POLITECNICO DI TORINO
Repository ISTITUZIONALE

Parallel batching with multi-size jobs and incompatible job families

Original
Parallel batching with multi-size jobs and incompatible job families / Druetto, A., Pastore, E., Rener, E.. - In: TOP. - ISSN 1134-5764. - ELETTRONICO. - 31:2(2023), pp. 440-458. [10.1007/s11750-022-00644-2]

Availability:
This version is available at: 11583/2971541 since: 2023-06-22T14:25:28Z

Publisher:
Springer

Published
DOI:10.1007/s11750-022-00644-2

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

26 November 2026

TOP
https://doi.org/10.1007/s11750-022-00644-2

ORIGINAL PAPER

Parallel batching with multi-size jobs and incompatible job families

Alessandro Druetto[1] · **Erica Pastore**[2] · **Elena Rener**[2]

Received: 11 April 2022 / Accepted: 4 September 2022
© The Author(s) 2022

Abstract

Parallel batch scheduling has many applications in the industrial sector, like in material and chemical treatments, mold manufacturing and so on. The number of jobs that can be processed on a machine mostly depends on the shape and size of the jobs and of the machine. This work investigates the problem of batching jobs with multiple sizes and multiple incompatible families. A flow formulation of the problem is exploited to solve it through two column generation-based heuristics. First, the column generation finds the optimal solution of the continuous relaxation, then two heuristics are proposed to move from the continuous to the integer solution of the problem: one is based on the price-and-branch heuristic, the other on a variable rounding procedure. Experiments with several combinations of parameters are provided to show the impact of the number of sizes and families on computation times and quality of solutions.

Keywords Parallel batch scheduling · Column generation · Incompatible job families · Multi-size jobs

Mathematics Subject Classification 90C27 · 90B35 · 90C39

Alessandro Druetto, Erica Pastore and Elena Rener contributed equally to this work.

✉ Alessandro Druetto
alessandro.druetto@unito.it

Erica Pastore
erica.pastore@polito.it

Elena Rener
elena.rener@polito.it

[1] Dipartimento di Informatica, Università di Torino, Via Pessinetto 12, Turin 10149, Italy

[2] Department of Management and Production Engineering, Politecnico di Torino, c.so Duca degli Abruzzi 24, Turin 10129, Italy

Published online: 21 September 2022

1 Introduction

In the batch scheduling problem, jobs are processed on each machine grouped in batches. The batch scheduling can be parallel or serial, depending on the processing rule of the jobs within the batch: if the jobs in the batch are processed simultaneously, this is the case of a parallel batch; otherwise, if the jobs within the batch are processed one after another, the batch is serial. Typical examples of parallel batches are furnaces and ovens that can be used, for instance, in mould manufacturing and semiconductor industries (Liu et al. 2016; Ozturk et al. 2012); instead, serial batches are usually exploited when a setup is needed before processing a new batch. In this paper, we address the parallel batch scheduling problem. Jobs in each batch are processed in parallel, so the processing time of the whole batch is equal to the maximum processing time amongst the jobs that compose it. The objective is to compose the batches and to sequence them, to optimize a performance measure. Usually, batches have a maximum size, depending on the technological characteristics of the process (Potts and Kovalyov 2000); for instance, the batch might have a maximum weight, or a maximum volume. Thus, to form the batches, the size constraint must be taken into account.

The batch scheduling problem is typical of semiconductor industries, mold manufacturing (Liu et al. 2016), medical device sterilization (Ozturk et al. 2012), heat-treating ovens (Mönch and Unbehaun 2007), chemical processes in tanks or kilns (Takamatsu et al. 1979), semiconductor and wafer fabrication industries (Mönch et al. 2013), and testing of electrical circuits (Hulett et al. 2017). Also additive manufacturing (AM) often requires batch production to optimize the chamber space (Zhang et al. 2020). However, here the processing times depend on different factors than those of conventional production, and the resulting scheduling problem could be different.

Sometimes, batch production needs to address multiple sizes in composing the batches. For instance, in AM, chambers can produce various parts simultaneously, either by placing products in the 2-dimensional space or by stacking products and hence using the 3-dimensional space of the chamber. Thus, when creating the batches, constraints on several dimensions must be considered; for instance, if the AM technology in use is able to stack parts in the chamber, then there is a maximum vertical span (height) and a maximum horizontal area that cannot be exceeded. These dimensions (e.g. height, horizontal area) will be called *sizes* in the paper. Similarly, other industries could have the same batch requirements. Moreover, for technological reasons, in shop floors where various product families are produced, batches must be composed of jobs of the same family. This is the case for families that need different manufacturing operations.

Due to the current industrial challenges, this paper addresses the single machine batch scheduling problem with multiple sizes and incompatible job families. The aim is to find the batch schedule that minimizes the total completion time; thus, with the Graham's three-field standard notation (Graham et al. 1979), the addressed problem is defined as $1|p\text{-}batch, s_{ij} \leq b_i, incomp| \sum C_j$ (where *p-batch* defines the parallel batching, $s_{ij} \leq b_i$ the multiple sizes, and *incomp* the incompatible job families).

The remainder of the paper is structured as follows. The relevant literature on the batch scheduling problem is reviewed in Sect. 2. The problem is formalized in Sect. 3, and the solution approaches are presented in Sect. 4. The numerical results are discussed in Sect. 5, while Sect. 6 concludes the paper.

2 Literature review

The batch scheduling problem has been addressed for a few years by many researchers (Ikura and Gimple 1986) because of its many fields of application. From the complexity point of view, Uzsoy (1994) first proved the NP-hardness for the $1|p\text{-}batch,\ s_j \leq b|\sum C_j$ problem, where jobs have different sizes, equal processing times and must be batched on a single machine. Since batching has raised the interest of many researchers in the scheduling field, there are many applications and problem variants that have been addressed in the literature. A complete overview of batching problems is given by Potts and Kovalyov (2000), who reviewed both parallel and serial batching problems and discussed major results and algorithms on the subject.

In the following, a brief review of the literature on single machine parallel batch scheduling problems with total completion time as performance measure, incompatible job families, and/or multiple sizes is discussed. Some examples of recent papers that address the parallel batch scheduling with other objective functions are: Muter (2020), Emde et al. (2020), Tan et al. (2018), Shahidi-Zadeh et al. (2017).

First, some variations of the single size batch scheduling problem have been studied. For instance, Azizoglu and Webster (2000) generalized the model of Uzsoy (1994) to the weighted case $1|p\text{-}batch, s_j \leq b|\sum w_j C_j$ with arbitrary job sizes and weights. They extended the existing branch and bound procedure to this case and exploited some dominance properties. Rafiee Parsa et al. (2016) addressed the $1|p\text{-}batch, s_j \leq b|\sum C_j$ problem by proposing an ant colony optimization metaheuristic. Alfieri et al. (2021) recently formulated a new MIP for the same problem based on a graph model, where the nodes of the graph represent job positions and arcs represent batches. The proposed model is solved by means of a column generation technique (Desrosiers and Lübbecke 2005) and allows to compute effective lower bounds. The model is also extended to the parallel machines environment. A column generation approach was also developed by Ozturk (2020), who considered identical parallel machines. The problem is here decomposed into two stages: first, a column generation is used to generate batches, and second these batches are scheduled on machines using an integer linear model.

With respect to incompatible job families, the batch scheduling problem was addressed by Azizoglu and Webster (2001), who adapted the solution procedure used in Azizoglu and Webster (2000) for the single family case to the incompatible job family case $1|p\text{-}batch, s_j \leq b, incomp|\sum w_j C_j$, where the jobs in the same family have equal processing times. The same problem was also solved through several heuristic approaches by Dobson and Nambimadom (2001), who addressed a version of the $1|p\text{-}batch, s_j \leq b, incomp|\sum w_j C_j$ problem, where the jobs of a family share the same processing time and the objective function is the mean weighted flow time.

To the authors' knowledge, this is the first attempt to address the batch scheduling problem with total completion time minimization, and jobs with multiple sizes and incompatible families. As the problem is NP-hard, a flow formulation is exploited to solve it through two column generation-based heuristics. The column generation finds a continuous-relaxed solution, then two different heuristics from the literature are used to move from the continuous to the integer solution of the problem: the first is based on the so-called price-and-branch heuristic (Desrosiers and Lübbecke 2005), the other on a variable rounding procedure (Druetto and Grosso 2022). An extensive experimental campaign was run to compare the two heuristics, which both prove to be very effective for this scheduling problem. As the results will show, variable rounding is the most effective both in terms of computation time and quality of solution. Other useful insights for practical applications will be derived from the results.

3 Problem definition

The single machine batching problem is considered, in which a set $N = \{1, \dots, n\}$ of jobs must be grouped in batches, and the batches must be scheduled on the machine. Setup times between batches are assumed to be negligible and are not considered.

The machine processes jobs in batches, and batches must respect physical constraints, such as maximum height, volume, weight, and so on. Let m be the number of different sizes to be addressed (e.g. $m = 3$ if the physical constraints imply maximum height, volume and weight), and b_i be the batch capacity associated to the size i (with $i = 1, \dots, m$). Each job j has a processing time p_j, and m size values s_{ij}, with $i = 1, \dots, m$. For instance, if batches must respect a maximum volume (b_1) and a maximum height (b_2), then each job j will be characterized by its own volume s_{1j} and its own height s_{2j}.

In the machine, jobs must be clustered in batches, such that the sum of job sizes s_{ij} of all jobs in a batch does not exceed the batch dimension b_i, $i = 1, \dots, m$.

In addition, jobs are grouped into families that represent different process requirements. Hence, jobs from different families must be processed in different batches. There are n_f families, and each job j belongs to a family f, $f = 1, \dots, n_f$.

A solution to the problem is made by a batch schedule, i.e., a sequence of feasible batches $S = B_1, \dots, B_{n_B}$. The total number of batches n_B is not known a priori, but it depends on how jobs are clustered in each solution. However, the number of batches n_B must be in the range between n_f and n, as there should exist at least one batch for each family, and there are at maximum n batches each composed of one single job.

All jobs in a batch B are assumed to be processed simultaneously and the processing time p_B of batch B equals the longest processing time of jobs contained in it i.e., $p_B = \{\max p_j \ \forall j \in B\}$. Also, a job is completed when all other jobs in the same batch are completed, i.e., the completion time C_j of each job equals the completion time of the batch it belongs to. The aim is to find the sequence $S = B_1, \dots, B_{n_B}$ that minimizes the total completion time.

In the paper, the three combinations of constraints will be considered i.e., the problem $1|p\text{-}batch, s_{ij} \leq b_i, incomp| \sum C_j$, the multi-size problem without

families $(1|p\text{-}batch, s_{ij} \leq b_i| \sum C_j)$, and the family problem with single size-jobs $(1|p\text{-}batch, s_j \leq b, incomp| \sum C_j)$.

4 Two column generation-based heuristics

The flow model used by Alfieri et al. (2021) is exploited and adapted to the problem at hand to develop two heuristic algorithms. The flow model considers the feasible batches as binary variables and associates a cost to each of them, whose sum is to be minimized in the objective function. Usual flow constraints guarantee the flow preservation from a source node to the last and a second set of constraints guarantee that each job is scheduled exactly once (for the details on the flow formulation, the reader is referred to Alfieri et al. (2021)). Both heuristics rely on an initial column generation algorithm, which solves the continuous relaxation of the flow model. Then, the heuristics use different techniques to move from the continuous solution found by the column generation to the final integer solution of the problem, as will be explained in the following.

4.1 The `CG-LB` column generation algorithm

The column generation starts from a restricted formulation of the relaxed continuous flow model and iteratively selects promising variables (i.e., promising batches) through a so-called pricing procedure. The column generation finds as an output the optimal solution of the continuous relaxation of the flow model. This solution is then used as a starting point to find the integer solution of the initial problem in the heuristics. The details of the column generation are explained in the following.

First, an initialization phase is needed. Jobs are first sorted according to the Shortest Processing Time (SPT) rule, which is optimal for the scheduling of all jobs on a serial machine with the $\sum C_j$ objective function (Chandru et al. 1993). Then, some feasible batches are generated by clustering together the sorted jobs until one of the maximum batch dimensions is reached, or until a job belonging to a different family is selected. When the batch is closed, it is placed on all possible schedule positions, and then, a new batch is developed with the same rule. The final output of the initialization is the set H, which contains a set of feasible batches.

The set H is set as the initial column set in the column generation, which is used to solve the continuous relaxation of the flow model. The algorithm iterates between solving the Restricted Master Problem (RMP, i.e., the problem with only a subset of variables) and the pricing problem (the problem of selecting new columns to be added to the RMP) until the continuous optimum is found.

In the pricing procedure, the dual multipliers associated with the RMP are computed to find the most promising variables, i.e., those with the most negative reduced cost, to be included in the next iteration of the RMP.

The pricing problem searches for the minimum reduced cost $\bar{c}_{ikB^*}$ associated to arc (i, k, B^*), among all arcs (i, k, B) that correspond to feasible batches. Let v_j be the dual multiplier corresponding to the constraints that guarantee that all jobs are included in

the final schedule, and let u_i, u_k be the dual multipliers corresponding to the flow maintenance constraints. Recall that each job j belongs to a family f_j and has m different sizes s_{hj} ($h \in \{1, \dots, m\}$), corresponding to the m batch capacities b_h; furthermore, the batch cardinality $|B|$ must be equal to $(k - i)$. Then, the problem can be formulated as follows.

$$\bar{c}_{ikB^*} = \min_B \left\{ \begin{array}{l} c_{ikB} - \sum_{j \in B} v_j \ : \\ \quad \sum_{j \in B} s_{hj} \leq b_h \quad \forall h \in \{1, \dots, m\}, \\ \quad f_j = f_{j'} \quad \forall j, j' \in B, \\ \quad |B| = (k - i) \end{array} \right\} - (u_i - u_k) \tag{1}$$

For each new possible feasible batch B, the evaluation of the corresponding reduced cost depends on the position in the sequence, on the processing time and on the jobs included in the batch. By isolating the parts depending on the position in an external loop that considers every i, k with $k > i$, the problem at hand is reduced to the computation of the feasible batch containing $k - i$ jobs with maximum value. This problem can be formulated as a cardinality-constrained multi-weight knapsack, which has to be solved for every starting and ending position. Indeed, jobs are the items, their dual multipliers are profits, and their sizes are the weights.

The solution space of the multi-weight knapsack problem with cardinality constraint is explored via a dynamic programming algorithm. Specifically, for every pair of i, k with $k > i$, and for each job $r = 1, \dots, n$, let $g_r(\tau_1, \dots, \tau_m, l)$ be the optimal knapsack with capacities τ_h for all $h \in \{1, \dots, m\}$, cardinality $l = k - i$, and that considers only jobs $r, r + 1, \dots, n$. The binary variable $y_j \in \{0, 1\}$ equals 1 if job j is selected for the knapsack, 0 otherwise. Then, the algorithm searches for the following values.

$$\begin{aligned} g_r(\tau_1, \dots, \tau_m, l) = \max \Bigg\{ & \sum_{j=r}^{n} v_j y_j \ : \ \sum_{j=r}^{n} s_{1j} y_j \leq \tau_1, \ \dots, \ \sum_{j=r}^{n} s_{mj} y_j \leq \tau_m, \\ & \sum_{j=r}^{n} y_j = l, f_j = f_r, \ y_j \in \{0, 1\} \Bigg\} \end{aligned} \tag{2}$$

This can be computed with the following recursion as in Kellerer et al. (2004):

$$g_r(\tau_1, \dots, \tau_m, l) = \max \left\{ \begin{array}{ll} g_{r'}(\tau_1 - s_{1r}, \dots, \tau_m - s_{mr}, l - 1) + v_r & (y_r = 1) \\ g_{r'}(\tau_1, \dots, \tau_m, l) & (y_r = 0) \end{array} \right. \tag{3}$$

where r' is the next job in the Longest Processing Time (LPT) ordering that belongs to the same family of the job that started the recursion. In this way, family incompatibilities are addressed. The boundary conditions are as follows:

$$g_r(\tau_1, \dots, \tau_m, 1) = \left\{ \begin{array}{ll} v_r \ \text{if} \ s_{hr} \leq \tau_h & (y_r = 1) \\ 0 \ \ \text{otherwise} & (y_r = 0) \end{array} \right. \quad r = 1, \dots, n, \ \tau_h = 0, \dots, b_h \tag{4}$$

$$g_r(\tau_1, \dots, \tau_m, 0) = 0 \quad r = 1, \dots, n, \ \tau_h = 0, \dots, b_h \tag{5}$$

$$g_r(\tau_1, \ldots, \tau_m, l) = -\infty \quad \text{if } l > n - r + 1 \text{ or } \tau_h < 0 \tag{6}$$

Equations (4)–(6) hold for all $h \in \{1, \ldots, m\}$ since all sizes must be considered. It is worth noting that the dynamic programming state space increases of one dimension for every size of the jobs.

The variables with negative reduced cost found through this pricing procedure are added to the RMP, and the procedure is repeated until no negative reduced costs are found. The final solution of the column generation is the optimum (CG-UB) for the continuous problem, and a lower bound for the integer problem. This algorithm, called Column Generation Lower Bound, will be referred to as CG-LB in the following.

4.2 The CG-UB and VR-UB heuristic procedures

Two heuristics are proposed to find an integer solution for the problem, and both starts from the lower bound given by the column generation CG-LB.

The first heuristic is called Column Generation Upper Bound (CG-UB), and it is the same used by Alfieri et al. (2021). Once the continuous optimum CG-LB is found, the variable domain of the problem is changed from continuous to binary, and the MIP is solved to get a heuristic integer solution, namely CG-UB. This evaluation can be quite slow, especially when the number of jobs increases. Solving such a MIP to the optimum is NP-hard, indeed. Some strategies to speed up the evaluation could include stopping the evaluation after a fixed number of open nodes (but that amount should increase with the number of considered jobs), or after a certain time limit has been reached (but it can lead to large CG-UB/CG-LB gaps), or after a certain gap is reached (but the timing to reach such a gap could be too slow again). A time limit will be set for the numerical results of Section 5.

The second heuristic is the Variable Rounding Upper Bound, namely VR-UB, and consists in the variable rounding procedure proposed in Druetto and Grosso (2022). Differently from CG-UB, this approach generates good upper bounds within shorter computation times. Promising variables are sequentially fixed to 1 i.e., a batch is forced to be included in the solution. Then, the leftover part of the RMP is re-optimized with continuous variables, up to the point when the entire sequence is established. Between each rounding and re-optimization step, the feasibility is restored through a modified version of the initialization algorithm. This modified version is run only on the not-fixed part of the sequence and considers only jobs not included in the fixed batches yet.

Algorithm 1 Variable Rounding Upper Bound

1: $z \leftarrow$ `CG-LB` computed over $G(V, A')$; ▷ generates also batches in A'
2: $n \leftarrow |N|$, $k \leftarrow 1$;
3: **while** $k \leq n$ **do**
4: $i \leftarrow k$; ▷ new starting position is old ending position
5: $\mathbf{x} \leftarrow$ nonzero basic variables of $G(V, A')$;
6: $j \leftarrow$ index of $\max\{\mathbf{x}\}$ with starting position i; ▷ batch with highest flow
7: $k \leftarrow$ ending position of selected batch j;
8: **if** $x_j = 1$ **then** ▷ if x_j is already integer, skip to the next
9: `fix` $x_j := 1$; ▷ variable x_j rounded to be 1
10: **continue**
11: **end if**
12: `fix` $x_j := 1$; ▷ variable x_j rounded to be 1
13: FeasRestore(G, k); ▷ feasibility restoration on G from k to end
14: $z \leftarrow$ `CG-LB` computed over $G(V, A')$; ▷ re-optimization of the problem
15: **end while**
16: `VR-UB` $\leftarrow$ `CG-LB` computed over $G(V, A')$; ▷ is an integral solution

The detailed variable rounding procedure is summarized in Algorithm 1, and it works as follows. First, `CG-LB` is computed and the RMP is populated. Then, among all columns that start from $i = 1$ i.e., all batches in the first position of the sequence, the one whose flow value is the closest to 1 is selected and enforced to be part of the solution. Also, no other columns can start from the same position. The procedure is repeated on the remaining part of the problem, by keeping all the already generated columns. Then, again, a new selection is made among all columns that start from $i = k$, where k was the ending position of the column fixed in the previous step.

The column generation and variable rounding procedures are iterated until the value $i = |N| + 1$ is reached, i.e., when there exist a sequence of batches from 1 to $|N| + 1$, where all non-zero flow variables are equal to 1; this is the heuristic solution `VR-UB` for the original problem.

As shown in Algorithm 1, during the variable rounding procedure, another optimization of the algorithm is made (lines 8-11 of Algorithm 1). If the selected column already has a flow with value 1 i.e., it is already integer in the continuous optimum, there is no need to re-optimize the model, since the value obtained would be the same as before and no new columns would be generated.

5 Numerical results

Random instances are generated to test the proposed algorithm. The generation approach used in Alfieri et al. (2021), Rafiee Parsa et al. (2016), Uzsoy (1994) is here replicated.

In the experiment, some factors are varied to test the proposed approach in various scenarios. Specifically, the number of jobs n has been varied in $n = \{20, 40, 60, 80, 100, 200\}$, as the larger is n, the more complex is the problem to be solved. Jobs have various sizes, and each size can be sampled from two different uniform distributions: $\sigma_{10} = U(1, 10)$ and $\sigma_5 = U(1, 5)$; the results should confirm that smaller intervals make the problem more complex, as more feasible batches can be created. The evaluated numbers of sizes per job are $m = \{1, 2, 3\}$, to test the efficiency of the approach in the multi-size cases. For each size, a distribution must be chosen between σ_{10} and σ_5 and, for each value of m, all possible combinations of σ_{10} and σ_5 are evaluated. Last, the tested numbers of families are $n_f = \{1, 3, 5, 7, 10\}$, to address the case of incompatible job families. All in all, 174 combinations of factors are tested. For each combination, 10 instances are solved, thus leading to 1740 experiments. For conciseness purposes, only results for $n = \{20, 60, 100, 200\}$ are shown below; the trend highlighted by this subset of experiments is confirmed by the experiments on the other sizes.

Some system characteristics have been fixed as parameters in all the instances. Specifically, for each job, the processing time is sampled from a uniform distribution $U(1, 100)$. If more than one family is considered, the family is randomly assigned to every job with equal probability. Also, for each size, the batch capacity b_i is fixed to 10, as commonly used in the literature (Azizoglu and Webster 2000; Rafiee Parsa et al. 2016; Alfieri et al. 2021).

Beside the instances generated as mentioned above, additional tests are run with $b_i = 50$ and with size distribution $\sigma_{50} = U(1, 50)$. The aim is to test the algorithm performance in the case where jobs have sizes with higher granularity. The results are presented separately in Appendix A.

Two heuristic algorithms are compared: the column generation combined with the MIP solver (`CG-UB`), and the column generation with the variable rounding procedure (`VR-UB`). Both are described in Sect. 4.

The proposed methods are developed in C++, and the optimization procedure is done by calling the CPLEX solver, version 12.9. Tests are run on a computer having a 3.70 GHz Intel i7 processor with 32 GB RAM.

In the following, the results are shown separately for the following cases: multi-size with no families (Table 1); incompatible families with one size (Tables 2 and 3); multi-size and incompatible families (Table 4, which shows only the cases of 3 and 7 families for reasons of conciseness). Results of single-size single-family instances are not reported in the paper for reasons of conciseness.

In all the tables, each row reports various statistics for a single combination of factors (grouping together the 10 instances) on the performance of the CPLEX's integer solution (`CG-UB`) and of the variable rounding procedure (`VR-UB`). Specifically, each row is characterized by:

- the number of jobs n;
- the distributions for each size (columns s_1, s_2, s_3 contain respectively the distribution of size 1, 2 or 3, if jobs have one, else −);
- the number of incompatible families n_f;

Table 1 Algorithm performance on the multi-size case

n	s_1, s_2, s_3	α	CG−UB				VR−UB			
			Avg t	Max t	Avg % g	Max % g	Avg t	Max t	Avg % g	Max % g
20	$\sigma_{10}\,\sigma_{10}-$	7	0.03	0.1	0.27	1.5	0.02	0.1	0.20	0.8
20	$\sigma_{10}\,\sigma_5-$	6	0.03	0.1	0.52	2.6	0.02	0.1	0.36	1.9
20	$\sigma_5\,\sigma_5-$	2	0.05	0.1	1.13	3.5	0.02	0.1	1.11	3.5
60	$\sigma_{10}\,\sigma_{10}-$	0	0.19	0.4	0.30	0.9	0.14	0.2	0.48	1.3
60	$\sigma_{10}\,\sigma_5-$	0	0.50	1.7	0.53	1.1	0.30	0.5	0.86	2.0
60	$\sigma_5\,\sigma_5-$	0	10.74	35.9	2.01	3.4	0.97	1.3	1.85	2.5
100	$\sigma_{10}\,\sigma_{10}-$	1	1.17	2.3	0.16	0.3	0.99	1.4	0.36	0.8
100	$\sigma_{10}\,\sigma_5-$	0	8.55	34.5	0.60	1.2	1.79	2.3	1.08	2.1
100	$\sigma_5\,\sigma_5-$	0	97.66	>100.0	3.76	6.6	4.11	4.6	1.84	2.5
200	$\sigma_{10}\,\sigma_{10}-$	0	20.91	>100.0	0.12	0.3	7.91	11.9	0.29	0.5
200	$\sigma_{10}\,\sigma_5-$	0	97.40	>100.0	0.51	2.0	11.51	14.9	0.60	1.1
200	$\sigma_5\,\sigma_5-$	0	>100.00	>100.0	56.80	69.7	27.91	30.3	1.37	2.1
20	$\sigma_{10}\,\sigma_{10}\,\sigma_{10}$	10	0.03	0.1	0.00	0.0	0.03	0.1	0.00	0.0
20	$\sigma_{10}\,\sigma_{10}\,\sigma_5$	8	0.04	0.1	0.06	0.6	0.03	0.1	0.07	0.7
20	$\sigma_{10}\,\sigma_5\,\sigma_5$	3	0.05	0.1	0.78	2.4	0.05	0.1	1.29	4.8
20	$\sigma_5\,\sigma_5\,\sigma_5$	0	0.07	0.1	1.94	4.2	0.07	0.1	2.34	5.1
60	$\sigma_{10}\,\sigma_{10}\,\sigma_{10}$	7	0.18	0.3	0.12	0.8	0.34	0.5	0.14	0.8
60	$\sigma_{10}\,\sigma_{10}\,\sigma_5$	1	0.25	0.4	0.34	0.9	0.60	0.9	0.70	1.8
60	$\sigma_{10}\,\sigma_5\,\sigma_5$	0	0.40	0.9	0.49	1.2	0.87	1.3	1.08	3.6
60	$\sigma_5\,\sigma_5\,\sigma_5$	0	7.24	24.0	1.62	2.3	2.09	2.7	1.81	3.0
100	$\sigma_{10}\,\sigma_{10}\,\sigma_{10}$	4	0.83	1.1	0.03	0.1	1.84	3.0	0.08	0.4
100	$\sigma_{10}\,\sigma_{10}\,\sigma_5$	4	1.21	2.0	0.10	0.3	2.69	6.0	0.24	0.7
100	$\sigma_{10}\,\sigma_5\,\sigma_5$	0	6.08	11.0	0.51	0.8	5.75	7.4	0.94	2.0
100	$\sigma_5\,\sigma_5\,\sigma_5$	0	82.57	>100.0	1.88	5.0	9.04	11.4	1.33	2.2
200	$\sigma_{10}\,\sigma_{10}\,\sigma_{10}$	4	5.76	8.6	0.03	0.1	13.90	26.7	0.08	0.4
200	$\sigma_{10}\,\sigma_{10}\,\sigma_5$	0	12.05	22.5	0.10	0.2	25.21	47.7	0.28	0.7
200	$\sigma_{10}\,\sigma_5\,\sigma_5$	0	90.16	>100.0	0.34	0.6	44.27	62.1	0.69	1.4
200	$\sigma_5\,\sigma_5\,\sigma_5$	0	>100.00	>100.0	22.85	68.9	78.35	96.2	1.14	2.1

- the number of reached optima α (where the optimum is reached in the case $LB = UB$), knowing that the optimum is reached when the continuous CG−LB is characterized by all integer variables;
- the average (*Avg t*) and maximum (*Max t*) computation time over the 10 instances, expressed in seconds;
- the average (*Avg % g*) and maximum (*Max % g*) percentage gap computed on the single instance as:

$$\mathbf{gap_{CG\text{-}UB}} = \frac{\mathbf{CG\text{-}UB - CG\text{-}LB}}{\mathbf{CG\text{-}UB}} \times 100, \qquad \mathbf{gap_{VR\text{-}UB}} = \frac{\mathbf{VR\text{-}UB - CG\text{-}LB}}{\mathbf{VR\text{-}UB}} \times 100.$$

Table 2 Algorithm performance on the family case for σ_{10}

n	s_1	n_f	α	CG-UB				VR-UB			
				Avg t	Max t	Avg % g	Max % g	Avg t	Max t	Avg % g	Max % g
20	σ_{10}	3	6	0.04	0.1	0.54	2.7	0.01	0.1	1.28	5.7
20	σ_{10}	5	7	0.03	0.1	0.44	1.7	0.01	0.1	0.49	2.1
20	σ_{10}	7	7	0.04	0.2	0.41	2.3	0.01	0.1	0.42	2.3
20	σ_{10}	10	7	0.04	0.2	0.22	1.7	0.01	0.1	0.43	3.6
60	σ_{10}	3	0	0.29	0.5	1.34	2.3	0.15	0.2	1.97	4.2
60	σ_{10}	5	0	0.19	0.4	0.65	2.0	0.09	0.2	1.17	2.9
60	σ_{10}	7	3	0.18	0.5	0.76	3.6	0.06	0.1	1.08	5.7
60	σ_{10}	10	2	0.12	0.2	0.83	1.9	0.06	0.1	1.11	2.7
100	σ_{10}	3	0	2.51	3.6	0.76	1.1	1.24	1.7	1.31	2.2
100	σ_{10}	5	0	1.29	1.8	0.92	1.4	1.01	1.7	1.70	2.7
100	σ_{10}	7	0	1.28	3.0	0.81	1.5	0.81	1.2	1.18	2.5
100	σ_{10}	10	0	0.97	1.8	1.21	2.3	0.74	1.0	1.87	3.7
200	σ_{10}	3	0	53.45	>100.0	0.58	1.4	8.63	12.1	1.01	1.6
200	σ_{10}	5	0	16.33	24.0	0.74	1.1	8.05	10.3	1.52	2.5
200	σ_{10}	7	0	16.94	49.2	0.83	1.2	6.58	8.0	1.39	1.9
200	σ_{10}	10	0	11.09	23.5	0.80	1.4	5.95	7.3	1.39	2.2

Table 3 Algorithm performance on the family case for σ_5

n	s_1	n_f	α	CG-UB				VR-UB			
				Avg t	Max t	Avg % g	Max % g	Avg t	Max t	Avg % g	Max % g
20	σ_5	3	3	0.05	0.1	1.12	2.7	0.01	0.1	1.43	4.2
20	σ_5	5	4	0.04	0.1	1.34	4.1	0.01	0.1	2.15	6.5
20	σ_5	7	5	0.04	0.1	0.63	3.7	0.01	0.1	1.09	4.5
20	σ_5	10	6	0.04	0.1	0.99	3.2	0.01	0.1	1.25	4.4
60	σ_5	3	0	2.05	5.3	2.49	4.1	0.69	1.0	3.30	7.1
60	σ_5	5	0	0.75	1.8	2.72	5.0	0.51	0.7	3.40	5.9
60	σ_5	7	0	0.61	0.9	2.79	7.1	0.40	0.6	3.51	7.0
60	σ_5	10	0	0.28	0.5	2.02	3.2	0.26	0.4	2.75	4.7
100	σ_5	3	0	87.51	>100.0	3.19	5.4	3.50	4.2	3.11	4.6
100	σ_5	5	0	24.05	>100.0	2.39	3.9	2.72	3.1	3.34	5.3
100	σ_5	7	0	8.13	16.3	2.49	4.2	2.30	2.7	3.02	4.9
100	σ_5	10	0	3.00	5.2	2.02	3.1	2.02	2.5	2.66	4.4
200	σ_5	3	0	>100.00	>100.0	40.70	72.1	23.97	29.8	2.35	3.1
200	σ_5	5	0	>100.00	>100.0	39.83	70.6	18.71	21.8	2.94	3.7
200	σ_5	7	0	>100.00	>100.0	5.41	8.9	17.14	21.2	3.23	4.2
200	σ_5	10	0	>100.00	>100.0	3.86	6.1	14.75	18.1	3.00	3.7

Table 4 Algorithm performance on the multi-size family case

				CG-UB				VG-UB			
n	s_1, s_2, s_3	n_f	α	Avg t	Max t	Avg % g	Max % g	Avg t	Max t	Avg % g	Max % g
20	$\sigma_{10}, \sigma_{10}, -$	3	9	0.02	0.1	0.06	0.6	0.01	0.1	0.31	3.1
20	$\sigma_{10}, \sigma_{10}, -$	7	10	0.02	0.1	0.00	0.0	0.01	0.1	0.00	0.0
20	$\sigma_{10}, \sigma_5, -$	3	6	0.04	0.1	0.86	3.5	0.01	0.1	0.90	3.9
20	$\sigma_{10}, \sigma_5, -$	7	4	0.04	0.1	0.40	2.4	0.01	0.1	1.33	3.1
20	$\sigma_5, \sigma_5, -$	3	2	0.07	0.1	1.18	3.8	0.02	0.1	1.61	3.9
20	$\sigma_5, \sigma_5, -$	7	2	0.07	0.2	1.92	5.0	0.01	0.1	2.79	5.9
60	$\sigma_{10}, \sigma_{10}, -$	3	2	0.13	0.2	0.29	1.6	0.10	0.2	0.68	1.7
60	$\sigma_{10}, \sigma_{10}, -$	7	5	0.09	0.2	0.16	0.6	0.06	0.1	0.36	1.9
60	$\sigma_{10}, \sigma_5, -$	3	0	0.33	0.7	1.00	2.0	0.18	0.4	1.41	3.5
60	$\sigma_{10}, \sigma_5, -$	7	2	0.12	0.2	0.82	2.3	0.11	0.2	1.09	4.2
60	$\sigma_5, \sigma_5, -$	3	0	1.36	2.8	2.38	3.7	0.77	1.0	3.79	6.2
60	$\sigma_5, \sigma_5, -$	7	0	0.43	0.8	2.46	3.8	0.43	0.7	4.14	6.1
100	$\sigma_{10}, \sigma_{10}, -$	3	3	0.78	1.4	0.32	0.6	0.77	1.3	0.57	1.1
100	$\sigma_{10}, \sigma_{10}, -$	7	2	0.47	0.8	0.45	1.1	0.61	0.9	0.73	1.5
100	$\sigma_{10}, \sigma_5, -$	3	0	1.77	4.0	0.50	0.8	1.39	1.9	0.93	1.9
100	$\sigma_{10}, \sigma_5, -$	7	0	1.00	1.4	0.96	2.1	0.99	1.4	1.52	2.4
100	$\sigma_5, \sigma_5, -$	3	0	75.00	>100.0	2.26	4.2	3.59	4.3	2.42	3.5
100	$\sigma_5, \sigma_5, -$	7	0	8.74	27.1	2.17	3.0	2.55	3.3	2.98	5.4
200	$\sigma_{10}, \sigma_{10}, -$	3	1	6.36	9.8	0.17	0.3	6.35	9.2	0.34	1.0
200	$\sigma_{10}, \sigma_{10}, -$	7	0	4.83	8.2	0.25	0.6	4.99	6.3	0.53	0.9
200	$\sigma_{10}, \sigma_5, -$	3	0	52.38	>100.0	0.54	0.9	11.12	13.2	1.16	1.5
200	$\sigma_{10}, \sigma_5, -$	7	0	14.58	41.4	0.81	1.4	9.06	12.5	1.63	1.9
200	$\sigma_5, \sigma_5, -$	3	0	>100.00	>100.0	28.97	68.9	24.27	27.7	2.24	2.7
200	$\sigma_5, \sigma_5, -$	7	0	>100.00	>100.0	6.38	8.7	20.17	23.7	2.50	3.1
20	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	10	0.04	0.1	0.00	0.0	0.03	0.1	0.00	0.0
20	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	10	0.03	0.1	0.00	0.0	0.02	0.1	0.00	0.0
20	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	7	0.07	0.2	0.56	4.5	0.04	0.1	0.48	3.8
20	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	8	0.03	0.0	0.12	0.9	0.03	0.1	0.47	3.8
20	$\sigma_{10}, \sigma_5, \sigma_5$	3	4	0.08	0.1	0.91	3.6	0.04	0.1	1.16	3.9
20	$\sigma_{10}, \sigma_5, \sigma_5$	7	6	0.04	0.1	0.88	3.8	0.04	0.1	1.19	3.8
20	$\sigma_5, \sigma_5, \sigma_5$	3	1	0.09	0.2	2.38	6.5	0.06	0.1	2.99	6.9
20	$\sigma_5, \sigma_5, \sigma_5$	7	3	0.05	0.1	1.61	5.0	0.05	0.1	2.50	5.9
60	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	8	0.16	0.2	0.03	0.2	0.31	0.8	0.16	1.3
60	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	7	0.13	0.2	0.18	1.3	0.32	0.8	0.36	1.6
60	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	3	0.26	0.7	0.35	0.8	0.44	0.7	0.43	0.8
60	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	6	0.16	0.3	0.37	1.8	0.44	1.3	0.59	2.7
60	$\sigma_{10}, \sigma_5, \sigma_5$	3	0	0.50	0.9	0.88	1.5	0.91	1.7	1.51	2.8
60	$\sigma_{10}, \sigma_5, \sigma_5$	7	0	0.27	0.4	1.05	2.2	0.64	0.9	1.76	3.2
60	$\sigma_5, \sigma_5, \sigma_5$	3	1	1.39	2.6	1.84	2.9	1.61	2.5	2.14	4.3
60	$\sigma_5, \sigma_5, \sigma_5$	7	0	0.36	0.4	1.98	3.3	1.19	1.8	2.61	5.2

Table 4 (continued)

				CG−UB				VG−UB			
n	s_1, s_2, s_3	n_f	α	Avg t	Max t	Avg % g	Max % g	Avg t	Max t	Avg % g	Max % g
100	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	6	1.08	1.5	0.04	0.2	1.04	1.5	0.07	0.6
100	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	5	0.40	0.6	0.20	0.8	1.05	1.8	0.23	0.8
100	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	2	1.31	1.9	0.20	0.5	2.25	4.2	0.36	1.1
100	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	1	0.65	0.9	0.42	1.1	2.33	4.0	0.80	2.4
100	$\sigma_{10}, \sigma_5, \sigma_5$	3	0	2.56	6.3	0.58	1.4	4.05	6.5	1.17	2.4
100	$\sigma_{10}, \sigma_5, \sigma_5$	7	0	1.10	1.4	0.81	1.8	4.07	8.5	1.32	3.4
100	$\sigma_5, \sigma_5, \sigma_5$	3	0	43.04	>100.0	1.57	2.5	7.97	10.3	2.04	3.3
100	$\sigma_5, \sigma_5, \sigma_5$	7	0	3.63	5.9	1.69	2.4	6.45	10.2	2.88	5.2
200	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	3	3	5.07	6.7	0.03	0.2	15.40	26.6	0.09	0.2
200	$\sigma_{10}, \sigma_{10}, \sigma_{10}$	7	4	2.92	4.9	0.10	0.4	8.98	16.5	0.15	0.6
200	$\sigma_{10}, \sigma_{10}, \sigma_5$	3	0	8.88	12.3	0.17	0.3	19.70	27.1	0.36	1.0
200	$\sigma_{10}, \sigma_{10}, \sigma_5$	7	1	6.13	10.0	0.27	0.6	21.78	29.5	0.60	1.1
200	$\sigma_{10}, \sigma_5, \sigma_5$	3	0	38.87	>100.0	0.46	0.8	40.75	61.2	0.91	1.5
200	$\sigma_{10}, \sigma_5, \sigma_5$	7	0	11.20	16.7	0.60	1.0	32.97	44.9	1.10	1.8
200	$\sigma_5, \sigma_5, \sigma_5$	3	0	>100.00	>100.0	9.52	13.5	71.87	80.8	1.86	2.3
200	$\sigma_5, \sigma_5, \sigma_5$	7	0	>100.00	>100.0	4.12	5.9	59.99	70.6	2.37	3.4

The column generation procedure that generates promising batches and builds a fractional solution works very fast. Finding the optimal integer solution using those batches is instead a time consuming issue for large instances. For this reason, a time limit of 100 CPU s has been set.

First, both the CG−UB and VR − UB approaches are very fast. Indeed, for instances up to 60 jobs, the overall computation time is of the order of one second. Across all instances, the largest computation time for VR − UB is 96.2 s, while CG−UB sometimes reaches the 100 second time limit.

The instances with sizes distributed following σ_5 are more difficult than those following σ_{10}. Indeed, if comparing the instances in Tables 2 and 3 with 200 jobs, the average computation time of CG−UB moves from the order of 20 s for σ_{10} cases to times larger than 100 s for σ_5. The same is true for VR − UB, although computation times are lower for both cases. As σ_5 reduces the maximum size a job can have, there are more possible combinations of jobs in a single batch, thus both computation times and gaps increase. However, if more than one size constraint is considered (Tables 1 and 4), even with σ_5 the number of combinations decreases, and the dynamic programming that searches for feasible batches is able to cut uninteresting combinations. Thus, the greater the number of job sizes, the faster the algorithm works.

The number of jobs negatively impacts the computation time and the gaps, both for CG−UB and VR − UB. Specifically, CG−UB often needs more than 100 s to solve instances with 200 jobs, especially in the cases of σ_5 size distributions. Instead, VR − UB is able to handle these instances in most cases, but its computation time largely

increases with respect to the instances with a small number of jobs. Moreover, with smaller numbers of jobs, both algorithms are more likely able to find optimal solutions; indeed, the number of optimal solutions α is larger with smaller instances.

When incompatible families are addressed, a noticeable difference in computation times can be seen when the number of families increases. This is shown both in Tables 2 and 3 and in Table 4. In the former tables, the average computation time decreases with the increase of the number of families. For instance, with $n = 100$ and $s_1 = \sigma_{10}$, the average computation time goes from 2.51 with 3 families to 0.97 with 10 families for the CG−UB, and from 1.24 to 0.74 for the VR − UB. When multi-size is involved, the same decrease in computation time is shown in Table 4; interestingly, if σ_5 distribution is given to the sizes, the difference in computation time with different families is even larger. As an example, if comparing the cases with $n = 100$; $m = 3$; $s_i = \sigma_{10}\ \forall i$ and $n = 100$; $m = 3$; $s_i = \sigma_5\ \forall i$, the difference in the average computation times from 3 to 7 families for the CG−UB moves from the order of 0.68 s to the order of 39.41 s. In the same instances, for the VR − UB approach, there is no difference between computation times of 3 and 7 families for the σ_{10} instances, and there is a difference of the order of 1.52 s for the σ_5 instances. As explained in Section 4, the search in the state space from job r that belongs to family f can be restricted to the subset of jobs belonging to the same family. This modification of the search procedure does not impact on the structure of the state space, and leads to a decrease in computation time proportional to the number of families.

Lastly, comparing the results of CG−UB and VR − UB across all results, the proposed variable rounding tends to be faster than CG−UB, and obtains comparable or better performance. Specifically, for the single-size multi-family cases VR − UB is always faster than CG−UB. Indeed, Tables 2 and 3 show that both average and maximum computation times are smaller for VR − UB in all the cases. When the single-family multi-size cases are considered (Table 1), CG−UB and VR − UB have comparable performance in the instances with a small number of jobs and σ_{10} distribution. However, VR − UB achieves remarkable improvements in gaps for the cases with large numbers of jobs and sizes with σ_5 distributions. For instance, in the experiments with $n = 200$;$s_i = \sigma_5\ \forall i$, the average gap moves from 56.80 % (two sizes) and 22.85 % (three sizes) for CG−UB to 1.37 % (two sizes) and 1.14 % for VR − UB, respectively. It is worth noting that the large CG−UB gap for larger instances is due to the enforced time limit. In terms of computation times, VR − UB results to be faster than CG−UB in almost all the cases. Also, CG−UB often reaches the time limit when 200 jobs are considered. Further experiments have been conducted for the cases where CG−UB reaches the time limit, and the results show that the computation times are consistently larger than VR − UB. For the multi-size family case, the same considerations hold, proving the outperforming of VR − UB with respect to CG−UB.

Although some gaps are slightly better for CG − UB (except for the cases where the time limit is exceeded), the relative difference is not worth the extra time required in comparison to the faster VR − UB. For instance, consider the single-size multi-family case with $n = 200$; $s_1 = \sigma_{10}$ and $n_f = 7$ in Table 2. The algorithm CG − UB finds the optimum value (without reaching the time limit) in an average time of 16.94s, with a mean gap equal to 0.83%; on the same instances VR − UB takes only 8.00s in the

worst case (less than the half of CG − UB average case), with an average gap equal to 1.39%, i.e., only half a point worse than CG − UB.

As the additional tests of Appendix A show, the efficiency of the algorithms is not affected for instances with larger job sizes and batch capacity. The average gaps are competitive for both algorithms; however, they suffer from the computation point of view since the pricing procedure becomes more difficult in these cases.

6 Conclusions

The parallel batch scheduling problem has become more and more addressed by the scientific and industrial communities because of its applications in many industrial fields.

This paper addresses the $1|p\text{-}batch, s_{ij} \leq b_i, incomp| \sum C_j$ problem. To the authors' knowledge, this paper is the first attempt in the literature to consider multiple sizes and family incompatibility constraints together. Also, no assumptions are made on the distribution and/or the value of the processing times.

The solution approach is based on the flow formulation of the problem by Alfieri et al. (2021); this formulation is exploited to develop two column-generation based heuristics: one is based on the price-and-branch heuristic (CG − UB) of Alfieri et al. (2021), the other on the variable rounding procedure (VR − UB) proposed in Druetto and Grosso (2022). The column generation finds a continuous-relaxed solution, then the two heuristics are used to move from the continuous to the integer solution of the problem. An extensive experimental campaign compared the two heuristics, which both proved to be very effective for this scheduling problem. Indeed, the proposed approaches can handle instances up to 200 jobs, and both find very good optimality gaps in all the addressed instances. Moreover, with a small number of jobs, the proposed algorithms are able to find optimal solutions in most of the cases.

Numerical results show that the smaller the job sizes, the more difficult the batch scheduling problem becomes. Having more than one size constraint simplifies the problem from the computation standpoint. Also, having more families simplifies the problem, as the number of feasible combinations of jobs is reduced.

Also, comparing the two heuristics, interesting results emerged. In simple instances (i.e., with a small number of jobs, large job sizes and a small number of families) the difference between the two approaches is not appreciable. The real gain can be perceived in difficult instances, where both computation times and gaps largely decrease. Specifically, instances with 200 jobs can not be solved by the CG − UB approach in 100 second time limit, however the variable rounding VR − UB is able to achieve good gaps in less than the time limit. In general, almost all instances are solved within a minute and the gap rarely overcomes 5%. The variable rounding procedure is therefore shown to be a valid alternative to the CG − UB.

At its current state, the proposed approach is able to solve single-machine batch scheduling problems. Further research will be devoted to adapting the approach to parallel machines and to weighted completion times ($\sum w_j C_j$).

Additional Tests

Table 5 shows the results of some additional tests. In these tests, a different size distribution and a different batch capacity are considered. The instances were generated with all jobs sizes sampled from a uniform distribution $U(1, 50)$ while the batch capacity

Table 5 Results for higher granularity jobs

n	s_1, s_2, s_3	n_f	α	CG − UB				VR − UB			
				Avg t	Max t	Avg % g	Max % g	Avg t	Max t	Avg % g	Max % g
Multiple sizes, single family											
20	σ_{50}, σ_{50}	1	6	0.04	0.1	0.67	3.3	0.06	0.1	0.84	3.7
60	σ_{50}, σ_{50}	1	2	0.33	0.5	0.18	0.5	0.60	1.1	0.30	1.0
100	σ_{50}, σ_{50}	1	1	1.83	3.8	0.12	0.3	3.76	7.4	0.30	1.0
20	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	1	10	0.58	0.6	0.00	0.0	1.25	1.8	0.00	0.0
60	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	1	7	5.26	5.4	0.15	0.7	34.27	>100.0	0.23	1.3
100	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	1	4	18.19	20.0	0.06	0.3	>100.00	>100.0	0.12	0.6
Single size, multiple families											
20	σ_{50}	3	7	0.03	0.1	0.25	1.3	0.02	0.1	0.31	1.3
20	σ_{50}	5	6	0.03	0.1	0.72	3.4	0.02	0.1	0.84	3.9
20	σ_{50}	7	10	0.02	0.1	0.00	0.0	0.02	0.1	0.00	0.0
20	σ_{50}	10	8	0.02	0.1	0.29	2.2	0.02	0.1	0.29	2.0
60	σ_{50}	3	1	0.21	0.3	0.84	1.3	0.21	0.3	1.43	2.6
60	σ_{50}	5	0	0.23	0.3	0.97	1.7	0.13	0.2	1.29	2.4
60	σ_{50}	7	0	0.17	0.3	0.80	2.2	0.10	0.1	1.57	4.2
60	σ_{50}	10	0	0.15	0.2	0.78	1.5	0.09	0.1	1.09	2.5
100	σ_{50}	3	0	2.49	4.6	0.53	1.1	1.62	2.6	1.07	2.9
100	σ_{50}	5	0	2.50	8.3	1.33	2.0	1.38	2.0	1.89	2.9
100	σ_{50}	7	0	1.32	2.9	1.21	2.2	1.06	1.4	1.98	3.8
100	σ_{50}	10	0	0.91	1.2	0.73	1.6	0.94	1.5	1.44	2.8
Multiple sizes, multiple families											
20	σ_{50}, σ_{50}	3	9	0.03	0.1	0.05	0.5	0.04	0.1	0.14	1.4
20	σ_{50}, σ_{50}	7	7	0.03	0.1	0.26	1.5	0.04	0.1	0.72	4.0
60	σ_{50}, σ_{50}	3	3	0.26	0.4	0.37	1.6	0.67	1.2	0.82	2.3
60	σ_{50}, σ_{50}	7	5	0.19	0.2	0.23	0.9	0.47	0.9	0.38	1.8
100	σ_{50}, σ_{50}	3	1	1.07	1.4	0.24	0.8	2.77	4.6	0.45	1.3
100	σ_{50}, σ_{50}	7	3	0.75	1.4	0.36	1.3	2.24	4.2	0.65	1.6
20	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	3	9	0.6	0.6	0.22	2.2	1.33	1.8	0.21	2.1
20	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	7	9	0.6	0.6	0.02	0.2	1.26	1.7	0.09	0.9
60	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	3	6	5.27	5.4	0.16	0.6	30.27	58.0	0.22	0.8
60	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	7	7	5.24	5.3	0.12	0.9	27.73	58.7	0.30	1.2
100	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	3	5	17.66	17.9	0.12	0.6	>100.00	>100.0	0.23	0.7
100	$\sigma_{50}, \sigma_{50}, \sigma_{50}$	7	8	17.41	17.8	0.07	0.5	>100.00	>100.0	0.14	0.8

b_i is set to 50 for all sizes $i = 1, \ldots, m$. The tests are run for all combinations used in Sect. 5. For issues related to excessive RAM usage, tests for instances up to 100 jobs are run.

The results on computation times show that having jobs with *higher granularity* with regards to their packing in batches, i.e., jobs with sizes included in a wider interval, makes the problem more difficult to solve. The pricing procedure requires in fact to optimally solve the cardinality-constrained multi-weight knapsack, which becomes harder when the number of feasible batches that can be formed increases.

Also, as noted in Sect. 4, the dynamic programming state space increases of one dimension for every size of the jobs, and the magnitude of these state space dimensions are exactly the maximum batch capacities b_i for all $i \in \{1, \ldots, m\}$. Thus, having a larger batch capacity leads to a considerably higher memory usage.

With respect to the percentage gaps, the algorithms still perform well, with very good average and maximal gaps, showing that the quality is not affected by the granularity of job sizes.

Author Contributions Conceptualization: AD, EP, ER; Methodology: AD, EP, ER; Formal analysis and investigation: AD, EP, ER; Writing—original draft preparation: AD, EP, ER; Writing—review and editing: AD, EP, ER.

Funding Open access funding provided by Università degli Studi di Torino within the CRUI-CARE Agreement. No funds, grants, or other support was received.

Availability of data and materials The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Code availability The code developed for this work is available from the corresponding author on reasonable request.

Declarations

Conflict of interest The authors have no financial or proprietary interests in any material discussed in this article.

Ethics approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Alfieri A, Druetto A, Grosso A et al (2021) Column generation for minimizing total completion time in a parallel-batching environment. J Sched 24(6):569–588. https://doi.org/10.1007/s10951-021-00703-9

Azizoglu M, Webster S (2000) Scheduling a batch processing machine with non-identical job sizes. Int J Prod Res 38(10):2173–2184. https://doi.org/10.1080/00207540050028034

Azizoglu M, Webster S (2001) Scheduling a batch processing machine with incompatible job families. Comput Ind Eng 39(3–4):325–335. https://doi.org/10.1016/S0360-8352(01)00009-2

Chandru V, Lee CY, Uzsoy R (1993) Minimizing total completion time on batch processing machines. Int J Prod Res 31(9):2097–2121. https://doi.org/10.1080/00207549308956847

Desrosiers J, Lübbecke M (2005) A primer in column generation. Column Generation. Springer, Boston, pp 1–32. https://doi.org/10.1007/0-387-25486-2_1

Dobson G, Nambimadom RS (2001) The batch loading and scheduling problem. Oper Res 49(1):52–65. https://doi.org/10.1287/opre.49.1.52.11189

Druetto A, Grosso A (2022) Column generation and rounding heuristics for minimizing the total weighted completion time on a single batching machine. Comput Oper Res 139(105):639. https://doi.org/10.1016/j.cor.2021.105639

Emde S, Polten L, Gendreau M (2020) Logic-based benders decomposition for scheduling a batching machine. Comput Oper Res 113(104):777. https://doi.org/10.1016/j.cor.2019.104777

Graham RL, Lawler EL, Lenstra JK et al (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of discrete mathematics, vol 5. Elsevier, New York, pp 287–326. https://doi.org/10.1016/S0167-5060(08)70356-X

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. https://doi.org/10.1016/j.cie.2017.09.037

Ikura Y, Gimple M (1986) Efficient scheduling algorithms for a single batch processing machine. Oper Res Lett 5(2):61–65. https://doi.org/10.1016/0167-6377(86)90104-5

Kellerer H, Pferschy U, Pisinger D (2004) Knapsack problems. Springer, Berlin Heidelberg. https://doi.org/10.1007/978-3-540-24777-7

Liu J, Li Z, Chen Q et al (2016) Controlling delivery and energy performance of parallel batch processors in dynamic mould manufacturing. Comput Oper Res 66:116–129. https://doi.org/10.1016/j.cor.2015.08.006

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. https://doi.org/10.1016/j.cor.2006.02.003

Mönch L, Fowler JW, Mason SJ (2013) Production planning and control for semiconductor wafer fabrication facilities: modeling, analysis, and systems. Springer Science & Business Media, Berlin. https://doi.org/10.1007/978-1-4614-4472-5

Muter A (2020) Exact algorithms to minimize makespan on single and parallel batch processing machines. Eur J Oper Res 285(2):470–483. https://doi.org/10.1016/j.ejor.2020.01.065

Ozturk O (2020) A truncated column generation algorithm for the parallel batch scheduling problem to minimize total flow time. Eur J Oper Res 286(2):432–443. https://doi.org/10.1016/j.ejor.2020.03.044

Ozturk O, Espinouse ML, Mascolo MD et al (2012) Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates. Int J Prod Res 50(20):6022–6035. https://doi.org/10.1080/00207543.2011.641358

Potts CN, Kovalyov MY (2000) Scheduling with batching: a review. Eur J Oper Res 120(2):228–249. https://doi.org/10.1016/S0377-2217(99)00153-8

Rafiee Parsa N, Karimi B, Moattar Husseini S (2016) Minimizing total flow time on a batch processing machine using a hybrid max-min ant system. Comput Ind Eng 99:372–381. https://doi.org/10.1016/j.cie.2016.06.008

Shahidi-Zadeh B, Tavakkoli-Moghaddam R, Taheri-Moghadam A et al (2017) Solving a bi-objective unrelated parallel batch processing machines scheduling problem: a comparison study. Comput Oper Res 88:71–90. https://doi.org/10.1016/j.cor.2017.06.019

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. https://doi.org/10.1016/0098-1354(79)80031-9

Tan Y, Mönch L, Fowler JW (2018) A hybrid scheduling approach for a two-stage flexible flow shop with batch processing machines. J Sched 21(2):209–226. https://doi.org/10.1007/s10951-017-0530-4

Uzsoy R (1994) Scheduling a single batch processing machine with non-identical job sizes. Int J Prod Res 32(7):1615–1635. https://doi.org/10.1080/00207549408957026

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. https://doi.org/10.1080/00207543.2019.1617447

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.