Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes

Christian Gahm, Stefan Wahl and Axel Tuma

Chair of Business Administration, Production & Supply Chain Management, University of Augsburg, Augsburg, Germany

ABSTRACT
The scheduling of (parallel) serial-batch processing machines is a task arising in many industrial sectors. In the metal-processing industry for instance, cutting operations are necessary to fabricate varying metal pieces out of large base slides. Here, the (cutting) jobs have individual, arbitrary base slide capacity requirements (sizes), individual processing times and due dates, and specific material requirements (i.e. each job belongs to one specific job family, whereby jobs of different families cannot be processed within the same batch and thus are incompatible). In addition, switching of base metal slides and material dependent adjustments of machine parameters cause sequence-dependent setup times. All these conditions need to be considered while minimising total weighted tardiness. For solving the scheduling problem, a mixed-integer program and several tailor-made construction heuristics (enhanced by local search mechanisms) are presented. The experimental results show that problem instances with up to five machines and 60 jobs can be tackled using the optimisation model. The experiments on small and large problem instances (with up to 400 jobs) show that a purposefully used batch capacity limitation improves the solution quality remarkably. Applying the best heuristic to the data of two real-world application cases shows its huge potential to increase delivery reliability.

ARTICLE HISTORY
Received 25 March 2020
Accepted 25 June 2021

KEYWORDS
Serial batching; incompatible job families; sequence-dependent setup times; arbitrary sizes; total weighted tardiness; mixed-integer linear programme

1. Introduction
The serial-batch scheduling problem addressed in this paper originates from the metal-processing industry but can be found in various other sectors like the garment industry or industrial 3D printing. In many metal-processing companies, cutting operations performed by laser or water jet cutting machines are necessary to fabricate a wide variety of metal parts out of large base slides (cf., e.g. Helo, Phuong, and Hao 2019). Hereby, the cutting jobs should be grouped into batches to avoid unnecessary machine setups and/or to increase resource efficiency by avoiding waste. The batch scheduling problem at hand is classified as ‘serial batching’ problem. This means, the jobs grouped within a batch are processed sequentially and the processing time of a batch is equal to the sum of processing times of all the jobs assigned to this batch. In contrast, other batching problems, for example found in the semiconductor industry (Balasubramanian et al. 2004) or food production (Cheng et al. 2014), belong to the group of ‘parallel batching’ problems. Parallel batching means that the jobs grouped in a batch are processed in parallel and the batch processing time is determined by the maximum processing time of all the jobs assigned to this batch. The required machine setups are mandatory to extract completed parts (this type of material availability is called ‘batch availability’), to insert a new base slide, and to configure the machine. Because setup efforts depend on the base slide characteristics (mainly on material type and material thickness), sequence-dependent setup times must be considered. The base slide characteristics also define incompatible job families because jobs with different material type and/or thickness requirements cannot be cut out of the same base metal slide and therefore cannot be grouped in one batch. The metal parts have arbitrary shapes and sizes and the base slides have a limited capacity corresponding to their size. Thus, the jobs that can be grouped within one batch are not only controlled by their family but additionally must be controlled by a capacity restriction.

Since the considered production step of laser cutting is often the starting point of a multi-stage production process or part of a supply chain, delivery reliability is very important. Therefore, we pursue the objective of minimising total weighted tardiness. Additionally, whenever tardiness minimisation is achieved (i.e. tardiness is equal to zero), we pursue the objective of flowtime minimi-
sation to avoid unnecessary setups and reduce work-in-process inventories. The described problem characteristics, the interdependencies between the batching decision (job to batch assignment) and the scheduling decision (batch to machine allocation and sequencing of batches) make the problem challenging and seem to be the reason that the problem, despite its high practical relevance, has not been investigated so far.

Our contributions to literature are manifold:

- A batch scheduling problem of high practical relevance in several industries is described and analysed for the first time.
- A mixed-integer linear program (MILP) addressing batching and scheduling decisions simultaneously is presented. In addition, a group of new priority-rule based construction heuristics and a local search mechanism are proposed.
- Solving capabilities of the standard solver Gurobi and the performance of the proposed heuristics are evaluated by a comprehensive set of problem instances. Several heuristics from literature are used for solution method comparisons.
- General insights on the influence of problem instance characteristics on solving complexity are outlined. Furthermore, some crucial aspects for the development of solution methods are pointed out.

The structure of the paper is as follows. After a more detailed description of two application cases in section 2, we discuss the current state of literature regarding related batching problems in section 3. Next, we introduce the notation and specify the problem at hand (section 4), both of which serve as basis for the presentation of the MILP and the (construction) heuristics in section 5. In section 6, we propose an instance generation scheme and define the experimental design. Result analysis and findings are depicted in section 7. The paper concludes with a brief summary and an outlook on further research topics in section 8.

### 2. Application cases

The first considered application case is from the metal-processing industry. The company operates three identical laser cutting machines to cut out metal parts from base metal slides. The base slides define eleven job families depending on material type and thickness. The batching decision must ensure compliance with two conditions. First, jobs need to belong to the same job family, and second, the slide capacity must not be exceeded. This restriction is approximated by the areas of the minimum bounding rectangle of each job shape (defining the batch capacity requirement) and the area of the base metal slides (defining the maximum batch capacity). An approximation is necessary as the exact evaluation of the capacity restriction would require solving a two-dimensional nesting problem with irregular shapes (as illustrated in Figure 1) which itself is very complex and time consuming (cf., e.g., Mundim et al. 2018). Note that for solving this nesting subproblem within an iterative solution method for the batch scheduling problem, even computation times of a few seconds are troublesome.

A production day at the company comprises 16 h (two shifts) and the complete production programme is defined by a preliminary planning step. In this planning

![Figure 1. Job shapes and the two-dimensional nesting problem.](image-url)
3D-2 (m = 6, q = 4) & 3,375,000 cm$^3$ & 160 cm$^3$ & 609,869 cm$^3$ & 1 min. & 227 min. & 721 & 1,443 \\

To keep the literature review on a manageable size, we limit the analysed contributions to those addressing batch scheduling problems with batch availability, incompatible job families, and limited batch capacities (sizes). A general review on batch scheduling problems can be found in Potts and Kovalyov (2000), where also an extended classification scheme based on the standard $\alpha|\beta|\gamma$ classification scheme of Graham et al. (1979) is presented.

By default, the $\alpha$-field describes the processor (machine) environment with the following abbreviations:

- $S := $ single machine
- $P := $ parallel machines
- $F := $ flow shop
- $J := $ job shop

To indicate that batching decisions are necessary, we follow the approach of Potts and Kovalyov (2000) and use $\alpha \in \{S, P, F, J\}$ for describing processor environments in combination with batching decisions. As usual, for processor environments with a specific number of processors, the number of processors is added to the $\alpha$-field.

To reflect the current state of research concerning batch scheduling problems in detail, we extend the $\beta$-field by additional job characteristics. First, the basic batching type is depicted by $pb$ for parallel batching, $sb$ for serial batching, and $fb$ for fixed batching problems (in the last case, the batch processing time is independent of the jobs in a batch; cf., e.g. Ahmadi et al. 1992). Second, a fixed number of incompatible job families is marked by $if#$ (instead of #, also the number of considered job families can be specified), whereas an arbitrary number of incompatible job families is marked by $if*$. Differences concerning the batch capacity requirements (sizes) of jobs are depicted by $a_j = 1$ for identical capacity requirements for each job (i.e. sizes are equal to one) and $a_j$ for arbitrary capacity requirements (note that $a_j = 1$ is omitted in the following classifications as we assume this to be the standard case). In the context of limited batch capacities, we generally assume that not all jobs can be assigned to a single batch (sometimes depicted by $b < n$ in the case of $a_j = 1$). Regarding processing times, common processing times for all jobs are depicted by $p_j = p$, processing times defined by job families are depicted by $p_j = p_f$ and arbitrary job processing times $p_j$ are taken as standard case and thus, omitted in the following classifications. Note that we generally assume a $pb$-environment for the first two processing time characteristics if not stated otherwise and that the combination of $pb$ and $p_j = p$ is equal to $fb$.

Some authors like Chang, Damodaran, and Melouk (2004) differentiate batch scheduling approaches into problems with constant and with varying batch processing times. We adopt their concept and define varying batch-processing time problems as problems where the processing time of a batch depends on the individual jobs that are grouped in a batch (i.e. $p_j = p$ or $p_f$), whereas in constant batch-processing time problems, the processing time solely depends on the job family (i.e. $p_j = p_f$).
The γ-field denotes the scheduling objective. In this paper, we use the following abbreviations: \((w)C := \) (weighted) completion time, \(C_{\text{max}} := \) makespan, \(C_{b} := \) total batch completion time, \(C_{ih} := \) inventory holding costs, \((w)\bar{F} := \) (weighted) flow time, \(FY := \) feasibility, \(L_{\text{max}} := \) maximum lateness, \((w)T := \) (weighted) tardiness, \(T_{\text{max}} := \) maximum tardiness, \(TEC := \) total energy consumption, \(TP := \) throughput, \(TWL := \) total workload, \((w)U := \) number of (weighted) tardy jobs, \(UB := \) number of batches, and \(U_{\text{mean}} := \) mean resource utilisation.

The following literature review is structured by the characteristics number of incompatible job families and job processing times. To increase traceability of the analysis, we present three tables Tables A1–A3 (cf. appendix A-1) containing all analysed articles and highlight those that are most relevant for the paper at hand. For this purpose, articles that propose relevant optimisation models are marked bold and articles that propose interesting heuristic solution approaches are marked italic. Those articles are also discussed in detail in the following subsections. Note that we do not focus on the development of metaheuristics in this paper and therefore leave them out of the detailed analysis.

### 3.1. Fixed number of incompatible job families

The first group of papers considers a fixed number of incompatible job families (cf. Table A1 in appendix A-1). Within this group, the most interesting paper regarding solution approaches is Hung (1998), proposing a heuristic to tackle the \(fpb, if2, b, sj|T\)-problem (\(sj\) indicates family dependent setup times). The heuristic first allocates jobs to machines and subsequently uses an introduced dynamic programme for the batching and sequencing process. Because the authors do not consider sequence-dependent setup times, their dynamic programme and consequently the heuristic approach is not applicable to the present problem.

### 3.2. Arbitrary number of incompatible job families and common or family dependent processing times

The second group of studies covers an arbitrary number of incompatible job families and common processing times for all jobs or family dependent processing times respectively (cf. Table A2 in appendix A-1).

Uzsoy (1995) develops several algorithms based on propositions for the \(fpb, if\*, b, p_{j} = p_{f}|\sim-\)-problem (the \(\sim\)-symbol in the γ-field denotes that several objective functions are considered independently) and the objectives \(C, wC, C_{\text{max}}\) and \(L_{\text{max}}\). In addition, they present extensions for parallel machines, whereby the algorithm Batch Greedy Earliest Due Date (BGREDD) stands out. Although the objective \(L_{\text{max}}\) is addressed, the adaption could be worthwhile. Therefore, we integrate the heuristic into the set of solution methods to be evaluated in a comparative study. Balasubramanian et al. (2004) present two genetic algorithms (GA1 and GA2) and several construction heuristics for the \(\bar{P}|\bar{p}_{b}, if\*, b, p_{j} = p_{f}|wT\)-problem. Their computational results reveal that assigning jobs to batches before batches are assigned to machines (GA1, decomposition approach 1) yields better results than first assigning jobs to machines and then grouping the jobs to batches (GA2, decomposition approach 2). The authors also claim that their developed priority-rule heuristic ATC-BATC, based on the ATC (apparent tardiness cost) rule suggested by Vepsalainen and Morton (1987), provides reasonably good solutions in a relatively short amount of time even for large-sized problems (recent publications also using the ATC rule or variants are for instance Đurasević and Jakobović 2018; Gahm, Kanet, and Tuma 2019; or Maecker and Shen 2020). Therefore, we apply their ATC-BATC construction heuristic and the proposed enhancement by a ‘swapping’ local search mechanism (ATC-BATC-LS) in our comparative study (note that more details about the heuristics being evaluated in the comparative study are described in section 5.2). As the authors use the priority-rule EDD-EDD for benchmarking, we also integrate this rule in the comparative study. Mönch et al. (2005) extend the parallel machine problem of Balasubramanian et al. (2004) by ready times: \(\bar{P}|\bar{p}_{b}, if\*, b, p_{j} = p_{f}, r_{j}|wT\). They also investigate the two decomposition approaches and confirm the superiority of decomposition approach 1 (i.e. assigning jobs to batches before batches are scheduled on machines). In consequence, our proposed heuristics will also follow this approach. Regarding decomposition approach 1, Mönch et al. (2005) add two features to the ATC-BATC heuristic of Balasubramanian et al. (2004). First, they do not create a single batch per family to be potentially scheduled next but consider all possible batch combinations based on the jobs of the family that are not yet scheduled. To limit the number of combinations (i.e. the power set of available jobs), the parameter \(\text{thres}\) is used. After creating all these batches, the batch combinations are evaluated by a BATC index and the best one is scheduled on the available machine. Second, they present three modifications of the BATC index (BATC-I, BATC-II, and BATC-III), whereby BATC-II (aiming to increase the utilisation of batches) clearly outperforms BATC-I and BATC-III. Building on this, we integrate the construction heuristic ATC-C-BATCU (including both introduced features) in the comparative study. Almeder and Mönch (2011) continue working on the problem...
performing algorithm of Balasubramanian et al. (2004). Besides ATC-BATC as reference heuristic they do not propose any new construction heuristic approaches. Jia, Jiang, and Li (2013) take up the problem of Mönch et al. (2005) and additionally consider re-entrant product flows. They present a real-time closed loop control dispatching heuristic consisting of several sub-routines like GA, push logics, and pull logics. A very similar procedure of the same authors can be found in Jia, Jiang, and Li (2015). They investigate the static version of the problem (with family dependent due dates) and develop a scheduling algorithm based on a rolling horizon control strategy. With both approaches being part of a very specific on-line scheduling framework, they are not useful to develop new construction heuristics or for benchmarking purposes. Pearn, Hong, and Tai (2013) analyze the problem $\tilde{P}|pb, if*, b, p_j = p_f|wT$. They focus on the development of metaheuristics and compare them to the best performing algorithm of Balasubramanian et al. (2004). Besides ATC-BATC as reference heuristic they do not propose any new construction heuristic approaches. Malve and Uzsoy (2007) investigate the $\tilde{P}|pb, if*, b, r_j|Lmax$-problem. They present a family of iterative improvement heuristics (including enhancements of the heuristics proposed by Uzsoy 1995) and combine them with a GA. Because the iterative heuristics are strongly related to release dates and focus on the $Lmax$ objective, they are not appropriate for solving the problem at hand. Closely related to our subject of investigation is the study of Suppiah and Omar (2014): $\tilde{S}|sb, if*, b, s_g|wT$. They develop a MILP for solving small-size problem instances with up to 18 jobs. For large-scale problem instances with up to 300 jobs, they use three different dispatching rule combinations and compare their performance with a hybrid Tabu Search approach. Two of the construction heuristics use a rule for batch selection that contains an additional term to reflect sequence-dependent setup times. This extension has been first proposed by Lee and Pinedo (1997) for the parallel machine scheduling problem and adapted to a batch-scheduling problem (without job families) by Mason, Fowler, and Matthew Carlyle (2002). In contrast to Mason, Fowler, and Matthew Carlyle (2002), Suppiah and Omar (2014) omit the last term aiming to increase the utilisation of batches (cf. also Mönch et al. 2005) in their batch priority index. To investigate this specificity, the two construction heuristics EDD-BATCSs, and ATC-BATCSs are analysed by the comparative study (and also their EDD-EDD rule which is the same as the EDD-EDD heuristic proposed by Balasubramanian et al. 2004). Note that Perez, Fowler, and Carlyle (2005) discuss the same construction heuristics for the $\tilde{S}|pb, if*, b, p_j = p_f|wT$-problem. The MILP formulation proposed by Suppiah and Omar (2014) provides the starting point for our developed optimisation model, details will be discussed later on. Liu et al. (2016) focus on the same problem as Malve and Uzsoy (2007) but with the objectives total weighted tardiness and total energy consumption: $\tilde{P}|pb, if*, b, r_j|wT$, $TEC$. They propose an event-driven on-line scheduling strategy that finds a trade-off between delivery and energy performance. Since their simulation-based approach is not suitable for the development of construction heuristics, we do not apply their strategy.

### 3.3. Arbitrary number of incompatible job families and arbitrary processing times

The third group of papers considers an arbitrary number of incompatible job families and arbitrary processing times (cf. Table A3 in appendix A-1).

Malve and Uzsoy (2007) investigate the $\tilde{P}|pb, if*, b, r_j|Lmax$-problem. They present a family of iterative improvement heuristics (including enhancements of the heuristics proposed by Uzsoy 1995) and combine them with a GA. Because the iterative heuristics are strongly related to release dates and focus on the $Lmax$ objective, they are not appropriate for solving the problem at hand. Closely related to our subject of investigation is the study of Suppiah and Omar (2014): $\tilde{S}|sb, if*, b, s_g|wT$. They develop a MILP for solving small-size problem instances with up to 18 jobs. For large-scale problem instances with up to 300 jobs, they use three different dispatching rule combinations and compare their performance with a hybrid Tabu Search approach. Two of the construction heuristics use a rule for batch selection that contains an additional term to reflect sequence-dependent setup times. This extension has been first proposed by Lee and Pinedo (1997) for the parallel machine scheduling problem and adapted to a batch-scheduling problem (without job families) by Mason, Fowler, and Matthew Carlyle (2002). In contrast to Mason, Fowler, and Matthew Carlyle (2002), Suppiah and Omar (2014) omit the last term aiming to increase the utilisation of batches (cf. also Mönch et al. 2005) in their batch priority index. To investigate this specificity, the two construction heuristics EDD-BATCSs, and ATC-BATCSs are analysed by the comparative study (and also their EDD-EDD rule which is the same as the EDD-EDD heuristic proposed by Balasubramanian et al. 2004). Note that Perez, Fowler, and Carlyle (2005) discuss the same construction heuristics for the $\tilde{S}|pb, if*, b, p_j = p_f|wT$-problem. The MILP formulation proposed by Suppiah and Omar (2014) provides the starting point for our developed optimisation model, details will be discussed later on. Liu et al. (2016) focus on the same problem as Malve and Uzsoy (2007) but with the objectives total weighted tardiness and total energy consumption: $\tilde{P}|pb, if*, b, r_j|wT$, $TEC$. They propose an event-driven on-line scheduling strategy that finds a trade-off between delivery and energy performance. Since their simulation-based approach is not suitable for the development of construction heuristics, we do not apply their strategy.

### 3.4. Summary

Next to the confirmation that the $\tilde{P}|sb, if*, b, a_j, s_g|wT$-problem addressed in this paper has not yet been considered in literature, several findings are obtained: First, serial batching problems are hardly considered in literature and parallel batching problems dominate (this can be attributed to the broadly considered application case of wafer fabrication in the semiconductor industry). Second, processing times are mostly considered to
be depending on the job family. Third, only a few papers consider (sequence-dependent) setup times.

4. Problem description and notation

The serial-batch scheduling problem under investigation is described as follows. A set of \( n \) jobs \( J = \{j|j = 1, \ldots, n \in \mathbb{Z}_+\} \) has to be processed by one of the given identical parallel machines from set \( M = \{M_i|i = 1, \ldots, m \in \mathbb{Z}_+\} \). Generally, sequence-dependent setup times are required between the processing of two jobs. To reduce setup efforts, jobs are grouped in batches given that the total batch capacity requirement of all jobs of a batch is not exceeding a maximum batch capacity \( bc \). Besides individual (arbitrary) batch capacity requirements \( cr_j \) (with \( cr_j \leq bc \)), each job is characterised by its weight \( w_{ij} \), processing time \( p_j \) and due date \( d_j \). Furthermore, each job belongs to a job family \( f \) comprising of \( n_f \) jobs whereby the complete set of incompatible families is \( F = \{f|f = 1, \ldots, q \in \mathbb{Z}_+\} \). These job families are called incompatible since jobs of different families cannot be processed together in one batch. In consequence of the batching, machine setup times are only required after each batch and defined by \( s_{fg} \) for a setup from a batch containing jobs of family \( f \) to a batch containing jobs of family \( g \). Initial setup times (if a batch is the first on a machine) are defined by \( s_{0,f} \). This parameter can also be used to reflect specific initial setup states. Concerning the setup times, we generally assume that the triangular inequality \( s_{fg} + s_{gh} \geq s_{fh} \) holds as otherwise, setup times would be not efficient and no technical reasoning to that is given (cf., e.g. Shen and Buscher 2012). The processing time of a batch is defined by the sum of the processing times of all jobs in the batch (serial batching). Once processing of a batch is started, no jobs can be removed or added to the batch until processing is completed (batch availability). In addition, batch and job processing cannot be interrupted (no preemption) and all jobs are available for processing at the beginning (no release dates).

Because the considered production step is often embedded in a multi-stage production process and/or supply chain, the objective to minimise the total weighted tardiness is used to maximise delivery reliability. Additionally, we aim to minimise the total flow time (to reduce work-in-process inventories) whenever all jobs can be delivered in time (i.e. tardiness is equal to zero).

The problem of minimising total weighted tardiness on a single batch machine with incompatible job families is NP-hard since it is a reduction from the NP-hard single machine total tardiness problem when the batch size, or capacity, is a single job (cf. Lawler 1977 and Brucker et al. 1998). Therefore, also the problem with parallel machines is NP-hard. The NP-hard complexity suggests that any optimisation programme or exact solution method will likely run into computational difficulties if the number of jobs increases. This leads to the question which problem sizes or types of instances are still solvable with such solution methods.

5. Solution methods

To gain first insights on the solvability of the new serial-batch scheduling problem at hand, we propose a MILP and new priority-rule based construction heuristics.

5.1. Mixed-integer linear programme

The mixed-integer linear programme proposed for solving small problem instances generally bases on the optimisation model described in Suppiah and Omar (2014). In contrast to their model, we are not using a predefined number of batches but use a set of constraints to avoid empty batches being not at the end of the schedule. In addition, we reformulate some of the constraints used by Suppiah and Omar (2014).

The basic idea of the developed model is the provisioning of a sufficiently large number of empty batches for each family on each machine so that, if suitable, each job could be placed in a separate batch on any machine: \( B = \{B_b|b = 1, \ldots, n \in \mathbb{Z}_+\} \) (cf., e.g. Dobson and Nambadom 2001). Furthermore, we use the sets \( I_f \) containing all job indices that belong to family \( f \). To model the batching, machine allocation, and sequencing decisions, we use two sets of decision variables. The job to batch assignment decision is modelled by the binary variables \( X_{b,i,f} \):

\[
X_{b,i,f} = \begin{cases} 
1 & \text{if job } i \text{ belonging to family } f \text{ is assigned to batch } b \text{ on machine } i \\
0 & \text{otherwise} 
\end{cases}
\]

\( \forall b \in B; i \in M; j \in J; f \in F \)

The family to batch assignment is modelled by the binary variables \( Y_{b,i,f} \):

\[
Y_{b,i,f} = \begin{cases} 
1 & \text{if batch } b \text{ on machine } i \text{ contains jobs of family } f \\
0 & \text{otherwise} 
\end{cases}
\]

\( \forall b \in B; i \in M; f \in F \)

In addition, we use the following auxiliary variables:

- \( P_{b,i,f} \in \mathbb{Z}_+ := \text{processing time of batch } b \text{ on machine } i \text{ containing jobs of family } f \)
The objective function defined by (1) consists of two parts: The first part represents the dominant objective of minimising total weighted tardiness. The second part represents the subordinated objective of minimising total flow time.

\[
\text{Minimize } \left( \sum_{j \in f} w_j \cdot T_j \right) + \frac{TF}{(C_{\max} \cdot n \cdot 10)} \quad (1)
\]

The ordering of the objectives (due to the decision maker’s preferences) is achieved by defining the (constant) denominator to be greater than the nominator (i.e., the total flow time) in the second part. To assure that the denominator is greater than the TF, the upper bound of the maximum flow time of one job is estimated by the approximated makespan \( \hat{C}_{\max} \) (cf. (21) in section 6.1) and multiplied by the number of jobs. Because the makespan is approximated, we use a ‘safety’ factor of 10. In consequence of the second part always being smaller than one, it only becomes relevant if the first part is equal to zero, i.e., a flow time improvement is never achieved at the expense of increased weighted tardiness.

Constraints:

\[
\sum_{b \in B} \sum_{i \in M} X_{b \cdot i \cdot f} = 1 \quad \forall f \in F; \ j \in I_f \quad (2)
\]

\[
\sum_{f \in F} \sum_{j \in I_f} X_{b \cdot j \cdot f} \cdot cr_j \leq bc \quad \forall b \in B; \ i \in M \quad (3)
\]

\[
\sum_{f \in F} \sum_{j \in I_f} X_{b \cdot 1 \cdot j \cdot f} \geq \sum_{f \in F} \sum_{j \in I_f} X_{b \cdot j \cdot f} \quad \forall b \in B \land b > 1; \ i \in M \quad (4)
\]

\[
X_{b \cdot i \cdot f} \leq Y_{b \cdot i \cdot f} \quad \forall b \in B; \ i \in M; \ f \in F; \ j \in I_f \quad (5)
\]

\[
\sum_{f \in F} Y_{b \cdot i \cdot f} \leq 1 \quad \forall b \in B; \ i \in M \quad (6)
\]

\[
P_{b \cdot i \cdot f} \geq \sum_{j \in I_f} X_{b \cdot i \cdot j \cdot f} \cdot p_j \quad \forall b \in B; \ i \in M; \ f \in F \quad (7)
\]

\[
C_{i \cdot f} \geq s_{i \cdot f} + P_{i \cdot i \cdot f} - \text{bigM} \cdot (1 - Y_{1 \cdot i \cdot f}) \quad \forall i \in M; \ f \in F \quad (8)
\]

\[
C_{b \cdot i \cdot f} \geq C_{b \cdot 1 \cdot i \cdot f} + [s_{j \cdot f} - \text{bigM} \cdot (1 - Y_{b \cdot 1 \cdot i \cdot f})] + [P_{b \cdot i \cdot f'} - \text{bigM} \cdot (1 - Y_{b \cdot i \cdot f'})] \quad \forall b \in B \land b > 1; \ i \in M; \ f, f' \in F \quad (9)
\]

\[
C_{j \cdot i \cdot f} \geq C_{b \cdot i \cdot f} - \text{bigM} \cdot (1 - X_{b \cdot i \cdot f}) \quad \forall b \in B; \ i \in M; \ f \in F; \ j \in I_f \quad (10)
\]

\[
T_{j \cdot i \cdot f} \geq (C_j - d_j) \quad \forall j \in J \quad (11)
\]

\[
TF = \sum_{j \in I_f} C_{j \cdot i \cdot f} \quad (12)
\]

Constraints set (2) ensures that every job has to be assigned to exactly one batch and set (3) guarantees that the total capacity requirement of all jobs assigned to the same batch does not exceed the maximum batch capacity. The constraints defined by (4) allow only batches at the end of the schedule to be empty. The family of the batches is determined by (5) and (6) assures that only one family is assigned to a batch (i.e., that only jobs of the same family are assigned to a batch). Equations (7) determine the processing time per batch. Constraints (8) define the completion time of the first batch on each machine and constraints (9) the completion times of all following batches. The completion time per job is defined by (10) and its tardiness by (11). Constraint set (12) determines the total flow time. Within all corresponding constraints, the following bigM-value is appropriate: \( \text{bigM} = \hat{C}_{\max} \cdot n. \)

### 5.2. Multi-start construction heuristics

The developed multi-start construction heuristics base on the general priority rule dispatching procedure as for instance used by the ATC-BATC heuristic proposed by Balasubramanian et al. (2004). This procedure iteratively adds new batches to the schedule whenever one of the parallel machines comes free. Thereby, at each decision point, one or more batches for each job family are created and the batch with the ‘highest’ priority will be scheduled as early as possible. Batches per family are created based on priority indices of the family’s unscheduled jobs. The prioritised jobs are assigned to a batch in a specific priority-index based sequence as long as the batch capacity is not exceeded. Since all construction heuristics being evaluated in the comparative study follow this procedure (despite BGRED proposed by Uzsoy 1995), we can illustrate the main course of action by a generic pseudo-code (parameters etc. are described in the following paragraphs):

The corresponding priority-rule specific parameters are listed in Table 2 and the corresponding specifications of the job priority-rules and the batch priority-rules are
listed in Table 3 and Table 4, respectively. Whenever one of the priority-rules contains elements that are not relevant for the problem at hand (e.g. release dates or batch utilisation factors based on the number of jobs) we omit or adapt them. Besides the basic setup of the heuristics, Table 2 also lists the additional parameters used by the heuristics.

The two parameters $\kappa_1$ and $\kappa_2$ are so called look ahead parameters and the parameter $\text{thres}$ controls the number of jobs that are used to create and evaluate batch combinations of one family. The parameter $\beta$ is a newly introduced parameter that controls the batch utilisation. Note that ATCS-BATC(\beta) and ATCS-C-BATCS(\beta) are the newly developed construction heuristics, the others are derived from literature (cf. section 3). The ‘-C-’ in ATC-C-BATCU and ATCS-C-BATCS(\beta) signals that batch combinations are used.

\textbf{Table 2. Construction heuristics with parameters.}

| JobPriorityRule | JobSortingRule | BatchPriorityRule | BatchSelectionRule | $\kappa_1$ | $\kappa_2$ | Uses batch combinations (thread) |
|-----------------|----------------|-------------------|-------------------|----------|----------|-------------------------------|
| EDD-EDD         | $\pi_j = d_j$ | non-decreasing    | min. due date     | min      | -        | -                            |
| EDD-BATCs       | $\pi_j = d_j$ | non-decreasing    | (15) max          | x        | x        | -                            |
| ATC-BATC        | (13) non-increasing |                | max              | x        | -        | -                            |
| ATC-BATCs       | (13) non-increasing |                | max              | x        | -        | -                            |
| ATCS-BATC(\beta)| (14) non-increasing |                | max              | x        | x        | x                            |
| ATCS-C-BATCS(\beta)| (14) non-increasing |            | max              | x        | x        | x                            |

\textbf{Table 3. Job priority rules with decision point t and a preceding batch with jobs of family f.}

| ATC             | $\pi_j(t) = \frac{\sum_{j \in T} p_{j,t}}{\max(p_{j,t} - c_j, 0)}$ |
|-----------------|---------------------------------------------------------------------|
| ATCS            | $\pi_j(t, f) = \frac{\sum_{j \in T} p_{j,t}}{\max(p_{j,t} - c_j, 0)}$ |

Details about the ATC and the ATCS rule, their rationality and idea, are for instance explained in Vepsalainen and Morton (1987) and Lee and Pinedo (1997), respectively.
Table 4. Batch priority rules with decision point \( t \) and a preceding batch with jobs of family \( f \).

| BATCSs | \( \Pi_b(t, f) = \frac{w_f}{\bar{p}_b} \cdot \exp \left( -\frac{\max(d_j - pt_b - t, 0)}{\kappa_1 \bar{p}_b} \right) \) \cdot \exp \left( -\frac{d_f}{\kappa_2 \bar{p}_b} \right) \) with \( \bar{w}_b = \sum_{j \in B_b} w_j \). | \( \Pi_b(t) = \sum_{j \in B_b} \pi_j(t) \) | \( \Pi_b(t, f) = \sum_{\gamma \neq j \in B_b} \pi_j(t, f) \) |
|---|---|---|---|
| BATC | \( \Pi_b(t) = \sum_{j \in B_b} \pi_j(t) \) \( \Pi_b(t, f) = \sum_{j \neq \gamma \in B_b} \pi_j(t, f) \) | \( \Pi_b(t) = \sum_{j \in B_b} \pi_j(t) \) \( \Pi_b(t, f) = \sum_{j \neq \gamma \in B_b} \pi_j(t, f) \) | \( \Pi_b(t, f) = \sum_{\gamma \neq j \in B_b} \pi_j(t, f) \) |

Regarding the batch priority rules depicted in Table 4, we would like to emphasise that the first rule (15) uses mean values based on the jobs assigned to a batch, whereas the other rules sum up individual job values ((17)) or job priorities ((16) and (18)). In this way, the two latter rules transfer individual job ‘apparent tardiness costs’ to a corresponding batch ‘apparent tardiness costs’ value. Thus, each batch has its own ‘apparent tardiness costs’ and the scheduling decision is then based on prioritised batches (like prioritised jobs).

The filling of batches for maximising batch capacity utilisation seems to be appropriate for constant batch-processing time problems but not for the varying batch-processing time problem at hand. Because the processing time of a batch depends on all the jobs that are grouped in a batch, batches with a lower capacity utilisation lead to shorter batch processing times in most cases. This in turn could be more suitable with regard to the objective of minimising total weighted tardiness. Therefore, we introduce the parameter \( \beta \in \mathbb{R} \) (with \( 0 < \beta \leq 1 \)) to control the batch utilisation during the batching decision. Because we cannot assume any longer that each job can be assigned to a batch \( (c_r \leq bc) \), we have to guarantee that each job is assignable regarding its size and use a ‘actualized’ maximum batch capacity \( bc^{ACT} \) for controlling the batch utilisation:

\[
bc^{ACT} = \max \{ \beta \cdot bc, \max \{ c_r \mid \forall j \in J \} \}.
\]  

As it is known that ATC related priority rules lead to good results regarding total weighted tardiness when look ahead parameters are set appropriately (cf., e.g. Balasubramanian et al. 2004 or Almeder and Mönch 2011), we use a multi-start heuristic approach that inherently performs a grid search for all specified combinations of \( \kappa_1, \kappa_2, \text{thres} \), and \( \beta \). Furthermore, the instance related calculation procedures for \( \kappa_1 \) and \( \kappa_2 \) as proposed by Lee and Pinedo (1997) are considered (cf. Appendix A-2). The following parameters are used by the grid search: \( \kappa_1 \in \{ 0.5, 1.0, \ldots, 5.0 \} \) (cf., e.g. Balasubramanian et al. 2004 or Almeder and Mönch 2011), \( \kappa_2 \in \{ 0.1, 0.2, \ldots, 1.6 \} \) (cf., e.g. Lee and Pinedo 1997 or Park, Kim, and Lee 2000), \( \text{thres} \in \{ 10, 15 \} \) (cf. Mönch et al. 2005), and \( \beta \in \{ 0.50, 0.55, \ldots, 1.00 \} \). Concerning the \( \beta \)-parameter, we are not using smaller values than 0.50 because the actual batch capacity \( bc^{ACT} \) also depends on the maximum capacity requirement of all jobs and therefore, smaller \( \beta \)-values may not have any impact (cf. (19)). Of course, the repeated execution of the heuristics within the multi-start environment increases computation times but we assume that computational efforts are reasonable due to solution quality improvements. This will be analysed later on.

5.3. Local search

To improve the solutions obtained by the ATC-BATC, Balasubramanian et al. (2004) propose a corrective technique called ‘swapping’. This local search mechanism keeps the assignment of batches to machines and the sequence of the batches and tries to interchange jobs between batches of the same family (across all machines). Our proposed local search mechanism works in a same way. Starting with the batch having the earliest start time, we iterate through all jobs of this batch and evaluate swappings of this job with jobs in batches with a later start time and the same family. Whenever a swap is feasible (regarding batch capacity) and leads to an improvement (i.e. the objective value decreases), jobs are swapped and the search is restarted with the batch having the earliest start time. Because the artificial batch capacity limitation is no longer suitable during the local search, we do not use the parameter \( \beta \) and \( bc^{ACT} \) but the original batch capacity \( bc \). If no improvements can be achieved, the batch with the next earliest start time is examined. To avoid exceeding computation times, a swap is discarded if the relative improvement is smaller than \( 1.0 \times 10^{-4} \). This value is used to fully explore the local search’s potential to improve the tardiness objective and to explore the potential of improving the flow time objective to a certain extent. The search terminates when all jobs of all batches have been examined.

Because Balasubramanian et al. (2004) consider a batching problem where the capacity restriction bases
on the number of jobs in a batch, the capacity restriction is not a restriction for job swapping in their case. As we consider arbitrary batch capacity requirements, the capacity restriction will likely prohibit many swaps and therefore, solution improvements might be not that large. The approach of Balasubramanian et al. (2004) is named ATC-BATC-LS and our developed approaches are named ATCS-BATCS(β)-LS and ATCS-C-BATCS(β)-LS in the remainder of this paper.

6. Experimental design

To investigate the capabilities of standard solvers and the MILP, to evaluate the performance of the proposed heuristics compared to heuristics from literature, and to gain general insights on the influence of problem instance characteristics on solving complexity, a comprehensive set of problem instances is needed.

6.1. Instance generation scheme

For generating problem instances with specific characteristics, we basically follow the procedures used by Mehta and Uzsoy (1998), Dobson and Nambimadom (2001), and Perez, Fowler, and Carlyle (2005). Basic parameters of the instances are the number of parallel machines \( m \) \((m = 1 \) represents the single machine case), the total number of jobs \( n \), and the number of incompatible job families \( q \). As emphasised by Pearn, Hong, and Tai (2013) and Shen, Gupta, and Buscher (2014), the assignment of jobs to families has a great impact on the total incurred setup time in a schedule and thus, on the total weighted tardiness. To assess this effect, we use two job to family assignment modes: The distribution of jobs to families is either determined by a discrete uniform distribution (UD) or a normal distribution (with a discretization function; ND). By using the ND assignment, many of the \( q \) or a normal distribution (with a discretization factor \( \eta \)).

Next, the mean processing time per batch is estimated by \( \bar{p}_b = \bar{n}_b \cdot \bar{p} \) (with \( \bar{p} = \sum_{j \in J} p_j/n \)). Together with the setup time severity factor \( \eta \) (also used by Suppiah and Omar 2014), the mean setup time is determined by \( \bar{s} = \eta \cdot \bar{p}_b \) and setup times are drawn from a discrete uniform distribution restricted by \([1, 2\bar{r}]\). To respect the triangular inequality for the setup times, the randomly generated setup times are sorted in non-decreasing order and the symmetric family setup time matrix is filled row by row and column by column. Setup times between the same family \( f \) \((sf_f)\) are the first \( q \) smallest generated setup times. The initial setup time \( s_{0,f} \) of a family \( f \) is equal to the setup time between batches of the same family \( sf_f \). For instance, for five families, the randomly determined setup time sequence \([29, 44, 65, 70, 75, 83, 89, 90, 103, 117, 126, 132, 141, 147, 148]\) leads to the following setup time matrices:

\[
f = \begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 \\
0 & 29 & 44 & 65 & 70 & 75
\end{pmatrix}
\]

\[
f = \begin{pmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 29 & 83 & 89 & 90 & 103 \\
2 & 83 & 44 & 117 & 126 & 132 \\
3 & 89 & 117 & 65 & 141 & 147 \\
4 & 90 & 126 & 141 & 70 & 148 \\
5 & 103 & 132 & 147 & 148 & 75
\end{pmatrix}
\]

An alternative approach to generate setup times respecting the triangular inequality is described in Rocha et al. (2008).

To obtain reasonable due dates, we first estimate the makespan \( \bar{C}_{\text{max}} \) depending on the parameters calculated so far: The approximated total number of setups \( tns = n/\bar{n}_b \) and the mean setup time \( \bar{s} \) (after generating the actual setup times) are used to approximate the total setup time: \( \bar{t}_{\text{st}} = tns \cdot \bar{s} \). Finally, the makespan is approximated by

\[
\bar{C}_{\text{max}} = (n \cdot \bar{p} + \bar{t}_{\text{st}})/m \tag{20}
\]

Note that this makespan approximation bases on the idea of the makespan’s lower bound calculation assuming job preemption (cf., e.g. Malve and Uzsoy 2007).

According to Baker (2013), most studies addressing tardiness related problems use the two parameters tardiness factor \( tf \) (also called tightness factor) and due date range factor \( rdd \) for guiding instance generation. Hereby,
the due date tightness factor $tf$ controls the approximated percentage of tardy jobs (a higher $tf$ value means stricter due dates) and the parameter $rdd$ controls the range of due dates (a higher value of $rdd$ means a larger variance). We also use these parameters and follow the approach of Balasubramanian et al. (2004) to randomly generate due dates by a discrete uniform distribution restricted by $[\text{max}(0, \lambda - (\lambda \cdot rdd/2)), \lambda + (\lambda \cdot rdd/2)]$ with $\lambda = \frac{C_{\text{max}}}{(1 - tf)}$. Hereby, we slightly adapted the lower bound of the interval to avoid zero due dates.

### 6.2. Experimental setup

To evaluate the capabilities of standard solvers and the MILP as well as the performance of the heuristics, we use two sets of problem instances. The first set (S1) contains small instances with up to 60 jobs and five machines, the second set (S2) contains large instances with up to 400 jobs and ten machines. Because not all instance parameter combinations are reasonable with regard to the capacity requirement scenarios, we restrict our experiments to those combinations with a sufficiently large approximated number of batches per machine. The resulting parameter combinations for S1 are depicted in Table 5.

These 31 basic instance parameter combinations (defining the number of jobs $n$, the number of parallel machines $m$, and the capacity requirement scenarios CRS1 to CRS4) are combined with the parameters listed in Table 6 to form the complete set of instances.

For each parameter combination we generate five instances. For set S1, this leads to 2,880 instances with $m = 1$, 2,160 instances with $m = 3$, 1,200 instances with $m = 4$ and 1,200 instances with $m = 5$ (7,440 problem instances in total).

For problem instance set S2, we generate instances with $n \in \{100, 200, 400\}$, $m \in \{1, 3, 5, 10\}$, and $q \in \{3, 5, 10, 20\}$. All other parameters are like for S1 and we have no limitations regarding appropriate parameter combinations. Accordingly, we have 15,360 instances in set S2. All generated instances are available for download at Mendeley Data (with best-known objective values of the applied heuristics; see Gahm 2021).

All heuristics and a tool for the management of the experiments are implemented in Java 10 and GAMS 24.4.5 was used to implement the MILP and to apply the standard solver Gurobi 8.11 and CPLEX 12.8. The experiments have been executed on workstations with an Intel® XEON® CPU E5-2690 with 3.0 GHz and 64 GB RAM.

### 7. Experimental results

In the first part of our analysis, we investigate the capabilities of two standard solver and the MILP. In the second part, we compare the performance of the new construction heuristics with the heuristics from literature and the results obtained with the MILP. In addition, with analyse the performance with regard to instances from the two application cases. Finally, several general insights on the structure of ‘good’ solutions are presented.

#### 7.1. Solving capabilities of standard solvers

With a preliminary study based on 24 additionally generated instances showing a slightly better performance of Gurobi compared to CPLEX, we use Gurobi in the following experiments. Details about the preliminary study are given in Appendix A-3.

To assess the solving capabilities of Gurobi with the proposed MILP, we depict the number of problem instances solved with a specific relative MIP gap in Table 7 (with a maximum computation time of 3,600 s).

The results in Table 7 show that the solvability of problem instances strongly depends on the number of machines and the number of jobs. Larger instances lead to computational difficulties which makes the development and use of heuristics necessary.

Regarding capacity requirement scenarios, we can report that instances with a higher mean batch capacity requirement per job (CRS3 and CRS4) are easier to solve compared to those with a lower mean capacity requirement per job (CRS1 and CRS2). The reason for this is that higher batch capacity requirements lead to fewer

### Table 5. Reasonable instance parameter combinations.

| Parameter | Values |
|-----------|--------|
| $m$       | 1, 3, 5, 10 |
| $n$       | 100, 200, 400 |
| $q$       | 3, 5, 10 |
| $rdd$     | 0.5, 2.5 |
| $tf$      | 0.25, 0.75 |

### Table 6. Further instance parameters.

| Parameter                   | Values |
|-----------------------------|--------|
| Number of families ($q$)    | 3, 5, 10 |
| Job to family assignment mode ($jtfam$) | UD, ND |
| Percentage of tardy jobs ($tf$) | 0.3, 0.6 |
| Due date range ($rdd$)      | 0.5, 2.5 |
| Setup time severity ($s$)   | 0.25, 0.75 |
possibilities regarding the batching decision and thus to a smaller solution space. Concerning the job to family assignment mode (jtfam) and due date range (rrd), no peculiarities can be observed. The effects of the parameters number of job families (q) and setup time severity (η) are as expected: i.e. a higher number of families often leads to less batches per family (facilitating the batching decision) and higher setup times (η = 0.75) often lead to less batches (consequently less batches have to be scheduled). In contrast, the percentage of tardy jobs (tf) remarkably influences solving capabilities. This can be explained on the grounds that a lower percentage (tf = 0.3) offers a higher possibility to achieve a tardiness of zero and to provide better lower bounds.

### 7.2. Heuristic performance

To assess the solution quality of the discussed heuristics, we use the key figure ’mean relative improvement versus the worst objective function value (MRIW)’ (cf. Valente and Schaller 2012 or Gahm, Kenet, and Tuma 2019). The reason for this is that objective function values could be close to zero (cf. (1)) and thus, relative improvements related to the best objective function value are troublesome. The MRIW used here is defined as follows: For a given problem instance, the objective function value achieved by a specific solution method s is depicted by $v_s$ and $v_{WORST}$ ($v_{BEST}$) represents the worst (best) objective function value of all considered solution methods. Then, the MRIW of solution method s is calculated by $RIW_s = (v_{WORST} - v_s) / v_{WORST} \times 100$. Based on this definition of $RIW_s$, mean values (MRIWs) for each solution method with regard to all or a subset of problem instances can be reported. Additionally, we report the number of times a solution method s achieves the best objective function

| rel. MIP gap ∈ | [0, 0.05] | [0.05, 0.25] | [0.25, 0.5] | [0.5, 0.75] | Total number of instances |
|---------------|-----------|-------------|-------------|-------------|--------------------------|
| $m = 1$       | 253       | 14          | 1           | 4           | 2,880                    |
| $m = 3$       | 151       | 2           | 1           | 2           | 2,160                    |
| $m = 4$       | 54        | 0           | 0           | 0           | 1,200                    |
| $m = 5$       | 49        | 0           | 0           | 0           | 1,200                    |
| $n = 15$      | 203       | 11          | 2           | 5           | 1,440                    |
| $n = 30$      | 265       | 5           | 0           | 1           | 2,640                    |
| $n = 60$      | 39        | 0           | 0           | 0           | 3,360                    |
| CRS = 1       | 2         | 1           | 1           | 3           | 960                      |
| CRS = 2       | 29        | 14          | 0           | 2           | 1,680                    |
| CRS = 3       | 242       | 0           | 0           | 2           | 2,400                    |
| CRS = 4       | 234       | 1           | 1           | 1           | 2,400                    |
| $q = 3$       | 111       | 6           | 1           | 3           | 2,480                    |
| $q = 5$       | 183       | 4           | 0           | 1           | 2,480                    |
| $q = 10$      | 213       | 6           | 1           | 2           | 2,480                    |
| jtfam = DND   | 158       | 8           | 1           | 4           | 3,720                    |
| jtfam = DUD   | 249       | 8           | 1           | 2           | 3,720                    |
| $\eta = 0.25$| 128       | 10          | 0           | 1           | 3,720                    |
| $\eta = 0.75$| 379       | 6           | 2           | 5           | 3,720                    |
| $tf = 0.3$    | 502       | 16          | 2           | 4           | 3,720                    |
| $tf = 0.6$    | 502       | 0           | 0           | 2           | 3,720                    |
| $rrd = 0.5$   | 295       | 0           | 1           | 2           | 3,720                    |
| $rrd = 2.5$   | 212       | 16          | 1           | 4           | 3,720                    |

Table 8. MRIW and BEST regarding instance set S1 (small instances).

| m   | Number of instances | Gurobi | BDG-BDD | EDD-BDD | EDDB-ATCS | ATC-BATC | ATC-BATC+S | ATC-C-BATC | ATC-C-BATC/β | ATCS-C-BATC+S | ATC-BATC-LS | ATCS/C-BATC-LS | ATCS/C-BATC-LS |
|-----|---------------------|--------|---------|---------|-----------|----------|------------|------------|--------------|---------------|-------------|----------------|----------------|
| 1   | 15                  | 960    | 61.8    | 12.8    | 17.0      | 43.5     | 43.8       | 47.8       | 34.6         | 58.6          | 57.4        | 44.8           | 59.1           |
| 2   | 960                 | 557    | 44.9    | 4.6     | 44.9      | 58.7     | 61.4       | 72.4       | 69.6         | 60.6          | 63.4        | 60.6           | 73.6           |
| 3   | 960                 | 43.2   | 22.7    | 11.1    | 56.0      | 58.7     | 61.4       | 72.4       | 69.6         | 60.6          | 63.4        | 60.6           | 73.6           |
| 4   | 960                 | 163    | 16.1    | 137     | 8         | 78.3     | 72.4       | 69.6       | 60.6          | 63.4          | 60.6        | 60.6           | 73.6           |
| 5   | 960                 | 163    | 16.1    | 137     | 8         | 78.3     | 72.4       | 69.6       | 60.6          | 63.4          | 60.6        | 60.6           | 73.6           |
| Mean MRIW | 53.4   | 28.5    | 35.4    | 55.9    | 60.6      | 62.4     | 33.0       | 73.5       | 70.9         | 62.9          | 75.2        | 74.5           | 73.6           |
Table 9. Mean CT regarding instance set S1 (small instances).

| m  | n  | Number of instances | Gurobi | BGRED | EDD-EDD | EDD-BATCSs | ATC-BATC | ATC-BATCS | ATC-C-BATCU | ATCS-BATCS(β) | ATCS-C-BATCS(β) | ATC-BATC-LS | ATCS-BATCS(β)-LS | ATCS-C-BATCS(β)-LS |
|----|----|---------------------|--------|-------|---------|------------|----------|---------|------------|----------------|-----------------|--------------|-----------------|-----------------|
| 1  | 15 | 960                 | 3,571.2| 0.8   | 1.0     | 74.3       | 5.6      | 78.5    | 0.0        | 0.9            | 2.6             | 11.7         | 1.6             | 3.0             |
| 30 | 960| 3,591.2             | 1.4    | 1.9   | 154.5   | 11.7      | 319.4    | 0.5     | 3.7       | 62.7           | 29.4            | 4.1           | 3.7             | 73.4            |
| 60 | 960| 3,601.4             | 3.5    | 4.3   | 356.7   | 27.1      | 638.5    | 5.9     | 8.1       | 816.4          | 85.3            | 8.3           | 930.7           |                 |
| 1  | 30 | 480                 | 3,523.4| 0.8   | 1.0     | 80.3      | 6.2      | 83.5    | 0.0        | 2.0            | 5.3             | 14.9         | 2.1             | 6.1             |
| 30 | 720| 3,601.2             | 1.6    | 2.0   | 166.0   | 12.6      | 359.4    | 0.5     | 4.3       | 70.6           | 30.3            | 4.3           | 80.4            |                 |
| 60 | 960| 3,602.2             | 3.6    | 4.4   | 360.0   | 27.3      | 645.3    | 5.9     | 8.2       | 840.9          | 88.8            | 8.4           | 975.5           |                 |
| 4  | 30 | 480                 | 2.4    | 3.1   | 271.4   | 20.0      | 274.2    | 0.5     | 3.2       | 68.6           | 31.5            | 4.6           | 78.2            |                 |
| 60 | 720| 3,600.8             | 5.0    | 7.8   | 683.1   | 50.4      | 693.8    | 6.2     | 8.8       | 910.4          | 89.5            | 9.2           | 1,047.0         |                 |
| 5  | 30 | 480                 | 3.5    | 5.7   | 273.5   | 20.6      | 283.0    | 0.5     | 3.3       | 65.7           | 32.4            | 4.7           | 76.2            |                 |
| 60 | 720| 3,593.5             | 2.5    | 3.1   | 273.8   | 20.8      | 283.0    | 0.5     | 3.2       | 65.7           | 32.4            | 4.7           | 76.2            |                 |

Table 10. Absolute MRIW differences of instances solved to ‘optimum’.

| m  | n  | Num. Instances (< rel. MIP gap < 0.05): Total |
|----|----|---------------------------------------------|
| 1  | 15 | 126                                         |
| 30 | 92 | 45.3                                        |
| 60 | 35 | 55.3                                        |
| 3  | 15 | 77                                          |
| 30 | 72 | 58.7                                        |
| 60 | 2  | 100.0                                       |
| 4  | 30 | 52                                          |
| 5  | 30 | 49                                          |

MeanMRIW 56.5 39.7 19.3 14.7 18.0 37.7 4.4 4.2 12.3 3.6 2.8 6.3 1.5 2.3 10.8 4.1 1.4 3.1

value (BEST; i.e. \( v = v_{BEST} \)) and the computation time (CT).

Table 8 presents the key figures MRIW (upper value) and BEST (lower value, italic) for instance set S1 with regard to the instance characteristics number of machines \( (m) \) and number of jobs \( (n) \). Best (highest) values per row are marked bold. Table 9 depicts the corresponding computation times.

Table 8 shows that the new heuristics explicitly developed for the problem at hand outperform the heuristics from literature in terms of solution quality. Furthermore, we can report that the additional computational efforts (cf. Table 9) required by the consideration of batch combinations (ATCS-C-BATCS(\( \beta \)) and ATCS-C-BATCS(\( \beta \)-LS) have no positive impact on the solution quality. Instead, solution quality slightly decreases. This can be explained by the applied job priority and batch priority rules (cf. (14) and (18)): If batch combinations are considered, batches containing many jobs that are not that ‘urgent’ could have a higher priority compared to other batches with few jobs that are ‘urgent’. Although this could lead to a better local decision, it can negatively influence the global result.

The small additional computational effort (cf. Table 9) of the local search is justified by the improved solution quality. Albeit every swapping of jobs by the local search requires an evaluation of the complete schedule (due to serial-batching), the additional computational effort is low due to the individual batch capacity requirement (size) of the jobs limiting the number of feasible swaps. This is not the case when the batch capacity is restricted by the number of jobs as with many other problems.

To assess the absolute performance of the heuristics, we calculate the absolute MRIW differences between the MRIW achieved by Gurobi and the MRIW achieved by the other solution methods: MRIW\(_{Gurobi} - \) MRIW. Table 10 reports the results for those problem instances Gurobi achieved a relative MIP gap lower than 0.05 (as we assume that these instances are solved to optimum). Best (lowest) values per row are marked bold.

The absolute comparison in Table 10 underlines the superiority of the newly proposed heuristics.
### Table 11. MRIW and BEST regarding instance set S2 (large instances).

| m | n   | ATC-BATCS | ATC-BATC | ATC-BATC-LS | ATCS-BATCS(β) | ATCS-BATCS(β)-LS |
|---|-----|-----------|----------|-------------|---------------|-----------------|
| 1 | 100 | 28.0      | 15.4     | 24.3        | 55.4          | 61.7            |
| 100 | 81 | 28.0      | 15.4     | 24.3        | 55.4          | 61.7            |
| 200 | 30.8 | 19.1     | 31.8     | 59.0        | 70.0          | 76.7            |
| 400 | 31.1 | 20.4     | 36.2     | 59.4        | 70.0          | 76.7            |
| 6    | 327 | 8        | 26       | 58.9        | 69.1          |                 |
| 200 | 172 | 84       | 19.1     | 31.8        | 55.4          | 61.5            |
| 400 | 178 | 17.0     | 33.6     | 59.0        | 72.4          |                 |
| 100 | 200 | 30.8     | 20       | 24          | 66.9          |                 |
| 5    | 400 | 28.2     | 19.5     | 34.9        | 64.0          |                 |
| 100 | 21   | 34.8     | 13       | 71          | 69.1          |                 |
| 400 | 18   | 19.5     | 11       | 13          | 64.0          |                 |
| 5    | 200 | 28.2     | 19.5     | 34.9        | 64.0          |                 |
| 100 | 170 | 20.2     | 13.3     | 23.4        | 60.9          |                 |
| 3    | 200 | 24.1     | 16.6     | 29.0        | 59.7          |                 |
| 400 | 34.8 | 0.3      | 13       | 71          | 69.1          |                 |
| 5    | 18   | 13.5     | 9        | 13          | 64.0          |                 |
| 100 | 400 | 28.2     | 19.5     | 34.9        | 64.0          |                 |
| 100 | 200 | 24.1     | 16.6     | 29.0        | 59.7          |                 |
| 400 | 18   | 13.5     | 9        | 13          | 64.0          |                 |
| 5    | 200 | 24.1     | 16.6     | 29.0        | 59.7          |                 |
| 100 | 170 | 20.2     | 13.3     | 23.4        | 60.9          |                 |
| 10    | 100 | 1.8      | 0.1      | 0.3         | 23.1          |                 |
| 100 | 1.8 | 0.1      | 0.3      | 23.1        | 67.9          |                 |
| Mean MRIW |          | 22.4     | 16.3     | 29.8        | 58.5          | 67.2            |

### Table 12. Mean CT [s] regarding instance set S2 (large instances).

| m | n   | ATC-BATCS | ATC-BATC | ATC-BATC-LS | ATCS-BATCS(β) | ATCS-BATCS(β)-LS |
|---|-----|-----------|----------|-------------|---------------|-----------------|
| 1 | 100 | 2.1       | 0.1      | 0.3         | 26.2          | 26.6            |
| 200 | 4.6 | 0.3       | 4.5      | 64.0        | 73.8          |                 |
| 400 | 13.2 | 0.9      | 91.4     | 198.3       | 366.3         |                 |
| 3 | 100 | 2.1       | 0.2      | 0.3         | 27.1          | 27.4            |
| 200 | 4.8 | 0.3       | 3.4      | 66.4        | 74.2          |                 |
| 400 | 13.5 | 1.0      | 56.9     | 203.5       | 344.5         |                 |
| 5 | 100 | 2.1       | 0.2      | 0.3         | 26.4          | 26.8            |
| 200 | 4.6 | 0.3       | 3.4      | 64.8        | 72.2          |                 |
| 400 | 13.4 | 0.9      | 59.6     | 202.1       | 325.7         |                 |
| 10 | 100 | 1.8       | 0.1      | 0.3         | 23.1          | 23.3            |
| 200 | 4.5 | 0.3       | 3.1      | 61.7        | 67.9          |                 |
| 400 | 13.4 | 1.0      | 57.5     | 199.6       | 316.8         |                 |

Based on the results depicted in Tables 8–10, we limit the performance analysis regarding the large instances (S2) to the two new heuristics ATCS-BATCS(β) and ATCS-BATCS(β)-LS and the heuristics ATC-BATCSs, ATC-BATC, and ATC-BATC-LS from literature. The key figures MRIW (upper value) and BEST (lower value; italic) are reported with regard to the instance characteristics number of machines (m) and number of jobs (n) in Table 11 (note that for each combination of m and n, 1,280 instances have been investigated). Best (highest) values per row are marked bold. Table 12 depicts the corresponding computation times.

The results in Table 11 regarding large instances confirm the previous results on the superiority of ATCS-BATCS(β) and ATCS-BATCS(β)-LS in terms of solution quality. This superiority is underlined by the statistical comparison of ATC-BATC-LS, ATCS-BATCS(β), and ATCS-BATCS(β)-LS by two-sided, pairwise t-tests on each instance subset depicted in Table 11. The tests are used to evaluate whether the difference of the MRIWs (with regard to one instance subset) is statistically significant ($< 0.05$; with degrees-of-freedom $df = 1279$) or not. As the results in Table A5 (Appendix A-4) show, ATCS-BATCS(β)-LS performs significantly better than the two other solution methods and ATCS-BATCS(β) performs significantly better than ATC-BATC-LS.

In addition, the last two columns in Table 11 emphasise the suitability of the local search mechanism: the solution quality can be remarkably improved at the expense of higher computation times (cf. Table 12).

The increasing computation times in Table 12 indicate that for solving very large-scale problem instances with thousands of jobs, more sophisticated local search
Table 13. Application case results.

|                | MRIW     | Mean CT [s]         |
|----------------|----------|---------------------|
|                | WEDD     | ATCS-BATCS(\(\beta\)) | ATCS-BATCS(\(\beta\))-LS | WEDD | ATCS-BATCS(\(\beta\)) | ATCS-BATCS(\(\beta\))-LS |
| 3D-1 (m = 5, q = 4) | 0.0      | 76.1                | 83.0 | 0.1 | 82.2 | 129.8 |
| 3D-2 (m = 6, q = 4) | 0.0      | 69.1                | 81.3 | 0.3 | 638.1 | 16,427.8 |
| LC             | 0.0      | 40.6                | –    | 2.6 | 10,116.8 | –    |
| Mean           | 0.0      | 60.0                | 82.2 | 1.1 | 4,203.6 | 8,278.8 |

Table 14. Solution comparison.

|                | Mean number of batches | Mean utilisation per batch [%] | Total setup time | MRIW | Mean \(\beta\) |
|----------------|------------------------|--------------------------------|------------------|------|-----------------|
|                | ATC-BATC-LS            | ATCS-BATCS(\(\beta\))        | ATC-BATC-LS      | ATCS-BATCS(\(\beta\)) | ATCS-BATCS(\(\beta\))-LS |      |
| \(n\) = 100   | 44.0                   | 51.7                           | 80.3             | 4,187,177 | 3,672,484 | 20.6 |
| \(\eta\) = 0.25 | 43.8                  | 48.9                           | 80.2             | 12,281,123 | 9,819,575 | 27.7 |
| \(n\) = 200   | 82.8                   | 94.7                           | 85.8             | 8,162,233  | 6,338,525 | 28.4 |
| \(\eta\) = 0.75 | 83.1                  | 91.0                           | 85.8             | 24,178,510 | 17,980,414 | 33.0 |
| \(n\) = 400   | 159.8                  | 179.7                          | 89.5             | 16,233,419 | 11,516,314 | 33.1 |
| \(\eta\) = 0.75 | 159.5                 | 174.0                          | 89.6             | 47,668,224 | 34,008,545 | 35.7 |
| Mean           | 95.5                   | 106.7                          | 85.2             | 29.8        | 58.5          | 0.87 |

mechanisms or metaheuristics are required to calculate high quality solutions more efficiently.

7.3. Application cases

The problem instances analysed in this section are generated based on the production environments of the two real-world application cases described in section 2. In Table 13, the aggregated mean results for the three sets of instances are presented by the key figures MRIW and CT. Besides the heuristics described so far, we use a WEDD-rule based heuristic to reflect the current state of planning in both companies. In this heuristic, job priorities are calculated by the WEDD \((d_j/w_j)\) instead of the ATCS rule and the batch with the minimum WEDD priority is allocated first.

The results in Table 13 show that a remarkable improvement in terms of the delivery reliability is achieved by the proposed heuristics (compared to WEDD). Because of the high computation times of ATCS-BATCS(\(\beta\)) due to the large number of jobs (4,588.8 on average) for the laser cutting (LC) case, we omitted applying ATCS-BATCS(\(\beta\))-LS for solving (even without local search, computation times are quite high).

Figure 2. Histogram of \(\beta\)-values leading to the best solutions.
The reasons for that are the higher complexity of the priority rules (compared to WEDD) and the three parameters \( \kappa_1, \kappa_2, \) and \( \beta \) that lead to 1,771 calculated schedules by the multi-start heuristic. If we would use instance related parameter specifications like the one depicted in appendix A-2, only a single value for each parameter would be used for calculating a single solution and thus, mean computations times would decrease to approximately 5.7 s. Of course, this would affect the solution quality. Thus, these results also show that more efficient improvement methods would be very desirable.

7.4. General insights

To gain general insights about the structure of ‘good’ solutions, we compare the solutions (for instance set S2) obtained by the best benchmark heuristic considered in this paper (ATC-BATC-LS) with the solutions obtained by ATCS-BATCS(\( \beta \)). For this comparison, we additionally use the key figures mean number of batches, total setup time, and mean capacity utilisation per batch (\( [\%] \)) in Table 14.

The key figure values in Table 14 clearly indicate that the ‘artificial’ batch capacity limitation by the parameter \( \beta \) leads to a higher number of batches and a corresponding lower batch utilisation but also indicate that this does not result in higher total setup times and that the solution quality of ATCS-BATCS(\( \beta \)) is remarkably better. The last column in Table 14 shows that for higher \( \eta \)-values higher \( \beta \)-values are suitable.

The histogram in Figure 2 shows the influence of different \( \beta \)-values in more detail and illustrates that the best solutions of more than the half of the 15,360 problem instances in set S2 are calculated by ATCS-BATCS(\( \beta \)) with a \( \beta \) smaller than 0.95. The figure also illustrates that not a single \( \beta \)-value is suitable but it must be set appropriately (as it is also important for the parameters \( \kappa_1 \) and \( \kappa_2 \)). In consequence of these results, we can conclude that the artificial reduction of the batch capacity by parameter \( \beta \) is suitable to improve the solution quality in terms of total weighted tardiness. This is because the larger number of batches leads to ‘better’ batch completion times without resulting in too high setup times.

8. Concluding remarks

In this paper, we address the \( P|sb, if*, b, a_j, s_{fg}|wT \)-problem that is highly relevant in different manufacturing sectors but has not been considered in literature so far. For solving the problem, we propose a MILP and several heuristics. The experiments show that the standard solver Gurobi is basically able to solve problem instances with up to 60 jobs and 5 machines. However, solving the MILP with Gurobi shows the necessity to solve the problem with heuristics if the number of jobs increases. In consequence, a group of four heuristics are developed. Unique feature point of these heuristics is the artificial reduction of batch utilizations. Hereby, the batch utilisation is controlled by a parameter in a way that smaller batches without containing non-urgent jobs are created. The experimental results show that this feature leads to a general superiority of the new heuristics compared to heuristics from literature. Applying the new heuristic ATCS-BATCS(\( \beta \)) on three instance sets from real-world application cases reveals its great potential: total weighted tardiness can be improved between 16.8% and 88.3%.

Main suggestion for further research is the development of more sophisticated solution methods taking the findings of this paper into account. Particularly smarter mechanisms for the batching decision (e.g. to limit the range of job priorities in one batch) are of interest. Furthermore, the experiments have shown that the multi-start approach leads to a high solution quality but also that computation times become high for large-scale instances (with a number of jobs higher than 1,000). For such instances, a suitable reduction of the applied heuristic parameter values would be very precious (e.g. by functions based on instance characteristics or by machine learning; cf., e.g. Mönch, Zimmermann, and Otto 2006). A similar effect for large-scale instances is observed when using the local search mechanism: solution quality is improved but on cost of high computation times. Regarding this aspect, the development of meta-heuristics like Variable Neighborhood Search or Tabu Search is a promising research topic.

Another aspect for further research and the major limitation of the study is the capacity approximation approach to decide on feasible batches. The rough approximation as used in this contribution (based on the area and the volume of the items, respectively) as well as in literature could lead to infeasible batches. Methods from machine learning may be suitable for a better approximation.

Finally, we would like to remark that the proposed MILP and the heuristics are not only applicable to the described application cases but also to many other cutting, packing, or additive manufacturing problems using other technologies (e.g. stereolithography, selective laser sintering/melting, multi jet modelling or colour jet printing; cf., e.g. Gardan 2016).

Disclosure statement

No potential conflict of interest was reported by the author(s).
Notes on contributors

**PD Christian Gahm** (Dr., Dipl.-Inf.), born in 1979, studied Computer Science at the University of Augsburg with a focus on software engineering and programming languages. Since November 2006 he is a research assistant at the Chair of Production & Supply Chain Management. Since his doctorate in 2010, Mr. Gahm is head of the research group ‘Production Management’ and completed his habilitation in business administration in June 2021. His research interests are energy-aware and serial-batch scheduling as well as production planning concepts and advanced planning systems for special purpose machinery. In addition to courses in the areas of Supply Chain Management and Operations Management, he is responsible for the courses on ERP systems (SAP) at the Institute of Business Administration. The knowledge transfer of research results and of the latest methods into business practice is of great importance to him and has already led to great success in many practical projects.

**Stefan Wahl** (M.Sc.), born in 1991, studied Industrial Engineering at the University of Augsburg with a focus on materials resource management and sustainability. Since April 2017 he is a research assistant and doctoral candidate at the Chair of Production & Supply Chain Management and concerned with serial-batch scheduling and production planning. In teaching, Mr. Wahl imparts content from Operations Management, Discrete Event Simulation and ERP systems (SAP).

**Professor Dr. Axel Tuma**, born in 1963, graduated as an industrial engineer from the University of Karlsruhe in 1991. He worked there as a research assistant at the Institute of Industrial Management and Industrial Production until his doctorate in 1994 on the use of ‘artificial intelligence’ methods for emission-oriented production control. Following his doctorate, Tuma moved to the University of Bremen as a research assistant, where he completed his habilitation in business administration in February 2000. Until his move to the University of Augsburg, he worked there as a university lecturer in the field of sustainable ERP systems. Since 2001, Mr. Tuma holds the Chair of Production & Supply Chain Management at the Institute of Business Administration (University of Augsburg).

**ORCID**

Christian Gahm [http://orcid.org/0000-0002-9707-203X](http://orcid.org/0000-0002-9707-203X)

**References**

Ahmadi, Javad H., Reza H. Ahmadi, Sriram Dasu, and Christopher S. Tang. 1992. “Batching and Scheduling Jobs on Batch and Discrete Processors.” *Operations Research* 40 (4): 750–763. doi:10.1287/opre.40.4.750.

Almeder, Christian, and Lars Mönch. 2011. “Metaheuristics for Scheduling Jobs with Incompatible Families on Parallel Batch Machines.” *Journal of the Operational Research Society* 62 (12): 2083–2096. doi:10.1057/jors.2010.186.

Avramidis, A. N., K. J. Healy, and Reha Uzsoy. 1998. “Control of a Batch-Processing Machine: A Computational Approach.” *International Journal of Production Research* 36 (11): 3167–3181. doi:10.1080/002075498192355.

Azizoglu, Meral, and Scott Webster. 2001. “Scheduling a Batch Processing Machine with Incompatible Job Families.” *Computers & Industrial Engineering* 39 (3–4): 325–335. doi:10.1016/S0360-8352(01)00009-2.

Baker, Kenneth R. 2013. “Computational Results for the Flow-shop Tardiness Problem.” *Computers & Industrial Engineering* 64 (3): 812–816. doi:10.1016/j.cie.2012.12.018.

Balasubramanian, Hari, Lars Mönch, John W. Fowler, and Michele E. Pfund. 2004. “Genetic Algorithm Based Scheduling of Parallel Batch Machines with Incompatible job Families to Minimize Total Weighted Tardiness.” *International Journal of Production Research* 42 (8): 1621–1638. doi:10.1080/00207540310001636994.

Bilyk, Andrew, Lars Mönch, and Christian Almeder. 2014. “Scheduling Jobs with Ready Times and Precedence Constraints on Parallel Batch Machines Using Metaheuristics.” *Computers & Industrial Engineering* 78: 175–185. doi:10.1016/j.cie.2014.10.008.

Brucker, Peter, Andrei Gladky, Jan Hoogeveen, Mikhail Y. Kovalyov, Chris N. Potts, T. Tautenhahn, and Steef van de Velde. 1998. “Scheduling a Batch Machine.” *Journal of Scheduling* 1 (1): 31–54.

Cakici, Eray, Scott J. Mason, John W. Fowler, and H. N. Geismar. 2013. “Batch Scheduling on Parallel Machines with Dynamic job Arrivals and Incompatible job Families.” *International Journal of Production Research* 51 (8): 2462–2477. doi:10.1080/00207543.2012.748227.

Chakhlevitch, Konstantin, Celia A. Glass, and Hans Kellerer. 2011. “Batch Machine Production with Perishability Time Windows and Limited Batch Size.” *European Journal of Operational Research* 210 (1): 39–47. doi:10.1016/j.ejor.2010.10.033.

Chandru, Vijaya, Chung-Yee Lee, and Reha Uzsoy. 1993. “Minimizing Total Completion Time on a Batch Processing Machine with job Families.” *Operations Research Letters* 13 (2): 61–65. doi:10.1016/0167-6377(93)90030-K.

Chang, P.-Y., Purusothaman Damodaran, and S. Melouk. 2004. “Minimizing Makespan on Parallel Batch Processing Machines.” *International Journal of Production Research* 42 (19): 4211–4220. doi:10.1080/00207540410001711863.

Cheng, Bayi, Junfeng Cai, Shanlin Yang, and Xiaoxuan Hu. 2014. “Algorithms for Scheduling Incompatible job Families on Single Batching Machine with Limited Capacity.” *Computers & Industrial Engineering* 75: 116–120. doi:10.1016/j.cie.2014.06.014.

Chiang, Tsung-Che, Hsueh-Chien Cheng, and Li-Chen Fu. 2010. “A Memetic Algorithm for Minimizing Total Weighted Tardiness on Parallel Batch Machines with Incompatible job Families and Dynamic job Arrival.” *Computers & Operations Research* 37 (12): 2257–2269. doi:10.1016/j.cor.2010.03.017.

Cicirello, Vincent A. 2009. “Weighted Tardiness Scheduling with Sequence-Dependent Setups: A Benchmark Problem for Soft Computing.” In *Applications of Soft Computing*. Vol. 52, edited by Erel Avineri, Mario Köppen, Keshav Dahal, Yos
Sunitiyoso, and Rajkumar Roy. 189–198. Advances in Soft Computing. Berlin, Heidelberg: Springer.

Dauzère-Pérès, Stéphane, and Lars Mönch. 2013. “Scheduling Jobs on a Single Batch Processing Machine with Incompatible Job Families and Weighted Number of Tardy Jobs Objective.” Computers & Operations Research 40 (5): 1224–1233. doi:10.1016/j.cor.2012.12.012.

Dobson, Gregory, and Ramakrishnan S. Nambimadom. 2001. “The Batch Loading and Scheduling Problem.” Operations Research 49 (1): 52–65. doi:10.1287/opre.49.1.52.11189.

Duenyas, Izak, and John J. Neale. 1997. “Stochastic Scheduling of a Batch Processing Machine with Incompatible Job Families.” Annals of Operations Research 70: 191–220. doi:10.1023/A:1018922104670.

Dupont, Lionel, and Clarisse Dhaenens-Flipo. 2002. “Minimizing the Makespan on a Batch Machine with non-Identical Job Sizes: An Exact Procedure.” Computers & Operations Research 29 (7): 807–819. doi:10.1016/S0305-0548(00)00078-2.

Durasević, Marko, and Domagoj Jakobović. 2018. “A Survey of Dispatching Rules for the Dynamic Unrelated Machines Environment.” Expert Systems with Applications 113: 555–569. doi:10.1016/j.eswa.2018.06.053.

Emde, Simon, Lukas Polten, and Michel Gendreau. 2020. “Logic-based Benders Decomposition for Scheduling a Batch Machine.” Computers & Operations Research 113: 104777. doi:10.1016/j.cor.2019.104777.

Erramilli, Vishnu, and Scott J. Mason. 2008. “Multiple Orders per Job Batch Scheduling with Incompatible Jobs.” Annals of Operations Research 159 (1): 245–260. doi:10.1007/s10479-007-0286-x.

Fanti, M. P., B. Maione, G. Piscitelli, and B. Turchiano. 1996. “Heuristic Scheduling of Jobs on a Multi-Product Batch Processing Machine.” International Journal of Production Research 34 (8): 2163–2186. doi:10.1080/00207549608905019.

Fowler, John W., Nipa Phojanamongkolkij, K. C. Cochran, and D. C. Montgomery. 2002. “Optimal Batching in a Wafer Fabrication Facility Using a Multiproduct G/G/c Model with Batch Processing.” International Journal of Production Research 40 (2): 275–292. doi:10.1080/00207540110081489.

Fu, Qing, Appa I. Sivakumar, and Kunpeng Li. 2012. “Optimization of Flow-Shop Scheduling with Batch Processor and Limited Buffer.” International Journal of Production Research 50 (8): 2267–2285. doi:10.1080/00207543.2011.565813.

Gahm, Christian. 2021. “Problem Instances for the Parallel Serial-Batch Scheduling Problem with Incompatible Job Families, Sequence-Dependent Setup Times, and Arbitrary Sizes.” Mendeley Data V1.

Gahm, Christian, John J. Kanet, and Axel Tuma. 2019. “On the Flexibility of a Decision Theory-Based Heuristic for Single Machine Scheduling.” Computers & Operations Research 101: 103–115. doi:10.1016/j.cor.2018.09.005.

Gao, Yuan, Jinjiang Yuan, C. T. Ng, and Edwin T. C. Cheng. 2019. “A Further Study on two-Agent Parallel-Batch Scheduling with Release Dates and Deteriorating Jobs to Minimize the Makespan.” European Journal of Operational Research 273 (1): 74–81. doi:10.1016/j.ejor.2018.07.040.

Gardan, Julien. 2016. “Additive Manufacturing Technologies: State of the art and Trends.” International Journal of Production Research 54 (10): 3118–3132. doi:10.1080/00207543.2015.1115909.
Appendix

A.1. Literature analysis

A.2. Approximation of look-ahead parameters

The look ahead parameters $\kappa_1$ and $\kappa_2$ are calculated as described by Lee and Pinedo (1997):

$$
\kappa_1 = \begin{cases} 
1.2 \cdot \ln(\mu) - \vartheta & \text{if } \tau < 0.5 \\
1.2 \cdot \ln(\mu) - \vartheta & \text{if } \eta < 0.5 \text{ and } \mu > 5 \\
1.2 \cdot \ln(\mu) - \vartheta - 1 & \text{if } \tau < 0.5, \eta < 0.5, \text{ and } \mu > 5 \\
1.2 \cdot \ln(\mu) - \vartheta & \text{otherwise}
\end{cases}
$$

$$
\kappa_2 = \begin{cases} 
\frac{\tau}{1.8 \cdot \sqrt{\eta}} & \text{if } \tau < 0.8 \\
\frac{\tau}{2.0 \cdot \sqrt{\eta}} & \text{otherwise}
\end{cases}
$$

with $\mu = n/m$, $\vartheta = (d_{\text{max}} - d_{\text{min}})/C_{\text{max}}$, $\tau = 1 - (\tilde{d}/C_{\text{max}})$, $\eta = s/p$ and $C_{\text{max}}$ approximated as described in section 6.1 (cf. (20)).

A.3. Results of the preliminary standard solver testing

Table A4 lists the relative improvements of Gurobi and CPLEX (compared to each other). The maximum computation time has been set to one hour and both solvers have been executed with standard parameters (as also done by Dauzère-Pérès and Mönch, 2013).
### A.4. Statistical performance comparison.

#### Table A1. Literature with if #.

| Scheduling environment | Basic batching type | Batch capacity requirements (sizes) | Processing times | Setup times | Objective criterion/criteria | Opt. programmes (e.g., MILP) | Exact solution methods | Heuristics | Metaheuristics |
|------------------------|---------------------|-------------------------------------|------------------|-------------|-------------------------------|--------------------------------|---------------------|-------------|---------------|
| Chandru, Lee, and Uzsoy (1993) | S | x | x | X | X | C | - | x | - |
| Hung (1998) | S/P | x | x | x | x | T | - | x | x |
| Mehta and Uzsoy (1998) | S | x | x | x | x | T | - | x | x |
| Kurz (2003) | P | x | x | x | x | wT | - | - | - |
| Nong, Ng, and Cheng (2008) | S | x | x | x | x | Cmax | - | - | x |
| Lin and Liao (2012) | F/2 | x | x | x | x | C, Cmax, T | - | x | - |
| Tang et al. (2017) | S | x | x | x | Cmax | - | - | - |
| Kovalyov and Šešok (2019) | S | x | x | x | x | Cmax, U | - | - | - |
| Passchyn and Spieksma (2019) | J | x | x | x | p_i = p_j | F | - | - | - |
| Emde, Polten, and Gendreau (2020) | S | x | x | x | x | Lmax | x | x | x |
Table A2. Literature with \( if \) and \( p_j = p \) or \( p_j = p_\ell \).

| Scheduling environment | Basic batching type | Batch capacity requirements (sizes) | Setup times | Objective criterion/criteria | Opt. programmes (e.g., MILP) | Exact solution methods | Heuristics | Metaheuristics |
|------------------------|---------------------|-------------------------------------|-------------|-------------------------------|------------------------------|-----------------------|------------|---------------|
| Ahmadi et al. (1992)   | \( \tilde{F} \)     | x                                   | x           | C, \( \text{Cmax} \)         | x \( \times \) \( \times \) | \( \times \) |
| Uzsoy (1995)           | \( \tilde{S}/\tilde{P} \) | \( a \_i = a \_f \) | \( C, wC, \text{Cmax, Lmax} \) | x \( \times \) \( \times \) | \( \times \) |
| Duenyas and Neale (1997)| \( \tilde{S} \)       | x                                   | \( a_i = a_f \) | Cih                          | \( \times \) \( \times \) |
| Kempf, Uzsoy, and Wang (1998)| \( \tilde{S} \) | x                                   | \( C, \text{Cmax} \) | x \( \times \) | \( \times \) |
| Azizoglu and Webster (2001) | \( \tilde{S} \) | x                                   | \( wF \) | x \( \times \) \( \times \) |
| Dobson and Namdudam (2001)| \( \tilde{S} \)       | x                                   | \( wT \) | \( \times \) \( \times \) |
| Fowler et al. (2002)   | \( \tilde{P} \)       | x                                   | C, \( \text{Cmax} \) | \( \times \) \( \times \) | \( \times \) \( \times \) |
| Balasubramanian et al. (2004) | \( \tilde{P} \) | x                                   | U           | \( \times \) \( \times \) | \( \times \) |
| Koh et al. (2004)      | \( \tilde{P} \)       | x                                   | C, \( \text{Cmax} \) | \( \times \) \( \times \) |
| Jolai (2005)           | \( \tilde{S} \)       | x                                   | U           | \( \times \) \( \times \) | \( \times \) |
| Koh et al. (2005)      | \( \tilde{S} \)       | x                                   | C, \( \text{Cmax} \) | x \( \times \) \( \times \) |
| Mönch et al. (2005)    | \( \tilde{P} \)       | x                                   | \( wT \) | \( \times \) \( \times \) |
| Perez, Fowler, and Carlyle (2005) | \( \tilde{S} \) | x                                   | \( wT \) | \( \times \) \( \times \) |
| Tangudu and Kurz (2006) | \( \tilde{S} \)       | x                                   | \( wT \) | \( \times \) \( \times \) |
| Erramilli and Mason (2008)| \( \tilde{S} \) | x                                   | \( wT \) | x \( \times \) \( \times \) |
| Kashan and Karimi (2008)| \( \tilde{S} \)       | x                                   | \( wC \) | \( \times \) \( \times \) |
| Kurz and Mason (2008)  | \( \tilde{S} \)       | x                                   | \( wT \) | \( \times \) \( \times \) |
| Chiang, Cheng, and Fu (2010) | \( \tilde{P} \) | x                                   | \( wT \) | \( \times \) \( \times \) |
| Jula and Leachman (2010)| \( \tilde{P} \)       | x                                   | TP, UB      | \( \times \) \( \times \) |
| Almeder and Mönch (2011)| \( \tilde{P} \)       | x                                   | \( wT \) | \( \times \) \( \times \) |
| Chakhlevitch, Glass, and Kellerer (2011): \( p_j = p \) | \( \tilde{S} \)       | x                                   | \( FY \) | \( \times \) \( \times \) |
| Gokhale and Mathirajan (2011)| \( \tilde{S} \) | x                                   | \( wT \) | x \( \times \) \( \times \) |
| Fu, Sivakumar, and Li (2012): permutation flow shop | \( \tilde{P} \)       | \( a_i = a_f \) | Cmax       | x \( \times \) \( \times \) |
| Tajan, Sivakumar, and Gershwin (2012) | \( \tilde{P} \) | x                                   | C           | x \( \times \) \( \times \) |
| Yao, Jiang, and Li (2012) | \( \tilde{S} \)       | x                                   | U, wU       | x \( \times \) \( \times \) |
| Dauzère–Péres and Mönch (2013)| \( \tilde{P} \) | x                                   | \( wT \) | \( \times \) \( \times \) |
| Jia, Jiang, and Li (2013)| \( \tilde{P} \)       | x                                   | TWL         | \( \times \) \( \times \) |
| Pearn, Hong, and Tai (2013) | \( \tilde{P} \) | x                                   | \( wT \) | \( \times \) \( \times \) |
| Bilyk, Mönch, and Almeder (2014) | \( \tilde{P} \) | x                                   | \( wT \) | \( \times \) \( \times \) |
| Jia, Jiang, and Li (2015)| \( \tilde{P} \)       | x                                   | \( wT \) | \( \times \) \( \times \) |
| Li (2017)               | \( \tilde{P} \)       | x                                   | Cmax        | \( \times \) \( \times \) |
| Zhang et al. (2017)    | \( \tilde{F}_2 \)     | x                                   | C           | \( \times \) \( \times \) |
| Huynh and Chien (2018) | \( \tilde{P} \)       | x                                   | Cmax        | x \( \times \) \( \times \) |
| Shi, Huang, and Shi (2018)| \( \tilde{S}/\tilde{P} \) | x                                   | \( wC, \text{Cmax} \) | x \( \times \) \( \times \) |
| Huang, Shi, and Shi (2019) | \( \tilde{F} \) | x                                   | \( wC \) | x \( \times \) \( \times \) |
| Wang et al. (2019)     | \( \tilde{F}_2 \)     | x                                   | \( wC \) | x \( \times \) \( \times \) |
### Table A3. Literature with $if^*$ and $pj$.  

| Scheduling environment | Basic batching type | Batch capacity requirements (size) | Setup times | Objective criterion/criteria | Opt. programmes (e.g., MILP) | Exact solution methods | Heuristics | Metaheuristics |
|------------------------|---------------------|-----------------------------------|-------------|-------------------------------|-----------------------------|------------------------|------------|----------------|
| Fanti et al. (1996)    | $p^*$               | $x$                               | $x$         | Cmax                          | --                          | --                     | $x$         | --             |
| Avramidis, Healy, and Uzsoy (1998) | $f^*$               | $x$                               | $x$         | $F$                           | --                          | --                     | $x$         | --             |
| Oey and Mason (2001)   | $J$                 | $x$                               | $x$         | $wT$                          | --                          | --                     | $x$         | --             |
| Dupont and Dhaenens–Flipo (2002) | $S$                 | $x$                               | $x$         | Cmax                          | --                          | --                     | $x$         | --             |
| Mathirajan, Sivakumar, and Chandru (2004) | $P^*$               | $x$                               | $x$         | $U$                           | --                          | --                     | $x$         | --             |
| Gupta and Sivakumar (2006) | $S$                 | $x$                               | $x$         | $T, Tmax, U$                  | --                          | --                     | $x$         | --             |
| Malve and Uzsoy (2007) | $P^*$               | $x$                               | $x$         | $Lmax$                        | --                          | --                     | $x$         | --             |
| Geiger and Uzsoy (2008) | $S$                 | $x$                               | $x$         | Cmax                          | --                          | --                     | $x$         | --             |
| Nong et al. (2008)     | $S$                 | $x$                               | $x$         | Cmax                          | --                          | --                     | $x$         | --             |
| Lei and Guo (2011)     | $F^*$               | $x$                               | $x$         | $T, Tmax, wU$                 | --                          | --                     | $x$         | --             |
| Cakici et al. (2013)   | $P^*$               | $x$                               | $x$         | $wC$                          | $x$                         | $x$                     | $x$         | --             |
| Cheng et al. (2014)    | $S$                 | $x$                               | $x$         | $Cb, Cmax$                    | $x$                         | $x$                     | $x$         | --             |
| Li and Chen (2014)     | $S$                 | $x$                               | $x$         | $U, wU$                       | --                          | $x$                     | $x$         | --             |
| Suppia and Omar (2014) | $S$                 | $x$                               | $x$         | $wT$                          | $x$                         | $x$                     | $x$         | --             |
| Liu et al. (2016)      | $P^*$               | $x$                               | $x$         | $wT, TEC$                     | --                          | --                     | $x$         | --             |
| Gao et al. (2019)      | $S$                 | $x$                               | $x$         | Cmax                          | $x$                         | --                     | $x$         | --             |
| Li, Li, and Huang (2019) | $S$                 | $x$                               | $x$         | $Lmax$                        | $x$                         | $x$                     | $x$         | --             |
| He, Xu, and Lin (2020) | $S$                 | $x$                               | $x$         | $Cmax, Lmax$                  | $x$                         | $x$                     | $x$         | --             |
| Huang, Wang, and Jiang (2020) | $S$             | $x$                               | $x$         | Cmax                          | $x$                         | $x$                     | $x$         | --             |

### Table A4. Preliminary tests.  

| $m$ | $n$ | $q$ | $jt$ | $fm$ | cap. req. sc. | $\eta$ | $tf$ | $rrd$ | Gurobi (%) | Cplex (%) | Lower bound |
|-----|-----|-----|------|------|--------------|-------|------|------|------------|-----------|-------------|
| T1  | 1   | 15  | 3    | DND  | CRS1         | 0.25  | 0.3  | 0.5  | 26.43      | 0.00      | 2,08E-03    |
| T2  | 1   | 15  | 3    | DUD  | CRS1         | 0.25  | 0.3  | 0.5  | 0.00       | 25.64     | 7,46E-04    |
| T3  | 1   | 15  | 5    | DUD  | CRS2         | 0.75  | 0.3  | 0.5  | 26.43      | 0.00      | 2,08E-03    |
| T4  | 1   | 30  | 3    | DUD  | CRS3         | 0.25  | 0.3  | 0.5  | 17.20      | 0.00      | 1,35E-04    |
| T5  | 1   | 30  | 3    | DUD  | CRS1         | 0.75  | 0.6  | 2.5  | 17.47      | 0.00      | 8,28E-06    |
| T6  | 1   | 30  | 5    | DUD  | CRS3         | 0.25  | 0.6  | 2.5  | 0.00       | 25.64     | 7,46E-04    |
| T7  | 3   | 15  | 3    | DND  | CRS3         | 0.25  | 0.3  | 0.5  | 0.00       | 25.64     | 7,46E-04    |
| T8  | 3   | 15  | 3    | DND  | CRS4         | 0.75  | 0.3  | 0.5  | 0.00       | 25.64     | 7,46E-04    |
| T9  | 3   | 15  | 5    | DUD  | CRS3         | 0.75  | 0.3  | 0.5  | 0.00       | 25.64     | 7,46E-04    |
| T10 | 3   | 15  | 5    | DND  | CRS4         | 0.75  | 0.3  | 0.5  | 0.00       | 25.64     | 7,46E-04    |
| T11 | 3   | 30  | 5    | DND  | CRS3         | 0.75  | 0.6  | 2.5  | 0.00       | 66.52     | 1,00E-04    |
| T12 | 3   | 30  | 10   | DUD  | CRS4         | 0.25  | 0.6  | 2.5  | 0.00       | 66.52     | 1,00E-04    |
| T13 | 4   | 30  | 3    | DUD  | CRS3         | 0.25  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T14 | 4   | 30  | 5    | DUD  | CRS4         | 0.75  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T15 | 4   | 30  | 10   | DUD  | CRS3         | 0.25  | 0.6  | 2.5  | 0.00       | 66.52     | 1,00E-04    |
| T16 | 4   | 60  | 3    | DND  | CRS2         | 0.75  | 0.6  | 2.5  | 0.00       | 66.52     | 1,00E-04    |
| T17 | 4   | 60  | 5    | DUD  | CRS3         | 0.25  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T18 | 4   | 60  | 10   | DUD  | CRS4         | 0.75  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T19 | 5   | 30  | 3    | DUD  | CRS3         | 0.25  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T20 | 5   | 30  | 5    | DUD  | CRS4         | 0.75  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T21 | 5   | 30  | 10   | DUD  | CRS3         | 0.25  | 0.6  | 2.5  | 0.00       | 66.52     | 1,00E-04    |
| T22 | 5   | 60  | 3    | DUD  | CRS2         | 0.75  | 0.6  | 2.5  | 0.00       | 66.52     | 1,00E-04    |
| T23 | 5   | 60  | 5    | DUD  | CRS3         | 0.25  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |
| T24 | 5   | 60  | 10   | DUD  | CRS4         | 0.75  | 0.3  | 0.5  | 0.00       | 66.52     | 1,00E-04    |

Mean 15.99 10.31  
Max 88.17 66.52
### Table A5. Statistical comparison.

| SM-B | SM-A vs. ATC-BATC-LS | ATC-BATC-LS vs. ATCS-BATCS(β) | ATCS-BATCS(β) vs. ATCS-BATCS(β)-LS |
|------|----------------------|-------------------------------|-------------------------------------|
| Instance subset | Rdiff [%] | p-value | Signif. | Rdiff [%] | p-value | Signif. | Rdiff [%] | p-value | Signif. |
| 1 | 100 | -56.2 | 1.01E-201 | y | -60.7 | 3.24E-280 | y | -10.1 | 1.01E-181 | y |
| 200 | -46.0 | 1.25E-164 | y | -53.0 | 2.57E-275 | y | -12.8 | 6.06E-230 | y |
| 400 | -39.1 | 8.66E-138 | y | -48.3 | 7.69E-272 | y | -15.1 | 2.02E-254 | y |
| 2 | 100 | -56.9 | 1.75E-258 | y | -61.5 | 0.00E+00 | y | -10.7 | 2.58E-160 | y |
| 200 | -51.4 | 2.95E-236 | y | -57.1 | 0.00E+00 | y | -11.8 | 4.54E-195 | y |
| 400 | -45.5 | 5.96E-212 | y | -51.9 | 0.00E+00 | y | -11.8 | 1.06E-222 | y |
| 3 | 100 | -54.5 | 1.52E-269 | y | -60.3 | 0.00E+00 | y | -12.8 | 5.90E-178 | y |
| 200 | -50.3 | 4.52E-257 | y | -56.5 | 0.00E+00 | y | -12.4 | 4.91E-199 | y |
| 400 | -48.2 | 6.77E-238 | y | -54.6 | 0.00E+00 | y | -12.2 | 9.36E-210 | y |
| 5 | 100 | -52.7 | 1.69E-269 | y | -60.4 | 0.00E+00 | y | -14.7 | 2.52E-225 | y |
| 200 | -45.4 | 8.17E-248 | y | -53.4 | 0.00E+00 | y | -13.8 | 1.83E-215 | y |
| 400 | -46.2 | 8.22E-249 | y | -53.6 | 0.00E+00 | y |             |             |             |

*Rdiff* = relative difference of MRWIs: \[
\frac{MRIW(SM-A) - MRIW(SM-B)}{\max\{MRIW(SM-A), MRIW(SM-B)\}}
\]