Learning-augmented heuristics for scheduling parallel serial-batch processing machines

Aykut Uzunoglu, Christian Gahm, Stefan Wahl, Axel Tuma

Angaben zur Veröffentlichung / Publication details:

Uzunoglu, Aykut, Christian Gahm, Stefan Wahl, and Axel Tuma. 2023. "Learning-augmented heuristics for scheduling parallel serial-batch processing machines." Computers & Operations Research 151: 106122. https://doi.org/10.1016/j.cor.2022.106122.

Learning-augmented heuristics for scheduling parallel serial-batch processing machines

Aykut Uzunoglu, Christian Gahm, Stefan Wahl *, Axel Tuma

Chair of Business Administration, Production & Supply Chain Management, Augsburg University, D-86135 Augsburg, Germany

A B S T R A C T

The addressed machine scheduling problem considers parallel machines with incompatible job families, sequence-dependent setup times, limited batch capacities, and arbitrary sizes combined with the serial-batch processing characteristic (i.e., the processing time of a batch is equal to the sum of processing times of all jobs grouped in a batch). The primary objective is the minimization of the total weighted tardiness, and a subordinate (secondary) objective is the minimization of the flow time. This scheduling problem arises in many production environments like cutting operations (metal-processing industry or garment industry) or in industrial 3D printing. For solving this problem, we propose a new multi-start construction heuristic with controlled batch urgencies. Furthermore, to improve solution efficiency, we use machine learning methods that are appropriate for multi-target regression with dependent outputs (i.e., Neural networks) to minimize the number of starts by predicting the most suitable heuristic parameters. Hereby, different learning aspects and pipeline parameters must be considered. Additionally, we apply a mixed-integer linear program and a local search mechanism with advanced termination criteria for solution improvement.

To evaluate the performance of the new heuristic, we use an exhaustive set of small, large, and very large instances (with symmetric Euclidean, asymmetric Euclidean, and arbitrary sequence-dependent setup times) and heuristics from the literature. The results indicate the superiority of the new, learning-augmented heuristics in terms of solution quality and computation times.

1. Introduction

The scheduling of parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times, and arbitrary sizes arises in different production environments like industrial 3D printing (cf., e.g., Helo et al., 2019) or laser-cutting in the metal-processing industry (cf., e.g., Gahm et al., 2022). The basic task of this scheduling problem (illustrated for the laser-cutting case in Fig. 1) is the grouping of a set J of n jobs ($J = \{J_j | j = 1, \ldots, n \in \mathbb{Z}^{>}\}$) into batches $B = \{B_b | b = 1, \ldots, o \in \mathbb{Z}^{>}\}$ (the number of batches o is not given but never larger than n) and the scheduling (machine allocation and sequencing) of the formed batches on a set of m identical parallel machines $M = \{M_i | i = 1, \ldots, m \in \mathbb{Z}^{>}\}$. Hereby, the term "identical parallel machines" not only indicates identical processing speeds but also an identical maximum batch capacity bc for all machines. Each job is characterized by its weight w_j, processing time p_j, due date d_j, and an individual (arbitrary) batch capacity requirement cr_j (size), whereby $cr_j \leqslant bc$ must hold.

In addition, each job belongs to a job family f comprising n_f jobs (the complete set of incompatible job families is $F = \{F_f | f = 1, \ldots, q \in \mathbb{Z}^{>}\}$ and a set I_f containing all jobs that belong to the family f). The job families are characterized as "incompatible" because jobs of different families cannot be processed together in one batch (e.g., due to technical restrictions). The grouping of multiple jobs belonging to the same family into batches is done to reduce setup efforts as sequence-dependent setup times between the processing of two batches (jobs) are mandatory. The setup times are defined by $s_{f,g}$ for a setup from a batch containing jobs of family f to a batch containing jobs of family g ($s_{0,f}$ depicts initial setup times for family f). Because the processing of jobs in a batch is performed in a sequential (serial) way, the processing time of a batch is equal to the sum of the processing times of all jobs in the batch. Thus, the machines are called serial-batching machines.

Further assumptions are that each machine can process at most one batch at a time, that one batch can only be processed by one machine at a

* Corresponding author.
E-mail address: stefan.wahl@wiwi.uni-augsburg.de (S. Wahl).

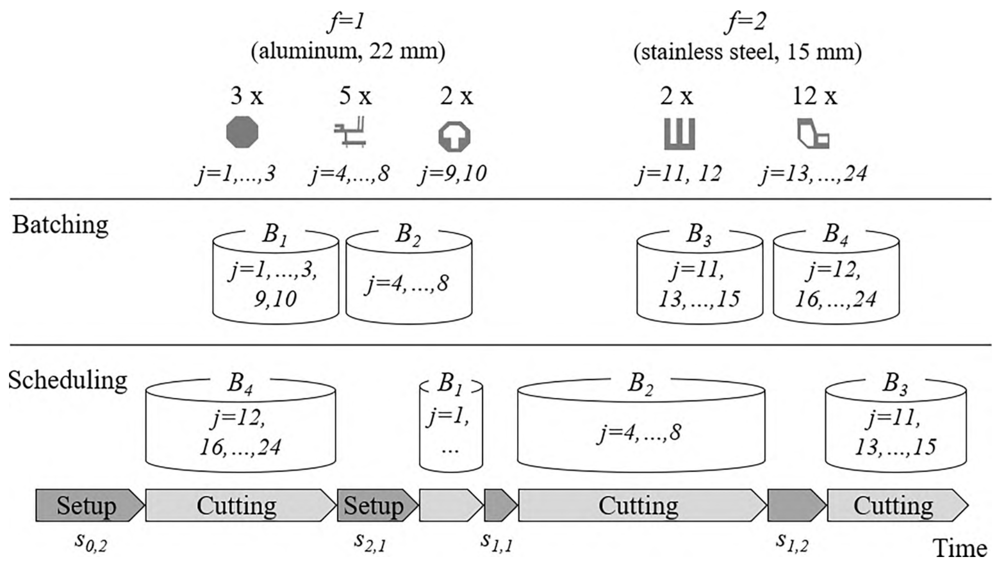

Fig. 1. The serial-batch scheduling problem.

time, that all jobs are available for processing at the beginning (no release dates), that batch and job processing cannot be interrupted (no pre-emption), and that once processing of a batch has started, jobs cannot be added or removed. This assumption is called "batch availability" (in contrast to "item (job) availability" with jobs being completed and available immediately after their processing on a machine). In consequence, the completion time of a job is equal to the completion time of the batch a job is assigned to. The primary scheduling objective is the minimization of the total weighted tardiness to maximize delivery reliability as the considered production environments are often part of a multi-stage production process and/or supply chain. If all jobs can be delivered in time and the weighted tardiness is equal to zero, we additionally use the objective of total flowtime minimization (to reduce work-in-process inventories) as a subordinate (secondary) objective. The lexicographical ordering of both objectives is illustrated by the objective function (cf., Appendix A-1).

The described NP-hard problem was first investigated and classified as $\widetilde{P}|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$-problem by Gahm et al. (2021) (details on the used problem classification scheme can also be found there). It is different from most other batch scheduling problems with limited batch capacities because it is not necessarily suitable to maximize batch utilization. This is due to the very specific combination of characteristics, particularly due to the weighted tardiness objective and the serial-batch characteristic: Batches with a lower utilization result in more batches with a lower processing time and thus, batches can be scheduled more flexibly according to the due dates of the jobs. Of course, more batches also result in additional setups delaying batch start and completion times. This relationship is exploited by the ATCS-BATCS(β) heuristic (and its variants) proposed by Gahm et al. (2021), where the β-parameter is used to control the batch utilization. Because the setting of this parameter and the two look-ahead parameters of the heuristic remarkably influence the solution quality, a multi-start heuristic approach performing a full grid search is applied. This leads to very competitive results concerning the solution quality but also leads to high computation times for large-scale problem instances with several thousand jobs (as can be found in some industries). Therefore, the goal of this contribution is to propose new solution methods that are competitive regarding solution quality and that are particularly efficient to solve large-scale problem instances.

With this report, we extend the study of Gahm et al. (2021) in several ways and make the following contributions to the literature:

- A new multi-start construction heuristic based on parametrized controlled batch urgencies is presented.
- To improve solution efficiency (i.e., reduce computation times with a competitive solution quality), we propose using machine learning (i.e., Artificial neural networks, NN) to predict the most suitable heuristic parameters for the new heuristic and for a heuristic from the literature. Hereby, the challenge is that a multi-target regression with dependent outputs must be performed.
- In addition, we propose several heuristics that use a mixed-integer linear program or a local search with advanced termination criteria for improving solution quality.
- For the training of the NN, its validation, hyper-parameter tuning, and heuristic parameter tuning, 74,688 instances have been generated. The performance of the new heuristics and the heuristic from the literature is evaluated by an additional set of 18,672 instances with up to 3,200 jobs. The complete data for scheduling and machine learning is made freely available for download.

The remainder of this paper is organized as follows. In section 2, related work on the different aspects is discussed. The basic solution methods, two construction heuristics, and two improvement procedures are presented in section 3. In section 4, the enhancement of the construction heuristics to learning-augmented heuristics is described. The instance generation procedure is depicted in section 5. In section 6, key figures and parameter tuning are described, and several analyses of the proposed solution methods are performed. The paper closes with a summary and further research topics in section 7.

2. Related work

The related work section is divided into three parts: First, we analyze existing scheduling approaches addressing the problem at hand (or closely related ones). Second, learning-augmented scheduling methods are discussed. Third, machine learning methods that are appropriate for the parameter prediction are investigated.

2.1. Serial-batch scheduling approaches

An exhaustive literature review regarding the problem addressed in this paper is presented by Gahm et al. (2021). Thus, we only give a brief review of the most closely related literature and focus on scheduling approaches considering serial-batch processing machines, an arbitrary number of incompatible job families, and sequence-dependent setup times related to job families. Altogether, we identified 13 articles in the literature having these characteristics. Since the availability type "item availability" in combination with regular objective functions (like the weighted tardiness objective) fundamentally changes the problem, the solution methods proposed by Jordan & Drexl (1998), Voß & Witt (2007), Shen & Buscher (2012), Giglio, 2015, Shahvari & Logendran (2016), and Shahvari & Logendran (2017) cannot be applied directly to the problem at hand where batch availability must be considered but can only provide some suggestions. This also holds for the second group of articles, namely Naddef & Santos (1988), Shen et al. (2013), and Shen & Gupta (2018), who tackle batching problems with unrestricted maximum batch capacities (i.e., batches can be of arbitrary size).

Pei et al. (2017) address a problem with a single serial-batch processing machine, deteriorating processing times, and a batch capacity requirement equal to one (i.e., the number of jobs in a batch is limited by their amount): $\widetilde{S}|sb,\ if^*,\ b,\ s_{fg}|\sim$. The authors investigate the objectives makespan, maximum tardiness, maximum lateness, maximum earliness, number of tardy jobs, and total completion time, and define dominance properties related to these objectives. As the proposed exact algorithms are based on these properties, they cannot be transferred to the problem considered here.

A hybrid job-shop problem minimizing the makespan is considered by Castillo & Gazmuri (2015) ($h\widetilde{J}|sb,\ if^*,\ b_i,\ a_j,\ s_{fg}|wT$). Hereby, batches are created once and remain fixed throughout the production process. Setup times are not only sequence-dependent but also job family and machine-dependent. In addition, machine-specific batch capacity lower and upper bounds are given. The authors propose three Genetic algorithm (GA) variants with randomly generated initial solutions and different crossover operators and compare their performance to Simulated Annealing and a random search heuristic. The very specific representation of individuals and the related operators, the hardly documented schedule generation scheme, the exploitation of the batch capacity lower bound to determine a minimum number of batches, and the makespan objective, make their GAs hardly applicable to the problem considered here.

Suppiah & Omar (2014) discuss a very similar problem ($\widetilde{S}|sb,\ if^*,\ b,\ s_{fg}|wT$) as described in the introduction, except that a single machine is considered and batch capacities are restricted by the number of jobs. For solving, the authors present a mixed integer linear program, a hybrid Tabu Search, and three dispatching-rule heuristics. Without further reasoning, the authors artificially limit the number of batches before applying their solution methods and evaluate them with regard to the number of batches. Nevertheless, their dispatching heuristics have been applied to the problem at hand by Gahm et al. (2021), but have been outperformed by their presented heuristics.

To the best of our knowledge, only Gahm et al. (2021) investigate the same problem ($\widetilde{P}|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$) as addressed in this paper. They propose a mixed-integer linear program, several new priority-rule based construction heuristics, and a local search improvement procedure for solving. The construction heuristics (ATCS-BATCS(β) and its variants) use the "Apparent tardiness cost with setups" (ATCS)-priority rule for determining job and batch priorities, whereby the introduced β-parameter controls the batch utilization (more detail can be found in 3.1). To simplify the reading, we abbreviate their construction heuristic by BATCS-b in the following.

Their experimental results show that their heuristics based on "controlled batch utilization" outperform all approaches from the literature. However, due to the multi-start design of the heuristics (used to perform a grid search on heuristic parameters), computation times are quite high for very large instances with thousands of jobs (an order of magnitude found in practice). Besides this result, the authors make some suggestions for solution method development that are taken up in this work and discussed in the corresponding paragraphs.

2.2. Learning-augmented scheduling methods

The basic idea of learning-augmented scheduling methods (algorithms, optimization programs, heuristics, or metaheuristics) is to improve the solution quality of schedules and/or reduce computation times by enhancing a scheduling method with automated "learning". The general term "learning" is used here because the allocation of learning methods to the fields "Machine learning", "Artificial intelligence", "Deep learning", and "Statistics" is not always clear. In the literature, five basic approaches (scopes) for learning-augmented scheduling can be found: Learning is used

- to directly make scheduling (allocation and/or sequencing) decisions.
- for the automated design of new scheduling methods (heuristics).
- to select the most suitable scheduling method from a set of existing methods.
- to determine the most suitable parameters of a scheduling method.
- to predict the objective value achieved by a specific scheduling method.

Often, the second and third approaches are summarized under the term "hyper-heuristic". Burke et al. (2010) define hyper-heuristics as an "[…] automated methodology for selecting or generating heuristics to solve hard computational search problems".

Besides the different scopes, we differentiate learning-augmented scheduling methods by their learning mode (time of learning): online learning, i.e., during solving a single problem instance, and offline learning, i.e., gathering knowledge from a set of already solved problem instances. Because the learning mode directly influences computation times (online learning can remarkably increase the computation time for a single instance), this aspect is crucial and thus, analyzed for all learning-augmented scheduling methods discussed in the following.

Although we propose to use the fourth approach in this paper, we would like to give a brief overview of selected papers addressing the other approaches first (additional literature addressing learning-augmented scheduling methods can be found in Burke et al., 2013, Priore et al., 2014, Branke et al., 2016, Nguyen et al., 2017a, and Bertolini et al., 2021).

The approach of directly learning scheduling decisions with NNs is for instance pursued by El-Bouri et al. (2000) and Weckman et al. (2008). Agarwal et al. (2006) use an adaptive online reinforcement learning method and Chen et al. (2019) propose an online reinforcement learning-based assigning policy. Further approaches using NNs directly for scheduling decisions are described in the review of Akyol & Bayhan (2007).

Most approaches to the design of new scheduling methods use Genetic programming: cf., e.g., Geiger et al. (2006), Geiger & Uzsoy (2008), Nguyen et al. (2013), Pickardt et al. (2013), Branke et al. (2015), Freitag & Hildebrandt (2016), or Nguyen et al. (2017b). Genetic programming can be used for online learning and offline learning, but most often the offline learning approach is used. Hereby, usually, a simulation model is applied (cf., e.g., Geiger & Uzsoy, 2008). Another frequently used offline learning method is (inductive) decision trees: cf., e.g., Kim et al. (1998), Koonce & Tsai (2000), Li & Olafsson (2005), or Olafsson & Li (2010). A review of the automated design of scheduling heuristics can be found in Branke et al. (2016).

For the (real-time) selection of scheduling methods in production environments with stochastic job arrivals, offline trained machine learning methods (mostly NN, but also Support vector machines, Inductive decision trees, Q-learning, or Deep Q networks) are used very frequently: cf., e.g., Priore et al. (2006). Mouelhi-Chibani & Pierreval (2010), Shiue et al. (2012), Priore et al. (2018), Stricker et al. (2018), or Waschneck et al. (2018). But also static environments are considered: cf., e.g., Azadeh et al. (2012), Azadeh et al. (2013), Shiue et al. (2018), or Lin et al. (2019). The approaches show that offline learning provides the possibility to reasonably decide in real-time on the most suitable scheduling method (often a dispatching rule in a stochastic, dynamic context) and that most often NNs are used.

The selection of scheduling methods is closely related to the prediction of objective values, i.e., the method(s) with the probably best performance will be selected. However, the purpose of a sole objective value prediction can be different. For instance, Gahm et al. (2022) use objective value predictions for the anticipation of base-level reactions in a hierarchical production planning system. In the literature, makespan predictions, which can be used for fast order acceptance decisions or delivery date confirmations, are very common (cf., e.g., Akyol, 2004, Raaymakers & Weijters, 2003 or Shafaei et al., 2011).

Parameter determination by machine learning is not as widespread in the literature as the approaches described so far.

The determination of the two look-ahead parameters of the ATCS rule ("Apparent Tardiness Cost with Setups"; cf., equation in section 3.1) by Machine learning was first proposed by Kim et al. (1995). Their offline approach uses NNs to estimate the look-ahead parameters κ_1 and κ_2 for solving the $S|s_{jk}|wT$-problem (s_{jk} indicates sequence-dependent setup times between jobs). Based on four features (number of jobs, due date tightness, due date range, and setup time severity), the NN performs a multi-target regression and predicts both parameters simultaneously. The mean percentage improvements compared to the ATCS with curve-fitted parameters are remarkable and range from 8.8 % to 22.2 %.

Park et al. (2000) investigate the $P|s_{jk}|wT$-problem and also use the ATCS rule for solving it. To determine the two look-ahead parameters, they use an offline trained NN with five features (due date tightness, due date range, setup time severity, job-machine factor, and setup time range). The first four features are also used by Lee et al. (1997) to develop an instance characteristics-related curve-fitted parameter function. In contrast to Kim et al. (1995), Park et al. perform single target regressions and predict κ_1 and κ_2 by two separately trained NNs. The data basis for the training consists of 4,320 samples resulting from the aggregation of 40,320 problem solutions (calculated by ATCS with a grid search for a given set of parameter combinations). Hereby, the parameters of all best solutions of a problem instance are aggregated by their mean values and further aggregated for all instances with identical factors (also by mean values). The evaluation shows that ATCS parametrized by the two NNs outperforms the approach of Lee et al. (1997) by 6 % on average.

To solve the $\widetilde{P}|pb,\ if^{*},\ b,\ p_j\ = p_f,\ r_j|wT$-problem, Mönch et al. (2006) compare the performance of NNs and inductive decision trees to estimate a single look-ahead parameter for their BATC dispatching heuristic. They use five features for their prediction: batch-machine factor (mean number of batches per machine), due date tightness, due date range, ready time tightness, and ready time range. Based on these features and a small training data set, the authors show that their offline learning approaches lead to high-quality schedules regarding total weighted tardiness.

For the $P|r_j,\ s_{jk}|wT$- problem, several adaptations of the ATCS and other rules incorporating job release times are presented by Pfund et al. (2008). The authors use a grid search to calculate multiple schedules and select the best. To reduce computational efforts, a regression analysis is performed offline to determine equations for each of the three used parameters. Their results show that computation times are remarkably reduced, but at the cost of a 9 % increase in total weighted tardiness.

A unified four-phase method to determine a single robust heuristic parameter setting for composite dispatching rules is proposed by Chen et al. (2010). Although their approach is not fully automated, it offline "learns" heuristic parameters based on calculated features (e.g., for the ATCS rule). A small validation study with the ATCS rule and the $P|\,s_{jk}|wT$-problem shows that the new approach outperforms the fitted parameter function proposed by Lee & Pinedo (1997). The advantage of the approach is that the parameters must not be recalculated if a new problem instance must be solved. However, the usage of identical parameters for all instances contradicts the finding that instance-specific parameters achieve the best solution quality (cf., e.g., Park et al., 2000 or Balasubramanian et al., 2004).

A cooperative dispatching rule for online decisions in a stochastic, dynamic flow shop ($F|\,dyn|T$; dyn signals that stochastic and dynamic job arrivals are considered) has been proposed by El-Bouri (2012). Their dispatching rule incorporates current information about downstream machines and uses a proximity factor to control the influence of this information on the incumbent dispatching decision. To determine the proximity factor, the authors use multiple regression models based on the job arrival rate and the due date tightness. The offline trained models can immediately provide appropriate parameters and the cooperative dispatching rule remarkably outperforms other dispatching rules from the literature.

Heger et al. (2016) propose a combined offline and online learning approach to dynamically adjust the two parameters of the ATCS dispatching rule for a stochastic, dynamic flow shop with sequence-dependent setup times ($F|\,dyn,\ s_{jk}|T$). Based on offline simulations, a Gaussian process regression model is trained and used online to estimate the best parameter settings. Their final simulation study shows that the approach achieves improvements of over 5 % compared to the best constant parameter settings.

To the best of our knowledge, only Shahrabi et al. (2017) address the learning-augmented parameter configuration of a metaheuristic. The authors use the Q-learning algorithm for learning the best parameters for their Variable neighborhood search developed to solve a stochastic, dynamic job shop problem ($J|\,dyn|F$). Hereby, the parameters are determined by a Reinforcement learning (RL) agent at any decision point.

The dynamic adjustment of the two ATCS parameters for the $HF|\,dyn|T$-problem is addressed by Heger & Voss (2021). The authors propose to use RL as an online hyper-heuristic. The RL agent is trained using a temporal difference method. The evaluation shows that dynamically adjusted parameters can reduce the mean tardiness by up to 5 %.

The analysis of learning-augmented scheduling methods concerning the parameter determination approach results in the following conclusions: first, instance-specific parameters outperform static parameters; second, offline-trained learning-augmented scheduling methods provide lower computation times (even enabling real-time decisions); Third, more sophisticated regression methods (e.g., NNs) outperform simple methods (e.g., linear regression).

2.3. Appropriate learning methods

The learning task at hand is a multi-target regression (also called multi-output, multi-variate, or multi-response regression) as the heuristic parameters to be predicted (the dependent variables) are continuous (cf., section 3.1). In contrast, in multi-class (multi-label) learning tasks a single discrete category (multiple discrete categories) must be identified (cf., e.g., Murphy, 2013 or Goodfellow et al., 2016). In addition, the heuristic parameters have interdependencies as can be easily seen by the parameters κ_1 and κ_2 from the ATCS rule described in equation (1) in section 3. In consequence, a multi-target regression with dependent (correlated) outputs must be performed and the learning task is called a "multi-target regression with dependencies" (MTRD).

Since the main purpose of the parameter prediction is to avoid a time-consuming full-grid search for identifying the most suitable parameter configuration(s), we propose using offline learning as learning mode.

To fulfill the MTRD task, one of the following learning paradigms could be used: Supervised learning (SL), Unsupervised learning (UL), or Reinforcement learning (RL) (or their combination; e.g., Deep learning; cf., e.g., Goodfellow et al., 2016). Because UL is usually used to discover structures in the data (e.g., clusters or patterns), UL is not applied to the learning task at hand. Since the considered batch scheduling problem is static and deterministic, RL cannot exploit its main advantage (to dynamically react on changing environments) and thus, the high computational efforts for training RL methods are not necessarily required. Therefore, we propose to use SL having a lower training effort due to efficient learning methods. Another reason for using SL is that we have information about appropriate labels (parameter values) and thus, training data for offline learning can be efficiently provided.

In their survey on multi-target regression, Borchani et al. (2015) differentiate between two main categories of multi-target machine learning methods: problem transformation methods and algorithmic adaptation methods. In problem transformation methods, multiple targets are independently predicted and either used directly (single-target method), combined in a stack (multi-target regressor stacking), or combined in regressor chains. Algorithmic adaptation methods extend single-output regression methods to handle multi-outputs and thus, are able to consider the dependency between the output variables. For instance, Support vector machine and Regression tree adaptations can be found in the literature (cf., e.g., Melki et al., 2017; for additional literature on multi-target regression, cf., e.g., Borchani et al., 2015, Spyromitros-Xioufis et al., 2016, Waegeman et al., 2019, Xu et al., 2020, or Rodríguez et al., 2021).

For the MTRD task, we decided to use NNs for several reasons: First, NNs are applicable for MTRD learning tasks without adaptations (cf., e.g., Spyromitros-Xioufis et al., 2016 or Reyes & Ventura, 2019); second, NNs have shown good performance for parameter prediction in the literature (cf., section 2.2); third, NNs are perfectly suited to be trained offline; and fourth, NNs are an SL method based on a parametric model with the advantage that they often have lower response times compared to non-parametric models (cf., Murphy, 2013). However, also other methods like Support vector machines or decision tree-based methods would be applicable. Furthermore, the application of RL methods where a problem instance is considered as a separate episode and the RL agent is trained by solving sample problem instances, could be possible. Hereby, the action taken by the agent may determine the heuristic parameters. Note, that – to the best of our knowledge – our learning-augmented scheduling methods are the first that use predictions for more than one parameter and that consider dependencies between the parameters.

3. Solution methods

3.1. Controlled batch utilization heuristic

The construction heuristics for our problem are based on the general priority rule dispatching procedure for batch scheduling problems as described in Gahm et al. (2021) and used by several authors (e.g., Balasubramanian et al., 2004, Almeder & Mönch, 2011, or Suppiah & Omar, 2014). The BATCS-b (originally abbreviated ATCS-BATCS(β)) heuristic proposed by Gahm et al. (2021) uses the ATCS rule (1) to determine job priorities $ATCS_j$ depending on the decision point t (i.e., the earliest latest batch completion time of all machines) and the family f of the latest scheduled batch of the corresponding machine:

$$ATCS_j(t,\ f) = \frac{w_j}{p_j}\cdot\exp\left(-\frac{\max\{d_j - p_j - t,\ 0\}}{\kappa_1\cdot\overline{p}_t}\right)\cdot\exp\left(-\frac{s_{f f_j}}{\kappa_2\cdot\overline{s}_t}\right) \tag{1}$$

with $\overline{p}_t$ = mean processing time of all unscheduled jobs, $\overline{s}_t$ = mean setup time between all families with unscheduled jobs, and f_j = family of job j. The two parameters κ_1 and κ_2 are so-called look-ahead parameters used for individually adjusting the "slack" component and the "setup avoidance" component separately. More information on the rationality of the ATCS rule can be found, for instance, in Vepsalainen & Morton (1987), Lee et al. (1997), or Lee & Pinedo (1997).

The individual job priorities $ATCS_j$ are then used to create a batch for each family: jobs are added to a batch in non-increasing order of their priorities (as long as the maximum batch capacity is not exceeded). Hereby, Gahm et al. introduce an additional parameter $\beta\in\mathbb{R}$ (with $0<\beta\leqslant 1$) to control the utilization of batches. This is done because the batch processing time depends on all assigned jobs and therefore, batches with a lower capacity utilization may result in lower batch processing times that can be suitable regarding the objective of minimizing total weighted tardiness. To guarantee that each job can still be assigned to a batch, the following formula is used to determine the "actualized" maximum batch capacity: $bc^{ACT} = \max\{\beta\cdot bc;\ \max\{cr_j|\forall j\in J\}\}$. After creating a batch for each family, batch priorities $\Pi_b(t,\ f) = \sum_{j\in B_b} ATCS_j(t,f)$ for all determined batches are calculated, and the batch with the highest priority is scheduled next.

Because the appropriate configuration of all three parameters κ_1, κ_2, and β is very important to calculate good solutions, Gahm et al. propose using a multi-start heuristic performing a grid search for applying different parameter configurations for κ_1, κ_2, and β: $\kappa_1\in \mathrm{K}_1 = \{0.5, 1.0, \ldots, 5.0\}\wedge\kappa_2\in K_2 = \{0.1, 0.2, \ldots, 1.6\}$ or $\kappa_1^*\wedge\kappa_2^*$ combined with $\beta\in \mathrm{B} = \{1.00, 0.95, \ldots, 0.50\}$. Here, κ_1^* and κ_2^* are calculated as defined in Appendix A-2 in Gahm et al., 2021 and originally proposed by Lee & Pinedo (1997). The complete parameter grid contains 1,771 parameter configurations $((|\mathrm{K}_1|\cdot|K_2|+1)\cdot|\mathrm{B}|=(10\cdot16+1)\cdot11)$, whereby each configuration defines values for κ_1, κ_2, and β. The experimental results have shown the superiority of this approach compared to other heuristics from the literature.

3.2. The new 'Controlled batch urgency heuristic'

In our new construction heuristic, we propose to improve the batching decision by controlling the jobs of a batch according to their "urgency" given by their ATCS priority $ATCS_j$ instead of influencing the batching decision by actualizing the maximum batch capacity. For this, we use a parameter $\delta\in\mathbb{R}$ (with $0<\delta\leqslant 1$). This parameter, together with the current maximum ATCS value of family f, defines a minimum ATCS value for family f: $ATCS_f^{MIN}(t) = ATCS_f^{MAX}(t)\cdot\delta$ (with $ATCS_f^{MAX}(t) = \max\{ATCS_j|\ \forall j\in\widetilde{I}_f(t)\}$ and set $\widetilde{I}_f(t)$ containing all unscheduled jobs of family f at decision point t). In consequence, only jobs with sufficiently high urgency, i.e., $ATCS_j\geqslant ATCS_f^{MIN}(t)$, are added to the current batch of a family f (at decision point t). Of course, the maximum batch capacity bc must be respected by the batching decision. According to the parameter δ, we name this heuristic "BATCS-d". Like for the BATCS-b heuristic, we propose a multi-start approach performing a grid search for applying different parameter configurations for κ_1, κ_2, and δ: $\kappa_1\in \mathrm{K}_1\wedge\kappa_2\in \mathrm{K}_2$ or $\kappa_1^*\wedge\kappa_2^*$ combined with $\delta\in\Delta = \{0.0,\ 0.1,\ \ldots,\ 0.9\}$. The resulting parameter grid contains 1,610 configurations $((|\mathrm{K}_1|\cdot|K_2|+1)\cdot|\Delta|=(10\cdot16\ +\ 1)\cdot10)$, whereby each configuration defines values for κ_1, κ_2, and δ. Note that $\delta = 0.0$ means that no restriction on the ATCS priority $ATCS_j$ is given and that we do not set $\delta = 1.0$ because this would most likely result in batches only containing a single job.

3.3. Improvement procedures

Improvement procedures are used to improve existing feasible solutions. Hereby, special focus is paid to the trade-off between solution quality and computation time.

3.3.1. Mixed-integer linear program

Based on the assumption that the previously described construction heuristics provide appropriate solutions, we use the mixed-integer linear program (MILP) proposed by Gahm et al. (2021) as an improvement procedure. The complete model is described in Appendix A-1. Compared to the model of Gahm et al., we slightly adapted constraint set (8) that now enables an arbitrary number of jobs in consecutive batches and allows the auxiliary variables to be continuous values (as the applied

Table 1
Gurobi parameter settings.

		τ [s]	*grb-gap*	*grb-mct* [s]	*grb-ovt* [s]	*grb-lbt* [s]	*grb-mrl*
Improvement	*GI1*	240	0.001 (0.1 %)	(τ - CT^{MSH}) • 0.5	30	30	1.0x10^{-4}
	GI2	240	0.001 (0.1 %)	(τ - CT^{MSH}) • 0.5	60	60	1.0x10^{-4}
Standalone	*GST*	3,600	0.001 (0.1 %)	τ• 0.5	180	180	1.0x10^{-4}

standard solver Gurobi 9.11 has shown to perform better with continuous auxiliary variables).

To apply Gurobi as an improvement procedure (abbreviated by GRB in the following) in an efficient manner, we not only use the basic termination criteria "maximum computation time" and/or "relative MIP gap" but introduce the additional criteria "minimum computation time", "maximum time without objective value improvement", and "maximum time without lower bound improvement". The first criterion "maximum computation time" terminates Gurobi after a given amount of time (τ). When using GRB as an improvement procedure, τ limits the computation time for the multi-start heuristic (CT^{MSH}) and the computation for Gurobi (CT^{GRB}), i.e., CT^{MSH} + CT$^{GRB}\leqslant \tau$. The second criterion terminates Gurobi if the incumbent relative gap (defined by Gurobi: MIPgap = |lb-bestOV| / (1.0x10^{-99} + |bestOV|)) is not greater than a given value (*grb-gap*). The third criterion guarantees a minimum computation time (*grb-mct*) and after that time, two additional criteria can lead to termination before τ is reached: first, if the objective value cannot be improved for a given time (*grb-ovt*), Gurobi is terminated; second, if the lower bound cannot be improved for a given minimum value *grb-mrl* ((lb^{NEW} - lb^{OLD}) / lb^{NEW} > *grb-mrl*) in a given time (*grb-lbt*), Gurobi is terminated. The additional termination criteria are introduced as we observed in preliminary tests that the best solutions are calculated relatively early during the optimization and that much time is spent to improve the lower bound, which is particularly difficult when total (weighted) tardiness is the objective. The first observation is corroborated by the computational results of Trindade et al. (2018) for a related problem. To use these additional termination criteria for the two heuristics BATCS-b-ML-GRB and BATCS-d-ML-GRB, we implement a callback and call Gurobi from Java.

Because GRB is used as an improvement procedure and as a standalone solution method, different termination criteria and parameters are used accordingly (cf., Table 1).

Further adjusted parameters of Gurobi are as follows (for explanations see the Gurobi documentation): *Threads* = 4, *ConcurrentMIP* = 1, *Method* = 4, *IntFeasTol* = 1.0x10^{-9}, *MIPGapAbs* = 0, and *IntegralityFocus* = 1.

3.3.2. Local search

The second proposed improvement procedure is a local search (LS) as described by Balasubramanian et al. (2004) and Gahm et al. (2021), enhanced by advanced termination criteria. The local search enhanced heuristic proposed by Gahm et al. (2021) is named BATCS-b-LS (instead of ATCS-BATCS(β)-LS) in the following and is used for comparisons in the experimental study (in addition to BATCS-b). The local search preserves the batch-to-machine assignment and the sequence of batches on the machines as we assume that the proposed construction heuristics generally make reasonable decisions based on the ATCS rule. To improve the constructed solutions, the local search solely swaps jobs between batches of the same family. Hereby, not only "late" jobs are considered because almost any job swapping influences batch completion times and thus, the completion time (tardiness) of all jobs in both batches (due to the serial batching). The swapping is repeated until no further improvements can be achieved, i.e., the swapping of any pair of jobs does not result in a lower objective value (for more details see Gahm et al., 2021).

To avoid exceeding computation times, Gahm et al. (2021) propose terminating the search if the relative improvement ($\Delta I = (ov^{BEST} - ov^{NEW})/ov^{BEST}$) is smaller than $1.0\text{x}10^{-4}$. We follow this approach but explicitly differentiate between weighted tardiness improvements and total flow time improvements. Accordingly, we introduce two parameters ε^{WT} and ε^{FT} (one for each objective criterion) that control the termination if the relative improvement is smaller than ε^{WT} or ε^{FT}, respectively. Furthermore, we also use the global time limit τ that terminates the search if the total computation time (including the time for the multi-start heuristic CT^{MSH} and the local search CT^{LS}) reaches τ (i.e., CT^{MSH} + CT^{LS} $\leqslant\tau$ must hold).

For defining the parameters ε^{WT}, ε^{FT}, and τ, we use three combined termination settings (cf., Table 2): The first setting *TS1* only uses the global time limit, which is set to 240 seconds. The time limit of 240 seconds is used because preliminary experiments have shown that the basic variants of the learning-augmented heuristics are capable to provide solutions within 120 seconds (cf., Table 10 and Table 11) and thus, sufficient time for improvement should be given by 240 seconds. 240 seconds (or four minutes) can also be seen as a reasonable time to solve problem instances with 3,200 jobs when applying the heuristics in a manufacturing company.

The second termination setting *TS2* uses the improvement termination criteria as proposed in the literature and dynamically adjusts τ depending on the number of jobs. The third setting *TS3* uses different values for ε^{WT} and ε^{FT} and dynamically adjusts τ.

In the experimental study, we not only investigate the performance of BATCS-b-LS but also the combination of BATCS-d with the local search (named BATCS-d-LS). To provide tight benchmarks for the learning-augmented heuristics, we use termination setting *TS4* (with large maximum computation times) for BATCS-b-LS and BATCS-d-LS in our experiments.

4. Learning-augmented heuristics: Parameter estimation by Neural networks

As discussed before, the multi-start heuristics performing full grid searches with many parameter configurations lack solution efficiency, i.e., achieve an appropriate solution quality but at the expense of high computation times (particularly for large-scale problem instances with hundreds or thousands of jobs; cf., Table 12 and Table 13 in Gahm et al., 2021). This is because the grid contains 1,771 parameter configurations for BATCS-b and 1,610 parameter configurations for BATCS-d. To

Table 2
Local search parameter settings.

TS1	$\tau = 240$			$\varepsilon^{WT} = 0$	$\varepsilon^{FT} = 0$
TS2	n < 100: $\tau = 60$	100 ≤ n < 600: $\tau = 120$	n ≥ 600: $\tau = 240$	ε^{WT}=1.0 × 10^{-4}	ε^{FT}=1.0 × 10^{-4}
TS3	n < 100: $\tau = 60$	100 ≤ n < 1,000: $\tau = 120$	n ≥ 1,000: $\tau = 180$	ε^{WT}=1.0 × 10^{-6}	ε^{FT}=1.0 × 10^{-3}
TS4	n ≤ 1,600: $\tau = 3{,}600$	n > 1,600: $\tau = 10{,}800$		$\varepsilon^{WT} = 0$	$\varepsilon^{FT} = 0$

improve solution efficiency, we propose using NNs to predict suitable parameter configurations (κ_1, κ_2, β and κ_1, κ_2, δ) for creating learning-augmented heuristics. The heuristics based on the learned parameters are named BATCS-b-ML and BATCS-d-ML in the following. The combination of these two basic variants with the GRB and LS improvement procedures results in four additional learning-augmented heuristics: BATCS-b-ML-GRB, BATCS-d-ML-GRB, BATCS-b-ML-LS, and BATCS-d-ML-LS.

To perform the parameter configuration prediction, we propose to use NNs because they can accomplish MTRD without adaptations. However, mostly independent of the specific SL method, several components of the "machine learning pipeline" must be determined (cf., e.g., Gahm et al., 2022). Since we have observed that the literature on learning-augmented scheduling methods often does not include all the information needed to implement the learning methods, we place special emphasis on the thorough description of the entire process, all required components (e.g., data acquisition and preparation, feature vector definition, dimension reduction, input scaling, output scaling, loss function, loss weighting, and hyper-parameter tuning procedures), and the use of the predicted parameter values. In this paper, a combination of components and their specification is called an "ML pipeline configuration" and is identified by PC_y.

4.1. Data acquisition and preparation

First, a sufficiently large sample data set X must be made available. To provide such a data set, we use 93,360 $\widetilde{P}|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$-problem instances (cf., section 5) and solve each with the two basic multi-start heuristics BATCS-b and BATCS-d. Due to the grid search with pre-defined discrete parameter values, it is generally possible that different parameter configurations achieve the best objective value for one problem instance and thus, a set of multiple best solutions results. In terms of machine learning, this results in samples with identical features (independent variables representing the problem instance) and different targets (dependent variables, i.e., the heuristic parameters). Since such (training) samples are usually not appropriate for standard regression models, we have to solve this so-called "identical features - different targets" problem (IF-DT), which – to the best of our knowledge – has not been discussed in the literature yet. For solving the IF-DT problem, we propose to aggregate the set of best solutions and their target values to a single sample per instance. Hereby, the procedure is designed such that the new target values can be used to calculate the best solutions if applied by the corresponding heuristic. To achieve this, simply calculating the mean value for every target dimension is not sufficient but a more sophisticated procedure is required. The aggregation procedure is described by the following pseudo-code:

createSampleTargets$(S = \{(\beta^1, \kappa_1^1, \kappa_2^1), (\beta^2, \kappa_1^2, \kappa_2^2), \ldots\})$.

$\overline{\beta} := \text{mean}\{\beta^1, \beta^2, \ldots\}$ // get mean of all β-values.

$\beta' := \underset{\beta \in B(S)}{\text{argmin}}\{|\beta - \overline{\beta}|\}$ // get the closest "original" value from all β in S to $\overline{\beta}$.

$\overline{\kappa}_1 := \text{mean}\{\kappa_1^1, \kappa_1^2, \ldots | \forall \kappa_1^z \in S\, with\, \beta = \beta'\}$ // get mean of all κ_1-values in all parameter combinations in S having $\beta = \beta'$

$\kappa_1' := \underset{\kappa_1 \in K_1(S)}{\text{argmin}}\{|\kappa_1 - \overline{\kappa}_1|\}$

$\overline{\kappa}_2 := \text{mean}\{\kappa_2^1, \kappa_2^2, \ldots | \forall \kappa_2^z \in S\, with\, \beta = \beta' \wedge \kappa_1 = \kappa_1'\}$

$\kappa_2' := \underset{\kappa_2 \in K_2(S)}{\text{argmin}}\{|\kappa_2 - \overline{\kappa}_2|\}$

return$(\beta', \kappa_1', \kappa_2')$.

Of course, the sample aggregation and all the training and parameter tuning described in the following have to be performed independently for both heuristics (BATCS-b-ML and BATCS-d-ML).

Note that the used sample data set and its (statistical) properties have an important influence on the generalization capabilities and thus, on the parameter estimation quality of an ML method in practical applications. To avoid so-called "sampling biases", the sample data set must be representative for the complete (diverse) instance space of real-world instances. For a more detailed discussion on generalization, we refer to Gahm et al. (2022). The authors also propose a prediction (process) framework that can be easily adapted to the prediction task at hand and that can be used to integrate new problem instances (with new properties) or to investigate other ML methods.

4.2. Feature engineering

The second pipeline component is the feature vector representing a problem instance. Here, we follow two basic approaches: the first one uses a small number of features based on complex instance characteristics (called "complex feature"-vector; CF) and the second one uses a large number of aggregated simple instance characteristics (called "aggregated feature"-vector; AF). The CF-vector consists of 12 features that are listed in Table 3 and partly taken or inspired by the literature (cf., Park et al., 2000; Mönch et al., 2006).

The AF-vector consists of 85 features. Five features are identical to the first five features of Table 3 (indicated by an asterisk) and the other 80 are calculated by eight basic instance data parameters and ten aggregation functions. The basic instance data parameters are the job processing times (p_j), job due dates (d_j), job weights (w_j), job capacity requirements (cr_j), job capacity requirement to processing time ratios (cr_j/p_j), job due date to weight ratios (d_j/w_j), setup times ($s_{f,g}$), and the number of jobs per family (n_f). The corresponding values of a single instance are aggregated to features by the following aggregation functions: *MIN*, *MAX*, *SUM*, *MED* (median), *VAR* (variance), *Q1* (first quartile), *Q3* (third quartile), *P10* (10 % percentile), *P90* (90 % percentile), and *SKEW* (Fisher-Pearson coefficient of skewness).

Table 3
Feature definitions of the CF-vector.

Feature	Description
n	*Number of jobs
m	*Number of identical parallel machines
q	*Number of incompatible job families
bc	*Maximum batch capacity
$\widetilde{M}$	*Approximated makespan (cf., (4))
$\widetilde{T}$	Tardiness factor: $\widetilde{T} = 1 - \overline{d}/\widetilde{M}$ (with the mean of all due dates $\overline{d} = \sum_{j\in J} d_j/n$); cf., e.g., Park et al. (2000) or Mönch et al. (2006).
$\widetilde{R}$	Due date range: $\widetilde{R} = (\max\{d_j \mid \forall j \in J\} - \min\{d_j \mid \forall j \in J\})/\widetilde{M}$; cf., e.g., Park et al. (2000).
$\widetilde{N}$	Approximated number of jobs per batch: $\widetilde{N} = bc/\left(\sum_{j\in J} cr_j/n\right)$
$\widetilde{B}$	Approximated number of batches: $\widetilde{B} = n/\widetilde{N}$
$\widetilde{F}$	Batch machine factor: $\widetilde{F} = \widetilde{B}/m$; cf., Mönch et al. (2006) for a similar feature and also the job machine factor used by Park et al. (2000).
$\widetilde{E}$	Setup time severity factor: $\widetilde{E} = \left(\sum_{f\in F}\sum_{f'\in F} s_{ff'}/q\right)/\widetilde{P}$ with the approximated mean batch processing time $\widetilde{P} = \widetilde{N}\cdot\left(\sum_{j\in J} p_j/n\right)$; cf., Park et al. (2000)
$\widetilde{S}$	Setup times range: $\widetilde{S} = (\max\{s_{ff'} \mid \forall f, f' \in F\} - \min\{s_{ff'} \mid \forall f, f' \in F\})/\max\{p_j \mid \forall j \in J\}$; cf., Park et al. (2000)

4.3. ML pipeline configurations

An optional component of a machine learning pipeline is “dimension reduction”. If the number of features is unnecessarily large (i.e., has more features than the variance in the data justifies) and the machine learning method is not capable to handle very large vectors (efficiently), dimension reduction can be used. Hereby, a most suitable trade-off between the benefits and drawbacks of more or fewer features must be considered (cf., Gahm et al., 2022). Because the CF-vector is already very small, we only use dimension reduction for the AF-vector. Hereby, we investigate if a dimension reduction by Principal Component Analysis (PCA) with a remaining variance of 0.98 is suitable or not.

A further component that is often used in a machine learning pipeline is “scaling mechanisms” as many methods require or perform better on scaled feature values. Therefore, we use a normalization (min-max) scaling between 0.0 and 1.0 for scaling all feature values (input scaling). Additionally, we investigate if the normalization of the outputs (output scaling) has a positive influence or not.

The component “loss function” is used to measure the prediction error during training, validation, and testing. Because larger errors are penalized stronger, we decided on the root mean squared error (RMSE). Equation (2) illustrates the RMSE for BATCS-b, sample data set X, and pipeline configuration PC_y (parameters with a circumflex indicate predictions and index s represents a single data sample):

$$RMSE(X,\ PC_y) = \sqrt{\frac{1}{|X|}\sum_{s\in X}(\beta^s - \widehat{\beta}^s)^2 + (\kappa_1^s - \widehat{\kappa}_1^s)^2 + (\kappa_2^s - \widehat{\kappa}_2^s)^2} \tag{2}$$

A natural and maybe more precise loss function to measure prediction errors would be using the heuristic with the estimated parameters and comparing the resulting objective values. However, this approach is not possible during training because NNs usually are based on the (stochastic) gradient descent algorithm (which is the de facto standard) that requires a differentiable function. Nevertheless, we use this approach in the validation and testing phase and compare the objective values achieved by the heuristics with the original grid search and the predicted parameters. In equation (2), the penalty terms for the three parameters have an identical influence on the overall error. To investigate if this assumption is correct, we slightly adapt (2) to analyze different loss weighting factors (ω, ω_1, and ω_2) controlling the influence of each term:

$$RMSE(X,PC_y,\omega,\omega_1,\omega_2) = \sqrt{\frac{1}{|X|}\sum_{s\in X}\omega(\beta^s - \widehat{\beta}^s)^2 + \omega_1(\kappa_1^s - \widehat{\kappa}_1^s)^2 + \omega_2(\kappa_2^s - \widehat{\kappa}_2^s)^2} \tag{3}$$

With that, it is also possible to include prior knowledge of the importance of the different parameters. Naturally, for BATCS-d-ML, all β must be replaced by δ in equation (3).

Because the estimation of heuristic parameters by NNs is not exhaustively studied in the literature, the most suitable components and parameters of the machine learning pipeline must be determined in addition to the standard hyper-parameter tuning of the NN. Table 4 depicts the 18 investigated pipeline configurations PC_1 to PC_{18}.

Because instance set *L* (cf., section 5) contains instances with all instance attribute combinations (cf., section 5) for $n \in\{100, 200, 400\}$, we are interested in whether the generalization capabilities of the NN trained with data sample *L* are sufficient for predictions with more or fewer jobs. Data sample *S-L-XL* contains all kinds of instances. The two discussed types of feature vectors are also analyzed, whereby PCA is only applied to the AF-vector as it comprises 85 features. In addition, output scaling and different loss weights are investigated.

Table 4
Machine learning pipeline configurations.

Pipeline configuration	Data sample	Feature vector	Dimension reduction	Output scaling (*osc*)	Loss weights (ω, ω_1, ω_2)
PC_1	*L*	*CF*			*1,1,1*
PC_2	*S-L-XL*	*CF*			*1,1,1*
PC_3	*L*	*CF*		*True*	*1,1,1*
PC_4	*S-L-XL*	*CF*		*True*	*1,1,1*
PC_5	*L*	*CF*			*2,3,1*
PC_6	*S-L-XL*	*CF*			*2,3,1*
PC_7	*L*	*AF*			*1,1,1*
PC_8	*S-L-XL*	*AF*			*1,1,1*
PC_9	*L*	*AF*		*True*	*1,1,1*
PC_{10}	*S-L-XL*	*AF*		*True*	*1,1,1*
PC_{11}	*L*	*AF*			*2,3,1*
PC_{12}	*S-L-XL*	*AF*			*2,3,1*
PC_{13}	*L*	*AF*	*PCA*		*1,1,1*
PC_{14}	*S-L-XL*	*AF*	*PCA*		*1,1,1*
PC_{15}	*L*	*AF*	*PCA*	*True*	*1,1,1*
PC_{16}	*S-L-XL*	*AF*	*PCA*	*True*	*1,1,1*
PC_{17}	*L*	*AF*	*PCA*		*2,3,1*
PC_{18}	*S-L-XL*	*AF*	*PCA*		*2,3,1*

4.4. Hyper-parameter tuning

The hyper-parameter tuning of the feedforward NN with back-propagation is based on the following parameters and value sets (values and basic settings have been identified by preliminary experiments): *number of neurons in the first hidden layer* $\in$\{64, 128, 256, 512, 1024\}, *number of neurons in the second hidden layer* $\in$\{64, 128, 256, 512, 1024\}, *drop-out rate* $\in$\{0.1, 0.2, 0.3\}, and *L2 regularization rate* $\in$\{0.001, 0.002, 0.01\}. The *learning rate* for the Adadelta Optimizer is fixed to 1.0 and the number of *epochs* is set to 50 for the hyper-parameter tuning and to 100 for all following trainings.

4.5. Use of the prediction

The response of the NN is a predicted parameter configuration either for BATCS-b-ML ($\widehat{\beta}$, $\widehat{\kappa}_1$, and $\widehat{\kappa}_2$) or for BATCS-d-ML ($\widehat{\delta}$, $\widehat{\kappa}_1$, and $\widehat{\kappa}_2$). Preliminary experiments have shown that the direct application of the predicted parameter configuration leads to a remarkably lower solution quality compared to the full grid search (11.4 percentage points for BATCS-b-ML and 15.0 percentage points for BATCS-d-ML; cf., Table 19 and Table 20 in Appendix A-2). Because the lower solution qualities are not acceptable from our point of view (even though computation times are lower than 17 seconds for every instance size) and because predictions are uncertain by definition, we do not only use the predicted values directly but also use them to define “reduced” parameter grids (compared to the original parameter grids with 1,771 and 1,610 configurations). The goal hereby is to determine grids that are much smaller than the original ones (to reduce computation times) but simultaneously achieve a comparable solution quality. To control this trade-off, we use a grid size parameter $\zeta \in \{1, 2, 3\}$ to select two, four, or six parameter values from the sets K_1, K_2, B, and Δ to determine parameter grids of different sizes. Fig. 2 illustrates the relationship between parameter prediction and different grid sizes depending on the parameter ζ (for a single sample instance). In Fig. 2, darker points mark better objective values.

Parameter values are selected in a way to balance greater and lower values compared to the predicted value as long minimum and maximum values of the sets are respected. For instance, if the predicted parameter values are $\widehat{\kappa}_1 = 1.13$, $\widehat{\kappa}_2 = 1.41$, and $\widehat{\beta} = 0.725$, the parameters

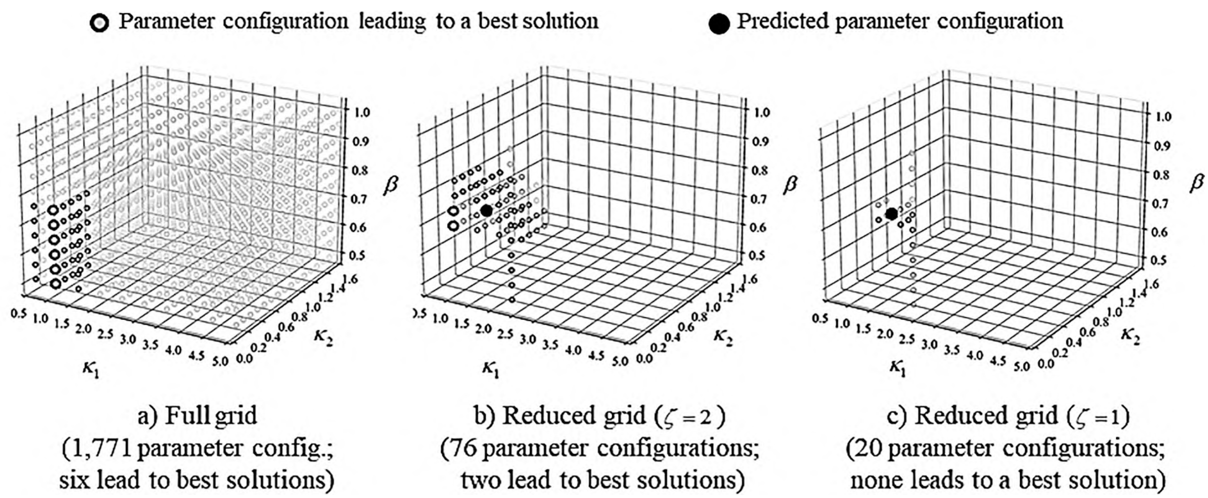

Fig. 2. Illustration of different grid sizes.

Table 5
Reduced grid sizes depending on the grid size parameter ζ.

ζ	κ_1 values with $\widehat{\kappa}_1 = 1.13$	κ_2 values with $\widehat{\kappa}_2 = 1.41$	β values with $\widehat{\beta} = 0.725$	Size of the reduced grid
1	1.0, 1.5	1.4, 1.5	0.75, 0.7	$1 + 2 \cdot 2 \cdot 2 + 11 = 20$
2	0.5, 1.0, 1.5, 2.0	1.3, 1.4, 1.5, 1.6	0.8, 0.75, 0.7, 0.65	$1 + 4 \cdot 4 \cdot 4 + 11 = 76$
3	0.5, 1.0, 1.5, 2.0, 2.5, 3.0	1.1, 1.2, 1.3, 1.4, 1.5, 1.6	0.85, 0.8, 0.75, 0.70, 0.65, 0.6	$1 + 6 \cdot 6 \cdot 6 + 11 = 228$

depicted in Table 5 are used to define the reduced parameter grids (depending on ζ).

Note that in the last column, 1 represents the predicted parameter configuration and 11 represents the combination of κ_1^* and κ_2^* (calculated as proposed by Lee & Pinedo, 1997) with the eleven β-values fromB. The predicted parameter configuration is added to the reduced grid because the preliminary tests have shown that the predictions returned by some pipeline configurations achieve the best objective values for some instances (cf., columns "Num. BEST" in Table 19 and Table 20 in Appendix A-2). The procedure to determine reduced grids for BATCS-d works accordingly and the concrete setting of ζ is part of the parameter tuning.

Besides the 18 pipeline configurations, we investigate three grid size parameter settings for determining the grid size parameter (cf., Table 6). With this, the trade-off between solution quality and computation time can be controlled: a higher ζ leads to a larger grid and thus can improve solution quality but increases computation time.

The first setting (*GS1*) defines a fixed ζ for all instances. The second setting (*GS2*) dynamically adjusts ζ depending on the number of jobs (related to the different instance sets described in section 5). Thereby, computation times for larger instances are reduced. Based on some initial experiments, we observed that for instances with 800 jobs, the solution quality decreases. In consequence, we define a third setting (*GS3*).

Table 6
Grid size parameter settings.

GS1	$\zeta = 2$					
GS2	$n < 100$:	$\zeta = 3$	$100 \leqslant n < 600$:	$\zeta = 2$	$n \geq 600$:	$\zeta = 1$
GS3	$n < 100$:	$\zeta = 3$	$100 \leqslant n < 1,000$:	$\zeta = 2$	$n \geq 1,000$:	$\zeta = 1$

5. Instance generation

To generate instances for evaluating the different solution methods (including the machine learning methods), we use the same procedure for instance generation as described by Gahm et al. (2021). Although the authors have made their instances available for download, we do not use these because we would like to additionally investigate different setup time characteristics (i.e., symmetric Euclidean, asymmetric Euclidean, and arbitrary setup times) and very large instances with more than 3,000 jobs. Therefore, to have a consistent data set, we decided to newly generate all instances. Altogether, we generate three sets of instances: small instances (*S*), large instances (*L*), and very large instances (*XL*). Distinguishing attributes of the instance sets *S*, *L*, and *XL* are the number of jobs n, the number of parallel machines m (with $m = 1$ representing the single machine case), and the number of incompatible job families q. Before describing the instance sets in detail, the procedure to determine the data of an instance and the used instance attributes and their values are depicted.

The maximum batch capacity bc is set to 50 for all instances. A job's batch capacity requirement cr_j is drawn from a discrete uniform distribution restricted by a specific capacity requirement scenario (*crs*) C1 to C4, defined by [1, 12], [1, 25], [1, 50], or [13, 38], respectively. The processing time p_j and the weight w_j of a job are drawn from discrete uniform distributions restricted by [1, 100] and [1, 10]. The job-to-family assignment (*jtfam*) follows two modes: either by a discrete uniform distribution (UD) or by a normal distribution (with a discretization function; ND). Note that with the ND-assignment mode, a few families contain most of the jobs and thus, the total setup time is likely to be smaller than with the UD assignment (because setup times between the same family are smaller than between different families).

The determination of the sequence-dependent setup times is based on the approximated mean batch capacity requirement $\overline{r} = \sum_{j \in J} cr_j / n$ (with the already drawn capacity requirements cr_j), the approximated mean number of jobs per batch $\widetilde{n}_b = \lceil bc / \overline{r} \rceil$, and the approximated mean processing time per batch $\widetilde{p}_b = \widetilde{n}_b \cdot \overline{p}$ (with the mean processing time $\overline{p} = \sum_{j \in J} p_j / n$ that is based on the already drawn processing times). In combination with the setup time severity factor η, a target mean setup time $\widetilde{s} = \eta \cdot \widetilde{p}_b$ is defined and consequently, setup times are drawn from a discrete uniform distribution restricted by $[1, \ 2\widetilde{s}]$. To generate the setup times, we first draw a set of setup times based on the given interval and the setup time type defines their number (i.e., for symmetric Euclidean setup times $(q^2+q)/2$ values and for asymmetric Euclidean and arbitrary setup times q^2 values must be drawn). After sorting these setup times in non-decreasing order, the first q smallest values define the setup time between batches of the same family f (s_{ff}) and the initial setup time of family f (s_{0f}). The remaining setup times of the sequence are then used to fill the setup time matrix row by row and column by column to obtain symmetric Euclidean (SE) setup times (i.e., setup times that respect the triangular inequality and $s_{ff'} = s_{f'f}$) and to obtain asymmetric Euclidean (AE) setup times (i.e., setup times that respect the triangular inequality and $s_{ff'} \neq s_{f'f}$). In the case of arbitrary setup times (AR), we shuffle the remaining setup times and then fill the setup time matrix in the same way. For instance, for $q = 3$, the randomly determined setup time sequences {29, 65, 75, 89, 90, 103}, and {29, 44, 65, 70, 75, 83, 89, 90, 103} lead to the following setup time matrices:

Symmetric Euclidean (SE)				Asymmetric Euclidean (AE)				Arbitrary (AR)			
$f=$	1	2	3	$f=$	1	2	3	$f=$	1	2	3
1	29	89	90	1	29	70	75	1	29	89	70
2	–	65	103	2	83	44	89	2	83	44	103
3	–	–	75	3	90	103	65	3	90	75	65

For determining appropriate due dates, we start with the estimation of the makespan $\widetilde{C}_{\max}$. With the approximated number of setups $tns = n/\widetilde{n}_b$ and the mean setup time $\bar{s}$ (after generating the actual setup times as described above), the total setup time is approximated by $tst = tns \cdot \bar{s}$ and leads to the following makespan approximation (cf., Gahm et al., 2021):

$$\widetilde{C}_{\max} = (n \cdot \overline{p} + tst)/m \tag{4}$$

This makespan approximation follows the idea of a makespan's lower bound (relaxation) by allowing job preemption (cf., e.g., Malve & Uzsoy, 2007). Based on $\widetilde{C}_{\max}$, we follow the broadly accepted procedure from the literature (cf., e.g., Balasubramanian et al., 2004) by determining due dates with regard to the attribute "tightness (tardiness) factor" tf (a higher value means stricter due dates) and "due date range factor" rdd (a higher value means a larger variance). Accordingly, due dates are randomly drawn from a discrete uniform distribution restricted by $[\max\{0,\ \lambda-(\lambda \cdot rdd/2)\},\ \lambda+(\lambda \cdot rdd/2)]$ with $\lambda = \widetilde{C}_{\max} \cdot (1-tf)$. The maximum function of the interval's lower bound guarantees due dates not lower than zero.

In Table 7, the instance attributes that are identical for all instance sets are depicted.

Instance set S contains instances with $n \in${15, 30, 60}, $m \in${1, 3, 4, 5}, $q \in${3, 5, 10}, and $crs \in${C1, C2, C3, C4}. As discussed in Gahm et al. (2021), not all of these attribute combinations are reasonable and thus, we restrict S to the same 31 combinations of n, m, and crs as depicted in Table 5 in Gahm et al. (2021). Combining these 31 combinations with the three values for q and the 48 parameter combinations listed in Table 7 leads to 4,464 attribute combinations in set S. For each combination, we randomly generate five instances and thus, instance set S contains 22,320 instances.

For generating instance set L, we use the attributes $n \in${100, 200, 400}, $m \in${1, 3, 4, 5, 10}, $q \in${3, 5, 10, 20}, and $crs \in${C1, C2, C3, C4} in combination with those depicted in Table 7. Accordingly, we have 11,520 attribute combinations and thus, 57,600 instances in set L (five instances for each attribute combination). Note that we do not have any

Table 7
Common instance attributes.

Parameter	Values	Number of values
Job-to-family assignment mode (*jtfam*)	UD, ND	2
Setup time severity (η)	0.25, 0.75	2
Setup time type (*st*)	SE, AE, AR	3
Tardiness factor (*tf*)	0.3, 0.6	2
Due date range (*rdd*)	0.25, 0.75	2
		$\prod$ 48

Table 8
Specific instance parameters of set *XL*.

n	m	q
=800	∈{5, 10, 20}	∈{10, 20, 40}
=1,600	∈{10, 20}	∈{20, 40}
=3,200	=20	=40

restrictions regarding appropriate combinations for set L.

To keep the overall computation times in a reasonable time range, we use special attribute combinations for instance set XL: Here, we use the attributes depicted in Table 7, $crs \in${C1, C2, C3, C4}, and the combinations of n, m, and q as depicted in Table 8. Following these settings, we have 2,688 combinations and 13,440 instances in set XL (five instances for each attribute combination).

Summarizing, our experiments base on three instance sets S, L, and XL containing 93,360 instances in total ($|S|$=22,320, $|L|$=57,600, and $|XL|$=13,440). This large set of instances is required as we not only evaluate different solution methods but also need a sufficiently large base of samples for the machine learning.

To make the following results of our experiments traceable and reproduceable, and to enable future research on scheduling methods and prediction methods, we provide two data sets for download on Mendeley data: The first data set (Gahm, 2022a) contains the complete scheduling data including the objective values of the developed solution methods. The second data set (Gahm, 2022b) provides the complete data for the machine learning (e.g., feature vectors and labels).

6. Experimental study

To ensure an unbiased evaluation of the learning-augmented heuristics, we split the complete data set of problem instances for the four phases of our experimental study into the sets D^H, D^T, D^{V80}, and D^E (cf., Fig. 3). For getting a stratified sampling, we allocate the five instances that have been generated for each instance attribute combination to different sets used by the four phases. For the NN hyper-parameter tuning performed with each pipeline configuration, set D^H contains the first two instances ($|D^H|$=37,344). The training of the tuned pipeline configurations is executed with instance set D^T containing the first three instances of each instance attribute combination $|D^T| = 56{,}016$.

To validate the best NN learning pipeline and the tuning of the heuristic parameters, only the respective fourth instances are used. To keep computational efforts at a manageable level, we only use 80 randomly selected instances from the fourth instances to create set D^{V80}. D^{V80} contains ten instances for each $n \in${30, 60, 100, 200, 400, 800, 1,600, 3,200} (because computation times with 15-job instances are lower than 50 milliseconds and thus biased by operating system activities, we do not consider such instances during parameter tuning). In this phase, not the loss function defined by equation (3) is used but the predicted heuristic parameters are applied within the heuristics and the objective function (5) is used as "loss function". For the final testing and evaluation of the tuned learning augmented heuristics, only the respective fifth instances are used and combined to set D^E with 18,672 instances. In this way, we can guarantee that no instance is used in phase IV that was already part of the development and tuning process.

All heuristics and a tool for the management of the experiments are implemented in Java 10 and the standard solver Gurobi version 9.11 is used. For the machine learning, "Keras" (keras.io), "TensorFlow" (tensorflow.org), and "scikit-learn" (Pedregosa et al., 2011) are used. The machine learning and all experiments have been executed on workstations with an Intel® XEON® CPU E5-2690 with 3.0 GHz and 64 GB RAM.

To reduce computation times, we use a task-parallel (also called result-parallel; cf., e.g., Kaminsky, 2010) implementation to perform several "starts" of the multi-start heuristics on parallel threads of a

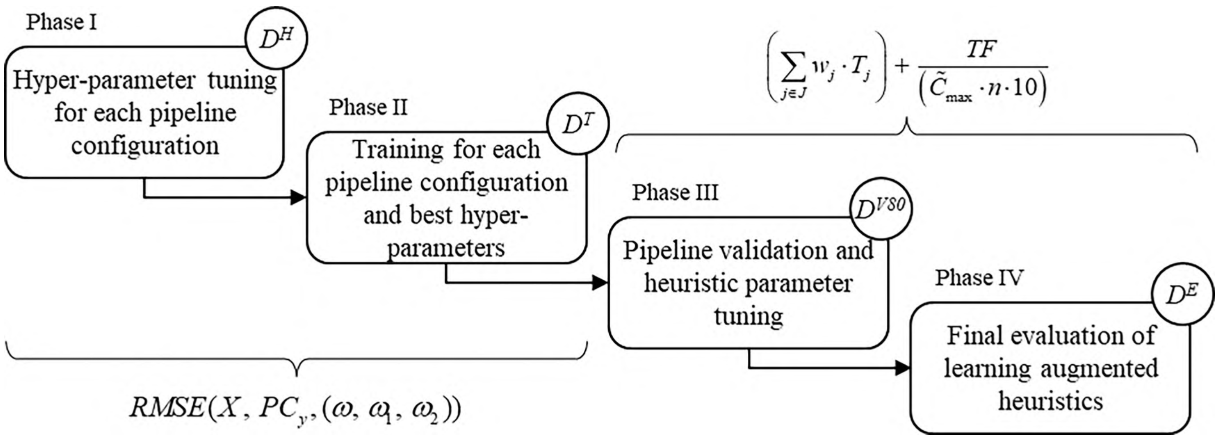

Fig. 3. Parameter tuning and evaluation process.

shared memory multiprocessor. The number of parallel threads (*th*) is set to four for all experiments. The only exceptions to this setting are the experiments to investigate the performance gain of the parallelization (cf., section 6.5). To implement the task-parallel execution, we use the "Parallel Java 2 PJ2" API and middleware (available at jimihford.github.io/pj2/).

6.1. Key figures

To assess the solution quality, we use the key figure "mean relative improvement versus the worst objective function value" (MRIW; cf., Valente & Schaller, 2012; Gahm et al., 2021). The MRIW is used since objective function values could be close to zero for some instances and thus, relative improvements versus the best objective function value are troublesome. The MRIW is specified as follows: For a given problem instance and objective function, the relative improvement versus the worst result for a solution method M_s (with $s = 1, \ldots, z$) is indicated by RIW_s. Let v_{best} and v_{worst} indicate the best (minimum) and worst (maximum) objective function value of all z solution methods (regarding a single problem instance and a single objective function). If $v_{best} = v_{worst}$, then RIW_s is set to zero. Otherwise, $RIW_s = |(v_{worst} - v_s)/v_{worst} \cdot 100|$, with v_s being the objective function value of solution method M_s. Based on this definition of RIW_s, a mean RIW (MRIW) for each solution method regarding a specific (sub-)set of instances (e.g., sets D^H, D^T, D^{V80}, and D^E) and solution method (sub-)set can be calculated.

In addition to the MRIW, we use the key figure S.BEST to indicate the number of "single-best" solutions calculated by one of the solution methods. "Single-best" means that no other solution method has also achieved the best (minimum) objective value for a specific instance.

The key figure computation time (CT) indicates the total runtime of a solution method and includes the following components (depending on the specific solution method): feature vector computation time (i.e., the time needed to compute the features for the machine learning model), prediction response time (i.e., the time needed by the machine learning model to return a prediction), heuristic computation time, Gurobi optimization time, and local search time.

6.2. Parameter tuning

The purpose of parameter tuning for heuristics is to identify the most suitable trade-off between solution quality and computation time. The tuning of the ML-pipeline configuration, grid size, Gurobi, and the local search parameter settings are based on 80 randomly selected instances from instance set D^V having a different number of jobs: 10 instances for each $n \in \{30, 60, 100, 200, 400, 800, 1{,}600, 3{,}200\}$. Because computation times with 15-job instances are lower than 50 milliseconds and thus biased by operating system activities, we do not consider such instances during parameter tuning. The full factorial experiment design is depicted in Table 9.

To simplify traceability, we only present summarizing tables that substantiate our conclusions in the following. As we are interested in the most suitable efficiency trade-off, all tables depict mean RIWs and mean CTs related to the number of jobs (as n is most responsible for the computation time).

Table 9
Experiment design of the parameter tuning.

Heuristic	Num. of pipeline config.	Num. of grid size parameter settings	Num. of LS parameter settings	Num. GRB parameter settings	Num. of comb.	Num. of instances	Num. of exp.
BATCS-b		–	–	–	1	80	80
BATCS-d		–	–	–	1	80	80
BATCS-b-ML	18	3	–	–	54	80	4,320
BATCS-d-ML	18	3	–	–	54	80	4,320
BATCS-b-ML-GRB	18	3	–	2	108	30	3,240
BATCS-d-ML-GRB	18	3	–	2	108	30	3,240
BATCS-b-ML-LS	18	3	3	–	162	80	12,960
BATCS-d-ML-LS	18	3	3	–	162	80	12,960
SUM					650		41,200

6.2.1. Machine learning

The results of the hyper-parameter tuning, the pipeline configuration, and the grid size parameter setting analysis (Phase I, II, and III in Fig. 3) are summarized in Table 10 for BATCS-b-ML and Table 11 for BATCS-d-ML (the results for all pipeline configurations are depicted in Table 21 and Table 22 in Appendix A-3). Note that the MRIWs in the following result tables are related to the specified set of instances and the depicted solution methods.

Based on the parameter tuning results for BATCS-b-ML depicted in Table 10, we conclude that ML-pipeline configuration PC_8 combined with grid size parameter setting *GS3* provides the best efficiency trade-off. With these settings, the solution quality decreases only by 0.9 percentage points (compared to BATCS-b) but computation times can be reduced by 94.3 % (on average; compared to BATCS-b). Note that the small deviations in the computation times between GS2 and GS3 for the same number of jobs (except $n = 800$) result from rounding, unit scaling to minutes, and biases due to operating system activities.

The results of the pipeline configuration and grid size parameter setting analysis for BATCS-d-ML are presented in Table 11.

For BATCS-d-ML, we conclude that ML-pipeline configuration PC_6 combined with grid size parameter setting *GS3* provides the best efficiency trade-off. With these settings, the solution quality can even be improved by 1.0 percentage points (compared to BATCS-d) and computation times can be reduced by 93.7 % (on average; compared to BATCS-d). The improvement is traced back to instances for which the predicted parameter configuration leads to the overall best solution.

Summarizing the ML-pipeline configuration analysis, we can report that the complete set of instances (*S-L-XL*) is required to achieve a suitable generalization by the NN, that both feature vectors have their justification, that dimension reduction is not suitable for the AF feature vector, and that output scaling and loss weighting can be helpful.

The best NN hyper-parameters for BATCS-b-ML(PC_8) with the pipeline setting *S-L-XL*, *AF*, and *(1,1,1)* are as follows: *number of neurons in the first hidden layer* = 64, *number of neurons in the second hidden layer* = 64, *drop-out rate* = 0.1, and *L2 regularization rate* = 0.001. For BATCS-d-ML(PC_6) with *S-L-XL*, *CF*, and *(2,3,1)*, the best NN hyper-parameters are the following: *number of neurons in the first hidden layer* = 512, *number of neurons in the second hidden layer* = 64, *drop-out rate* = 0.1, and *L2 regularization rate* = 0.001.

Based on the previous results, we only depict those parameter tuning results that are related to the most suitable machine learning parameters in the following.

6.2.2. Gurobi

For tuning the GRB termination criteria, we only use instances with up to 100 jobs because model preparation and root relaxation solving for instances with 200 jobs require longer than the available 240 seconds (limited by the global time limit).

The results in Table 12 show that for BATCS-b-ML(PC_8, *GS3*)-GRB less restrictive termination criteria defined by *GI2* are required for a high

Table 10
Grid size parameter settings analysis for BATCS-b-ML (best ML-pipelines).

	MRIW (D^{V80})									Mean CT							
n	30	60	100	200	400	800	1,600	3,200	MEAN	30	60	100	200	400	800	1,600	3,200
BATCS-b										[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
	25.0	23.8	21.5	13.0	6.0	13.7	5.3	4.6	14.1	1.5	1.6	4.1	12.8	42.4	6.1	25.3	131.5
BATCS-b-ML(PC_4) with *S-L-XL, CF, osc*, and *(1,1,1)*																	
GS1	19.8	19.3	19.1	5.8	3.9	14.8	11.8	4.7	12.4	0.1	0.1	0.3	0.6	1.7	0.3	1.1	5.5
GS2	21.1	20.2	19.1	5.8	3.9	14.1	10.7	4.7	12.4	0.1	0.2	0.3	0.5	1.3	0.1	0.3	1.5
GS3	21.1	20.2	19.1	5.8	3.9	14.8	10.7	4.7	12.5	0.1	0.2	0.3	0.5	1.4	0.2	0.3	1.5
BATCS-b-ML(PC_8) with *S-L-XL, AF*, and *(1,1,1)*																	
GS1	23.7	15.8	19.8	6.9	8.5	15.2	10.9	3.7	13.0	0.1	0.1	0.3	0.6	1.4	0.2	0.9	5.2
GS2	25.0	19.7	19.8	6.9	8.5	9.8	8.5	2.3	12.5	0.1	0.2	0.2	0.5	1.4	0.1	0.2	1.5
GS3	**25.0**	**19.7**	**19.8**	**6.9**	**8.5**	**15.2**	**8.5**	**2.3**	**13.2**	**0.1**	**0.2**	**0.3**	**0.5**	**1.3**	**0.2**	**0.3**	**1.6**
BATCS-b-ML(PC_{10}) with *S-L-XL, AF, osc*, and *(1,1,1)*																	
GS1	19.1	15.4	18.8	7.9	10.8	15.6	6.1	2.1	12.0	0.1	0.1	0.2	0.5	1.3	0.2	0.8	5.0
GS2	21.1	19.9	18.8	7.9	10.8	11.6	6.0	1.3	12.2	0.1	0.2	0.3	0.5	1.3	0.1	0.3	1.5
GS3	21.1	19.9	18.8	7.9	10.8	15.6	6.0	1.3	12.7	0.1	0.2	0.3	0.5	1.4	0.3	0.3	1.5

Table 11
Grid size parameter settings analysis for BATCS-d-ML (best ML-pipelines).

	MRIW (D^{V80})									Mean CT							
n	30	60	100	200	400	800	1,600	6,200	MEAN	30	60	100	200	400	800	1,600	3,200
BATCS-d										[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
	46.5	34.8	38.5	54.5	31.0	60.9	48.0	57.4	46.4	0.8	1.3	4.7	14.3	42.2	6.3	28.2	169.4
BATCS-d-ML(PC_1) with *L, CF*, and *(1,1,1)*																	
GS1	30.0	32.9	36.4	51.7	39.1	52.7	45.7	53.1	42.7	0.1	0.1	0.3	0.6	1.5	0.3	1.2	6.6
GS2	38.3	34.2	36.4	51.7	39.1	50.3	49.0	52.3	43.9	0.1	0.2	0.3	0.6	1.6	0.1	0.4	1.8
GS3	38.3	34.2	36.4	51.7	39.1	52.7	49.0	52.3	44.2	0.1	0.2	0.3	0.6	1.5	0.2	0.3	1.6
BATCS-d-ML(PC_4) with *S-L-XL, CF, osc*, and *(1,1,1)*																	
GS1	36.3	31.1	36.0	52.4	32.2	59.9	51.0	55.3	44.3	0.1	0.1	0.3	0.6	1.6	0.2	1.3	7.6
GS2	39.4	34.7	36.0	52.4	32.2	57.7	54.1	54.2	45.1	0.1	0.2	0.3	0.6	1.6	0.1	0.3	2.0
GS3	39.4	34.7	36.0	52.4	32.2	59.9	54.1	54.2	45.3	0.1	0.2	0.3	0.6	1.6	0.2	0.3	2.1
BATCS-d-ML(PC_6) with *S-L-XL, CF*, and *(2,3,1)*																	
GS1	32.1	27.1	38.5	51.1	44.0	59.6	54.2	57.0	45.5	0.1	0.1	0.3	0.6	2.0	0.3	1.5	7.7
GS2	42.0	34.6	38.5	51.1	44.0	58.0	52.9	56.2	47.2	0.1	0.2	0.3	0.6	1.5	0.1	0.4	2.1
GS3	**42.0**	**34.6**	**38.5**	**51.1**	**44.0**	**59.6**	**52.9**	**56.2**	**47.4**	**0.1**	**0.2**	**0.3**	**0.7**	**1.5**	**0.2**	**0.3**	**2.0**
BATCS-d-ML(PC_{10}) with *S-L-XL, AF, osc*, and *(1,1,1)*																	
GS1	37.1	23.7	36.1	49.8	30.9	59.0	47.1	56.0	42.5	0.1	0.1	0.3	0.6	1.6	0.2	1.3	7.5
GS2	42.9	33.0	36.1	49.8	30.9	55.7	49.5	54.5	44.0	0.1	0.2	0.3	0.6	1.5	0.1	0.3	2.0
GS3	42.9	33.0	36.1	49.8	30.9	59.0	49.5	54.5	44.4	0.1	0.2	0.3	0.6	1.6	0.3	0.4	2.0

Table 12
GRB-termination parameter analysis.

	MRIW (D^{V80})				Mean CT [s]			Mean rel. best OV CT [%]		
	30	60	100	MEAN	30	60	100	30	60	100
BATCS-b-ML(PC_8, *GS3*)-GRB										
GI1	45.3	34.7	28.3	36.1	171.6	136.9	164.5	80.3	79.1	67.1
GI2	**46.2**	**36.4**	**29.3**	**37.3**	**211.6**	**205.9**	**223.6**	**79.7**	**73.9**	**66.6**
BATCS-d-ML(PC_6, *GS3*)-GRB										
GI1	**51.9**	**42.5**	**47.4**	**47.3**	**146.4**	**167.4**	**175.4**	**58.7**	**75.4**	**71.1**
GI2	51.9	42.6	47.4	47.3	184.3	212.4	226.3	56.8	75.0	61.1

Table 13
LS-termination parameter analysis.

	MRIW (D^{V80})									Mean CT							
n	30	60	100	200	400	800	1,600	3,200	MEAN	30	60	100	200	400	800	1,600	3,200
BATCS-b-ML(PC_8, *GS3*)-LS										[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
TS1	27.9	32.0	32.1	31.6	37.9	42.1	32.6	36.3	34.1	0.1	0.3	0.8	13.6	58.9	2.0	3.1	4.0
TS2	27.9	32.0	32.1	31.5	37.8	42.0	33.6	37.2	34.2	0.1	0.2	0.7	8.4	34.0	1.7	3.0	4.0
TS3	**27.9**	**31.9**	**32.1**	**31.3**	**37.2**	**39.2**	**31.7**	**32.9**	**33.0**	**0.1**	**0.3**	**0.8**	**4.4**	**15.1**	**1.1**	**2.3**	**3.0**
BATCS-d-ML(PC_6, *GS3*)-LS										[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
TS1	46.7	39.5	44.6	57.4	59.0	65.0	57.5	71.7	55.2	0.1	0.3	0.7	3.0	43.3	1.5	2.6	4.0
TS2	46.7	39.5	44.6	56.8	58.5	64.7	58.2	71.8	55.1	0.1	0.2	0.7	2.2	23.0	1.1	2.7	4.0
TS3	**46.7**	**39.4**	**44.6**	**57.0**	**57.9**	**63.7**	**57.3**	**70.8**	**54.7**	**0.1**	**0.2**	**0.6**	**1.7**	**14.0**	**0.9**	**2.1**	**3.1**

solution quality, whereas BATCS-d-ML(PC_6, *GS3*)-GRB achieves with *GI1* the same solution quality as with *GI2* but with lower computation times. The values in the last three columns indicate that the best objective values are not achieved at the end of the optimization but between 56.8 % and 80.3 % of the total Gurobi computation time. This underpins the assumption underlying the proposed GRB termination criteria.

6.2.3. Local search

For the tuning of the LS termination parameters, Table 13 presents the MRIW and the mean CT regarding BATCS-b-ML(PC_8, *GS3*) and BATCS-d-ML(PC_6, *GS3*), respectively.

From the results in Table 13, we conclude that the LS can remarkably improve the solution quality and that the termination setting *TS3* (with the adjusted global time limit of 180 seconds and different values for ε^{WT} and ε^{FT}) achieves the best efficiency trade-off.

6.3. Problem-solving complexity

To analyze the influence of instance attributes on the solving complexity, we depict the relative number of instances that could be solved "optimally", i.e., with a MIPgap < 0.05, in Table 14. The problem of assessing the solving complexity based on the MIPgap lies in the fact that this value not only depends on the objective value but also on the lower bound, and that in most cases, lower bounds for tardiness-oriented objective functions have poor quality. Nevertheless, we assume that it can be used as an indicator for solving complexity. In the experiments, we use all instances from set D^E with a number of jobs not greater than 100. Note that GRB as a standalone solution method is not able to solve instances with 100 jobs within the time limit of 3,600 seconds and thus, the corresponding results are not depicted in Table 14.

In Table 14, the attributes with the highest influence on the solving complexity are marked bold. The tardiness factor *tf* and the due date range factor *rdd* have a strong influence on the solving complexity. This can be explained by the effect that lower values offer a higher possibility to result in a tardiness close to zero and thus, better lower bounds can be calculated. Because capacity requirements scenario C1 leads to "smaller" jobs and thus in a larger number of possible batch combinations, C1 shows the highest solving complexity (compared to C2, C3, and C4). A lower setup time severity leads to lower setup times and thus, a larger number of batches can be suitable (that in turn increases problem-solving complexity). Somehow surprising is the result that asymmetric Euclidean (AE) setup times result in a higher problem-solving complexity.

Table 14
Relative number of instances solved with a MIPgap < 0.05 (with D^E).

	Number of instances	GRB	Number of instances	BATCS-b-ML(PC_8, *GS3*)-GRB (*GI2*)	BATCS-d-ML(PC_6, *GS3*)-GRB (*GI1*)
$m = 1$	1,728	9.8 %	2,496	14.3 %	15.4 %
$m = 3$	1,296	10.1 %	2,064	14.0 %	16.2 %
$m = 4$	720	10.0 %	1,488	13.6 %	16.0 %
$m = 5$	720	8.8 %	1,488	13.4 %	15.0 %
$m = 10$	–	–	768	5.1 %	5.5 %
$n = 15$	864	13.5 %	864	12.2 %	12.3 %
$n = 30$	1,584	14.1 %	1,584	17.9 %	19.9 %
$n = 60$	2,016	4.8 %	2,016	13.6 %	15.7 %
$n = 100$	–	–	3,840	11.0 %	12.6 %
CRS = C1	**576**	**1.9 %**	**1,536**	**4.1 %**	**3.8 %**
CRS = C2	1,008	6.1 %	1,968	9.6 %	11.3 %
CRS = C3	1,440	12.5 %	2,400	16.8 %	19.7 %
CRS = C4	1,440	12.8 %	2,400	18.0 %	19.5 %
$q = 3$	1,488	11.4 %	2,448	14.3 %	16.5 %
$q = 5$	1,488	10.3 %	2,448	14.9 %	16.4 %
$q = 10$	1,488	7.6 %	2,448	12.4 %	13.6 %
$q = 20$	–	–	960	7.3 %	8.6 %
jtfam = DND	2,232	10.9 %	4,152	13.5 %	15.2 %
jtfam = DUD	2,232	8.6 %	4,152	12.7 %	14.2 %
$\boldsymbol{\eta = 0.25}$	**2,232**	**4.9 %**	**4,152**	**6.9 %**	**8.5 %**
$\eta = 0.75$	2,232	14.6 %	4,152	19.3 %	20.9 %
st = AR	1,488	11.0 %	2,768	13.8 %	15.9 %
***st* = AE**	**1,488**	**4.3 %**	**2,768**	**6.4 %**	**7.7 %**
st = SE	1,488	14.0 %	2,768	19.1 %	20.7 %
tf = 0.3	2,232	18.8 %	4,152	25.6 %	28.9 %
***tf* = 0.6**	**2,232**	**0.8 %**	**4,152**	**0.6 %**	**0.6 %**
rdd = 0.5	2,232	10.2 %	4,152	18.9 %	20.0 %
***rdd* = 2.5**	**2,232**	**9.4 %**	**4,152**	**7.3 %**	**9.4 %**

Table 15
Mean CT [s] (aggregated by number of jobs, with D^E).

n	GRB (*GST*) [m]	BATCS-b					BATCS-d				
			-LS (*TS4*)	-ML(PC_8, *GS3*)				-LS (*TS4*)	-ML(PC_6, *GS3*)		
					-LS (*TS3*)	-GRB (*GI2*)				-LS (*TS3*)	-GRB (*GI1*)
15	30.6	0.3	5.4	0.1	0.1	134.4	0.2	2.2	0.1	0.1	123.5
30	31.1	0.7	2.8	0.1	0.1	190.3	0.4	1.8	0.1	0.1	142.0
60	40.7	1.7	2.8	0.2	0.3	214.8	1.1	2.3	0.2	0.3	155.9
100	–	4.4	5.0	0.2	0.5	212.5	3.9	4.9	0.3	0.4	166.6
200	–	12.8	17.9	0.5	4.4	–	13.8	19.2	0.5	2.6	–
400	–	40.8	175.5	1.3	33.6	–	49.7	156.0	1.5	24.3	–
800	–	323.9	934.2	8.6	75.0	–	410.3	721.5	10.8	64.6	–
1,600	–	1,300.9	2,519.2	16.3	149.7	–	2,045.2	2,626.3	18.1	131.9	–
3,200	–	7,291.6	10,387.2	80.8	183.3	–	6,957.1	10,825.0	94.2	180.9	–

6.4. Solution method evaluation

To evaluate the solution efficiency of the solution methods GRB, BATCS-b, BATCS-d, BATCS-b-LS, BATCS-d-LS, and the six learning-augmented heuristics, we depict CT, MRIW, and S.BEST in the following tables (Table 15, Table 16, and Table 17, respectively). The “-“ in some cells of the tables indicate that the corresponding solution method was not suitably applicable to instances with the depicted number of jobs. Bold values in the tables indicate the best key figure values per row.

The computation times in Table 15 and the MRIWs in Table 16 show that the proposed learning-augmented heuristics achieve a remarkably good solution efficiency.

The application of reduced grids based on estimated parameters leads to high computation time reductions and a very competitive solution quality that becomes even better for some instance subsets.

The results in Table 16 show that the heuristics with “controlled batch urgency” outperform the heuristics with “controlled batch utilization” in terms of the mean solution quality. It can also be seen that with a number of jobs larger than 30, GRB as a standalone solution method is no longer appropriate and that the BATCS-d-ML-GRB is able to provide the best solutions for instances with up to 100 jobs (on average, 15 % can be solved with a MIPgap lower than 0.05).

Regarding the overall solution quality, BATCS-d-LS and BATCS-d-ML-LS achieve the highest MRIWs of all solution methods capable to solve instances with an arbitrary number of jobs (on average, the MRIWs are 46.2 and 45.47, respectively). This result is all the more impressive when considering that the mean computation times of BATCS-d-ML-LS are an order of magnitude lower than that of BATCS-d-LS.

In Table 17, the single best solutions (S.BEST) per solution methods are depicted (note that BATCS-b, BATCS-b-ML, BATCS-d, and BATCS-d-ML are not listed as they cannot have any single best solutions).

Analyzing S.BEST, we can see that BATCS-d-LS has the highest score, but we can also see that the more efficient heuristic BATCS-d-ML-LS has also a high S.BEST value. However, the mean absolute deviation of the RIWs of BATCS-d-ML-LS and BATCS-d-LS (regarding all those instances

Table 16
Mean RIWs [%] (aggregated by number of jobs, with D^E).

n	GRB (*GST*)	BATCS-b					BATCS-d				
			-LS (*TS4*)	-ML(PC_8, *GS3*)				-LS (*TS4*)	-ML(PC_6, *GS3*)		
					-LS (*TS3*)	-GRB (*GI2*)				-LS (*TS3*)	-GRB (*GI1*)
15	**30.76**	7.08	9.34	2.35	5.69	30.69	19.72	20.86	16.28	17.86	30.68
30	22.83	15.59	22.01	11.21	19.78	39.92	34.94	37.84	32.44	36.24	**43.86**
60	5.95	55.50	61.93	53.64	61.52	67.23	67.62	70.00	67.16	69.71	**71.86**
100	–	10.19	27.01	2.83	25.22	22.59	34.77	42.09	31.37	40.26	**43.26**
200	–	8.78	29.70	2.64	28.94	–	35.31	**44.58**	33.70	43.88	–
400	–	7.34	33.22	2.62	32.36	–	36.40	47.68	36.93	**47.92**	–
800	–	4.90	34.75	4.19	32.33	–	39.32	51.37	42.63	**52.36**	–
1,600	–	4.47	34.27	5.34	28.75	–	41.56	**53.03**	43.51	50.79	–
3,200	–	4.49	26.30	5.70	19.96	–	42.62	49.59	46.26	**50.18**	–

Table 17
Total number S.BEST (aggregated by number of jobs, with D^E).

n	Num. of instances	GRB (*GST*)	BATCS-b			BATCS-d		
			-LS (*TS4*)	-ML (PC_8, *GS3*)		-LS (*TS4*)	-ML (PC_6, *GS3*)	
				-LS (*TS3*)	-GRB (*GI2*)		-LS (*TS3*)	-GRB (*GI1*)
15	864	**244**	1	0	161	3	0	141
30	1,584	264	14	0	407	46	0	**766**
60	2,016	32	80	0	257	160	0	**1,255**
100	3,840	–	330	192	252	866	8	**1,633**
200	3,840	–	431	353	–	**1,057**	474	–
400	3,840	–	524	290	–	**1,119**	602	–
800	1,728	–	342	88	–	486	319	–
1,600	768	–	115	60	–	345	125	–
3,200	192	–	26	22	–	51	59	–
SUM	18,672	540	1,863	1,005	1,077	4,133	1,587	3,795

Table 18
Influence of the task-parallel implementation on BATCS-d-ML (with D^{V60}).

n	Mean Speedup(n, th) = CT(1) / CT(th)							Efficiency(n, th) = Speedup(n, th) / th						
th=	2	3	4	5	6	7	8	2	3	4	5	6	7	8
100	1.7	2.3	2.7	2.9	2.9	3.4	3.4	0.83	0.77	0.67	0.58	0.49	0.49	0.43
200	1.8	2.5	3.1	3.5	3.8	4.2	4.3	0.92	0.82	0.77	0.69	0.63	0.60	0.54
400	1.9	2.7	3.4	4.1	4.7	4.9	5.3	0.94	0.91	0.84	0.82	0.78	0.70	0.66
800	1.9	2.9	3.7	4.5	5.1	5.7	6.0	0.97	0.95	0.93	0.89	0.85	0.81	0.75
1,600	2.0	2.9	3.8	4.7	5.4	6.1	6.4	0.98	0.96	0.96	0.94	0.90	0.87	0.80
3,200	2.0	3.0	4.0	4.8	5.5	6.0	6.3	0.99	0.99	1.00	0.97	0.92	0.85	0.79

that are solely best solved by BATCS-d-LS) is −4.99 percentage points. Thus, we can conclude that many of the best solutions calculated by BATCS-d-LS are not that much better than those of BATCS-d-ML-LS.

Summarizing the solution method evaluation, instances with up to 100 jobs are best solved by BATCS-d-ML-GRB (within <170 seconds) and instances with a larger number of jobs should be solved with BATCS-d-ML-LS. If the solution quality is most important, we recommend the combination (e.g., by a multi-start approach) of BATCS-d-ML-LS and BATCS-b-ML-LS.

6.5. *Influence of the task-parallel implementation*

To investigate the influence of the task-parallel implementation on the computation times of the multi-start heuristics, we use the parameter tuning instance set D^{V80} but restricted to n ∈{100, 200, 400, 800, 1,600, 3,200} to avoid biases by operating system activities (D^{V60}). In Table 18, mean CTs and the two key figures "Speedup" and "Efficiency" as defined by Kaminsky (2010) are presented (related to the number of jobs n and the used number of threads th). Note that all experiments have been executed with $\zeta = 2$ (grid size = 76).

Ideally, the speedup of a parallel program should be equal to the number of used threads th. As can be seen in Table 18, this is not the case for the implementation of BATCS-d-ML: for up to five threads, speedup and efficiency are sufficiently good for all n, whereas the efficiency decreases when using more threads. Low speedups (efficiencies) are caused by sequential program parts like data preparation (particularly for small n) and thread management, and by unused threads resulting from the grid size to the number of thread ratio (e.g., for $th = 8$, four threads remain unused for one cycle). Since four threads have shown to be most efficient, we use $th = 4$ in all experiments.

7. Summary and further research

The considered $\widetilde{P}|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$-problem is not only of high practical relevance in several industries but also scientifically interesting due to the rare literature addressing serial batch scheduling with tardiness-related objectives. The main goal of this paper was to develop and extend heuristics with a high solution efficiency, i.e., solution methods with a low computation time and a competitive solution quality. For this, we propose seven new heuristics: Three of them enhance the best heuristic from the literature (which is based on "controlled batch utilization") by using NN for parameter estimation (to avoid the computational efforts for an exhaustive parameter grid search). The other four heuristics are based on the idea of "controlled batch urgency" and three of these also incorporate NNs for estimating the most suitable parameters. For the development of the six learning-augmented heuristics, a sufficiently large data set has been required, and thus, we generated 93,360 problem instances with controllable characteristics. The experimental results show that the "controlled batch urgency"-based heuristics generally outperform the heuristics based on "controlled batch utilization". Furthermore, we can report that the learning-augmented heuristics clearly achieve the goal of a high solution efficiency as computation times can be considerably decreased (cf., Table 15) and the solution quality is not only competitive but even higher (cf., Table 16).

Although the solution quality is quite good, the mean RIWs of instances with fewer than 100 jobs indicate further potential for improvement. Therefore, a first topic for further research is the development of other heuristics, for instance, more sophisticated improvement methods or metaheuristics like Variable neighborhood search or Genetic algorithms. As we observed a relatively low batch utilization for both types of heuristics (on average 61.6 % for "controlled batch urgency" and 67.9 % for "controlled batch utilization"), a second topic could be the development of multi-objective optimization approaches that consider batch utilization related objective criteria (e.g., to avoid waste in the laser cutting application case). Further research topics could include analyzing other regression methods like Support vector machines or Inductive decision trees, the application of feature attribution methods or other methods for a more sophisticated feature engineering (model explainability or data leakage detections), a suitability analysis of the Reinforcement learning paradigm (an RL agent could learn to determine heuristic parameters), or the development of other approaches addressing the IF-DT problem. Finally, the integration of the new learning-augmented heuristics into a hyper-heuristic with the scope to learn selecting the most suitable heuristic might be worth investigating.

CRediT authorship contribution statement

Aykut Uzunoglu: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Christian Gahm:** Conceptualization, Methodology, Project administration, Software, Validation, Writing – original draft, Writing – review & editing. **Stefan Wahl:** Conceptualization, Formal analysis, Methodology, Writing – original draft, Writing – review & editing. **Axel Tuma:** Conceptualization, Resources, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All data is made available on Mendeley Data.

Appendix

A-1. Mixed-integer linear program

The MILP proposed by Gahm et al. (2021) has been slightly adapted to the following model (note that the number of batches o is set to be equal to the number of jobs n: $o = n$):

Decision variables:

Job to batch assignment decision is modeled by the binary variables $X_{b,i,j,f}$:

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

$\forall\, b \in B,\ i \in M; j \in J; f \in F$

The family-to-batch assignment is modeled by the binary variables $Y_{b,i,f}$:

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

$\forall\, b \in B;\ i \in M; f \in F$

Auxiliary variables:

- $P_{b,i,f} \in \mathbb{R}^{\geqslant}$:= processing time of batch b on machine i containing jobs of family f
- $C_{b,i} \in \mathbb{R}^{\geqslant}$:= completion time of batch b on machine i
- $C_j \in \mathbb{R}^{\geqslant}$:= completion time of job j
- $T_j \in \mathbb{R}^{\geqslant}$:= tardiness of job j
- $TF \in \mathbb{R}^{\geqslant}$:= total flow time

Objective function:

$$Minimize \left(\sum_{j\in J} w_j \cdot T_j\right) + \frac{TF}{\left(\widetilde{C}_{\max} \cdot n \cdot 10\right)} \tag{5}$$

Constraints:

$$\sum_{b\in B}\sum_{i\in M} X_{b,\, i,\, j,\, f} = 1$$

$$\forall\, f \in F;\ j \in I_f \tag{6}$$

$$\sum_{f\in F}\sum_{j\in I_f} X_{b,i,j,f} \cdot cr_j \leqslant bc$$

$$\forall\, b \in B;\ i \in M \tag{7}$$

$$\left(\sum_{f\in F}\sum_{j\in I_f} X_{b-1,i,j,f}\right) \cdot n \geqslant \sum_{f\in F}\sum_{j\in I_f} X_{b,i,j,f}$$

$$\forall\, b \in B \ \wedge \ b > 1;\ i \in M \tag{8}$$

$$X_{b,i,j,f} \leqslant Y_{b,i,f}$$

$$\forall\, b \in B;\ i \in M;\ f \in F;\ j \in I_f \tag{9}$$

$$\sum_{f\in F} Y_{b,i,f} \leqslant 1$$

$$\forall\, b \in B;\ i \in M \tag{10}$$

$$P_{b,i,f} \geqslant \sum_{j\in I_f} X_{b,i,j,f} \cdot p_j$$

$$\forall\, b \in B;\ i \in M;\ f \in F \tag{11}$$

$$C_{1,i} \geqslant s_{0,f} + P_{1,i,f} - bigM\cdot(1 - Y_{1,i,f})$$

$$\forall\, i \in M;\ f \in F \tag{12}$$

$$C_{b,i} \geqslant C_{b-1,i} + \left[s_{f,f'} - bigM\cdot(1 - Y_{b-1,i,f})\right] + \left[P_{b,i,f'} - bigM\cdot(1 - Y_{b,i,f'})\right]$$

$$\forall\, b \in B \wedge b > 1;\ i \in M;\ f, f' \in F \tag{13}$$

$$C_j \geqslant C_{b,i} - bigM\cdot(1 - X_{b,i,j,f})$$

$$\forall\, b \in B;\ i \in M;\ f \in F;\ j \in I_f \tag{14}$$

$$T_j \geqslant (C_j - d_j)$$

$$\forall\, j \in J \tag{15}$$

$$TF = \sum_{j \in J} C_j \tag{16}$$

Constraints set (6) ensures that every job has to be assigned to exactly one batch and set (7) 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 (8) allow only batches at the end of the schedule to be empty. The family of the batches is determined by (9), and (10) assures that only jobs of the same family are assigned to one batch. Equations (11) determine the processing time per batch. Constraints (12) define the completion time of the first batch on each machine and constraints (13) the completion times of all following batches. The completion time per job is defined by (14), and its tardiness by (15). The constraint set (16) determines the total flow time. Within all corresponding constraints, the following *bigM*-value is appropriate: $bigM = \widetilde{C}_{\max}\cdot n$.

A-2. Direct application of prediction results content

Table 19
Analysis of single predicted parameter configurations for BATCS-b-ML.

	MRIW (D^{V80})									Num. BEST	Mean CT							
n	30	60	100	200	400	800	1,600	3,200	MEAN		30	60	100	200	400	800	1,600	3,200
BATCS-b											[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
	42.0	60.5	42.3	51.6	61.1	63.3	68.8	84.2	59.2		1.5	1.7	4.8	11.6	33.9	5.3	21.3	108.9
	BATCS-b-ML															[s]	[s]	[s]
PC_1	24.9	37.4	24.9	31.4	48.9	53.4	56.6	72.2	43.7	2	0.04	0.05	0.13	0.15	0.24	0.86	2.71	13.67
PC_2	26.3	31.2	32.5	31.4	55.8	52.6	44.5	57.2	41.4	1	0.05	0.04	0.15	0.15	0.23	0.83	2.67	11.71
PC_3	23.6	21.5	33.6	40.3	54.2	52.9	56.6	73.0	44.5	1	0.04	0.04	0.13	0.15	0.26	0.86	2.67	11.99
PC_4	**28.6**	**38.2**	**30.6**	**33.9**	**53.0**	**55.1**	**62.8**	**80.1**	**47.8**	4	**0.05**	**0.05**	**0.13**	**0.15**	**0.24**	**0.86**	**2.68**	**13.31**
PC_5	20.3	33.3	30.3	31.6	48.2	46.1	49.2	66.0	40.6	0	0.04	0.04	0.13	0.15	0.23	0.84	2.77	12.74
PC_6	23.3	34.3	29.7	30.9	50.5	54.6	63.1	81.4	46.0	2	0.04	0.06	0.13	0.15	0.24	0.83	2.60	11.93
PC_7	14.8	30.2	26.2	27.9	46.5	50.5	62.3	35.2	36.7	0	0.04	0.04	0.14	0.16	0.23	0.88	2.81	12.73
PC_8	16.4	31.1	29.8	35.7	53.1	50.1	50.9	53.7	40.1	2	0.04	0.04	0.13	0.15	0.22	0.81	3.08	11.73
PC_9	12.8	35.4	32.0	32.9	52.1	57.8	42.3	12.2	34.7	1	0.04	0.05	0.13	0.15	0.23	0.86	2.81	12.82
PC_{10}	22.5	30.7	25.7	29.8	51.8	46.4	51.9	28.3	35.9	7	0.03	0.03	0.13	0.15	0.22	0.86	2.69	12.49
PC_{11}	17.4	25.8	33.1	39.8	50.0	52.3	43.8	32.4	36.8	3	0.04	0.05	0.13	0.15	0.28	0.85	2.63	11.55
PC_{12}	8.7	20.6	23.7	38.4	57.0	48.3	63.5	70.7	41.3	1	0.04	0.04	0.13	0.15	0.23	0.83	2.92	11.58
PC_{13}	8.1	25.2	8.1	23.2	10.6	13.6	47.3	74.0	26.3	0	0.07	0.04	0.26	0.17	0.24	0.80	2.66	11.85
PC_{14}	10.2	24.0	8.4	20.5	11.5	14.9	45.9	73.3	26.1	0	0.04	0.04	0.13	0.15	0.23	0.80	2.91	11.79
PC_{15}	8.1	23.0	5.1	22.2	17.8	14.3	47.7	74.0	26.5	0	0.04	0.04	0.13	0.15	0.23	0.81	2.60	11.87
PC_{16}	12.9	23.3	7.1	20.4	14.1	12.5	44.9	73.2	26.1	0	0.03	0.06	0.13	0.15	0.23	0.90	2.53	12.45
PC_{17}	6.7	24.4	6.9	24.2	17.4	17.1	47.5	74.2	27.3	0	0.05	0.04	0.12	0.15	0.26	0.85	2.66	12.85
PC_{18}	7.4	23.0	1.7	21.3	6.1	14.9	46.6	73.4	24.3	0	0.06	0.04	0.13	0.15	0.23	0.84	2.93	12.10

Table 20
Analysis of single predicted parameter configurations for BATCS-d-ML.

	MRIW (D^{V80})									Num. BEST	Mean CT							
n	30	60	100	200	400	800	1,600	3,200	MEAN		30	60	100	200	400	800	1,600	3,200
BATCS-d											[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
	61.2	73.2	58.6	71.9	73.3	84.3	79.7	87.9	73.7		0.8	1.5	4.9	16.9	43.7	6.4	27.4	147.7
	BATCS-d-ML															[s]	[s]	[s]
PC_1	19.6	40.7	45.0	36.4	54.5	60.9	56.8	59.6	46.7	1	0.04	0.04	0.12	0.14	0.24	0.92	2.76	11.30
PC_2	**32.9**	**49.7**	**49.9**	**60.6**	**65.9**	**75.0**	**62.5**	**72.8**	**58.7**	3	**0.05**	**0.04**	**0.12**	**0.15**	**0.22**	**0.94**	**3.78**	**16.09**
PC_3	26.0	19.9	38.2	60.2	58.2	60.2	63.6	48.8	46.9	2	0.04	0.04	0.12	0.15	0.23	0.88	2.74	11.98
PC_4	29.2	51.0	41.2	24.7	49.5	75.9	68.5	61.1	50.1	2	0.05	0.04	0.12	0.14	0.22	0.92	3.42	14.19
PC_5	36.2	50.2	40.8	61.0	62.6	58.5	58.0	65.2	54.1	0	0.04	0.04	0.12	0.15	0.23	0.85	2.93	13.03
PC_6	32.1	35.0	49.9	64.5	50.9	76.1	70.1	65.2	55.5	10	0.03	0.03	0.11	0.17	0.22	1.02	3.40	16.69
PC_7	30.6	43.9	32.8	51.4	61.2	66.2	78.7	68.1	54.1	0	0.04	0.04	0.12	0.16	0.23	0.91	2.99	12.93
PC_8	29.4	39.9	36.4	58.6	59.9	69.1	76.8	84.3	56.8	1	0.04	0.04	0.12	0.16	0.25	0.96	3.46	14.89
PC_9	23.0	43.6	46.2	63.0	65.3	64.1	67.8	72.6	55.7	1	0.04	0.05	0.13	0.16	0.22	0.93	2.87	13.68
PC_{10}	27.0	48.1	40.3	58.2	60.8	63.8	75.5	83.5	57.2	0	0.04	0.04	0.13	0.14	0.24	0.94	3.24	15.06
PC_{11}	22.7	43.3	40.5	52.0	66.8	63.9	73.7	80.0	55.4	0	0.05	0.04	0.12	0.15	0.23	0.93	3.04	13.32
PC_{12}	33.9	31.1	39.9	49.1	58.5	36.1	64.0	74.2	48.3	0	0.08	0.04	0.19	0.15	0.24	1.07	3.72	16.55
PC_{13}	25.4	38.4	19.6	43.5	36.4	36.0	60.2	75.1	41.8	0	0.04	0.04	0.12	0.15	0.22	0.90	2.91	13.68
PC_{14}	27.5	40.9	16.3	40.1	28.3	27.7	57.4	66.7	38.1	0	0.04	0.04	0.12	0.16	0.24	0.92	3.17	14.28
PC_{15}	25.3	39.5	18.6	43.0	35.0	37.5	61.5	74.4	41.8	0	0.04	0.04	0.12	0.14	0.23	0.89	3.05	13.28
PC_{16}	27.7	39.4	20.2	43.8	29.0	33.4	57.6	70.4	40.2	0	0.04	0.04	0.13	0.15	0.22	0.93	3.38	14.15
PC_{17}	25.8	38.7	16.3	41.7	33.4	38.2	61.3	74.2	41.2	0	0.09	0.04	0.12	0.14	0.23	0.92	2.92	13.42
PC_{18}	23.3	39.3	20.1	43.0	23.0	29.6	57.7	66.1	37.8	0	0.04	0.04	0.12	0.15	0.23	0.95	3.07	14.66

A-3. Parameter tuning results content

Table 21
Pipeline configuration analysis for BATCS-b-ML (mean values for all grid settings).

	MRIW (D^{V80})									Mean CT							
n	30	60	100	200	400	800	1,600	3,200	MEAN	30	60	100	200	400	800	1,600	3,200
BATCS-b										[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
	25.0	23.8	21.5	13.0	6.0	13.7	5.3	4.6	14.1	1.5	1.6	4.1	12.8	42.4	6.1	25.3	131.5
BATCS-b-ML																	
PC_1	17.6	19.2	18.6	5.4	6.3	12.1	6.5	0.4	10.8	0.1	0.2	0.3	0.5	1.3	0.1	0.6	2.9
PC_2	20.6	19.8	18.0	6.5	3.8	12.7	7.4	1.2	11.3	0.1	0.2	0.3	0.5	1.3	0.2	0.5	2.8
PC_3	18.6	20.1	20.4	9.3	5.7	12.2	4.6	0.5	11.4	0.1	0.2	0.3	0.5	1.5	0.2	0.5	2.8
PC_4	**20.6**	**19.9**	**19.1**	**5.8**	**3.9**	**14.6**	**11.1**	**4.7**	**12.5**	**0.1**	**0.2**	**0.3**	**0.5**	**1.4**	**0.2**	**0.5**	**2.8**
PC_5	17.8	19.2	11.1	6.5	5.7	8.5	1.1	0.4	8.8	0.1	0.1	0.3	0.5	1.3	0.1	0.5	3.0
PC_6	20.5	19.4	17.0	6.1	3.8	13.5	7.0	1.1	11.0	0.1	0.1	0.3	0.5	1.3	0.1	0.5	2.8
PC_7	19.8	17.1	17.0	5.2	5.8	12.1	3.5	0.5	10.1	0.1	0.2	0.3	0.5	1.3	0.2	0.5	3.0
PC_8	**24.5**	**18.4**	**19.8**	**6.9**	**8.5**	**13.4**	**9.3**	**2.7**	**12.9**	**0.1**	**0.2**	**0.3**	**0.5**	**1.4**	**0.1**	**0.5**	**2.8**
PC_9	19.6	17.6	17.3	6.8	5.8	11.0	3.4	1.5	10.4	0.1	0.2	0.3	0.5	1.3	0.1	0.4	2.7
PC_{10}	**20.4**	**18.4**	**18.8**	**7.9**	**10.8**	**14.3**	**6.0**	**1.6**	**12.3**	**0.1**	**0.2**	**0.3**	**0.5**	**1.3**	**0.2**	**0.5**	**2.7**
PC_{11}	20.2	18.3	18.0	5.1	3.9	13.5	3.4	0.5	10.4	0.1	0.2	0.2	0.5	1.3	0.1	0.5	2.8
PC_{12}	20.3	18.6	16.0	7.8	8.3	12.1	7.2	2.6	11.6	0.1	0.2	0.3	0.5	1.3	0.2	0.5	2.7
PC_{13}	15.4	16.0	4.7	6.5	2.4	5.3	3.6	0.9	6.9	0.1	0.2	0.2	0.5	1.4	0.1	0.4	2.6
PC_{14}	15.4	16.0	4.7	6.5	2.4	5.3	3.6	0.9	6.9	0.1	0.2	0.3	0.5	1.4	0.2	0.5	2.9
PC_{15}	15.4	16.0	4.7	6.5	2.4	5.3	3.6	0.9	6.9	0.1	0.1	0.3	0.5	1.3	0.1	0.4	2.5
PC_{16}	15.4	16.0	4.7	6.5	2.4	5.3	3.6	0.9	6.9	0.1	0.1	0.3	0.5	1.3	0.1	0.5	2.9
PC_{17}	15.4	16.0	4.7	6.5	2.4	5.3	3.6	0.9	6.9	0.1	0.2	0.3	0.5	1.4	0.1	0.4	2.6
PC_{18}	15.4	16.0	4.7	6.5	2.4	5.3	3.6	0.9	6.9	0.1	0.2	0.3	0.5	1.4	0.2	0.5	2.9

Table 22
Pipeline configuration analysis for BATCS-d-ML (mean values for all grid settings).

	MRIW (D^{V80})									Mean CT							
n	30	60	100	200	400	800	1,600	3,200	MEAN	30	60	100	200	400	800	1,600	3,200
BATCS-d										[s]	[s]	[s]	[s]	[s]	[m]	[m]	[m]
	46.5	34.8	38.5	54.5	31.0	60.9	48.0	57.4	46.4	0.8	1.3	4.7	14.3	42.2	6.3	28.2	169.4
BATCS-d-ML																	
PC₁	**35.5**	**33.8**	**36.4**	**51.7**	**39.1**	**51.9**	**47.9**	**52.6**	**43.6**	**0.1**	**0.2**	**0.3**	**0.6**	**1.5**	**0.2**	**0.6**	**3.3**
PC_2	34.8	28.7	35.4	51.5	30.7	57.8	49.0	52.0	42.5	0.1	0.2	0.3	0.6	1.6	0.2	0.7	3.7
PC_3	34.1	33.6	35.4	51.3	31.1	50.8	44.9	52.6	41.7	0.1	0.2	0.3	0.6	1.6	0.2	0.5	3.0
PC_4	**38.4**	**33.5**	**36.0**	**52.4**	**32.2**	**59.2**	**53.1**	**54.5**	**44.9**	**0.1**	**0.2**	**0.3**	**0.6**	**1.6**	**0.2**	**0.6**	**3.9**
PC_5	34.1	33.8	34.6	51.0	30.5	51.1	44.2	52.2	41.4	0.1	0.2	0.3	0.6	1.6	0.2	0.5	3.3
PC_6	**38.7**	**32.1**	**38.5**	**51.1**	**44.0**	**59.0**	**53.4**	**56.5**	**46.7**	**0.1**	**0.2**	**0.3**	**0.6**	**1.7**	**0.2**	**0.7**	**3.9**
PC_7	35.4	31.5	35.2	51.1	36.7	52.0	45.4	51.4	42.4	0.1	0.2	0.3	0.6	1.5	0.2	0.6	3.4
PC_8	41.6	30.4	34.6	49.5	30.7	57.2	48.2	53.2	43.2	0.1	0.2	0.3	0.6	1.5	0.2	0.7	3.7
PC_9	40.4	30.9	33.6	49.8	30.9	52.8	50.0	50.8	42.4	0.1	0.2	0.3	0.6	1.5	0.2	0.6	3.1
PC_{10}	**41.0**	**29.9**	**36.1**	**49.8**	**30.9**	**57.9**	**48.7**	**55.0**	**43.6**	**0.1**	**0.2**	**0.3**	**0.6**	**1.6**	**0.2**	**0.7**	**3.8**
PC_{11}	38.7	30.3	34.4	49.8	31.1	53.3	47.7	50.8	42.0	0.1	0.2	0.3	0.6	1.7	0.2	0.6	3.2
PC_{12}	38.3	27.4	32.8	49.8	30.3	57.7	50.1	55.7	42.8	0.1	0.2	0.3	0.6	1.6	0.2	0.6	3.8
PC_{13}	37.8	30.3	32.3	50.7	29.4	51.9	47.4	52.0	41.5	0.1	0.2	0.3	0.6	1.4	0.2	0.6	3.4
PC_{14}	38.2	28.4	32.2	49.8	29.4	52.5	48.2	52.1	41.3	0.1	0.2	0.3	0.6	1.7	0.2	0.6	3.5
PC_{15}	37.8	30.3	32.3	50.7	29.4	51.9	47.4	52.0	41.5	0.1	0.2	0.3	0.6	1.5	0.2	0.6	3.3
PC_{16}	38.2	28.4	32.2	49.8	29.4	52.5	48.2	52.1	41.3	0.1	0.2	0.3	0.7	1.8	0.2	0.6	3.4
PC_{17}	37.8	30.3	32.3	50.7	29.4	51.9	47.4	52.0	41.5	0.1	0.2	0.3	0.6	1.5	0.1	0.6	3.3
PC_{18}	38.2	28.4	32.2	49.8	29.4	52.5	48.2	52.1	41.3	0.1	0.2	0.3	0.6	1.6	0.2	0.6	3.5

References

Agarwal, A., Colak, S., Eryarsoy, E., 2006. Improvement heuristic for the flow-shop scheduling problem: an adaptive-learning approach. Eur. J. Oper. Res. 169 (3), 801–815. https://doi.org/10.1016/j.ejor.2004.06.039.

Akyol, D.E., 2004. Application of neural networks to heuristic scheduling algorithms. Comput. Ind. Eng. 46 (4), 679–696. https://doi.org/10.1016/j.cie.2004.05.005.

Akyol, D.E., Bayhan, G.M., 2007. A review on evolution of production scheduling with neural networks. Comput. Ind. Eng. 53 (1), 95–122. https://doi.org/10.1016/j.cie.2007.04.006.

Almeder, C., Mönch, L., 2011. Metaheuristics for scheduling jobs with incompatible families on parallel batching machines. J. Oper. Res. Soc. 62 (12), 2083–2096. https://doi.org/10.1057/jors.2010.186.

Azadeh, A., Negahban, A., Moghaddam, M., 2012. A hybrid computer simulation-artificial neural network algorithm for optimisation of dispatching rule selection in stochastic job shop scheduling problems. Int. J. Prod. Res. 50 (2), 551–566. https://doi.org/10.1080/00207543.2010.539281.

Azadeh, A., Shoja, B.M., Moghaddam, M., Asadzadeh, S.M., Akbari, A., 2013. A neural network meta-model for identification of optimal combination of priority dispatching rules and makespan in a deterministic job shop scheduling problem. Int. J. Adv. Manuf. Technol. 67 (5–8), 1549–1561. https://doi.org/10.1007/s00170-012-4589-y.

Balasubramanian, H., Mönch, L., Fowler, J.W., Pfund, M.E., 2004. Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness. Int. J. Prod. Res. 42 (8), 1621–1638. https://doi.org/10.1080/00207540310001636994.

Bertolini, M., Mezzogori, D., Neroni, M., Zammori, F., 2021. Machine Learning for industrial applications: a comprehensive literature review. Expert Syst. Appl. 175, 114820 https://doi.org/10.1016/j.eswa.2021.114820.

Borchani, H., Varando, G., Bielza, C., Larrañaga, P., 2015. A survey on multi-output regression. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 5 (5), 216–233. https://doi.org/10.1002/widm.1157.

Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015. Hyper-heuristic evolution of dispatching rules: a comparison of rule representations. Evol. Comput. 23 (2), 249–277. https://doi.org/10.1162/EVCO_a_00131.

Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M., 2016. Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. 20 (1), 110–124. https://doi.org/10.1109/TEVC.2015.2429314.

Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. 64 (12), 1695–1724. https://doi.org/10.1057/jors.2013.71.

Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Özcan, E., oodward, J.R. (2010). A Classification of Hyper-heuristic Approaches. In M. Gendreau & J.-Y. Potvin (Eds.), *International Series in Operations Research & Management Science: Vol. 146. Handbook of Metaheuristics* (2nd ed., 449–468), Springer Science+Business Media.

Castillo, F., Gazmuri, P., 2015. Genetic algorithms for batch sizing and production scheduling. Int. J. Adv. Manuf. Technol. 77 (1–4), 261–280. https://doi.org/10.1007/s00170-014-6456-5.

Chen, J.Y., Pfund, M.E., Fowler, J.W., Montgomery, D.C., Callarman, T.E., 2010. Robust scaling parameters for composite dispatching rules. IIE Trans. 42 (11), 842–853. https://doi.org/10.1080/07408171003685825.

Chen, S., Fang, S., Tang, R., 2019. A reinforcement learning based approach for multi-projects scheduling in cloud manufacturing. Int. J. Prod. Res. 57 (10), 3080–3098. https://doi.org/10.1080/00207543.2018.1535205.

El-Bouri, A., 2012. A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop. Comput. Oper. Res. 39 (7), 1305–1314. https://doi.org/10.1016/j.cor.2011.07.004.

El-Bouri, A., Balakrishnan, S., Popplewell, N., 2000. Sequencing jobs on a single machine: a neural network approach. Eur. J. Oper. Res. 126 (3), 474–490. https://doi.org/10.1016/S0377-2217(99)00302-1.

Freitag, M., Hildebrandt, T., 2016. Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization. CIRP Ann. Manuf. Technol. 65 (1), 433–436. https://doi.org/10.1016/j.cirp.2016.04.066.

Gahm, C., 2022a. Extended instance sets for the parallel serial-batch scheduling problem with incompatible job families, sequence-dependent setup times, and arbitrary sizes. Mendeley Data V2. https://doi.org/10.17632/rxc695hj2k.1.

Gahm, C., 2022b. Data set for learning heuristic parameters by a multi-target regression with dependent outputs. Mendeley Data V1. https://doi.org/10.17632/hm5d3cc45y.1.

Gahm, C., Uzunoglu, A., Wahl, S., Ganschinietz, C., Tuma, A., 2022. Applying machine learning for the anticipation of complex nesting solutions in hierarchical production planning. Eur. J. Oper. Res. 296 (3), 819–836. https://doi.org/10.1016/j.ejor.2021.04.006.

Gahm, C., Wahl, S., Tuma, A., 2021. Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes. Int. J. Prod. Res. 1–24 https://doi.org/10.1080/00207543.2021.1951446.

Geiger, C.D., Uzsoy, R., 2008. Learning effective dispatching rules for batch processor scheduling. Int. J. Prod. Res. 46 (6), 1431–1454. https://doi.org/10.1080/00207540600993360.

Geiger, C.D., Uzsoy, R., Aytuğ, H., 2006. Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. J. Sched. 9 (1), 7–34. https://doi.org/10.1007/s10951-006-5591-8.

Giglio, D., 2015. Optimal control strategies for single-machine family scheduling with sequence-dependent batch setup and controllable processing times. J. Sched. 18 (5), 525–543. https://doi.org/10.1007/s10951-015-0440-2.

Goodfellow, I., Bengio, Y., Courville, A., 2016. *Deep Learning. Adaptive Computation and Machine Learning*. The MIT Press, Cambridge, Massachusetts, London, England.

Heger, J., Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2016. Dynamic adjustment of dispatching rule parameters in flow shops with sequence-dependent set-up times. Int. J. Prod. Res. 54 (22), 6812–6824. https://doi.org/10.1080/00207543.2016.1178406.

Heger, J., Voss, T., 2021. Dynamically adjusting the k -values of the ATCS rule in a flexible flow shop scenario with reinforcement learning. Int. J. Prod. Res. 1–15 https://doi.org/10.1080/00207543.2021.1943762.

Helo, P., Phuong, D., Hao, Y., 2019. Cloud manufacturing – Scheduling as a service for sheet metal manufacturing. Comput. Oper. Res. 110, 208–219. https://doi.org/10.1016/j.cor.2018.06.002.

Jordan, C., Drexl, A., 1998. Discrete lotsizing and scheduling by batch sequencing. Manag. Sci. 44 (5), 698–713. https://doi.org/10.1287/mnsc.44.5.698.

Kaminsky, A., 2010. Building Parallel Programs: SMPs, Clusters, and Java. Course Technology, Boston.

Kim, C.-O., Min, H.-S., Yih, Y., 1998. Integration of inductive learning and neural networks for multi-objective FMS scheduling. Int. J. Prod. Res. 36 (9), 2497–2509. https://doi.org/10.1080/002075498192652.

Kim, S.-Y., Lee, Y.-H., Agnihotri, D., 1995. A hybrid approach to sequencing jobs using heuristic rules and neural networks. Prod. Plan. Control 6 (5), 445–454. https://doi.org/10.1080/09537289508930302.

Koonce, D., Tsai, S.-C., 2000. Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. Comput. Ind. Eng. 38 (3), 361–374. https://doi.org/10.1016/S0360-8352(00)00050-4.

Lee, Y.-H., Bhaskaran, K., Pinedo, M.L., 1997. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Trans. 29 (1), 45–52. https://doi.org/10.1080/07408179708966311.

Lee, Y.-H., Pinedo, M.L., 1997. Scheduling jobs on parallel machines with sequence-dependent setup times. Eur. J. Oper. Res. 100 (3), 464–474. https://doi.org/10.1016/S0377-2217(95)00376-2.

Li, X., Olafsson, S., 2005. Discovering dispatching rules using data mining. J. Sched. 8 (6), 515–527. https://doi.org/10.1007/s10951-005-4781-0.

Lin, C.-C., Deng, D.-J., Chih, Y.-L., Chiu, H.-T., 2019. Smart manufacturing scheduling with edge computing using multiclass deep Q network. IEEE Trans. Ind. Inf. 15 (7), 4276–4284. https://doi.org/10.1109/TII.2019.2908210.

Malve, S., Uzsoy, R., 2007. A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families. Comput. Oper. Res. 34 (10), 3016–3028. https://doi.org/10.1016/j.cor.2005.11.011.

Melki, G., Cano, A., Kecman, V., Ventura, S., 2017. Multi-target support vector regression via correlation regressor chains. Inf. Sci. 415–416, 53–69. https://doi.org/10.1016/j.ins.2017.06.017.

Mönch, L., Zimmermann, J., Otto, P., 2006. Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines. Eng. Appl. Artif. Intel. 19 (3), 235–245. https://doi.org/10.1016/j.engappai.2005.10.001.

Mouelhi-Chibani, W., Pierreval, H., 2010. Training a neural network to select dispatching rules in real time. Comput. Ind. Eng. 58 (2), 249–256. https://doi.org/10.1016/j.cie.2009.03.008.

Murphy, K.P., 2013. *Machine Learning: A Probabilistic Perspective* (4. print. (fixed many typos)). *Adaptive Computation and Machine Learning Series*. MIT Press, Cambridge, Mass.

Naddef, D., Santos, C., 1988. One-pass batching algorithms for the one-machine problem. Discret. Appl. Math. 21 (2), 133–145. https://doi.org/10.1016/0166-218X(88)90049-2.

Nguyen, S., Mei, Y., Zhang, M., 2017a. Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. 3 (1), 41–66. https://doi.org/10.1007/s40747-017-0036-x.

Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. Learning iterative dispatching rules for job shop scheduling with genetic programming. Int. J. Adv. Manuf. Technol. 67 (1–4), 85–100. https://doi.org/10.1007/s00170-013-4756-9.

Nguyen, S., Zhang, M., Tan, K.C., 2017b. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans. Cybern. 47 (9), 2951–2965. https://doi.org/10.1109/TCYB.2016.2562674.

Olafsson, S., Li, X., 2010. Learning effective new single machine dispatching rules from optimal scheduling data. Int. J. Prod. Econ. 128 (1), 118–126. https://doi.org/10.1016/j.ijpe.2010.06.004.

Park, Y., Kim, S., Lee, Y.-H., 2000. Scheduling jobs on parallel machines applying neural network and heuristic rules. Comput. Ind. Eng. 38 (1), 189–202. https://doi.org/10.1016/S0360-8352(00)00038-3.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., 2011. Scikit-learn: machine learning in python. J. Mach. Learn. Res. 12, 2825–2830.

Pei, J., Liu, X., Pardalos, P.M., Fan, W., Yang, S., 2017. Scheduling deteriorating jobs on a single serial-batching machine with multiple job types and sequence-dependent setup times. Ann. Oper. Res. 249 (1–2), 175–195. https://doi.org/10.1007/s10479-015-1824-6.

Pfund, M., Fowler, J.W., Gadkari, A., Chen, Y., 2008. Scheduling jobs on parallel machines with setup times and ready times. Comput. Ind. Eng. 54 (4), 764–782. https://doi.org/10.1016/j.cie.2007.08.011.

Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B., 2013. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. Int. J. Prod. Econ. 145 (1), 67–77. https://doi.org/10.1016/j.ijpe.2012.10.016.

Priore, P., Gómez, A., Pino, R., Rosillo, R., 2014. Dynamic scheduling of manufacturing systems using machine learning: an updated review. Artif. Intell. Eng. Des. Anal. Manuf. 28 (1), 83–97. https://doi.org/10.1017/S0890060413000516.

Priore, P., de La Fuente, D., Puente, J., Parreño, J., 2006. A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems. Eng. Appl. Artif. Intel. 19 (3), 247–255. https://doi.org/10.1016/j.engappai.2005.09.009.

Priore, P., Ponte, B., Puente, J., Gómez, A., 2018. Learning-based scheduling of flexible manufacturing systems using ensemble methods. Comput. Ind. Eng. 126, 282–291. https://doi.org/10.1016/j.cie.2018.09.034.

Raaymakers, W.H.M., Weijters, A., 2003. Makespan estimation in batch process industries: a comparison between regression analysis and neural networks. Eur. J. Oper. Res. 145 (1), 14–30. https://doi.org/10.1016/S0377-2217(02)00173-X.

Reyes, O., Ventura, S., 2019. Performing multi-target regression via a parameter sharing-based deep network. Int. J. Neural Syst. 29 (9), 1950014. https://doi.org/10.1142/S012906571950014X.

Rodríguez, J.J., Juez-Gil, M., López-Nozal, C., Arnaiz-González, Á., 2021. Rotation Forest for multi-target regression. Int. J. Mach. Learn. Cybern. https://doi.org/10.1007/s13042-021-01329-1.

Shafaei, R., Rabiee, M., Mirzaeyan, M., 2011. An adaptive neuro fuzzy inference system for makespan estimation in multiprocessor no-wait two stage flow shop. Int. J. Comput. Integr. Manuf. 24 (10), 888–899. https://doi.org/10.1080/0951192X.2011.597430.

Shahrabi, J., Adibi, M.A., Mahootchi, M., 2017. A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. Comput. Ind. Eng. 110, 75–82. https://doi.org/10.1016/j.cie.2017.05.026.

Shahvari, O., Logendran, R., 2016. Hybrid flow shop batching and scheduling with a bi-criteria objective. Int. J. Prod. Econ. 179, 239–258. https://doi.org/10.1016/j.ijpe.2016.06.005.

Shahvari, O., Logendran, R., 2017. An Enhanced tabu search algorithm to minimize a bi-criteria objective in batching and scheduling problems on unrelated-parallel machines with desired lower bounds on batch sizes. Comput. Oper. Res. 77, 154–176. https://doi.org/10.1016/j.cor.2016.07.021.

Shen, L., Buscher, U., 2012. Solving the serial batching problem in job shop manufacturing systems. Eur. J. Oper. Res. 221 (1), 14–26. https://doi.org/10.1016/j.ejor.2012.03.001.

Shen, L., Gupta, J.N.D., 2018. Family scheduling with batch availability in flow shops to minimize makespan. J. Sched. 21 (2), 235–249. https://doi.org/10.1007/s10951-017-0529-x.

Shen, L., Mönch, L., Buscher, U., 2013. An iterative approach for the serial batching problem with parallel machines and job families. Ann. Oper. Res. 206 (1), 425–448. https://doi.org/10.1007/s10479-013-1339-y.

Shiue, Y.-R., Guh, R.-S., Lee, K.-C., 2012. Development of machine learning-based real time scheduling systems: using ensemble based on wrapper feature selection approach. Int. J. Prod. Res. 50 (20), 5887–5905. https://doi.org/10.1080/00207543.2011.636389.

Shiue, Y.-R., Lee, K.-C., Su, C.-T., 2018. Real-time scheduling for a smart factory using a reinforcement learning approach. Comput. Ind. Eng. 125, 604–614. https://doi.org/10.1016/j.cie.2018.03.039.

Spyromitros-Xioufis, E., Tsoumakas, G., Groves, W., Vlahavas, I., 2016. Multi-target regression via input space expansion: treating targets as inputs. Mach. Learn. 104 (1), 55–98. https://doi.org/10.1007/s10994-016-5546-z.

Stricker, N., Kuhnle, A., Sturm, R., Friess, S., 2018. Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Ann. Manuf. Technol. 67 (1), 511–514. https://doi.org/10.1016/j.cirp.2018.04.041.

Suppiah, Y., Omar, M.K., 2014. A hybrid tabu search for batching and sequencing decisions in a single machine environment. Comput. Ind. Eng. 78, 135–147. https://doi.org/10.1016/j.cie.2014.10.010.

Trindade, R.S., de Araújo, O.C.B., Fampa, M.H.C., Müller, F.M., 2018. Modelling and symmetry breaking in scheduling problems on batch processing machines. Int. J. Prod. Res. 56 (22), 7031–7048. https://doi.org/10.1080/00207543.2018.1424371.

Valente, J.M., Schaller, J.E., 2012. Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. Comput. Oper. Res. 39 (9), 2223–2231. https://doi.org/10.1016/j.cor.2011.11.005.

Vepsalainen, A.P.J., Morton, T.E., 1987. Priority rules for job shops with weighted tardiness costs. Manag. Sci. 33 (8), 1035–1047. https://doi.org/10.1287/mnsc.33.8.1035.

Voß, S., Witt, A., 2007. Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: a real-world application. Int. J. Prod. Econ. 105 (2), 445–458. https://doi.org/10.1016/j.ijpe.2004.05.029.

Waegeman, W., Dembczyński, K., Hüllermeier, E., 2019. Multi-target prediction: a unifying view on problems and methods. Data Min. Knowl. Disc. 33 (2), 293–324. https://doi.org/10.1007/s10618-018-0595-5.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A., Kyek, A., 2018. Optimization of global production scheduling with deep reinforcement learning. Procedia CIRP 72, 1264–1269. https://doi.org/10.1016/j.procir.2018.03.212.

Weckman, G.R., Ganduri, C.V., Koonce, D.A., 2008. A neural network job-shop scheduler. J. Intell. Manuf. 19 (2), 191–201. https://doi.org/10.1007/s10845-008-0073-9.

Xu, D., Shi, Y., Tsang, I.W., Ong, Y.-S., Gong, C., Shen, X., 2020. Survey on multi-output learning. IEEE Trans. Neural Networks Learn. Syst. 31 (7), 2409–2429. https://doi.org/10.1109/TNNLS.2019.2945133.