Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 59-10 (2025) 1023–1028

A new parallel-batch scheduling problem with non-identical jobs, compatible families, and setup times

Annalisa Castelletti * **Renata Mansini** * **Lorenzo Moreschini** *

* *University of Brescia, Via Branze 38, 25125, Brescia, Italy (e-mail: annalisa.castelletti@unibs.it, renata.mansini@unibs.it, lorenzo.moreschini@unibs.it).*

Abstract: This paper addresses a new parallel-batch scheduling problem on a single batch-processing machine, inspired by an industrial heat treatment process. The goal is forming and sequencing batches with non-identical jobs belonging to compatible families while minimizing the makespan. The longest job in each batch determines the batch processing time, and setup times depend on the sequence of dominant families. We propose a mixed-integer linear programming (MILP) formulation for the problem and some valid inequalities. Computational experiments demonstrate the formulation's effectiveness and compare the performance of Cplex and Gurobi solvers.

Keywords: Parallel-batch scheduling, Single machine, Non-identical jobs, Sequence-dependent setups, Compatible families, Mixed Integer Linear Programming, Combinatorial optimization.

1. INTRODUCTION

Batch scheduling problems involve efficiently sequencing and timing jobs under resource constraints. They arise in different application fields, such as manufacturing, data center operations, and task planning in distributed systems, yet they are computationally challenging as many problems in this family are NP-hard.

Potts and Kovalyov (2000) classify batch scheduling *models* according to how jobs are grouped and processed. The *family scheduling model* (also known as *Serial-Batch* or *s-batch*) divides jobs into families based on common characteristics (such as material, color, or required machine settings). In this model, a batch is the largest subset of jobs belonging to the same family and consecutively processed on a machine that can handle one job at a time. Minimal or no setup times are needed within the same family, enabling continuous processing and reducing idle time; in contrast, substantial setup times are required when switching between families (due to operations such as machine reconfiguration or clean-up). This model is crucial in industries where setup times substantially impact production efficiency. The *batch machine model* (also known as *Parallel-Batch* or *p-batch*) gathers jobs to be processed simultaneously on the same machine. The longest job within a batch determines its processing time. This model addresses scenarios such as burn-in operations in semiconductor production, where simultaneous processing of multiple jobs is allowed and profitable.
Once jobs have been processed, their availability may or may not be immediate: under the *job availability* assumption (applicable only to the family scheduling model) they are immediately available once their processing completes, whereas under the *batch availability* assumption (common in the batch machine model) jobs become available only when the entire batch has been processed.
Other common features in both models include *identical* or *non-identical* jobs (e.g. differing in their processing times or sizes) and *compatible* or *incompatible* families (meaning that batches can be made of homogeneous or heterogeneous jobs). Finally, setup times may be required between two consecutive batches and they may depend on the batch sequence (*sequence-dependent*) or not (*sequence-independent*).

Motivated by industrial heat treatment applications, this paper explores a novel single batch-processing machine scheduling problem. Following the three-field notation $\alpha \,|\, \beta \,|\, \gamma$ by Graham et al. (1979), we address the $1 \,|\, s_j,\ p-batch,\ SU^k_{fg} \,|\, C_{max}$ scheduling problem, under the *p-batch* model with batch availability assumption. In this problem, non-identical jobs (s_j) must be processed simultaneously on a single (1) batch-processing machine to minimize the maximum completion time of the jobs (the *makespan* C_{max}) with compatible families. The problem also involves *dominant* families, which establish the machine settings. The order in which these families are processed defines the setup times (SU^k_{fg}). We call this problem the Single Batch-Processing Machine with Non-identical job sizes, Compatible families, and Sequence-dependent setup times (SBPM-NCS problem).
The paper provides some important contributions: (i) the introduction of a new p-batch scheduling problem, along with its mathematical formulation; (ii) the derivation of a lower bound on the number of non-empty batches, which is used to incorporate valid inequalities into the model; (iii) the development of a lower bound on the makespan of a sequence of subproblems, to improve the computational efficiency of our algorithm; (iv) computational results comparing the model's performance using different MIP solvers (Cplex and Gurobi).

10.1016/j.ifacol.2025.09.173

The paper is organized as follows. Section 2 reviews the current literature concerning parallel batch scheduling problems with non-identical jobs and setup times. Section 3 provides the problem definition and notation, whereas in Section 4, we propose a MILP model to solve the SBPM-NCS scheduling problem. Section 5 presents the experimental results obtained on synthetic instances. Finally, in Section 6, we resume our findings and present future research directions.

2. LITERATURE REVIEW

In this section, we focus on problems that share similar assumptions to the SBPM-NCS scheduling problem, especially those involving setup times on a single batch-processing machine, with or without identical jobs and compatible families. Readers interested in serial batching are referred to the dedicated section in Potts and Kovalyov (2000) and from now on we concentrate on parallel batching problems.

Dang and Kang (2004) deal with the real-world case of minimizing the total weighted completion time in a burn-in operation stage. In their problem, products must be divided into batches. The setup time of each batch equals the longest setup time for the included products; therefore, batch setups are sequence-independent. The authors develop a polynomial-time approximation algorithm with a performance guarantee of 2.

Motivated by a scheduling problem arising in semiconductor wafer fabrication, Li and Qiao (2008) develop an Ant-colony-optimization-based algorithm to solve the problem of minimizing the weighted tardiness of a single batch-processing machine with families of jobs and setup times. Under their assumptions, families are incompatible (i.e. each batch must contain jobs from the same family) and there are sequence-dependent setup times if and only if consecutive batches contain jobs from different families.

Qiu et al. (2024) deal with the single batch-processing machine lot-sizing and scheduling problem for a printed circuit board drilling production system. Jobs are identical, have a due date, and batch setup times are sequence-dependent. The authors propose an integer programming model that faces the problem as a whole and a two-stage model that first creates sub-lots and subsequently schedules them. They also developed a predictive-reactive scheduling method to minimize the total tardiness in a dynamic environment, relying on an adaptive genetic algorithm coupled with an iterated greedy search.

Cheng et al. (2023) investigate the p-batch scheduling problem with incompatible and deteriorating jobs, motivated by the soaking process under separate heating mode in the iron and steel industry. The authors consider three problem variants and provide exact mathematical formulations and approximated algorithms to minimize the makespan. The first two variants address the special case of equal job sizes and equal job processing times, respectively. The third one considers the general case of arbitrary sizes and processing times.

Setup times are required for consecutive batches involving incompatible families. Gong and Tang (2009) present a batch scheduling model on a single machine for a production and transportation problem, where several vehicles with unitary capacity travel between the machine and a holding area. The single machine processes several jobs simultaneously as a batch and requires a setup cost every time a new batch starts its process. The authors develop an integrated heuristic for both the scheduling and transportation problems, minimizing the sum of the total completion time and the total setup cost.

Kong et al. (2020) investigate the single batch-processing machine problem with deteriorating jobs, setup times, non-identical jobs, and rejection, motivated by the real case of the ingot soaking process. Setup times are required before each batch and jobs are not partitioned into families. The authors propose a hybrid algorithm that combines a heuristic procedure with a dynamic programming algorithm to solve the problem.

Rezaeian and Zarook (2018) propose a MILP formulation for the SBPM scheduling problem with arbitrarily sized jobs that arrive dynamically, incompatible families, sequence-dependent family setup times, and job preemption between batches. The authors aim to minimize the makespan and the maximum tardiness using a bi-objective genetic algorithm and an ϵ-constraint method.

To our knowledge, a makespan minimization batch scheduling problem with non-identical jobs, compatible families, and sequence-dependent setup times determined by dominant families in consecutive pairs of batches has not been studied so far.

3. PROBLEM STATEMENT

3.1 Problem description

Batch-processing machines play a crucial role in many manufacturing processes such as semiconductor fabrication, printed circuit boards production, heat treatment, thermochemical testing, and plastic injection molding. Consider the real case of heat treatment operations in metalworking industries where the single batch-processing machine is an oven with a limited capacity and jobs are semi-finished products to be treated inside the oven. Each semi-finished product has a minimum processing time (i.e. a minimum time to spend inside the oven) and belongs to a family (a category of products) that determines the minimum oven temperature required for its treatment. Each batch can be composed of products belonging to different families and we define its *dominant family* as the one that requires the highest minimum temperature among all families with jobs included in the batch. We assume that the temperature requirements for each family are distinct [1] . When a batch is processed, the dominant family determines the oven temperature. The processing time of the batch equals the maximum among the processing times of each semi-finished product inside the batch. The setup time, required to prepare the oven for processing the next batch, depends on the dominant families of the current and next batches. The problem asks to determine the number of batches, their composition, and their processing sequence to treat all semi-finished products while minimizing the *makespan*, i.e. the completion time of the last batch.

From now on we will refer to the semi-finished products as *jobs*, and to the industrial oven as a *batch-processing machine*.

[1] If two families share the same minimum temperature, they can be merged into a single family to satisfy this assumption.

3.2 Problem definition

Let J denote a finite set of jobs to be processed on a single batch-processing machine. Each job $j \in J$ has a size $s_j \in \mathbb{Z}^+$ and requires a minimum processing time $p_j \in \mathbb{Z}^+$. Any subset of jobs can be processed as a batch, provided that their total size is not larger than the machine capacity $B \in \mathbb{Z}^+$. The processing time taken by a batch equals the longest processing time among those of the jobs inside the batch. We assume batch availability: all jobs in a batch begin and finish their processing simultaneously and become available only when the entire batch has completed. Once the processing of the batch has begun, it cannot be stopped nor can other jobs be inserted in that batch.

Let F denote the index set of families. Job families form a partition $\mathcal{F} = \{J_f \subseteq J \mid f \in F\}$ of J, i.e. each job $j \in J$ belongs to exactly one family $J_f \in \mathcal{F}$. We will refer to a family either by the subset $J_f \subseteq J$ or simply by its index $f \in F$, and we will denote by $f_j \in F$ the index of the (unique) family containing the job j, i.e. $j \in J_{f_j}$. Each family J_f is associated with a minimum processing temperature $c_f \in \mathbb{Z}^+$ required by each of its jobs, and it is compatible with any other family (batches can be formed in a heterogeneous manner).

The setup time between two batches depends on their respective dominant family and, for each pair of families $f, g \in F$, the associated setup time $u_{fg} \in \mathbb{Z}^+$ is known. Moreover, the time required to start or shut down the batch-processing machine is determined by the dominant family in the first and last batch, respectively. Each family $f \in F$ has a specified startup time $v_f \in \mathbb{Z}^+$ and shutdown time $w_f \in \mathbb{Z}^+$.

The objective is to minimize the makespan, i.e. the completion time of the last batch, which equals the sum of all batch processing times plus the setup times for each pair of consecutive batches plus the startup time of the first dominant family and the shutdown time of the last one.

4. MATHEMATICAL FORMULATION

To model the SBPM-NCS problem, we assume a predefined number of batches allowing for empty ones. In any feasible solution, empty batches always follow non-empty ones ensuring correct evaluation of setup times. Therefore, a (optimal) solution for the model corresponds to the (optimal) solution of the SBPM-NCS problem obtained by discarding the trailing empty batches.

Let $K = \{1, 2, \ldots, \overline{k}\}$ denote the set of batches available. A straightforward bound on the number of batches in any feasible solution is the total number of jobs. Thus, we provisionally set $\overline{k} = |J|$. In Section 4.3, we describe a procedure to reduce this value and strength the formulation while preserving optimality.

The proposed MILP model uses the following variables:

- $x_{jk} \in \{0, 1\}$: equals 1 if job $j \in J$ belongs to batch $k \in K$, and 0 otherwise,
- $y_{fk} \in \{0, 1\}$: equals 1 if family $f \in F$ is dominant in batch $k \in K$, and 0 otherwise,
- $z_{kfg} \in \{0, 1\}$: equals 1 if family $f \in F$ is dominant in batch $k \in K \setminus \{\overline{k}\}$ and family $g \in F$ is dominant in the following batch, and 0 otherwise,
- $t_{fk} \in \{0, 1\}$: equals 1 if family $f \in F$ is dominant in the last non-empty batch $k \in K$, and 0 otherwise,
- $P_k \in \mathbb{R}^+$: equals the processing time of batch $k \in K$.

The formulation is as follows.

$$\min \sum_{k \in K} P_k + \sum_{f \in F} v_f y_{f1} + \sum_{\substack{f,g \in F \\ k \in K \setminus \{\overline{k}\}}} u_{fg} z_{kfg} + \sum_{\substack{k \in K \\ f \in F}} w_f t_{fk} \quad (1)$$

s.t.

$$P_k \geq p_j x_{jk}, \qquad \forall j \in J,\ \forall k \in K, \quad (2)$$

$$\sum_{k \in K} x_{jk} = 1, \qquad \forall j \in J, \quad (3)$$

$$\sum_{j \in J} s_j x_{jk} \leq B, \qquad \forall k \in K, \quad (4)$$

$$\sum_{f \in F} y_{fk} \leq 1, \qquad \forall k \in K, \quad (5)$$

$$y_{fk} \leq \sum_{j \in J_f} x_{jk}, \qquad \forall f \in F,\ \forall k \in K, \quad (6)$$

$$\sum_{f \in F} y_{f(k+1)} \leq \sum_{f \in F} y_{fk}, \qquad \forall k \in K \setminus \{\overline{k}\}, \quad (7)$$

$$\sum_{f \in F} c_{f_j} y_{fk} \geq c_{f_j} x_{jk}, \qquad \forall j \in J,\ \forall k \in K, \quad (8)$$

$$z_{kfg} \geq y_{fk} + y_{g(k+1)} - 1, \qquad \forall f, g \in F,\ \forall k \in K \setminus \{\overline{k}\}, \quad (9)$$

$$z_{kfg} \leq y_{fk}, \qquad \forall f, g \in F,\ \forall k \in K \setminus \{\overline{k}\}, \quad (10)$$

$$z_{kfg} \leq y_{g(k+1)}, \qquad \forall f, g \in F,\ \forall k \in K \setminus \{\overline{k}\}, \quad (11)$$

$$\sum_{f \in F} \sum_{k \in K} t_{fk} = 1, \quad (12)$$

$$t_{fk} \leq y_{fk}, \qquad \forall f \in F,\ \forall k \in K, \quad (13)$$

$$\sum_{f \in F} t_{fk} + \sum_{f \in F} y_{f(k+1)} \leq 1, \qquad \forall k \in K \setminus \{\overline{k}\}, \quad (14)$$

$$x_{jk} \in \{0, 1\}, \qquad \forall j \in J,\ \forall k \in K, \quad (15)$$

$$y_{fk} \in \{0, 1\}, \qquad \forall f \in F,\ \forall k \in K, \quad (16)$$

$$z_{kfg} \in \{0, 1\}, \qquad \forall f, g \in F,\ \forall k \in K \setminus \{\overline{k}\}, \quad (17)$$

$$t_{fk} \in \{0, 1\}, \qquad \forall f \in F,\ \forall k \in K, \quad (18)$$

$$P_k \geq 0, \qquad \forall k \in K. \quad (19)$$

The objective function (1) ensures the minimization of the makespan given by the sum of four terms: the sum of processing times of all batches, the startup time set by the dominant family in the first batch, the setup times due to the pairs of dominant families in consecutive batches, and the shutdown time set by the dominant family in the last non-empty batch. Constraints (2), combined with the objective function minimization, ensure that the processing time of each batch equals the maximum processing time among its jobs. Constraints (3) assign each job $j \in J$ to exactly one batch $k \in K$. Constraints (4) ensure that the total size of the jobs allocated in a batch does not exceed the machine capacity B. Constraints (5) guarantee that there exists at most one dominant family in each batch (including empty ones for which $\sum_{f \in F} y_{fk} = 0$).

Constraints (6) ensure that a family cannot be dominant in a batch if none of its jobs belong to it. Constraints (7) ensure that empty batches always come after the non-empty ones. Constraints (8) define the dominant family in each batch as the one requiring the highest temperature, relying on the condition that each batch has at most one dominant family. Constraints (9), (10) and (11) set the variable z_{kfg} equal to one if and only if family $f \in F$ is dominant in batch $k \in K$ and family $g \in F$ is dominant in the following one.
Next, Constraints (12), (13) and (14) ensure that the shutdown time is determined by exactly one dominant family that must be present in the last non-empty batch. Finally, conditions (15)-(19) define decision variables.

4.1 Lower bound on the number of batches

Formulation (1)–(19) is based on a predefined number of batches $\overline{k}$ upper bounded by the number of jobs (empty batches are allowed). To further strengthen the model, we compute a lower bound for the number of batches needed in any feasible solution. As each batch has a capacity that cannot be exceeded by the total size of the jobs it contains, it follows that

$$\hat{k} = \left\lceil \frac{\sum_{j \in J} s_j}{B} \right\rceil \tag{20}$$

provides a lower bound on the number of non-empty batches that must be present in any feasible solution. Therefore, the model can be strengthened with the addition of the following valid inequalities:

$$P_k \geq \min_{j \in J} p_j, \qquad \forall k \in K : 1 \leq k \leq \hat{k}, \tag{21}$$

$$\sum_{j \in J} x_{jk} \geq 1, \qquad \forall k \in K : 1 \leq k \leq \hat{k}, \tag{22}$$

$$\sum_{f \in F} y_{fk} = 1, \qquad \forall k \in K : 1 \leq k \leq \hat{k}, \tag{23}$$

$$\sum_{f,g \in F} z_{kfg} = 1, \quad \forall k \in K : 1 \leq k \leq \hat{k} - 1, \tag{24}$$

$$t_{fk} = 0, \qquad \forall f \in F,\ \forall k \in K : 1 \leq k \leq \hat{k} - 1. \tag{25}$$

Constraints (21) set the processing time of a non-empty batch to be greater than or equal to the minimum processing time among all jobs, and Constraints (22) ensure that non-empty batches contain at least one job. Constraints (23) force the existence of exactly one dominant family in non-empty batches (these inequalities are strictly better than (5)). Constraints (24) account for the setup time between two consecutive non-empty batches. It is worth noticing that these constraints are all valid inequalities, i.e. they do not exclude any feasible solution of the initial formulation.

4.2 Lower bounds on the makespan

Allowing more batches than necessary for an optimal solution leads to a weaker linear relaxation, thus resulting in a downgrade of the quality of solutions found within a given time limit by a Branch-&-Bound algorithm. Relying on the theoretical upper bound ($\overline{k} = |J|$) provides a MILP model that is often too weak to be solved within a reasonable amount of time.
To address this issue, we derive a formula to compute a lower bound for the makespan of any feasible solution using a fixed number $k' \in \mathbb{N}$ of non-empty batches. In Section 4.3, we show how this lower bound can be used to construct a MILP model with fewer than $|J|$ batches while guaranteeing global optimality for the SBPM-NCS problem. Let $j_1, j_2, \ldots, j_{|J|}$ be a permutation of the jobs such that $p_{j_i} \leq p_{j_{(i+1)}}$ for each i between 1 and $|J| - 1$. A lower bound on the makespan of any feasible solution using exactly $k' \in \{1, 2, \ldots, \overline{k}\}$ non-empty batches is given by

$$\delta_{k'} = \min \sum_{i=1}^{k'} p_{j_i} + \min_{f \in F} v_f + (k' - 1) \min_{f,g \in F} u_{fg} + \min_{f \in F} w_f, \tag{26}$$

as each of the four terms in (26) is a lower bound for the corresponding term appearing in the objective function (1). Note that the complexity to compute the bound is $O(|J| + |F|^2)$ (assuming the processing times to be sorted).

4.3 Solution strategy

To solve the SBPM-NCS problem optimally without relying on a MILP model with up to $\overline{k} = |J|$ non-empty batches, we select a small enough value for $\overline{k}$. This can be determined using $\overline{k} = \epsilon \hat{k}$, where $\hat{k}$ is the lower bound computed in (20) and $\epsilon \in [1, \frac{|J|}{\hat{k}}]$ is a manageable real coefficient. Let $\zeta^*_{\overline{k}}$ be the optimal solution value for the SBPM-NCS problem with at most $\overline{k}$ non-empty batches. We then compute the lower bound in (26) with $\overline{k}+1$ non-empty batches. If $\zeta^*_{\overline{k}} \leq \delta_{\overline{k}+1}$, we can conclude that such solution is also globally optimal as no solution with more than $\overline{k}$ batches can achieve a makespan lower than $\delta_{\overline{k}+1}$.

5. COMPUTATIONAL EXPERIMENTS

In this section, we present the experimental results obtained on synthetic instances using either Cplex 22.1.1 or Gurobi 11.0.3 to solve the MILP model. All computations have been executed inside a QEMU virtual machine on two Intel(R) Xeon(R) Gold 6140 CPU at 2.30GHz configured with 16 logical cores paired with 64 GB of RAM running Windows 10 Pro 22H2.

5.1 Instance generation

Param	Values	Param	Values
$\|J\|$	25	c_f	Uniform[200, 500]
$\|F\|$	5, 10, 15	λ	0.03
B	50, 75	σ	0.05
p_j	Uniform[1, 10]	τ	1.3
s_j	Uniform[5, 20]	η	Uniform[0.7, 1.3]

Table 1. Parameter values.

The SBPM-NCS scheduling problem has never been studied before, thus no benchmark instances are available in the literature. We generated synthetic instances to validate the effectiveness of our approach and compare the performance of Cplex and Gurobi when solving the proposed MILP model.

As reported in Table 1, to generate the instances, we set the number of jobs $|J| = 25$ and then defined six combinations $(B, |F|)$ varying the batch capacity B and the number of families $|F|$, with jobs assigned uniformly at random to families. For each combination, 5 instances have been generated for a total of 30 instances. In each instance, processing times p_j $(j \in J)$, job sizes s_j $(j \in J)$ and processing temperatures c_f $(f \in F)$ have been generated using discrete uniform distributions. Startup and shutdown times have been set proportionally to the temperature required by the (dominant) family appearing in the first and last batch, respectively, whereas setup times have been chosen in proportion to the difference of the temperatures required by each pair of families. To generate realistic instances we selected different coefficients $\lambda, \sigma, \tau \in \mathbb{R}^+$ to reflect the different amount of time required to lower or raise the temperature and we considered a short setup time when a family is dominant in consecutive batches (to account for machine maintenance operations). Finally, each startup, setup, and shutdown time has been altered by a random noise η and rounded to the next integer. These values have been computed as follows:

$$v_f := \lceil \eta \lambda c_f \rceil, \qquad \forall f \in F, \tag{27}$$

$$u_{fg} := \begin{cases} \lceil \eta \lambda (c_g - c_f) \rceil, & c_f < c_g, \\ \lceil \eta \tau \rceil, & c_f = c_g, \\ \lceil \eta \sigma (c_f - c_g) \rceil, & c_f > c_g, \end{cases} \qquad \forall f, g \in F, \tag{28}$$

$$w_f := \lceil \eta \sigma c_f \rceil, \qquad \forall f \in F. \tag{29}$$

Table 1 shows the chosen parameter values.

5.2 Computational results

All Cplex and Gurobi parameters have been left unchanged to their default value, except for the time limit set to 3600 seconds. For each instance, we computed the lower bound on the number of required batches $\hat{k}$ as in (20) and strengthened the model with the addition of valid inequalities (21) to (25). In the solution procedure, to avoid solving large MILP models allowing up to the number of batches equal to the number of jobs, we set $\overline{k} = \epsilon \hat{k}$ choosing $\epsilon = 2$ or $\epsilon = 2.5$.

Table 2 shows the results obtained by solving the MILP model for each instance, for each solver, and for each chosen value of ϵ for a total of 120 MILP models ($30 \times 2 \times 2$). The first column displays instance names using the pattern `instX_J_F_B` where `X` is an unique identifier of the instance, `J` is the number of jobs, `F` is the number of families available and `B` is the capacity of each batch.
The remaining part of the table is divided into two sections, one for each value of ϵ. Each section is internally divided into three parts. The first and the second ones display the performance of Cplex and Gurobi: columns *Obj*, *Gap* and *Time* show the objective function values, the MIP gaps, and the runtimes obtained by solving the model with both solvers. Runtimes lower than 3600 seconds always coincides with MIP gaps equal to 0%, meaning that the solver was able to find an optimal solution. The third part of both sections reports some statistics about the best (often optimal) solution available. Columns k^* and $\overline{k}$ show the number of batches used in the best available solution and the maximum number of batches chosen to build the model. For $\epsilon = 2$, the former is always half of the latter, meaning that the lower bound $\hat{k}$ actually provides a good bound on the number of batches. Next, column $\zeta^*_{\overline{k}}$ reports the best known objective value for the MILP model built using at most $\overline{k}$ batches, and column $\delta_{\overline{k}+1}$ displays the lower bound on the value of a feasible solution using at least $\overline{k} + 1$ batches. Whenever $\zeta^*_{\overline{k}} \leq \delta_{\overline{k}+1}$, it means that the optimal solution of the model built allowing only up to $\overline{k}$ batches is actually optimal for the SBPM-NCS problem (with any number of batches). A checkmark (✓) in column O appears whenever such condition is satisfied.
For $\epsilon = 2$ Cplex finds an optimal solution 18 times out of 30 instances (60%), while Gurobi reaches an optimal solution in 27 cases (90%). As expected, for $\epsilon = 2.5$ the overall performance worsen. Cplex finds one less optimal solution with the average runtime moving from 1651.04 seconds to 1814.70 seconds and the average gap changing from 4.81% to 7.32%. Similarly, Gurobi's average runtime increases from 588.98 seconds to 693.17 seconds, and the average MIP gap changes from 1.00% to 1.05%. Although the average of the objective values found by Cplex and Gurobi is roughly similar (56.03 and 56.13% vs. 55.87% and 55.87%), we obtained better overall performance by solving our models with Gurobi, which takes 63% less time on average. Focusing on the 12 out of 30 instances that reached the imposed time limit of 3600 seconds with Cplex, Gurobi finds the optimal solution for 9 of those, using a computational time lower than or equal to 1048.7 seconds for $\epsilon = 2$ and 1557.5 seconds for $\epsilon = 2.5$.
Regarding the optimality guarantee, we can prove that the solution of each MILP model is globally optimal for the SBPM-NCS problem in 19 cases out of 30 when $\epsilon = 2$ and in 28 when $\epsilon = 2.5$.

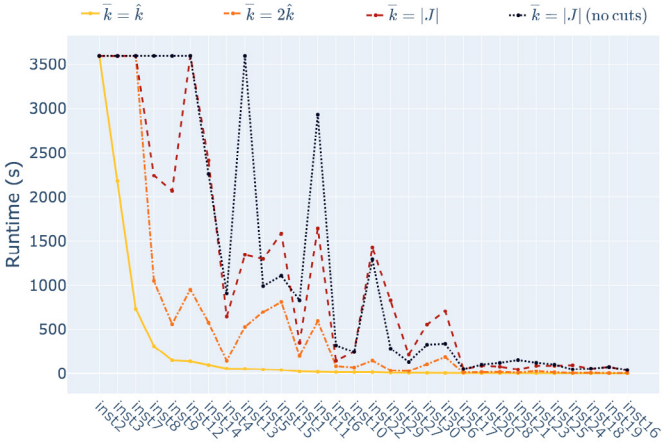

Fig. 1. Runtimes for different MILP models.

To further assess the benefits of solving MILP models as small as possible, in Figure 1 we compare the runtimes measured using different values of $\overline{k}$. For each instance, and each chosen value of $\overline{k}$, we display the shortest runtime achieved among Cplex and Gurobi. As we already noticed that $\epsilon = 1$ always provides a sufficient number of batches to solve each instance, the solid orange line represents the runtimes obtained setting $\overline{k} = \hat{k}$ and we use these values to sort the instances along the x-axis. This line always lies below all the others and all MILP models (except one) are optimally solved within the chosen time limit of 3600 seconds. Next, the dash-dotted red line,

Instance	$\epsilon = 2$											$\epsilon = 2.5$										
	Cplex			Gurobi			Stats					Cplex			Gurobi			Stats				
	Obj	Gap	Time	Obj	Gap	Time	k^*	$\overline{k}$	$\zeta^*_{\overline{k}}$	$\delta_{\overline{k}+1}$	O	Obj	Gap	Time	Obj	Gap	Time	k^*	$\overline{k}$	$\zeta^*_{\overline{k}}$	$\delta_{\overline{k}+1}$	O
inst1-25_5_50	60	2.5	3600.0	60	0.0	197.7	6	12	60	67	✓	60	0.0	2021.3	60	0.0	398.8	6	15	60	90	✓
inst2-25_5_50	64	15.4	3600.0	64	12.5	3600.0	7	14	64	90	✓	64	17.9	3600.0	64	12.5	3600.0	7	18	64	124	✓
inst3-25_5_50	78	18.1	3600.0	78	11.5	3600.0	8	16	78	116	✓	78	21.0	3600.0	78	11.5	3600.0	8	20	78	154	✓
inst4-25_5_50	64	0.0	1280.0	64	0.0	143.7	6	12	64	78	✓	64	0.0	2178.0	64	0.0	455.9	6	15	64	102	✓
inst5-25_5_50	69	5.0	3600.0	69	0.0	696.7	7	14	69	76	✓	69	8.8	3600.0	69	0.0	711.7	7	18	69	111	✓
inst6-25_10_50	56	0.0	166.5	56	0.0	84.5	7	14	56	56	✓	56	7.2	3600.0	56	0.0	198.3	7	18	56	86	✓
inst7-25_10_50	69	15.4	3600.0	68	5.9	3600.0	7	14	68	93	✓	69	23.4	3600.0	68	7.4	3600.0	7	18	68	129	✓
inst8-25_10_50	62	4.6	3600.0	62	0.0	1048.7	7	14	62	77	✓	62	18.0	3600.0	62	0.0	1557.5	7	18	62	109	✓
inst9-25_10_50	67	18.4	3600.0	65	0.0	558.1	7	14	65	72	✓	68	23.8	3600.0	65	0.0	973.0	7	18	65	102	✓
inst10-25_10_50	63	0.0	337.3	63	0.0	69.0	7	14	63	76	✓	63	0.0	1078.7	63	0.0	148.3	7	18	63	105	✓
inst11-25_15_50	58	19.1	3600.0	57	0.0	594.6	6	12	57	58	✓	58	12.6	3600.0	57	0.0	524.5	6	15	57	80	✓
inst12-25_15_50	64	22.1	3600.0	63	0.0	951.7	7	14	63	74	✓	63	20.5	3600.0	63	0.0	1194.6	7	18	63	107	✓
inst13-25_15_50	64	1.2	3600.0	64	0.0	528.3	7	14	64	68	✓	65	21.4	3600.0	64	0.0	1438.4	7	18	64	101	✓
inst14-25_15_50	59	9.4	3600.0	59	0.0	575.9	7	14	59	72	✓	61	24.6	3600.0	59	0.0	927.4	7	18	59	105	✓
inst15-25_15_50	57	13.2	3600.0	57	0.0	810.7	6	12	57	61	✓	57	10.6	3600.0	57	0.0	696.6	6	15	57	80	✓
inst16-25_5_75	45	0.0	7.6	45	0.0	4.5	4	8	45	44		45	0.0	19.0	45	0.0	6.8	4	10	45	56	✓
inst17-25_5_75	52	0.0	14.9	52	0.0	11.4	5	10	52	59	✓	52	0.0	82.7	52	0.0	18.5	5	13	52	77	✓
inst18-25_5_75	50	0.0	12.5	50	0.0	7.3	4	8	50	47		50	0.0	39.4	50	0.0	10.7	4	10	50	59	✓
inst19-25_5_75	45	0.0	7.5	45	0.0	3.2	4	8	45	36		45	0.0	22.0	45	0.0	11.4	4	10	45	44	
inst20-25_5_75	53	0.0	56.4	53	0.0	15.5	5	10	53	47		53	0.0	126.9	53	0.0	31.7	5	13	53	69	✓
inst21-25_10_75	46	0.0	33.9	46	0.0	9.8	4	8	46	38		46	0.0	71.5	46	0.0	17.4	4	10	46	51	✓
inst22-25_10_75	56	0.0	2667.3	56	0.0	146.1	5	10	56	65	✓	56	9.7	3600.0	56	0.0	263.1	5	13	56	86	✓
inst23-25_10_75	51	0.0	69.5	51	0.0	25.8	5	10	51	50		51	0.0	151.3	51	0.0	20.8	5	13	51	70	✓
inst24-25_10_75	44	0.0	25.9	44	0.0	5.4	4	8	44	33		44	0.0	34.9	44	0.0	7.7	4	10	44	43	
inst25-25_10_75	43	0.0	36.2	43	0.0	11.9	4	8	43	36		43	0.0	80.6	43	0.0	13.1	4	10	43	47	✓
inst26-25_15_75	48	0.0	104.2	48	0.0	185.0	4	8	48	46		48	0.0	432.5	48	0.0	66.5	4	10	48	59	✓
inst27-25_15_75	48	0.0	823.8	48	0.0	28.5	5	10	48	50	✓	48	0.0	547.2	48	0.0	43.2	5	13	48	68	✓
inst28-25_15_75	44	0.0	21.1	44	0.0	16.3	5	10	44	40		44	0.0	128.8	44	0.0	38.4	5	13	44	55	✓
inst29-25_15_75	50	0.0	607.4	50	0.0	34.1	5	10	50	50	✓	50	0.0	375.1	50	0.0	69.4	5	13	50	68	✓
inst30-25_15_75	52	0.0	59.3	52	0.0	105.1	5	10	52	48		52	0.0	251.2	52	0.0	151.4	5	13	52	68	✓
Avg	**56.03**	**4.81**	**1651.04**	**55.87**	**1.00**	**588.98**	**5.67**	**11.33**	**55.87**	**60.77**		**56.13**	**7.32**	**1814.70**	**55.87**	**1.05**	**693.17**	**5.67**	**14.47**	**55.87**	**83.50**	

Table 2. Computational results on benchmark instances using different MIP solvers and models.

the dashed brown, and the dotted black ones represent the best runtime obtained using $\overline{k} = 2\hat{k}$, $\overline{k} = |J|$, and $\overline{k} = |J|$ without enforcing valid inequalities (21) to (25), respectively. The red line (corresponding to the results displayed in Table 2) always lies below the brown and black ones, sometimes significantly (e.g. *inst8*, *inst9*, *inst12*, *inst14*), corroborating the efforts to minimize $\overline{k}$ as much as possible while preserving the optimality guarantee. Finally, although the black line represents the runtime of MILP models that are theoretically weaker than all the others, it sometimes falls below the brown one (e.g. *inst5*, *inst15*, *inst30*, *inst26*). This is probably due to the absence of valid inequalities (21) to (25) resulting in different search paths taken by the solvers when the Branch-&-Bound tree is explored.

6. CONCLUSION

In this paper, we introduce a new MILP Mbatch scheduling problem on a single machine, aiming to minimize the makespan while considering the processing times of non-identical jobs and the startup, setup, and shutdown times determined by dominant families. We propose a MILP formulation and present techniques to strengthen it, improving computational efficiency and solution quality when setting a time limit.

Future research may focus on extending the problem to multiple parallel machines and developing advanced approaches to address larger and more complex instances.

REFERENCES

Cheng, B., Yuan, H., Zhou, M., and Qi, T. (2023). Approximation algorithms for scheduling single batch machine with incompatible deteriorating jobs. *RAIRO - Operations Research*, 57(3), 1267 – 1284.

Dang, C. and Kang, L. (2004). Batch-Processing Scheduling with Setup Times. *Journal of Combinatorial Optimization*, 8(2), 137–146.

Gong, H. and Tang, L. (2009). Production-transportation scheduling model on a single batching machine. *2009 Chinese Control and Decision Conference, CCDC 2009*, 2760 – 2764.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P. Hammer, E. Johnson, and B. Korte (eds.), *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, 287–326. Elsevier.

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

Li, L. and Qiao, F. (2008). Aco-based scheduling for a single batch processing machine in semiconductor manufacturing. In *2008 IEEE International Conference on Automation Science and Engineering*, 85–90.

Potts, C.N. and Kovalyov, M.Y. (2000). Scheduling with batching: A review. *European Journal of Operational Research*, 120(2).

Qiu, J., Liu, J., Peng, C., and Chen, Q. (2024). A novel predictive–reactive scheduling method for parallel batch processor lot-sizing and scheduling with sequence-dependent setup time. *Computers & Industrial Engineering*, 189, 109985.

Rezaeian, J. and Zarook, Y. (2018). An efficient bi-objective genetic algorithm for the single batch-processing machine scheduling problem with sequence-dependent family setup time and non-identical job sizes. *Journal of Optimization in Industrial Engineering*, 11(2), 65 – 78.