.30	2.34	0.20	10
	σ_2	9.34	16.02	5.43	0	1.17	2.14	0.24	10
	σ_3	4.41	8.13	2.34	0	0.89	1.87	0.00	10
	σ_4	12.03	18.16	9.09	0	2.61	4.03	0.45	10
60	σ_1	49.28	77.33	38.61	0	0.91	2.03	0.23	10
	σ_2	48.86	59.69	33.98	0	0.98	1.90	0.34	10
	σ_3	33.78	41.74	22.84	0	0.49	1.09	0.00	10
	σ_4	45.86	66.77	24.59	0	2.57	3.97	0.94	10
80	σ_1	73.77	88.55	59.86	0	0.74	1.88	0.28	10
	σ_2	66.12	77.45	53.45	0	0.82	1.40	0.07	10
	σ_3	52.79	73.82	38.40	0	0.47	0.85	0.17	10
	σ_4	84.09	102.96	62.95	0	5.78	10.77	2.20	10

they are run on the same instance set. The availability of CG − LB allows us to confirm a narrower optimality gap for CG − UB.

The quality of CG − UB has been compared to the quality of the heuristic solution reached by CPLEX (CPLEX-UB) after 300 s of computation using model (1)–(9). The CPLEX optimality gap is generally well above 90% because the lower bound is zero or almost zero. Regardless, using the proposed stronger lower bound, a more realistic optimality gap can be computed for CPLEX as

$$\frac{\text{CPLEX-UB} - \text{CG} - \text{LB}}{\text{UB}^*} \cdot 100\%,$$
$$\text{with } \text{UB}^* = \min\{\text{CPLEX-UB}, \text{CG} - \text{UB}\}.$$

The gap for CG − UB is recomputed as (CG − UB−CG − LB)/ UB* · 100% for uniformity.

The comparison is reported in Table 5, in terms of average, worst and best gap, as in the previous comparison. Column #win counts the number of instances (out of 10) for which each algorithm achieves the best solution. In the case of a draw, a "win" is counted for both, so the two columns can sum to more than 10. Instances with $n = 20$, 40, 60, 80 and $b = 10$ have been tested. CPLEX ran for the full 300 s on all instances, without proving optimality for any of them. CG − UB ran with the same 60 s time limit as in Table 1. Basically, except for the small $n = 20$ instances, the CPLEX solution is consistently worse than CG − UB.

Parallel machines

With the same data, the $Pm|p-\text{batch}, s_j \le b|\sum C_j$ problem has been solved with $m = 2$, 3 and 5 machines. The tests have been limited to the case $b = 10$. The time limit for the branch and bound phase of the heuristic was raised to 180 s. The results are reported in Tables 6, 7 and 8. Apparently, increasing the number of machines has a very mild impact on the CPU time needed for computing the lower bound. The growth of the computational cost is much higher for the branch and bound phase, but with a certain variability on the four classes of instances, with classes σ_1 and σ_4 exhibiting the largest growth. Again, class σ_4 broke the time limit in all the instances. The quality of the solution, as measured by the percentage gap, does not suffer seriously, except for the case $n = 100$, $m = 5$, $\sigma = \sigma_4$. The worst average gap of 14.09% is caused by a single instance with a very large gap of 81.62%; if a larger but still acceptable time limit of 300 s is allowed, the average gap for this class decreases to 4.63% (max gap 12.56%).

Uzsoy's bound PR is easily extended to the parallel-machine case, allowing a relaxation to mb parallel machines. Tables 6, 7 and 8 also compare CG − LB with PR extended to the parallel-machine case. The ratio between the two bounds is apparently unaffected by the increase in m.

4.2 Evaluation of the branch and price approach

The branch and price exact algorithm has been tested on the single-machine $b = 10$ generated instances. The reference algorithm for the exact approaches on $1|p-\text{batch}, s_j \le b|$

Table 6 Results for CG − UB and CG − LB with $b = 10$ and 2 parallel machines

Param		Times (s)		Gap (%)			CG − LB/PR			
n	σ	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	σ_1	0.03	0.12	0.78	2.00	0.00	1.24	1.18	1.29	2
	σ_2	0.03	0.09	1.43	4.78	0.00	1.21	1.19	1.25	2
	σ_3	0.02	0.07	0.57	3.04	0.00	1.18	1.14	1.19	7
	σ_4	0.04	0.22	2.08	4.11	0.00	1.28	1.22	1.35	1
40	σ_1	0.06	0.82	1.14	2.28	0.01	1.19	1.16	1.24	0
	σ_2	0.05	0.90	1.08	1.73	0.23	1.16	1.13	1.18	0
	σ_3	0.03	0.22	0.75	1.39	0.00	1.18	1.13	1.26	1
	σ_4	0.13	5.01	2.07	3.73	0.37	1.18	1.15	1.22	0
60	σ_1	0.25	10.22	0.83	1.64	0.23	1.16	1.12	1.20	0
	σ_2	0.21	1.72	0.90	1.56	0.32	1.12	1.10	1.15	0
	σ_3	0.13	0.60	0.38	1.00	0.01	1.16	1.13	1.20	0
	σ_4	0.49	76.71	2.35	3.72	1.69	1.14	1.12	1.16	0
80	σ_1	0.62	45.96	0.76	1.94	0.21	1.14	1.10	1.17	0
	σ_2	0.54	40.97	0.71	1.00	0.08	1.11	1.09	1.13	0
	σ_3	0.35	3.29	0.46	0.76	0.18	1.14	1.12	1.18	0
	σ_4	1.30	limit	5.65	10.31	1.33	1.11	1.10	1.12	0
100	σ_1	1.08	19.16	0.44	0.77	0.07	1.13	1.11	1.14	0
	σ_2	0.79	3.66	0.41	0.73	0.12	1.13	1.11	1.16	0
	σ_3	0.59	1.63	0.21	0.66	0.00	1.15	1.11	1.20	2
	σ_4	2.25	limit	7.19	26.27	1.08	1.09	1.08	1.10	0

Table 7 Results for CG − UB and CG − LB with $b = 10$ and 3 parallel machines

Param		Times (s)		Gap (%)			CG − LB/PR			
n	σ	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	σ_1	0.03	0.13	0.52	1.29	0.00	1.24	1.18	1.29	3
	σ_2	0.03	0.10	1.15	2.90	0.00	1.21	1.19	1.25	2
	σ_3	0.03	0.08	0.45	2.49	0.00	1.18	1.14	1.19	7
	σ_4	0.05	0.30	1.77	3.61	0.00	1.30	1.23	1.36	1
40	σ_1	0.12	1.68	1.11	2.02	0.00	1.19	1.16	1.24	1
	σ_2	0.09	1.00	0.88	1.38	0.00	1.16	1.13	1.18	1
	σ_3	0.06	0.43	0.80	1.64	0.00	1.17	1.12	1.25	1
	σ_4	0.19	4.81	1.76	3.37	0.19	1.19	1.16	1.22	0
60	σ_1	0.35	3.89	0.73	1.54	0.24	1.16	1.12	1.20	0
	σ_2	0.32	3.47	0.86	1.50	0.29	1.12	1.10	1.15	0
	σ_3	0.24	1.09	0.38	1.12	0.00	1.15	1.13	1.19	1
	σ_4	0.62	76.82	2.00	3.92	1.01	1.15	1.12	1.16	0
80	σ_1	1.04	16.93	0.62	1.57	0.18	1.14	1.10	1.16	0
	σ_2	0.75	47.57	0.71	1.12	0.08	1.11	1.09	1.12	0
	σ_3	0.58	3.99	0.47	0.72	0.19	1.14	1.12	1.18	0
	σ_4	1.49	limit	5.23	13.93	1.20	1.12	1.10	1.13	0
100	σ_1	1.99	20.33	0.44	0.73	0.07	1.12	1.11	1.14	0
	σ_2	1.12	4.92	0.39	0.69	0.12	1.13	1.10	1.16	0
	σ_3	1.18	2.83	0.17	0.65	0.00	1.15	1.11	1.20	2
	σ_4	3.09	limit	5.44	19.24	1.58	1.09	1.08	1.10	0

Table 8 Results for CG − UB and CG − LB with $b = 10$ and 5 parallel machines

Param		Times (s)		Gap (%)			CG − LB/PR			
n	σ	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	σ_1	0.05	0.20	0.47	1.95	0.00	1.27	1.18	1.30	2
	σ_2	0.05	0.16	0.84	1.77	0.00	1.22	1.20	1.24	3
	σ_3	0.05	0.10	0.30	1.57	0.00	1.19	1.15	1.20	7
	σ_4	0.05	0.26	0.83	2.20	0.00	1.36	1.27	1.44	2
40	σ_1	0.23	1.94	0.91	1.59	0.24	1.19	1.17	1.23	0
	σ_2	0.18	1.62	0.71	1.25	0.00	1.16	1.14	1.19	1
	σ_3	0.15	0.68	0.72	1.79	0.02	1.17	1.13	1.24	0
	σ_4	0.27	6.02	1.28	2.43	0.32	1.21	1.17	1.26	0
60	σ_1	0.53	5.34	0.70	1.25	0.25	1.16	1.13	1.20	0
	σ_2	0.55	7.23	0.81	1.78	0.34	1.12	1.10	1.15	0
	σ_3	0.45	2.17	0.36	0.89	0.00	1.15	1.13	1.19	1
	σ_4	0.69	89.00	1.65	3.14	0.37	1.16	1.13	1.18	0
80	σ_1	1.33	17.41	0.61	1.56	0.11	1.14	1.11	1.16	0
	σ_2	0.93	38.02	0.62	0.99	0.01	1.11	1.09	1.12	0
	σ_3	0.78	4.37	0.39	0.60	0.16	1.14	1.12	1.17	0
	σ_4	1.53	limit	2.50	4.57	1.27	1.13	1.11	1.14	0
100	σ_1	2.76	20.14	0.43	0.79	0.05	1.12	1.11	1.14	0
	σ_2	1.41	6.34	0.38	0.61	0.12	1.13	1.10	1.15	0
	σ_3	1.54	3.50	0.16	0.63	0.00	1.14	1.11	1.19	2
	σ_4	3.22	limit	14.09	81.62	0.87	1.10	1.09	1.11	0

$\sum C_j$ is the branch and bound of Azizoglu and Webster (2000), which was developed for the weighted version $1|p-\text{batch}, s_j \leq b|\sum w_j C_j$, but the authors also reported results for the unweighted version. In the latter case, the lower bound of Azizoglu and Webster reduces to Uzsoy's bound. The branch and price procedure presented in Sect. 2.4 is based on the same branching scheme as Azizoglu and Webster (2000), with the same dominance conditions. The comparison considered here is between:

- The branch and price equipped with the proposed lower bound CG − LB obtained by column generation and the CG − UB heuristic for the root node upper bound—see Sect. 2.4.
- The branch and bound based on the same branching scheme, with the same dominance conditions, equipped with:
  - an evaluation of two different lower bounds of increasing complexity as described in Azizoglu and Webster (2000) Sect. 3, using the best one for the actual lower bound;
  - the procedure described in Azizoglu and Webster (2000) Sect. 2.4 for the root node upper bound and evaluated at every non-leaf node to further improve the quality of the upper bound.
- The node exploration policy in both algorithms was kept depth-first.

The results are reported in Table 9 for instances with up to 40 jobs. Cumulative results for all 40 instances are reported by number of jobs, across job size distributions. Specifically, for times, nodes and gap, the average value is reported, while for the number of found optima, the total sum over the 40 instances is reported.

Both exact methods were run with a 900 s time limit. The table compares the CPU time and the number of open nodes. Column Opt counts the number of certified optima obtained within the time limit. On the left part of the table, the results for the branch and price equipped with CG − LB and CG − UB are reported; the right side shows the results for the branch and bound equipped with a lower bound and the heuristic by Azizoglu and Webster.

By allowing the branch and price to run without a time limit, all optimal values of the 40-job instances were also collected, even though a few instances required several hours of computation; hence, the GAP columns report the percentage gap $100 \cdot (\text{UB} - \text{OPT})/\text{OPT}$ between the best upper bound UB obtained within the time limit and the exact optimum OPT for both algorithms.

The branch and bound has impressively fast node processing, since Uzsoy's bound has a very cheap running time. As the number of jobs increases, the more accurate lower bound obtained by column generation allows the branch and price to outperform the competitor. The latter algorithm is able to optimally solve all instances with up to 30 jobs and more

Table 9 Comparison of exact approaches

Param		Branch and price (CG − LB and CG − UB)							Branch and bound (Azizoglu and Webster 2000/depth-first)						
		Times (s)		Nodes		GAP		OPT	Times (s)		Nodes		GAP		OPT
n	σ	Avg	Max	Avg	Max	Avg	Max		Avg	Max	Avg	Max	Avg	Max	
10	σ_1	0.06	0.48	4.2	32	0.00	0.00	10	0.01	0.01	1266.3	2222	0.00	0.00	10
	σ_2	0.02	0.08	15.6	147	0.00	0.00	10	0.01	0.01	953.0	1910	0.00	0.00	10
	σ_3	0.01	0.05	5.4	54	0.00	0.00	10	0.01	0.01	514.9	1244	0.00	0.00	10
	σ_4	0.03	0.05	21.5	56	0.00	0.00	10	0.01	0.01	1895.6	3119	0.00	0.00	10
10		0.03	0.48	12.7	147	0.00	0.00	40	0.01	0.01	1157.4	3119	0.00	0.00	40
20	σ_1	1.64	4.27	824.0	2658	0.00	0.00	10	35.55	67.14	9,556,527.9	18,522,644	0.00	0.00	10
	σ_2	2.84	19.99	1639.6	11,820	0.00	0.00	10	14.07	34.69	3,553,628.2	9,094,599	0.00	0.00	10
	σ_3	4.38	38.05	3508.0	31,389	0.00	0.00	10	5.95	16.56	1,276,560.2	3,745,567	0.00	0.00	10
	σ_4	3.59	13.30	985.1	3351	0.00	0.00	10	30.64	56.14	7,439,538.0	14,484,092	0.00	0.00	10
20		3.11	38.05	1739.2	31,389	0.00	0.00	40	21.55	67.14	5,456,563.6	18,522,644	0.00	0.00	40
30	σ_1	3.83	19.05	715.9	4212	0.00	0.00	10	900.00	900.00	133,616,540.3	167,662,461	1.45	4.91	0
	σ_2	82.65	655.98	50,281.3	467,906	0.00	0.00	10	900.00	900.00	133,304,717.2	144,452,851	1.27	3.58	0
	σ_3	16.06	136.41	7353.7	67,290	0.00	0.00	10	900.00	900.00	129,637,409.5	145,358,435	0.14	1.05	0
	σ_4	90.90	307.28	11,255.6	35,881	0.00	0.00	10	900.00	900.00	126,362,353.9	139,369,516	2.16	5.35	0
30		48.36	655.98	17401.6	46,7906	0.00	0.00	40	900.00	900.00	130,730,255.2	167,662,461	1.26	5.35	0
40	σ_1	619.85	901.11	40,824.5	71,268	0.13	0.63	6	900.00	900.00	100,200,931.0	121,947,526	5.83	9.06	0
	σ_2	547.41	900.93	52,105.6	151,857	0.18	0.99	6	900.00	900.00	82,962,783.4	89,454,946	4.25	8.09	0
	σ_3	499.44	900.19	80,444.0	165,196	0.08	0.51	7	900.00	900.00	88,661,788.3	96,750,260	2.18	4.75	0
	σ_4	841.06	905.80	25,943.0	53,933	0.93	3.11	5	900.00	900.00	83,033,865.0	96,502,979	5.04	8.44	0
40		626.94	905.80	49,829.3	165,196	0.33	3.11	22	900.00	900.00	88,714,841.9	121,947,526	4.33	9.06	0

Table 10 Comparison ($\texttt{GAP} = 100 \cdot (\texttt{UB} - \texttt{OPT})/\texttt{OPT}$) between $\texttt{CG} - \texttt{UB}$ and real optima with $b = 10$

Param		Gap (%)		
n	σ	Avg	Worst	Best
10	σ_1	0.00	0.00	0.00
	σ_2	0.39	3.89	0.00
	σ_3	0.00	0.00	0.00
	σ_4	0.22	1.13	0.00
10		0.15	3.89	0.00
20	σ_1	0.26	0.63	0.00
	σ_2	0.44	1.54	0.00
	σ_3	0.01	0.14	0.00
	σ_4	0.74	3.12	0.00
20		0.36	3.12	0.00
30	σ_1	0.08	0.73	0.00
	σ_2	0.16	0.68	0.00
	σ_3	0.01	0.03	0.00
	σ_4	0.78	2.11	0.00
30		0.26	2.11	0.00
40	σ_1	0.30	0.92	0.01
	σ_2	0.33	0.99	0.00
	σ_3	0.11	0.51	0.00
	σ_4	1.32	3.11	0.25
40		0.51	3.11	0.00

than half of the 40-job instances, with optimality gaps mostly below 1%.

5 Final remarks

In this paper, column generation techniques for solving the $1|p-\text{batch}, s_j \le b|\sum C_j$ problem have been explored, generalizing such techniques to problems with parallel machines. The exponential size model (13)–(16), handled by means of column generation, allows one to find—to the authors' knowledge—the tightest known lower bound for $1|p-\text{batch}, s_j \le b|\sum C_j$. Embedded in a simple price and branch approach, it achieves high-quality solutions for instances up to 100 jobs in size, with certified optimality gaps. Thus, it can be claimed that model (13)–(16) is strong: its relaxation gives a sharp bound, and the columns generated can be effectively composed into high-quality feasible solutions. The comparison with state-of-the art (meta)heuristics like HMMAS is admittedly problematic because of the lack of available code and instances, but the price and branch heuristic is, in the authors' view, at least as accurate as the state-of-the-art heuristics, and faster and *simpler*,, since it mostly relies on a LP/MILP solver, with the addition of some ad hoc code. Embedded in an exact algorithm, the new lower bound allows us to extend the size of solvable

Table 11 Results for $\texttt{CG} - \texttt{UB}$ and $\texttt{CG} - \texttt{LB}$ with $b = 50$ and $\sigma = \sigma_5$ considering $m = \{1, 2, 3, 5\}$ parallel machines

Param		Times (s)		Gap (%)			$\texttt{CG} - \texttt{LB/PR}$			
n	m	CG-LB	CG-UB	Avg	Worst	Best	Avg	Min	Max	Opt
20	1	0.01	0.03	0.88	3.19	0.00	1.27	1.16	1.39	2
	2	0.01	0.04	0.66	2.95	0.00	1.26	1.15	1.36	4
	3	0.01	0.11	0.70	3.20	0.00	1.26	1.16	1.34	4
	5	0.02	0.12	0.50	1.51	0.00	1.28	1.18	1.36	4
40	1	0.04	0.52	0.90	1.51	0.00	1.23	1.19	1.29	1
	2	0.05	0.56	0.74	1.50	0.00	1.22	1.19	1.27	1
	3	0.11	0.97	0.64	1.39	0.00	1.22	1.18	1.26	1
	5	0.21	1.32	0.55	1.26	0.00	1.22	1.18	1.25	2
60	1	0.16	3.30	0.77	1.35	0.10	1.19	1.16	1.22	0
	2	0.26	1.58	0.51	0.90	0.12	1.18	1.16	1.21	0
	3	0.43	3.75	0.54	1.24	0.08	1.18	1.16	1.21	0
	5	0.83	4.79	0.46	0.79	0.07	1.18	1.15	1.21	0
80	1	0.57	19.06	0.61	1.15	0.18	1.17	1.15	1.20	0
	2	0.79	17.68	0.53	1.10	0.19	1.17	1.15	1.20	0
	3	1.15	33.10	0.50	0.97	0.15	1.17	1.15	1.19	0
	5	1.53	23.49	0.52	0.98	0.19	1.17	1.15	1.19	0
100	1	1.20	22.58	0.54	0.86	0.34	1.16	1.12	1.19	0
	2	1.64	48.68	0.50	0.81	0.30	1.16	1.12	1.18	0
	3	2.92	41.17	0.45	0.79	0.26	1.15	1.12	1.18	0
	5	3.54	64.40	0.47	0.82	0.25	1.15	1.12	1.18	0

Time limit for $\texttt{CG} - \texttt{UB}$ set to 60 s for the single-machine case, and to 180 s for the multiple-machine cases

$1|p-\text{batch}, s_j \leq b|\sum C_j$ instances towards 40 jobs. Having available all optimal solutions for the single-machine problems up to 40 jobs, the actual relative error of the CG − UB heuristic on such instances can be seen to be even lower than the figures estimated in the experiments of Sect. 4.1—see Table 10.

Although all the random distributions for job data mentioned in the literature were used in the tests, the interested reader might be worried about the relatively narrow distributions for job sizes given in classes $\sigma_1, \ldots, \sigma_4$. Hence, some tests were performed with capacity $b = 50$ and a new class σ_5, with a wider distribution of $s_j \in [1, 50]$. The results of CG − UB on such instances for the single- and parallel-machine cases as well are summarized in Table 11, showing that CG − UB still handles such instances within practical time limits, guaranteeing narrow optimality gaps.

The new model relies on Property 1 in order to express the linear objective function by means of "positional" coefficients. Property 2 is crucial for developing an efficient pricing procedure. The approach is flexible enough to be extended to problems with parallel machines with very limited effort, while it is probably not simple to extend it to weighted $\sum w_j C_j$ objectives.

Acknowledgements This work has been partially supported by "Ministero dell'Istruzione, dell'Universitá e della Ricerca" Award "TESUN-83486178370409 finanziamento dipartimenti di eccellenza CAP. 1694 TIT. 232 ART. 6".

Funding Open access funding provided by Universitá degli Studi di Torino within the CRUI-CARE Agreement.

References

Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network flows: Theory, algorithms, and applications*. Prentice-Hall Inc.

Azizoglu, M., & Webster, S. (2000). Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research, 38*(10), 2173–2184.

Beldar, P., & Costa, A. (2018). Single machine batch processing problem with release dates to minimize total completion time. *International Journal of Industrial Engineering Computations, 9*(3), 331–348.

Cabo, M., Possani, E., Potts, C. N., & Song, X. (2015). Split-merge: Using exponential neighborhood search for scheduling a batching machine. *Computers and Operations Research, 63,* 125–135.

Cachon, G., & Terwiesch, C. (2012). *Matching supply with demand: An introduction to operations management*. McGraw-Hill Education.

Damodaran, P., Kumar Manjeshwar, P., & Srihari, K. (2006). Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *International Journal of Production Economics, 103*(2), 882–891.

Desrosiers, J., & Lübbecke, M. E. (2011). *Branch-price-and-cut algorithms*.

Dupont, L., & Dhaenens-Flipo, C. (2002). Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure. *Computers and Operations Research, 29*(7), 807–819.

Foster, B. A., & Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly (1970–1977), 27*(2), 367–384.

Graham, R., Lawler, E., Lenstra, J., & Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. Johnson, & B. Korte (Eds.), *Discrete optimization II, volume 5 of annals of discrete mathematics* (pp. 287–326). Elsevier.

Jia, Z., Zhang, H., Long, W., Leung, J. Y., Li, K., & Li, W. (2018). A meta-heuristic for minimizing total weighted flow time on parallel batch machines. *Computers and Industrial Engineering, 125,* 298–308.

Jolai Ghazvini, F., & Dupont, L. (1998). Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes. *International Journal of Production Economics, 55*(3), 273–280.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*.

Kosch, S., & Beck, J. C. (2014). A new mip model for parallel-batch scheduling with non-identical job sizes. In H. Simonis (Ed.), *Integration of AI and OR techniques in constraint programming* (pp. 55–70). Springer International Publishing.

Li, S. (2017). Approximation algorithms for scheduling jobs with release times and arbitrary sizes on batch machines with non-identical capacities. *European Journal of Operational Research, 263*(3), 815–826.

Liu, J., Lin, Z., Chen, Q., & Mao, N. (2016). Controlling delivery and energy performance of parallel batchprocessors in dynamic mould manufacturing. *Computers & Operations Research, 66,* 116–129.

Malapert, A., Guéret, C., & Rousseau, L. (2012). A constraint programming approach for a batch processing problem with non-identical job sizes. *European Journal of Operational Research, 221*(3), 533–545.

Muter, I. (2020). Exact algorithms to minimize makespan on single and parallel batch processing machines. *European Journal of Operational Research, 285*(2), 470–483.

Mönch, L., Fowler, J. W., & Mason, S. J. (2012). *Production planning and control for semiconductor wafer fabrication facilities: Modeling, analysis, and systems, volume 52 of operations research/computer science interfaces series*. Springer Science & Business Media.

Ozturk, O. (2020). A truncated column generation algorithm for the parallel batch scheduling problem to minimize total flow time. *European Journal of Operational Research, 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. *International Journal of Production Research, 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. *International Journal of Production Research, 50*(20), 6022–6035.

Rafiee Parsa, N., Karimi, B., & Husseinzadeh Kashan, A. (2010). A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers and Operations Research, 37*(10), 1720–1730.

Rafiee Parsa, N., Karimi, B., & Moattar Husseini, S. M. (2016). Minimizing total flow time on a batch processing machine using a hybrid max–min ant system. *Computers and Industrial Engineering, 99,* 372–381.

Rafiee Parsa, N., Keshavarz, T., Karimi, B., & Moattar Husseini, S. M. (2019). A hybrid neural network approach to minimize total completion time on a single batch processing machine. *International Transactions of Operational Research* (in press).

Uzsoy, R. (1994). Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research, 32*(7), 1615–1635.

Wang, H. (2011). Solving single batch-processing machine problems using an iterated heuristic. *International Journal of Production Research, 49*(14), 4245–4261.