Annals of Operations Research
https://doi.org/10.1007/s10479-023-05541-w

ORIGINAL RESEARCH

A machine learning enhanced multi-start heuristic to efficiently solve a serial-batch scheduling problem

Aykut Uzunoglu[1] · **Christian Gahm**[1] · **Axel Tuma**[1]

Received: 18 January 2023 / Accepted: 8 August 2023

Abstract
Serial-batch scheduling problems are widespread in several industries (e.g., the metal processing industry or industrial 3D printing) and consist of two subproblems that must be solved simultaneously: the grouping of jobs into batches and the sequencing of the created batches. This problem's NP-hard nature prevents optimally solving large-scale problems; therefore, heuristic solution methods are a common choice to effectively tackle the problem. One of the best-performing heuristics in the literature is the ATCS–BATCS(β) heuristic which has three control parameters. To achieve a good solution quality, most appropriate parameters must be determined a priori or within a multi-start approach. As multi-start approaches performing (full) grid searches on the parameters lack efficiency, we propose a machine learning enhanced grid search. To that, Artificial Neural Networks are used to predict the performance of the heuristic given a specific problem instance and specific heuristic parameters. Based on these predictions, we perform a grid search on a smaller set of most promising heuristic parameters. The comparison to the ATCS–BATCS(β) heuristics shows that our approach reaches a very competitive mean solution quality that is only 2.5% lower and that it is computationally much more efficient: computation times can be reduced by 89.2% on average.

Keywords Serial batching · Incompatible job families · Sequence-dependent setup times · Arbitrary sizes · Total weighted tardiness · Heuristics · Machine learning

1 Introduction

The complexity of many industrial scheduling problems prevents their optimal solution and requires the application of heuristics or metaheuristics for efficiently calculating solutions. Such a hard-to-solve scheduling problem is the problem considered in this contribution: scheduling parallel serial-batch processing machines with incompatible job families, restricted batch capacities, arbitrary batch capacity demands, and sequence-dependent setup times. This problem arises for example in the metal processing industry where the (laser) cutting of metal sheets is one of the first and mandatory steps in the process (cf. e.g., Helo et al., 2019; Gahm et al., 2022b). The same scheduling problem also arises in the emerging

✉ Aykut Uzunoglu
aykut.uzunoglu@wiwi.uni-augsburg.de

[1] Production and Supply Chain Management, Augsburg University, 86135 Augsburg, Germany

area of additive manufacturing (i. e., industrial 3D printing; cf., e.g., Li & Zhang, 2018; Zhang et al., 2020; Toksarı & Toğa, 2022).

The specific problem with the weighted tardiness minimization objective was first described by Gahm et al. (2022b) and classified as $P|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$-problem. To solve the NP-hard problem, the authors formulate a mixed-integer linear program, propose a set of new heuristics, and benchmark those with heuristics from literature (with adaptations). Their evaluation shows that only small problem instances with up to 30 jobs can be solved to optimum. Since problem instances of up to 30 jobs are far from industrial-scale problems, which are typically in the order of several thousand jobs, mixed-integer linear programming cannot be considered for solving practical problems. In consequence, the authors propose several heuristics and their ATCS–BATCS(β) heuristic performs best regarding the trade-off between solution quality and computation time (to simplify reading, we will name this heuristic BATCS-b in the following). This BATCS-b heuristic is based on the ATCS priority rule (Lee & Pinedo, 1997) and uses a multi-start approach to perform a full grid search on the (discretized) parameter space stretched by three parameters (cf. Section 2.2). Due to this design, the computation time of the construction heuristic strongly increases with problem instances becoming large (real-world instances can consist of several thousand jobs; cf., Gahm et al., 2022b). This effect of multi-start full-grid searches is well-known and increases with the number of appropriate parameter configurations. However, the performance of many heuristics strongly depends on applying appropriate parameters, and thus, multi-start full-grid searches are very effective. Nevertheless, this comes at the cost of restarting the heuristic multiple times with different parameter configurations, resulting in long computation times. This aspect makes the application of multi-start heuristics in real-time decision scenarios, e.g., when the problem changes due to machine breakdowns or new urgent customer orders arrive, very challenging and thus prevents tardiness improvements of up to 83% (as reported in Gahm et al., 2022b).

Our contribution to literature is a new approach to improve the efficiency of multi-start heuristics through machine learning enhanced grid search (MLGS). To that, we propose to estimate the performance of individual parameter configurations regarding a single problem instance by machine learning models and use the performance predictions for creating a ranking of parameter configurations. Based on this ranking, we develop and analyze various strategies to select the most promising ones and create appropriate small grids. To the best of our knowledge, this ranking-based approach to determine a set of most suitable parameter configurations is completely new in the literature. Regarding the machine learning model, an Artificial Neural Network (NN), we develop and evaluate different "subsampling" strategies to determine samples that are most efficient for the training of the NN. In addition, we recommend controlling the size of the reduced grid with regard to the size of an instance (i.e., the number of jobs) to keep computation times manageable. The resulting heuristic with the machine learning enhanced grid search is called BATCS-b-MLGS in the following.

We would also like to point out that the application of our ranking-based "parameter tuning" approach is not limited to BATCS-b and can be applied or adapted by other parametrized heuristics or metaheuristics. In addition, the ranking of parameter configurations could be used in a different way: for example, to compute a set of promising "initial" solutions for population-based metaheuristics (e.g., to initialize the initial population of a Genetic Algorithm).

The structure of the paper is as follows. In Sect. 2, we first describe the scheduling problem and the BATCS-b heuristic in detail and then analyze the most relevant literature. In Sect. 3, all the components of the developed machine learning enhanced grid search are presented. Section 4 presents the results of the hyperparameter tuning for the Machine learning model

and the final testing results on a large test set of problem instances. In the closing Sect. 5, we summarize the findings and give an outlook on future research topics.

2 Related work

2.1 The scheduling problem

Basic task of the $P|sb,\ if^*,\ b,\ a_j,\ s_{f\,g}|wT$-problem is the grouping of n jobs ($J = \{J_j|\, j = 1,\ ...,\ n \in \mathbb{Z}^>\}$) into o batches ($B = \{B_b|\, b = 1,\ ...,\ o \in \mathbb{Z}^>\}$) and the machine allocation and sequencing (scheduling) of those batches on a set of m identical parallel machines ($M = \{M_i|\, i = 1,\ ...,\ m \in \mathbb{Z}^>\}$) with the objective of minimizing the total weighted tardiness. The maximum batch capacity bc is identical for all machines. Each job has individual weights w_j, processing times p_j, due dates d_j, and an individual (arbitrary) batch capacity requirement cr_j (size; whereby $cr_j \leq bc$ must hold). Furthermore, each job belongs to a job family f ($F = \{F_f|\, f = 1,\ ...,\ q \in \mathbb{Z}^>\}$), whereby job families are "incompatible" (that means jobs of different families cannot be processed together in one batch, e.g., due to technical or material restrictions). Because setups between the processing of two batches (jobs) are required, the batching of jobs of the same family is done to reduce setup efforts. Hereby, the setup times are family- and sequence-dependent: $s_{f,g}$ defined the setup time for a setup from a family f batch to a family g batch. Note that $s_{0,f}$ depicts initial setup times for family f at the beginning of a schedule.

Other assumptions include that each machine can process at most one batch at a time, that a batch can only be processed by one machine at a time, that all jobs are available for processing at the start (i.e., no release dates), that batch processing cannot be interrupted (i.e., no preemption), and that jobs cannot be added or removed once processing of a batch has started (i.e., batch availability). Consequently, a job's completion time equals the completion time of the batch to which a job is assigned.

2.2 The BATCS-b heuristic

To solve the aforementioned problem, Gahm et al. (2022b) developed the BATCS-b multi-start heuristic that is based on the "apparent tardiness cost with setups" (ATCS) priority rule to approximate the urgency of a single job and on the summation of job urgencies grouped in a batch to a "batch urgency". The idea of the ATCS rule goes back to the ATC rule proposed by Vepsalainen and Morton (1987) for a job shop problem with the weighted tardiness objective. Lee and Pinedo (1997) extended this rule by integrating sequence-dependent setup times for a parallel machine environment, also with the objective weighted tardiness. Since these publications, the ATC(S) rule has been used by many authors to efficiently calculate schedules for different kind of problems (cf., e.g., Heger et al., 2016; Lee, 2018; Maecker & Shen, 2020).

The BATCS-b heuristic computes the urgency $ATCS_j$ of job j at time step t based on job attributes, the family f of the last scheduled batch, and the two look-ahead (scaling) parameters κ_1 and κ_2:

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

In Eq. (1), $\overline{p}_t$ is the mean processing time of all unscheduled jobs, $\overline{s}_t$ the mean setup time between all families with unscheduled jobs, and f_j the family of job j. By knowing each job's ATCS-value, we can derive the batch ATCS (BATCS)-value $\Pi_b(t, f)$ for a given time step and family by summing up ATCS-values of jobs belonging to batch b: $\Pi_b(t, f) := \sum_{j \in b} ATCS_j(t, f)$.

In its basic version, the BATCS-b heuristic allocates jobs (ordered by their priorities) to a batch until no more jobs can be allocated without violating the batch capacity. However, using the total capacity of a batch may lead to worse solutions as jobs with a low urgency (priority value) are added to batches at the beginning of the schedule. This is because the batch processing time in serial-batching is the sum of its job's processing times and adding more jobs to a batch increases the batch processing time. To circumvent this, Gahm et al. (2022b) introduced the parameter $\beta \in (0, 1]$ to control the maximum batch utilization. This parameter defines how much of the batch capacity can be utilized and is given with $bc^{ACT} := \max\{\beta \cdot bc, \max\{cr_j | \forall j \in J\}\}$. The $\max\{cr_j | \forall j \in J\}$ term ensures that the "actualized" maximum batch capacity is not less than any job capacity requirement and that all jobs can be placed on a metal slide. Gahm et al. could report that introducing the parameter β improves the objective value remarkably.

Determining appropriate values for the parameters is necessary to achieve a satisfactory solution quality with ATC(S) based heuristics. Because there is no general rule or analytical method for obtaining good parameter configurations (containing values for β, κ_1 and κ_2) for the considered scheduling problem, Gahm et al. (2022b) use a multi-start grid search in their BATCS-b heuristic. For the multi-start heuristic, the authors define the sets for the parameters β, κ_1 and κ_2 as $\mathrm{B} := \{0.5, 0.55, \ldots, 1\}$, $\mathrm{K}_1 := \{0.5, 1, \ldots, 5\}$ and $\mathrm{K}_2 := \{0.1, 0.2, \ldots, 1.6\}$, respectively, and obtain a combined set of parameter configurations $\mathrm{B} \times \mathrm{K}_1 \times \mathrm{K}_2$ with 1760 ($= 11 \cdot 10 \cdot 16 = |\mathrm{B}| \cdot |\mathrm{K}_1| \cdot |\mathrm{K}_2|$) configurations. Furthermore, they use the instance characteristics-related calculation procedures proposed by Lee and Pinedo (1997) to determine additional $\tilde{\kappa}_1$ and $\tilde{\kappa}_2$ values. Since the procedures do not output a value for β, the estimated $\tilde{\kappa}_1$, $\tilde{\kappa}_2$ are combined with all β values to 11 new parameter combinations. In total, the BATCS-b heuristic evaluates 1771 parameter configurations to obtain best solutions.

The evaluation of the BATCS-b heuristic shows that it has a competitive solution quality compared to other heuristics. However, the multi-start full-grid search with 1771 parameter configurations results in very high computation times if the number of jobs increases (cf. Gahm et al., 2022b). Because of this lack of efficiency, we propose to use machine learning models to determine a single best parameter configuration (or at least a much smaller grid) to increase efficiency and simultaneously preserve solution quality.

2.3 Literature review on machine learning enhanced scheduling methods

Using machine learning models to improve the efficiency and solution quality of heuristics in machine scheduling dates back several decades. Since then, a large body of literature was published that goes in different methodological directions. We do not aim to give a comprehensive survey on applications of machine learning models within scheduling but give the reader a brief introduction in directions closely related to our approach to allow for categorizing this paper. We categorize previous research into approaches that: (1) choose the proper scheduling method from a set of methods, (2) predict the objective value achieved by a specific scheduling method, and (3) determine "good" parameters for a solution method.

To avoid misunderstandings in what follows, we want to clarify the ambiguous terms online and offline for machine learning models. We say that a model's learning mode is

"online" when the learning happens while solving a single problem instance, and "offline" when the model performs the learning on a set of already solved instances before solving the actual problem instance. One major benefit of offline models is the faster response time during model application since the learning task is already performed beforehand.

1. The problem of choosing a scheduling method is often seen in dynamic production environments (e.g., when the job arrival times are stochastic). Since these problems usually arise in a time-critical environment, offline-trained machine learning models are frequently used: cf., e.g., Piramuthu et al. (1994), Lee et al. (1997a), Priore et al. (2006), Mouelhi-Chibani and Pierreval (2010), Shiue et al. (2012), Priore et al. (2018), Stricker et al. (2018), Waschneck et al. (2018), or Park et al. (2021). These works chose a proper scheduling method based on the characteristics of the problem instance to solve either by machine learning models (often an inductive decision tree or an artificial neural network NN) or a genetic algorithm (which can be classified as a reinforcement learning approach in a broader sense). Even if not as prominent, method selection approaches are also considered in static production environments: cf., e.g., Azadeh et al. (2012), Azadeh et al. (2013), Shiue et al. (2018), or Lin et al. (2019). Since we do not focus on choosing the best method from a set of existing methods for a problem instance, we do not discuss these works in more detail.
2. Commonly in scheduling literature, objective value predictions are used to predict a makespan to use it as a "metric" for fast order acceptance decisions or delivery date confirmations (cf., e.g., Akyol, 2004; Raaymakers & Weijters, 2003; Shafaei et al., 2011). Examples of objective value predictions not directly related to scheduling problems are Neuenfeldt Júnior et al. (2019), which predict objective value to estimate lower bounds for 2D strip-packing problems, and Gahm et al. (2022a), which predict heights of 2D nesting problems to check constraints in a hierarchical production planning problem.
3. Literature in the third category, determining "good" parameters for a solution method for scheduling related problems, is of most interest for the purpose of our approach. Lee and Pinedo (1997) developed rules through experiments that determine values for both look-ahead parameters of the ATCS-rule. They use a curve fitting method that takes due date tightness, due date range, setup time severity, and a job-machine factor of an instance to obtain proper values for the parameters κ_1 and κ_2. Kim et al. (1995) proposed a NN to predict proper look-ahead parameters for the ATC(S) rule based on problem characteristics (such as due-date tightness and due-date range). Their computational tests show that the proposed approach outperforms the ATC rule with a fixed parameter configuration (i.e., the heuristic computes every instance with the same heuristic parameter configuration). Also, their extension for the scheduling problem with sequence-dependent setup times outperforms the method based on curve-fitted parameters by Lee and Pinedo (1997) (note that Lee et al. first published their work as a technical report in 1992 and in 1997 as a journal article). Park et al. (2000) also use NNs to predict suitable values for the two look-ahead parameters of the ATCS rule for a scheduling problem with identical parallel machines and sequence-dependent setup times. The authors propose using two independent NNs to predict κ_1 and κ_2 independently and investigate two feature vectors with four and five features, respectively. Their results reveal a positive influence of the new setup time-related feature and thus emphasize the importance of feature engineering. Mönch et al. (2006) extend the ATC rule for a parallel-batch scheduling problem. Like the BATCS-b heuristic, they aggregate each job's ATC value in a batch by summing the individual ATC values and use the sum to prioritize each batch. For determining an appropriate value for the single look-ahead parameter κ, they train an inductive decision

tree and a NN (multi-layer perceptron) with a feature vector consisting of five instance characteristics. They could show in a computational study with generated instances that a successful choice of the look-ahead parameter via machine learning models is possible. El-Bouri (2012) uses a multiple regression for determining a machines proximity factor in a composite dispatching rule to solve a dynamic flow shop problem. In his computational study, the proposed method outperforms (or is at least as good) six other methods that are typically favored for tardiness-related objectives. Heger et al. (2016) study dynamically changing (e.g., new job arrivals, stochastic processing times, or machine breakdowns) flow shop problems with sequence-dependent setup times. Due to the changing nature of the problem, they dynamically switch the heuristic depending on the current situation. For that, the authors use the ATCS rule and adjust the parameter κ_1 and κ_2 according to the current state. A Gaussian process regression model is trained offline on simulation runs to estimate the best parameter configuration for κ_1 and κ_2 online. Their computational study shows the superiority of the presented approach compared to fixed parameters. Shahrabi et al. (2017) use reinforcement learning to predict proper parameters for a variable neighborhood search procedure in a dynamic job shop environment. They could report that their approach outperforms priority rule-based heuristics and a general variable neighborhood search. In addition, the reinforcement learning approach leads to ongoing learning over time.

Similar to the previous contribution, Heger and Voss (2021) use the ATCS rule to solve a flexible flow shop in a dynamic environment and use reinforcement learning to find the parameters for the ATCS rule. Involving reinforcement learning in the solution methods enables the advantages of online training. This advantage can be observed in the evaluation, in which the authors' approach reduces the mean tardiness by up to 5%.

The analysis of the literature on determining "good" parameters shows that they all use a single estimated parameter configuration when the solution method makes a scheduling decision. However, since parameter estimates are by definition uncertain, it seems more suitable to use a few most appropriate parameter configurations rather than a single one. Therefore, in contrast to the existing literature, we propose to use a (small) set of most appropriate parameter configurations. Hereby, the challenge is to estimate which parameter configurations perform best and how many of them should be used in a reduced grid search to compute sufficiently good solutions in a reasonable time.

In addition, due to our ranking-based "parameter tuning" approach, only the relative performance between parameter configurations must be estimated (and not the absolute one) and thus, the estimation is more robust regarding prediction errors.

2.4 Summary

The $P|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$-problem can be efficiently solved by the BATCS-b multi-start heuristic if the number of jobs is not too large. If the number of jobs increases, the full grid search with 1771 parameter configurations becomes inefficient and a smaller grid or even a single parameter configuration should be used. To determine appropriate parameter configurations, machine learning models can be used. Hereby, several aspects must be considered:

First, the machine learning model and/or the training data preparation must consider the fact that different parameter configurations can lead to the best-known objective value.

Second, the three parameters of the BATCS-b heuristic are not independent and thus, complete parameter configurations should be predicted and not single values (as for instance

done by Park et al., 2000). As multiple parameter configurations can lead to the same best objective value, considering the parameter interdependency is especially important. Consider, for example, the following situation in which the parameter configurations (1.2, 2.5) and (2, 0.8) lead to the same best-known objective value for a problem instance. If the dependency is not considered, the machine learning models could output values like (1.2, 0.8), which are good predictions if considered independently but (may) lead to bad solution when applied in the heuristic. Therefore, the used machine learning models should incorporate the interdependent nature of the heuristic parameters.

Third, learning should take place offline to achieve high efficiency; thus, online learning approaches are not appropriate.

Fourth, because priority rule-based heuristics (like the BATCS-b heuristic) are sensitive to appropriate parameters and (parameter) predictions are uncertain by definition, not only a single prediction-based parameter configuration should be used but mechanisms to determine a (small) set of most suitable parameter configurations should be developed. For that, the trade-off between solution quality and computation time must be considered.

Due to these aspects, we propose a machine learning enhanced grid search that uses a ranking of parameter configurations (determined by estimated objective values) and investigates different ranking application strategies for determining grids. Hereby, the machine learning model considers the interdependency between the parameters as we use instance characteristics and parameters as features for the predictions.

3 Machine learning enhanced grid search (MLGS)

The developed machine learning enhanced grid search (MLGS) approach must not only consider the previously discussed aspects but also must be applicable to any type of problem instance (i.e., a suitable generalization of the machine learning model is required) and provide a reasonable trade-off between solution quality and computation time. To achieve this, our approach consists of several components depicted in Fig. 1 and described in the following subsections.

First, to provide a sufficient data basis for the machine learning model, an exhaustive diverse set of serial-batch problem instances must be prepared and solved by the BATCS-b heuristic. For the machine learning, a problem instance must be converted into a numerical representation. To that, the feature engineering provides two feature vectors with different elements. In addition to the problem instance-related features, the three parameters of the BATCS-b heuristic must be part of the vectors to enable the prediction of their performance. Because training the machine learning model with all combinations of instances and parameter configurations from the full grid search is not manageable, we present different "subsampling" strategies to choose suitable subsets of parameter configurations for speeding up the training. The combination of a subsampling strategy and one of the feature vectors is called a "data pipeline configuration" and we investigate different combinations. The data provided by one of the pipelines form the input of the machine learning model. From our previous summary in 2.4, we conclude that the only requirement for selecting an appropriate machine learning model is that it must be capable of performing regressions. As machine learning models' accuracy highly depends on their hyperparameters, we recommend performing a comprehensive hyperparameter tuning with one or more selected machine learning models. The tuned models with the highest accuracy are then used to estimate the performance (objective value) of all parameter configurations (of the full grid) for a given

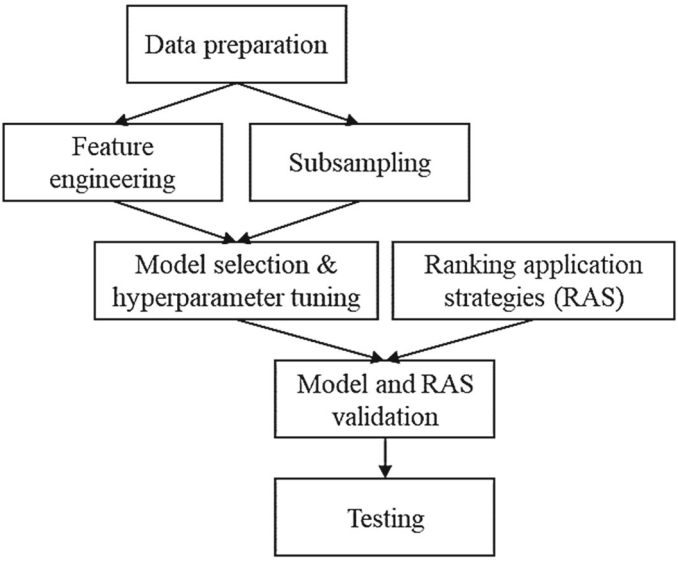

Fig. 1 Components of the machine learning enhanced grid search

instance. Based on these estimations, a ranking of the parameter configurations is created. To use this ranking in a most suitable way regarding solution quality and efficiency, different "ranking application strategies" are proposed. In the validation, combinations of data pipeline configurations, tuned models, and ranking application strategies are used to evaluate the performance of the resulting, differently parametrized BATCS-b-MLGS variants. Finally, the most promising BATCS-b-MLGS variants are evaluated by testing their performance on problem instances never used before in the development process.

3.1 Data preparation

The accuracy of a machine learning model depends on many factors (e.g., model type, feature engineering, and hyperparameters). One factor that is hard to compensate by the remaining components is the data set used for training the model. In many applications, providing a "good" data set is the major challenge when developing machine learning models. A fundamental assumption in machine learning theory is that both training and test data points (i.e., data points that have to be predicted) underlie identical and independent distributions (c.f., Bishop, 2006) and represent the "real world". Therefore, for machine learning, there exists an indispensable need of selecting a training data set representative for the test data set. Besides the training data sets' representativeness, the data sets' size also plays an essential role in obtaining an accurate Machine learning model. If the dependency between input and output of the learning problem is complex, a model with high capacity is needed (cf., Hastie et al., 2017 for model capacity/complexity). A high capacity in model-based learning (e.g., neural networks, linear regression, decision trees) often means that the model possesses many parameters (weights). Unfortunately, if the number of parameters is relatively high compared to the size of the data set and no penalizing terms for high model complexity are considered, the model is capable of "memorizing" the training data instead of finding a "generalization".

This well-known phenomenon would lead to highly accurate predictions on the training data but large errors on predictions for the test data (i.e., the model has a high variance, cf., bias-variance trade-off). So, a common way to combat this issue is to add a penalizing term on a model's complexity or use a large data set.

With these considerations, we build our data set I for the machine learning on a large set of instances that is provided for download at Mendeley data by Gahm (2022). This data set consists of 93,360 generated scheduling instances for the $P|sb,\ if^*,\ b,\ a_j,\ s_{fg}|wT$-problem. Those problem instances are divided into 18,672 instance classes, whereby each instance class contains five instances. The instance classes result from different combinations of instance attributes and thus provide a vast variety of instances. The most important instance attributes are the number of jobs $n \in \{15, 30, 60, 100, 200, 400, 800, 1600, 3200\}$, the number of machines $m \in \{1, 3, 4, 5, 10, 20\}$, and the number of incompatible job families $q \in \{3, 5, 10, 20, 40\}$. Further attributes are, for instance, setup time severity, tardiness factor, and due date range. These attributes primarily define how the job characteristics (e.g., processing times, due dates) are drawn from a distribution. More details on the data set can be found in the online description.

As the data set provides a high degree of diversity in its problem instances (e.g., number of jobs ranges from 15 to 3200, the number of machines from 1 to 20, and different types of distributions for drawing further job characteristics), we assume the requirements on the data set as described above are fulfilled. To make our study reproduceable for the scientific community, we did not integrate any real-world instances which could not be made available due to confidentiality issues. However, showing the effectiveness of our approach on this variety of problem instances justifies its applicability to real-world problems since manufacturers usually do not have such a high fluctuation in the attributes (e.g., the number of machines is mostly fixed, and the number of jobs does not vary as much as in the data set).

As strongly recommended for machine learning, different subsets of the total available data set are used for different development phases. Usually, 80% of the data for training and validation and 20% for testing. Therefore we split the complete set of scheduling instances into subsets and assign the different instances per instance class to different subsets: $D_P \subseteq I$ (with.

$P \subseteq \{1, .., 5\}$) indicates which of the five instances per class D_P contains (e.g., subset $D_{\{1,2\}}$ contains the first and the second instance of every instance class).

All 93,360 scheduling instances have been solved with the BATCS-b heuristic (with the full grid search that uses 1771 parameter configurations), and thus, 165,340,560 samples are available in total.

3.2 Feature engineering

The first element of a data pipeline configuration 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 1 and partly taken or inspired by literature (cf., Park et al., 2000; Mönch et al., 2006).

Most of the features are self-explanatory, but the approximated makespan $\tilde{M}$ requires further explanations. This feature needs the approximated total setup time $tst = tns \cdot \bar{s}$, which again has to compute the approximated total number of setups $tns = n/\tilde{n}_b$, the mean setup time $\bar{s}$, the approximated mean number of jobs per batch $\tilde{n}_b = \lceil bc/\bar{r} \rceil$, and the

Table 1 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
$\tilde{M}$	*Approximated makespan (cf., (2))
$\tilde{T}$	Tardiness factor: $\tilde{T} = 1 - \overline{d} \Big/ \tilde{M}$ (with mean of all due dates $\overline{d} = \sum_{j \in J} d_j \Big/ n$); cf. e.g., Park et al. (2000) or Mönch et al. (2006)
$\tilde{R}$	Due date range: $\tilde{R} = \left(\max\{d_j \mid \forall j \in J\} - \min\{d_j \mid \forall j \in J\}\right) \Big/ \tilde{M}$; cf., e.g., Park et al. (2000)
$\tilde{N}$	Approximated number of jobs per batch: $\tilde{N} = bc \Big/ \left(\sum_{j \in J} cr_j \Big/ n\right)$
$\tilde{B}$	Approximated number of batches: $\tilde{B} = n/\tilde{N}$
$\tilde{F}$	Batch machine factor: $\tilde{F} = \tilde{B} \Big/ m$; cf. Mönch et al. (2006) for a similar feature and also the job machine factor used by Park et al. (2000)
$\tilde{E}$	Setup time severity factor: $\tilde{E} = \left(\sum_{f \in F} \sum_{f' \in F} s_{f,f'} \Big/ q\right) \Big/ \tilde{P}$ with the approximated mean batch processing time $\tilde{P} = \tilde{N} \cdot \left(\sum_{j \in J} p_j \Big/ n\right)$; cf. Park et al. (2000)
$\tilde{S}$	Setup times range: $\tilde{S} = \left(\max\{s_{f,f'} \mid \forall f, f' \in F\} - \min\{s_{f,f'} \mid \forall f, f' \in F\}\right) \Big/ \max\{p_j \mid \forall j \in J\}$; cf. Park et al. (2000)

approximated mean batch capacity requirement $\overline{r} = \frac{1}{n} \sum_{j \in J} cr_j$. With these approximations, one can compute the makespan approximation according to (Gahm et al., 2022b):

$$\tilde{M} = (n \cdot \overline{p} + tst) \Big/ m \tag{2}$$

The AF-vector consists of 85 features. Five features are identical to the first five features of Table 1 (indicated by an asterisk) and the other 80 are calculated by eight basic instance characteristics and ten aggregation functions. The basic instance characteristics 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 ($cr_j \Big/ p_j$), job due date to weight ratios ($d_j \Big/ w_j$), 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).

For the actual prediction task, we extend the CF- and AF-vectors by candidate parameter configuration, so in total, the CF and AF-vectors have 15 and 88 features, respectively.

3.3 Subsampling

To achieve a sufficient generalization of the machine learning model, we must use the complete range of scheduling instances for training. At the same time, we must ensure that the training effort remains manageable. Therefore, we use the scheduling instance subset $D_{\{1,2\}}$

for the initial model training phase. However, using this scheduling instance subset with 37,344 instances results in 66,136,224 data points (due to the BATCS-b grid search with 1771 parameter configurations). Although innovations in hardware (e.g., parallelized computing) and theory (e.g., using rectifying linear units to overcome the vanishing gradient problem) made it possible to train large and complex models with large data sets, the computational efforts for learning with this number of data points becomes unmanageable high (as preliminary experiments have shown).

To reduce learning efforts, we introduce subsampling strategies that are developed to limit the number of parameter configurations (per scheduling instance) used for the learning but achieve sufficient prediction accuracy. Hereby, we differentiate between the set of parameter configurations that achieve best-known solutions ("best configurations", BC) and the set that contains those that lead to solutions with higher (worse) objective values ("worse configurations", WC). Because there can be more than one parameter configuration in set BC, we define a special representative that is in the "center" of all parameter configurations in BC. The procedure to determine this "central best configuration" (cBC) is described by the following pseudo-code:

determineCentralBestConfiguration$(\ BC = \{(\beta^1, \kappa_1^1, \kappa_2^1), (\beta^2, \kappa_1^2, \kappa_2^2), ...\}\ )$

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

$\beta' := \underset{\beta \in \text{B}(BC)}{\arg\min} \left\{ \left| \beta - \overline{\beta} \right| \right\}$ // get closest "original" value from all β in BC to $\overline{\beta}$

$\overline{\kappa}_1 := \text{mean}\{\kappa_1^1, \kappa_1^2, ... \mid \forall \kappa_1^j \in BC \text{ with } \beta^j = \beta'\}$ // get mean of all κ_1-values in all parameter // combinations in BC having $\beta = \beta'$

$\kappa_1' := \underset{\kappa_1 \in \text{K}_1(BC)}{\arg\min} \left\{ \left| \kappa_1 - \overline{\kappa}_1 \right| \right\}$

$\overline{\kappa}_2 := \text{mean}\{\kappa_2^1, \kappa_2^2, ... \mid \forall \kappa_2^j \in BC \text{ with } \beta^j = \beta' \wedge \kappa_1^j = \kappa_1'\}$

$\kappa_2' := \underset{\kappa_2 \in \text{K}_2(BC)}{\arg\min} \left\{ \left| \kappa_2 - \overline{\kappa}_2 \right| \right\}$

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

The sequence of the calculation ($\beta \rightarrow \kappa_1 \rightarrow \kappa_2$) is used as we observed that the performance of the heuristics is generally more sensitive regarding β compared to the two other parameters and that it is more sensitive to κ_1 than to κ_2.

In summary, three types of parameter configurations (cBC, BC, and WC) are available to perform subsampling on parameter configurations.

In the proposed subsampling strategies, we generally limit the number of parameter configurations to ten, and thus, we get different training data sets containing 373,440 data points. The strategies differ in the number of samples randomly drawn (by a uniform distribution) from BC and WC and if cBC is used. Table 2 depicts the eight subsampling strategies.

These strategies are defined to analyze the impact of the cBC and of the WC configurations on the accuracy of the models.

Table 2 Subsampling strategies

Subsampling strategy	Number of samples from		
	cBC	BC	WC
[1,9,0]	1	9	0
[0,5,5]	0	5	5
[1,4,5]	1	4	5
[0,2,8]	0	2	8
[1,2,7]	1	2	7
[0,1,9]	0	1	9
[1,0,9]	1	0	9
[0,0,10]	0	0	10

The eight resulting sample sets (named according to the subsampling strategies) and the two feature vectors are combined to 16 data pipeline configurations, for example, identified by *[0,1,9]-CF*.

3.4 Model selection and hyperparameter tuning

Multi-layered neural networks (NN) are commonly chosen as a machine learning model when high accuracy is needed, and interpretability is unimportant in the application. Since the interpretability of results is not necessary for our application, we use NN (from the tensorflow package in Python) for the prediction task. Of course, other machine learning models like regression trees or support vector machines could also be used, but the investigation of different models is out of the scope of this paper.

Besides the training of the weights itself, many hyperparameter decisions (e.g., number of neurons in the hidden layers, regularization, drop-out factor) must be made for the NN. Initial tests have shown that the complexity of NNs with two hidden layers is too restricted (a grid search returned NNs with the highest possible complexity, namely both hidden layers with 1024 neurons). Thus, we extended the NN by an additional third layer to avoid biasing by too simple models. This additional layer enlarges the number of weights in the NN and thus its capacity to represent more complex structures. We perform a hyperparameter grid search for all combinations of hyperparameter values depicted in Table 3 to identify the best hyperparameter setting for the model. To measure the error in this phase, we use the root mean squared error to assess the deviation between the predicted and actual objective values.

Table 3 Hyperparameter space for NN

Hyperparameter	Values
Number of neurons in hidden layer 1	128, 256, 512, 1024
Number of neurons in hidden layer 2	128, 256, 512, 1024
Number of neurons in hidden layer 3	128, 256, 512, 1024
Drop out factor	0.1, 0.2, 0.3
L2 regularization factor	0.001, 0.002, 0.010

The learning rate for the Adadelta optimizer is set to 1.0 and the number of epochs to 50 during the hyperparameter tuning phase.

To obtain models with highest prediction accuracy, we tune these hyperparameters (defining 576 combinations) with data set $D_{\{3\}}$ for each of the 16 data pipelines individually.

After identifying the best hyperparameter setting for each pipeline configuration, we train the machine learning models on 60% of the data ($D_{\{1,\,2,\,3\}}$).

3.5 Ranking application strategies

With the trained and tuned models, we are able to predict the objective value for each parameter configuration of the BATCS-b grid and a given scheduling instance i. Due to the immanent uncertainty of predictions, we are not only interested in the best predicted configuration but use the prediction to create a complete ranking R_i of parameter configurations of the BATCS-b grid for instance i. To handle the uncertainty and simultaneously achieve a sufficient trade-off between solution quality and computation times, we propose different ranking application strategies to form a final grid FG that is much smaller than the original grid. The BATCS-b-MLGS heuristic then uses this reduced FG_i to perform a multi-start grid search.

The first strategy *B1* assumes a perfect prediction and only uses the first (best) parameter configuration pc_1 of the ranking R_i to define FG_i. This strategy can be seen as a reference for the following more sophisticated ones.

The second strategy *B1-G* uses the best prediction pc_1 and stretches a grid around this configuration. This strategy aims to compensate for a prediction's deviations if it is close to a best parameter configuration but does not match exactly. Since the job size n has the most substantial influence on the computation time of BATCS-b, we vary the neighborhood (grid) size based on the number of jobs n of an instance to control the computation time. More precisely, we define a grid size parameter $\zeta \in \{1, 2, 3\}$ that indicates how many neighboring values ($(2\zeta + 1)^3$) in the reduced grid (below and above a predicted heuristic parameter) are added to the neighborhood (based on B, K_1, and K_2). For example, if the model predicts $\widehat{pc} = (\hat{\beta}, \hat{\kappa}_1, \hat{\kappa}_2) = (0.75, 1.0, 1.5)$ and we have $\zeta = 2$, the neighborhood of $\hat{\beta}$ is $\{0.6, 0.7, 0.75, 0.8, 0.85\}$, of $\hat{\kappa}_1$ is $\{0.8, 0.9, 1.0, 1.1, 1.2\}$, and of $\hat{\kappa}_2$ is $\{0.5, 1.0, 1.5, 2.0, 2.5\}$. So, in total, when ζ is equal to 1, 27 parameter configurations are in the neighborhood grid (NG_{pc}), 125 when $\zeta = 2$, and 343 when $\zeta = 3$. For different numbers of jobs, the grid size parameter ζ is defined by $\zeta = 3$ for $n_i \in [0,\ 100)$, $\zeta = 2$ for $n_i \in [100,\ 1000)$, and $\zeta = 1$ for $n_i \in [1000,\ 3200]$. This means that the neighborhood (or reduced grid as we refer to) decreases if the job size increases. Note that these settings have been developed based on preliminary tests to create final grids with desired sizes that achieve a sufficient solution quality and keep computation times below 240 s for instances with 3,200 jobs. Since the instance characteristic-related calculation procedures proposed by Lee and Pinedo (1997) to determine additional $\tilde{\kappa}_1$ and $\tilde{\kappa}_2$ values have shown their benefits, we use the 11 parameter configurations combing $\tilde{\kappa}_1$ and $\tilde{\kappa}_2$ with β to form LPG_i (Lee-Pinedo grid for instance i). Summarizing, strategy B1-G defines a final grid FG consisting of NG_{pc_1} and LPG_i.

In the third strategy *Bx*, we not only use the overall best ranked parameter configuration but add the first x parameter configurations $(pc_j)_{1 \le j \le x}$ from ranking R_i to the final grid FG_i that also contains LPG_i. To achieve desired final grid sizes, we use $x = (2\zeta + 1)^3$ with ζ as defined before.

The fourth strategy *B(k)-G* creates neighborhood grids based on the best k parameter configurations from ranking R_i to from the final grid FG_i. This follows the idea of "exploring" the parameter space by increasing k. As we use the same approach to create neighborhood

grids as before, the size of the FG_i would become very large with k becoming larger and thus, computation times would also increase. To avoid this effect, we reduce the size of the neighborhood grids to $\left\lceil {}^{1}/_{k}(2\zeta+1)^3 \right\rceil$ (with ζ as determined before). Since the different neighborhood grids may contain duplicates, we create the neighborhood grids one after the other (NG_{pc_k} starting with $k = 1$), eliminate duplicated parameter configurations after creating one of the grids, and increment k until the desired size of the final grid is achieved. For the same reason, we skip those parameter configurations in the ranking to create neighborhoods which are already part of the final grid. The following pseudo-code illustrates this procedure:

getFinalGrid*("B(k)-G", R_i, n_i, k, LPG_i)*

$\zeta := getGridSizeParameter(n_i)$ // grid size parameter based on the number of jobs
$maxConfigs := (2\zeta+1)^3$ // desired final grid size
$FG := \emptyset$
for each $pc \in R_i$ **do**
  if $pc \notin FG \wedge |FG| < maxConfigs$ **then**
    $RG := getReducedGrid(pc, \zeta)$
    $RG := RG \setminus FG$ // remove duplicates
    $RG := reduceToBest(RG, R_i, \left\lceil {}^{1}/_{k}(2\zeta+1)^3 \right\rceil)$
    // reduce to the desired number of best configurations
    for each $pc \in RG$ **do**
      if $|FG| < maxConfigs$ **then** $FG := FG \cup \{pc\}$
    next;
  end if;
next;
return $FG := FG \cup LPG_i$

As one may notice, *B1-G* is a special case of *B(k)-G* with $k = 1$. Nevertheless, we want to distinguish between these two strategies since they are conceptually different. *B1-G* selects the best-ranked parameter configuration and invests all efforts to exploit the neighboring configurations. In contrast, *B(k)-G* aims to find a trade-off between exploration and exploitation by varying k. A higher k results in a higher exploration of the parameter configuration space but the reduced grids of the parameter configurations are "exploited" less. This behavior is enforced by the decision to only span reduced neighborhood grids on top-ranked parameter configurations that are not yet in the final grid. In the validation, we use the *B(k)-G* strategy with $k \in \{3, 5, 9\}$ to investigate the trade-off between exploration and exploitation.

3.6 Validation and testing

In the validation, the twelve data pipeline configurations are combined with the six ranking application strategies *B1*, *B1-G*, *Bx*, *B(3)-G*, *B(5)-G*, and *B(9)-G* to determine the most suitable combinations. Hereby, a subset of scheduling instances of the previously unused data set $D_{\{4\}}$ is used. After the validation, the most promising data pipeline configurations are trained with data set $D_{\{1,2,3,4\}}$. In the final testing, the complete, preliminary unused data set $D_{\{5\}}$ is used to assess the performance of the most promising combinations identified by the validation.

4 Experimental results

The application of most machine learning methods requires a hyperparameter tuning; in our case, the tuning of the NN parameters described in the first subsection. In the following validation, we investigate the performance of the 16 pipeline configurations combined with the six ranking application strategies to determine the best-performing combinations in terms of solution quality and computation time. Finally, we compare the best nine BATCS-b-MLGS variants (i.e., the best three pipeline configurations combined with the best three ranking strategies) with BATCS-b.

For assessing 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., 2022b). This metric is used because the objective values of our weighted tardiness problem may be equal or very close to zero and setting the objective values in relation to a value close to zero distorts the interpretability of other metrics. For that, we define a set S of solution methods and are interested in the objective value $ov_{i,s}$ of the solution method $s \in S$ for an instance i. For instance, $ov_{i,BATCS-b}$ is the best objective value obtained by the BATCS-b heuristic (with all 1771 parameter configurations) or $ov_{i,\,BATCS-b-MLGS\ ([0,5,5],\,AF,\,B(5)-G)}$ is the objective value obtained after running the BATCS-b-MGLS heuristic (with pipeline *[0,5,5]-AF* and ranking application strategy *B(5)-G*). The relative improvement regarding instance i and solution method s to the worst objective value $ov_i^{\text{WORST}} = \max_{s \in S}\{ov_{i,s}\}$ is calculated as follows: if $ov_i^{\text{WORST}} > 0$: $RIW_{i,s} = (ov_i^{\text{WORST}} - ov_{i,s}) \cdot 100 \big/ ov_i^{\text{WORST}}$; otherwise: $RIW_{i,s} = 0$. We obtain the $MRIW_s$ for solution method s by calculating the mean of all $RIW_{i,s}$ for a given set of instances.

4.1 Hyperparameter tuning

Based on the hyperparameter space for the NN depicted in Table 3, we perform a full grid search for all twelve pipeline configurations to find the best hyperparameter setting for the NNs. For the initial training phase, we use instance subset $D_{\{1,\,2\}}$ with 40% of the data and in the hyperparameter tuning phase, we use instance subset $D_{\{3\}}$ with 20% of the data to test the performance of the machine learning models with an epoch size of 50 for the NN. Table 4 shows the hyperparameter settings that minimize the expected loss referring to the predicted objective values and the actual objective values. We use the mean squared error (MSE) to quantify the excepted loss because of its advantageous numerical properties, and we want to penalize higher deviations in the predictions more severely.

The hyperparameter tuning was conducted on three workstations and training all 576 hyperparameter combinations (cf. Table 3) for a single pipeline configuration takes 331 h on average. Differences in computation time between AF and CF vectors are not remarkable (334.9 h and 327.1 h, respectively). In total, all hyperparameter tuning efforts sum up to 5,297 h.

Even though finding the best-performing NN architecture is a very effortful task, it only has to be performed once for a manufacturing company and the architecture decision obtained can be used in the future (unless there are major changes in the structure of the problem instances of the company).

Table 4 Tuned hyperparameters

Pipeline configuration	Number of neurons in hidden layer 1	Number of neurons in hidden layer 2	Number of neurons in hidden layer 3	Drop out factor	L2 regularization factor
[1,9,0]-CF	1024	512	1024	0.1	0.002
[1,9,0]-AF	1024	256	512	0.1	0.001
[0,5,5]-CF	1024	1024	512	0.1	0.001
[0,5,5]-AF	1024	128	1024	0.1	0.002
[1,4,5]-CF	512	512	1024	0.1	0.002
[1,4,5]-AF	1024	256	512	0.2	0.010
[0,2,8]-CF	1024	1024	1024	0.1	0.002
[0,2,8]-AF	1024	256	512	0.1	0.010
[1,2,7]-CF	1024	1024	128	0.1	0.002
[1,2,7]-AF	512	512	1024	0.1	0.010
[0,1,9]-CF	1024	1024	1024	0.1	0.002
[0,1 9]-AF	1024	128	1024	0.1	0.010
[1,0,9]-CF	1024	1024	512	0.2	0.002
[1,0,9]-AF	1024	512	512	0.1	0.010
[0,0,10]-CF	512	1024	1024	0.1	0.002
[0,0,10]-AF	1024	1024	256	0.1	0.002

4.2 Validation

To evaluate the performance of the 16 data pipelines combined with the six ranking application strategies, we use 162 randomly selected instances from set $D_{\{4\}}$ (18 instances for each value of n). Since the hyperparameter tuning is completed in this stage, we use the set $D_{\{1,2,3\}}$ to train the NN. Table 5 shows the MRIWs [%] and computation times (CT; [s]) for BATCS-b-MLGS for each pipeline configuration and ranking application strategy combination.

The validation results depicted in Table 5, shows that the training of the NN only by the parameter configurations calculating best-known solutions (*[1,9,0]-CF* and *[1,9,0]-AF*) is outperformed by all the other ones. Based on this observation, we conclude that the model needs data points with"worse" objective values to achieve a more accurate ranking. An explanation for this phenomenon could be that the model needs to reduce the uncertainty for worse performing parameter configurations in order not to overestimate their performance.

Another interesting observation is that the pipeline configurations *[0,0,10]-CF* and *[0,0,10]-AF* achieve a good solution quality without learning with parameter configurations achieving best-known solutions. This supports the previous conclusion that "worse" solutions are important for the training of the models. Furthermore, we can see that the combination of parameter configurations calculating best-known solutions and randomly selected parameter configurations seem to be favorable and that the integrating the "central best configuration" within the training data provides some benefits.

We also observed that pipeline configurations with the AF-vector outperform the CF-vector pipeline configurations. From this, we can conclude that the NN is able to automatically "compute" the complex features from the CF-vector (which are based on problem specific expert knowledge).

Table 5 Key figures by pipeline configuration and ranking application strategy

Pipeline configuration	Ranking application strategy						Mean
	B1	*B1-G*	*Bx*	*B(3)-G*	*B(5)-G*	*B(9)-G*	
[1,9,0]-CF	23.16	66.34	66.59	66.55	66.68	66.78	59.35
[1,9,0]-AF	25.49	66.58	67.01	66.78	66.85	66.90	59.93
[0,5,5]-CF	55.49	68.61	69.34	69.13	69.28	69.33	66.86
[0,5,5]-AF	59.00	68.69	69.24	69.06	69.09	69.22	**67.38**
[1,4,5]-CF	57.13	68.58	69.02	69.03	68.99	69.04	66.96
[1,4,5]-AF	56.66	68.17	68.89	68.83	69.03	69.00	66.76
[0,2,8]-CF	56.21	68.51	69.35	69.16	69.17	69.18	66.93
[0,2,8]-AF	56.88	68.03	68.66	68.67	68.55	68.64	66.57
[1,2,7]-CF	56.97	67.91	69.41	68.89	69.14	69.45	66.96
[1,2,7]-AF	58.14	68.31	69.11	69.13	69.30	69.41	**67.23**
[0,1,9]-CF	59.15	68.04	68.95	68.53	68.92	68.92	67.08
[0,1 9]-AF	56.01	68.03	68.89	68.59	68.90	68.85	66.54
[1,0,9]-CF	55.16	67.44	68.76	68.21	68.69	68.66	66.15
[1,0,9]-AF	58.60	68.58	69.38	69.30	69.36	69.38	**67.43**
[0,0,10]-CF	56.31	68.10	69.09	69.00	69.16	69.10	66.79
[0,0,10]-AF	55.57	67.09	68.29	67.90	68.26	68.29	65.90
Mean MRIW	52.87	67.94	**68.75**	68.55	**68.71**	**68.76**	
Mean CT [s]	2.8	30.7	30.1	30.9	30.7	30.5	

Summarizing, the pipeline configurations *[0,5,5]-AF*, *[1,0,9]-AF*, and *[1,2,7]-AF* and the ranking application strategies *Bx*, *B(5)-G*, and *B(9)-G* achieved the highest and most robust solution qualities and thus, we use those in the final testing.

4.3 Testing

The final performance testing of the proposed BATCS-b-MLGS heuristic with different pipeline configurations and ranking applications strategies is based on test data set $D_{\{5\}}$ comprising 18,672 instances never used before in the development process. Furthermore, the NNs are trained on the data set $D_{\{1,2,3,4\}}$ with an epoch size of 200 to achieve a more accurate prediction. With the larger data set and higher epoch size, the three best-performing pipeline configurations *[0,5,5]-AF*, *[1,0,9]-AF*, and *[1,2,7]-AF* need training times of 2.1 h, 5.1 h and 5.2 h, respectively. The differences in the training time result from the different hyperparameter choices (cf., Table 4), leading to different number of weights to be learned ($1024 \cdot 128 + 128 \cdot 1024 = 262{,}144$; $1024 \cdot 512 + 512 \cdot 512 = 786{,}432$; and $512 \cdot 512 + 512 \cdot 1024 = 786{,}432$; respectively).

Table 6 depicts the MRIWs of the nine best BATCS-b-MLGS variants and BATCS-b. For the nine BATCS-b-MLGS variants, only the mean computation times are listed as deviations

Table 6 Performance comparison

n	MRIW [%]										CT [s]	
	BATCS-b-MLGS with									BATCS-b		
	[0,5,5]-AF			*[1,0,9]-AF*			*[1,2,7]-AF*				BATCS-b-MLGS	BATCS-b
	Bx	*B(5)-G*	*B(9)-G*	*Bx*	*B(5)-G*	*B(9)-G*	*Bx*	*B(5)-G*	*B(9)-G*			
15	3.28	3.07	3.17	4.75	4.38	4.69	4.88	4.78	**4.97**	7.93	0.05	0.21
30	4.49	4.75	4.40	4.71	4.23	4.50	**5.05**	4.72	4.96	7.94	0.12	0.53
60	4.37	4.36	4.50	2.62	2.35	2.46	**5.01**	4.83	4.99	6.55	0.28	1.38
100	5.44	5.41	5.36	4.33	4.10	4.28	6.63	6.56	**6.68**	10.14	0.27	3.90
200	4.69	4.60	4.61	2.97	2.87	2.94	**5.84**	5.76	5.72	8.19	0.77	12.04
400	3.64	3.47	3.59	3.20	3.14	3.09	**3.71**	3.64	3.68	6.33	2.80	47.29
800	1.52	1.52	1.62	1.04	0.89	1.00	**1.82**	1.65	1.71	2.27	22.27	322.17
1,600	2.14	2.25	2.19	2.49	2.38	**2.52**	2.32	2.26	2.24	4.42	30.49	1635.10
3,200	**4.05**	3.76	3.73	3.82	3.02	3.66	3.78	3.44	3.60	6.52	177.78	8938.72
Mean	4.11	4.06	4.07	3.30	3.10	3.22	**4.82**	4.71	4.78	7.28		

between their computation times result from operating system activities and not from their configurations (all have identical grid sizes related to n). We want to point out that the MRIW values in this stage are very different from the validation phase (Sect. 4.2) due to the new reference points of the MRIW metric. Since we have limited our final testing evaluation to the three best data pipeline configurations, the worst objective value (i.e., the highest for the weighted tardiness) among all solution methods for an instance is lower compared to the worst objective value in the validation stage. Of course, BATCS-b has the highest MRIWs because it runs the heuristic with all parameter configurations. The relatively low MRIW of BATCS-b of 7.28% over all instances indicates the overall good solution quality of the BATCS-b-MLGS variants (compared to BATCS-b).

The results in Table 6 show that BATCS-b-MLGS with the pipeline *[1,2,7]-AF* and ranking application strategy *Bx* achieves the highest mean solution quality among the nine variants. The small deviation by 2.46 percentage points between the mean MRIWs of BATCS-b-MLGS(*[1,2,7]-AF, Bx*) and BATCS-b illustrates the high solution quality. This value is all the more impressive when considering the computation times savings of about 89.2% regarding all kinds of instances and 98.0% regarding instances with 3,200 jobs (with a decrease of solution quality of only 2.74 percentage points). These results also emphasize the previous finding that the composition of the training data by the different types of parameter configurations (cBC, BC, and WC) influences the solution quality: *[1,2,7]* clearly outperforms the other pipelines.

In additionto the desired improvement in solution efficiency, the results show that different combinations of data pipeline configurations and ranking application strategies are preferable depending on the number of jobs (however, the performance differences are relatively low): For example, for instances with 3,200 jobs, BATCS-b-MLGS(*[0,5,5]-AF, Bx*) performs best and for instances with 1,600 jobs, BATCS-b-MLGS(*[1,0,9]-AF, B(9)-G*) performs best. For the practical application of our ranking base-based grid search in a manufacturing company, this means that depending on the number of jobs, the most preferred combination should be determined in a process as described and applied in this paper. For "small" instances with up to 400 jobs, the absolute time savings of our approach seems to be negligible for practical applications and thus, BATCS-b could be implemented.

5 Conclusions and avenues for further research

The literature on approaches that combine machine learning models with heuristic scheduling methods shows different methodological directions. Two directions are useful for the purpose of our paper, which is to select the most promising parameters for a heuristic: approaches that directly determine the parameters of a scheduling method and approaches that predict objective values. The analysis of the literature shows that none of the approaches in the literature consider a potential interdependence of parameters (as required for the BATCS-b heuristic) and/or do not consider the inherent uncertainty of single parameter predictions. Therefore, our approach fills a gap in the literature: The proposed machine learning enhanced grid search (applying multiple parameter configurations to cope with the uncertainty) uses a prediction-based ranking, taking into account interdependencies between parameters. Our extensive computational study shows that the BATCS-b-MLGS heuristic needs only a fraction of the computation time (-89.2%) needed for the full grid search used by BATCS-b and achieves a reasonably good solution quality (-2.46 percentage points lower compared to BATCS-b).

Based on the experimental results, we can report several interesting findings:

- The way in which the ranking is used by the ranking application strategy is of minor importance but using only the best parameter configuration is not recommended (solution quality decreases remarkably) as the uncertainty of the predictions is too high.
- To reduce the learning effort, we developed and analyzed different subsampling strategies to provide the data set for training. Regarding these strategies, we can report that the training of the NN should not only be based on the best-known solutions and that randomly selected "worse" solutions improve the accuracy of the resulting rankings. Furthermore, also not using best-known solutions at all leads to a reasonably good solution quality (which is particularly interesting when the computation of best-known solution is very effortful).
- In this context, it is also worth mentioning that the computation of a "central" best parameter configuration (based on all available best-known solutions) and its integration into the training data set is recommended.
- Depending on the characteristics of a problem instance (e.g., the number of jobs), different subsampling strategies achieve the best solution quality. Thus, the validation and testing process as applied in our study should be used to achieve the best results in real-world applications.

However, our analysis is not complete and could be extended in several ways: It should be investigated under which circumstances and application contexts the integration of "worse" sample solutions in the training positively impacts prediction accuracy. Also, other machine learning models capable of performing regressions (e.g., random forests or gradient boosted trees) could be analyzed and, on a higher level, analyzing the applicability of reinforcement learning approaches to the problem at hand and the impact on the computation time is interesting as we believe. Furthermore, our MLGS approach works on a discrete parameter space of 1771 parameter configurations. As it is generally capable of predicting objective values for "continuous" parameters, it is seeming to be worth to investigate using the prediction model to improve parameters in a more dynamic and flexible way. Furthermore, research on applying the MLGS approach to other parameterized heuristics would be interesting to show its general applicability. For population-based metaheuristics, the ranking could be used to compute promising initial solutions and also to determine "bad" solutions supporting the diversity of the initial population. One, rather evident, application of the performance prediction based ranking approach is to evaluate multiple heuristics/algorithms individually for every problem instance and only execute the top-ranked methods (algorithm selection based on Machine Learning).

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of 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.

References

Akyol, D. E. (2004). Application of neural networks to heuristic scheduling algorithms. *Computers & Industrial Engineering, 46*(4), 679–696. https://doi.org/10.1016/j.cie.2004.05.005

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. *International Journal of Production Research, 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. *The International Journal of Advanced Manufacturing, 67*(5–8), 1549–1561. https://doi.org/10.1007/s00170-012-4589-y

Bishop, C. M. (2006). Pattern recognition and machine learning. Springer. Online verfügbar unter https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

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

Gahm, C. (2022). Extended instance sets for the parallel serial-batch scheduling problem with incompatible job families, sequence-dependent setup times, and arbitrary sizes. Hg. v. V1. Mendeley Data (V1).

Gahm, C., Uzunoglu, A., Wahl, S., Ganschinietz, C., & Tuma, A. (2022). Applying machine learning for the anticipation of complex nesting solutions in hierarchical productiosn planning. *European Journal of Operational Research, 296*(3), 819–836. https://doi.org/10.1016/j.ejor.2021.04.006

Gahm, C., Wahl, S., & Tuma, A. (2022). Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes. *International Journal of Production Research, 60*(17), 5131–5154. https://doi.org/10.1080/00207543.2021.1951446

Hastie, T., Tibshirani, R., & Friedman, J. H. (2017). *The elements of statistical learning. Data mining, inference, and prediction. Springer series in statistics* (2nd ed.). Springer.

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. *International Journal of Production Research, 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. *International Journal of Production Research*. 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. *Computers & Operations Research, 110*, 208–219. https://doi.org/10.1016/j.cor.2018.06.002

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

Lee, C.-H. (2018). A dispatching rule and a random iterated greedy metaheuristic for identical parallel machine scheduling to minimize total tardiness. *International Journal of Production Research, 56*(6), 2292–2308. https://doi.org/10.1080/00207543.2017.1374571

Lee, C.-Y., Piramuthu, S., & Tsai, Y.-K. (1997). Job shop scheduling with a genetic algorithm and machine learning. *International Journal of Production Research, 35*(4), 1171–1191. https://doi.org/10.1080/002075497195605

Lee, Y.-H., Bhaskaran, K., & Pinedo, M. L. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions, 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. *European Journal of Operational Research, 100*(3), 464–474. https://doi.org/10.1016/S0377-2217(95)00376-2

Li, X., & Zhang, K. (2018). Single batch processing machine scheduling with two-dimensional bin packing constraints. *International Journal of Production Economics, 196*, 113–121. https://doi.org/10.1016/j.ijpe.2017.11.015

Lin, R., Li, W., & Chai, X. (2019). On-line scheduling with equal-length jobs on parallel-batch machines to minimise maximum flow-time with delivery times. *Journal of the Operational Research Society*. https://doi.org/10.1080/01605682.2019.1578626

Maecker, S., & Shen, L. (2020). Solving parallel machine problems with delivery times and tardiness objectives. *Annals of Operations Research, 285*(1–2), 315–334. https://doi.org/10.1007/s10479-019-03267-2

Mönch, L., Zimmermann, J., & Otto, P. (2006). Machine learning techniques for scheduling jobs with incompatible families and unequal ready times on parallel batch machines. *Engineering Applications of Artificial Intelligence, 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. *Computers & Industrial Engineering, 58*(2), 249–256. https://doi.org/10.1016/j.cie.2009.03.008

Neuenfeldt Júnior, A. N., Silva, E., Gomes, A. M., Soares, C., & Oliveira, J. F. (2019). Data mining based framework to assess solution quality for the rectangular 2D strip-packing problem. *Expert Systems with Applications, 118*, 365–380. https://doi.org/10.1016/j.eswa.2018.10.006

Park, J., Chun, J., Kim, S. H., Kim, Y., & Park, J. (2021). Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research, 59*(11), 3360–3377. https://doi.org/10.1080/00207543.2020.1870013

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

Piramuthu, S., Raman, N., & Shaw, M. J. (1994). Learning-based scheduling in a flexible manufacturing flow line. *IEEE Transactions on Engineering Management, 41*(2), 172–182. https://doi.org/10.1109/17.293384

Priore, P., La, F., David, D., Puente, J., & Parreño, J. (2006). A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems. *Engineering Applications of Artificial Intelligence, 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. *Computers & Industrial Engineering, 126*, 282–291. https://doi.org/10.1016/j.cie.2018.09.034

Raaymakers, W. H. M., & Weijters, A. J. M. M. (2003). Makespan estimation in batch process industries: A comparison between regression analysis and neural networks. *European Journal of Operational Research, 145*(1), 14–30. https://doi.org/10.1016/S0377-2217(02)00173-X

Shafaei, R., Rabiee, M., & Mirzaeyan, M. (2011). An adaptive neuro fuzzy inference system for makespan estimation in multiprocessor no-wait two stage flow shop. *International Journal of Computer Integrated Manufacturing, 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. *Computers & Industrial Engineering, 110*, 75–82. https://doi.org/10.1016/j.cie.2017.05.026

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. *International Journal of Production Research, 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. *Computers & Industrial Engineering, 125*, 604–614. https://doi.org/10.1016/j.cie.2018.03.039

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

Toksarı, M. D., & Toğa, G. (2022). Single batch processing machine scheduling with sequence-dependent setup times and multi-material parts in additive manufacturing. *The CIRP Journal of Manufacturing Science and Technology, 37*, 302–311. https://doi.org/10.1016/j.cirpj.2022.02.007

Valente, J. M. S., & Schaller, J. E. (2012). Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. *Computers & Operations Research, 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. *Management Science, 33*(8), 1035–1047. https://doi.org/10.1287/mnsc.33.8.1035

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

Zhang, J., Yao, X., & Li, Y. (2020). Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing. *International Journal of Production Research, 58*(8), 2263–2282. https://doi.org/10.1080/00207543.2019.1617447