MACHINE LEARNING IN COMBINATORIAL OPTIMIZATION – AN APPLICATION TO MACHINE SCHEDULING

AYKUT UZUNOGLU

Aykut Uzunoglu: *Machine Learning in Combinatorial Optimization – An Application to Machine Scheduling*, December 2023

Universität
Augsburg
University

MACHINE LEARNING IN COMBINATORIAL OPTIMIZATION – AN APPLICATION TO MACHINE SCHEDULING

Cumulative Doctoral Dissertation
at the Faculty of Business Administration and Economics
of the University of Augsburg

submitted in partial fulfillment of the requirements for the degree of
Doctor of Economic Sciences (Dr. rer. pol.)

submitted by
Aykut Uzunoglu, M.Sc.

December 2023

First reviewer: Prof. Dr. Axel Tuma
Second reviewer: Prof. Dr. Robert Klein
Chairman of the oral exam: Prof. Dr. Marco C. Meier
Date of oral exam: 22.02.2024

CONTENTS

LIST OF SCIENTIFIC CONTRIBUTIONS

The following published and submitted scientific contributions are presented within this doctoral dissertation. The articles are sorted following their order of publication. The journal rankings correspond to VHB-JOURQUAL3, published by the German Academic Association for Business Research (VHB).

Contribution A
(Published in the European Journal of Operational Research, ranked **A**)
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. *European Journal of Operational Research, 296(3), 819–836*. doi:10.1016/j.ejor.2021.04.006.

Contribution B
(Published in Computers & Operations Research, ranked **B**)
Uzunoglu, A., Gahm, C., Wahl, S., & Tuma, A. (2023a). Learning-augmented heuristics for scheduling parallel serial-batch processing machines. *Computers & Operations Research, 151, 106122*. doi:10.1016/j.cor.2022.106122 .

Contribution C
(Published in Annals of Operations Research, ranked **B**)
Uzunoglu, A., Gahm, C., & Tuma, A. (2023b). A machine learning enhanced multi-start heuristic to efficiently solve a serial-batch scheduling problem. *Annals of Operations Research*. doi:10.1007/s10479-023-05541-w.

Contribution D
(Submitted to Computers & Operations Research, ranked **B**)
Uzunoglu, A., Gahm, C., & Tuma, A. (2023c). Machine learning based Algorithm Selection and Genetic Algorithms for serial-batch scheduling. *Computers & Operations Research*.

LIST OF APPENDICES

LIST OF FIGURES

ACRONYMS

ASP	Algorithm Selection Problem
GA	Genetic Algorithm
ML	Machine Learning
MILP	Mixed-Integer Linear Program
MLGS	Machine Learning Grid Search
MLPP	Machine Learning Parameter Prediction
MLRP	Machine Learning Ranking Prediction
MRIW	Mean Relative Improvement to the Worst
NN	Neural Network
PSBIJF	Parallel Serial-Batch scheduling with Incompatible Job Families
RMSE	Root Mean Squared Error

1 INTRODUCTION

Machine Learning (ML) research has gained momentum in the recent decade and opened doors in applications (e.g., speech and image recognition, autonomous driving, chatbots, or generative models) that were hardly foreseeable for anyone. At its core, ML aims to find patterns in data (Bishop, 2006). ML's textbook definition comprises three branches: supervised learning, unsupervised learning (i.e., deducing structures in unlabeled data), and reinforcement learning (i.e., an agent adapts its action policy according to a reward obtained by the environment) (Russell & Norvig, 2021)[pp. 669–670]. In the context of the most widely used ML branch of supervised learning (LeCun et al., 2015; Jordan & Mitchell, 2015), ML models predict an outcome based on features. Such models are often functions $f : X \to Y$, i.e., the function f takes an input (feature) vector $x \in X$ and outputs a response $\hat{y} = f(x) \in Y$ as close as possible to its true label (target) $y \in Y$, which is only known for the training data set (most often $X = \mathbb{R}^n$, and $Y = \mathbb{R}^m$ in the case of regression or Y representing a finite set in the case of classification). The goal is to find a function f that describes the relationship between input x and output y, and to later apply that function to unseen data. This can be done by selecting a function (model) from a class of functions that minimizes a certain metric, like the distance between the target output y and the model's output $\hat{y} = f(x)$ predicted to an input x. This idea is not new, but searching for models in an "expressive" class of models, e.g., Neural Networks (NNs) with deep, "hidden" structures, was computationally too hard until recently. Not single but multiple developments (e.g., improved hardware, parallelization, or tailored model architectures for specific applications) have eased the application of such models and supported the uprise of ML (see e.g., Pandey et al., 2022 or Gu et al., 2018).

Optimization algorithms lie in the heart of many ML techniques (Rumelhart et al., 1986). An example for this would be the selection of models that minimize a certain error measure (e.g., mean squared error between the true and predicted label; see Hastie et al., 2009[p. 12]). Conversely, ML techniques have found their way into Operations Research methods and applications. At first glance, ML's probabilistic and mathematical optimization's exact nature may seem to contradict each other. However, ML can enhance optimization algorithms in many ways or even enable the development of algorithms that could not be realized without ML models. This becomes evident especially if we relax our optimality requirement for solutions and, therefore, allow using probabilistic models during the solution procedure for combinatorial optimization problems.

Bengio et al., 2021 introduced several perspectives on integrating ML models into solution methods in their review of ML applications for combinatorial optimization. They distinguished between imitation learning (i.e., finding a model that mimics the expected behavior of an expert), and experience-based learning (i.e., using reinforcement learning to generate policies according to a reward signal returned by interactions with the environment). The demonstrated examples range from rather general optimization approaches (e. g., approximating bound improvements by cutting planes or improving the node selection in branch-and-bound trees) to specialized approaches for specific problem types (e. g., learning a node selection policy for the traveling salesman problem or learning to predict suitable algorithm parameter configurations for a given problem instance). They also presented another perspective on integrating ML into optimization: using it alongside optimization algorithms. In this case, the (higher level) optimization algorithm calls the ML model to get predictions to a (lower level) decision. Unfortunately, using ML algorithms in optimization methods comes with the downside of neither having a guarantee for the gap to the optimal solution nor knowing if the answers of a model are feasible. Nevertheless, the authors argued that imitation learning's faster response time provides a valuable contribution to combinatorial optimization algorithms, and they "[. . .] strongly believe that this is just the beginning of a new era for combinatorial optimization algorithms" (Bengio et al., 2021).

Of course, ML methods are also present in the field of (machine) scheduling in which combinatorial optimization problems occur very frequently. To unify different wordings in the scheduling literature, Uzunoglu et al., 2023a refer to them as *learning-augmented scheduling methods* and categorize them as methods that:

- directly make scheduling decisions,
- automatically design new scheduling methods (e.g., heuristics),
- select the most suitable scheduling method from a set of existing methods (Algorithm Selection Problem),
- determine the most suitable parameters of a scheduling method (parameter tuning),
- predict the objective value achieved by a specific scheduling method.

This thesis introduces research in the latter three categories and validates its added value by incorporating it into a computationally hard (serial-batch) scheduling problem.

The remainder of this dissertation is structured as follows: Chapter 2 introduces the application case of a serial-batch scheduling problem, the potentials and challenges of using heuristic approaches for complex problems, and at which points ML models are used to support or improve the decisions made for that application case. Additionally, a brief introduction to ML theory is given. Chapter 3 summarizes the scientific contributions A-D and highlights the findings that enhance decision-making, and Chapter 4 presents the research articles, including their bibliographic information. The last chapter, Chapter 5 concludes with this thesis' scientific contributions and raises open questions that should be addressed in future research.

2 PRELIMINARIES

This chapter introduces foundations to get a better understanding of the constituent contributions of this thesis. The concepts and methods developed in this thesis are generally based on a scheduling application case from the metal-processing industry but are not limited to this case or industry. The application case presented in the following section is the main example used to validate and evaluate the applicability of the developed concepts in the contributions. Demonstrating these concepts and methods on a specific industrial example gives the reader an idea of how to integrate ML models in scheduling and discusses issues that must be (generally) addressed when using ML models. Choosing an example complex enough to be seen in practical applications is important since idealized examples omit (or relax) details essential for the implementation. Section 2.1 describes the hard-to-solve serial-batch scheduling problem (called Parallel Serial-Batch scheduling with Incompatible Job Families (PSBIJF) in the following) used as an example in this thesis' contributions. Next, Section 2.2 discusses how to tackle hard problems like the PSBIJF in Section 2.1 and presents the best-known heuristic(s) capable of solving large instances of that problem. Furthermore, Section 2.2 shows issues arising when using this heuristic and motivates the necessity of incorporating ML models into the solution procedure. Section 2.3 gives a brief introduction to ML theory to ease the understanding of concepts in the constituent contributions. The last section, Section 2.4, shows how the contributions in this thesis extend the decision-making for the PSBIJF.

2.1 APPLICATION CASE: A SERIAL-BATCH SCHEDULING PROBLEM

Manufacturers in the metal-processing industry often deal with serial-batch scheduling problems when laser-cutting machines cut out geometrical shapes (i.e., jobs) from a metal sheet in a serial manner, i.e., each shape is cut out one after another (Gahm et al., 2022b; Helo et al., 2019). Hereby, the operators must first decide which jobs to group in which batches, and second, they must define a sequence of execution (and a machine allocation when several exist; we assume parallel machines). The jobs have characteristics like material requirements (e.g., type and thickness), processing times, sizes (e.g., area demands), weights (often implied by a customer priority), or due dates that influence both decisions. Jobs can be grouped in the same batch only if they share the same job family (material requirements) and do not violate the capacity constraints of the batch, i.e., the jobs in a batch fit onto the metal sheet. The latter batch feasibility condition is a difficult geometrical problem and will be referred to as the *batch feasibility check* in what follows. The fact that processed jobs are only available if the whole batch is finished ("batch availability") introduces complexity to the decision-making process: Adding (or removing) a job to (from) a batch influences not only the job's completion time but also the batch's processing time (the batch processing time is the sum of all job processing times) and its completion, and thus, the completion of all jobs in the batch. Furthermore, if there is no idle time on the cutting machine, completion times of all subsequent batches (jobs) are affected. Additionally, the serial-batch scheduling problem considers sequence-dependent setup times between batches, depending on the current and upcoming batch's family. Setup times represent the time the operator needs to take out the processed jobs, insert a new metal sheet, and (possibly) adjust the machine to the new material properties if the batch families differ (Shen & Buscher, 2012).

Serial-batch scheduling problems may be part of a multi-stage production process or a complex supply chain (Gahm et al., 2022b). Given an aspired completion date for the (end) product, backward planning returns a due date for the items of the cutting process. Missing a due date should be avoided but, in contrast to deadlines, do not violate a constraint. In the presence of due dates, the objective is usually to minimize the (weighted) total tardiness of all jobs, which translates to increasing delivery reliability. If so, the effects of adding (removing) a job may contradict each other because if a job is tardy, allocating it to an earlier batch reduces its tardiness but increases the completion time of all jobs in this batch by the processing time of the added job.

This problem was classified as $P|sb, if*, b, a_j, s_{f,g}|wT$ according to the three-field notation (Gahm et al., 2022b) and is abbreviated in what follows as the PSBIJF problem (Parallel Serial-Batch scheduling with Incompatible Job Families).

Figure 1 shows an exemplary PSBIJF instance and a computed schedule for that instance. The upper part shows customer (or in-house) orders requiring specific shapes (jobs j_1 to j_6) with given material requirements (material type and thickness) and a due date. Stated in scheduling terminology, material requirements define the (incompatible) job families f_1 (aluminum, 22mm) and f_2 (stainless steel, 15mm). A job's size a_i is defined by the area of the shape (not depicted in Figure 1). The processing time of a job also depends on the shape (mainly on the cutting length), but the material type and the thickness may also impact it. The

Job	Shape	Job family (material, thickness)	Processing time	Due date
j_1, j_2	⬢	f_1 (aluminium, 22mm)	8	9
j_3		f_1 (aluminium, 22mm)	26	1
j_4	⬢	f_2 (stainless steel, 15mm)	8	2
j_5		f_2 (stainless steel, 15mm)	16	3
j_6		f_2 (stainless steel, 15mm)	16	5

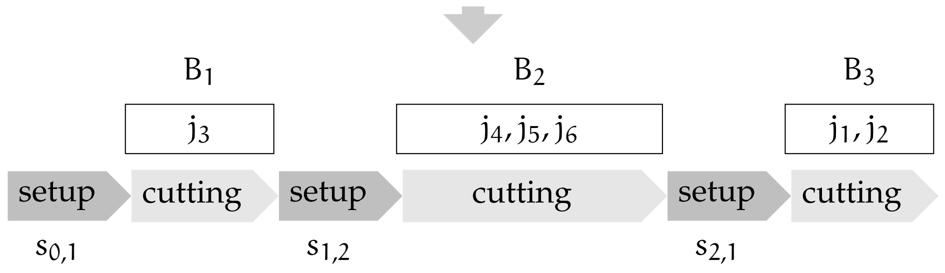


Figure 1: A serial-batch scheduling problem and its solution

lower part of the figure shows a computed schedule. Jobs are grouped into batches B_1, B_2 and B_3 and sequenced, aiming to minimize the total weighted tardiness. Because customers may have different priorities for the manufacturer, a job's tardiness is multiplied by a customer-specific weight in the objective function. Batch B_1 consists only of job j_3 and needs an initial setup time $s_{0,1}$ (with 0 indicating an initial "dummy" batch and 1 indicating family f_1). Job j_3 has the closest due date and, therefore, is processed without other jobs. Even though job j_4 has a similar due date, it belongs to family f_2 and hence can not be processed together. After that, a setup time $s_{1,2}$ is needed to change from family f_1 to f_2. Batch B_2 consists of three jobs (j_4, j_5, j_6) and needs more processing time. The same idea applies to batch B_3 and the setup time $s_{2,1}$. The next section discusses solution methods to compute schedules for this problem.

2.2 SOLVING THE PSBIJF

In manufacturing, practitioners frequently use solution methods from Operations Research in production planning and scheduling. The (combinatorial) optimization problems to be solved are often NP-hard, and therefore, solving large-scale real-world problems to optimality becomes often intractable. Relaxing the optimality requirement of problem solutions increases the flexibility of solution methods and enables trading in solution quality for computation time. Heuristics are popular choices to obtain reasonable solutions quickly but, in contrast to approximation algorithms, giving provable statements about their solution quality is difficult and are often evaluated empirically (Rardin & Uzsoy, 2001). This disadvantage is an acceptable toll to pay for manufacturers since the problem characteristics may change at short notice (e.g., due to new customer orders or machine breakdowns), leaving the manufacturers with little to no time to calculate (near) optimal solutions for complex problems. For the case of the PSBIJF, we can formulate it as a Mixed-Integer Linear Program (MILP; see Gahm et al., 2022b), allowing us to use sophisticated MILP-solvers (e.g., branch-and-cut). Unfortunately, finding a solution to the MILP is costly because methods like branch-and-cut algorithms fail to find reasonable lower bounds for their branching due to the "unhandy" total tardiness objective. In a computational study, Gahm et al., 2022b could only compute (non-optimal) solutions to problem instances with up to 100 jobs due to run time restrictions – a job number far too low for practical applications with usually several hundreds or thousands of jobs. Therefore, they proposed a new heuristic BATCS-b (Batch Apparent Tardiness Costs with Setup times and batch utilization control parameter b; see Vepsalainen and Morton, 1987 and Balasubramanian et al., 2004) to calculate schedules efficiently. Their heuristic decouples the batching and scheduling decision by creating batches with unscheduled jobs according to a priority rule (which needs two parameters to be set) when machines become available at a certain point t in the time horizon. It adds the most "urgent" job to the current batch if it does not violate the batch feasibility, and restarts the iteration with the remaining unscheduled jobs until no unallocated job is left. Next, the heuristic adapts the priority rule for batches to asses their urgency and selects the next batch in the sequence of a machine according to that priority. The heuristic repeats this routine until all jobs are scheduled. An essential feature of this heuristic is that it does not force "full" batches but stops if the altered batch capacity (by parameter b) is reached. In their computational study, this heuristic computes schedules for instances with up to 5,845 jobs quickly, but its solution quality highly depends on the used heuristic parameters. To increase the solution quality, Gahm et al., 2022b restart the heuristic multiple times with different parameter configurations (also known as *multi-start* heuristic), which massively increases the computation time.

The parameter configuration of a heuristic is an example in which the runtime of a heuristic may become a bottleneck. Setting suitable parameters for a heuristic immensely influences the solution quality of most parametrized solution methods. In practical applications, the algorithm's configuration often relies on an expert's expertise, leaving huge space for performance improvements. Unfortunately, finding a suitable parameter configuration is often time-consuming, involving executing the algorithm multiple times with different parameter configurations to evaluate their performance. In the case of continuous intervals for the parameter domains, it is common to discretize the intervals and perform a multi-start heuristic on all combinations, resulting in substantial computational efforts. Thus, even the fast response times of heuristics may not be sufficient in such situations.

This also holds for the hierarchical production planning example studied in this thesis. In general, hierarchical production planning consists of top-level and (several) base-level decision problems (see, e.g., Hax and Meal, 1973; Bitran et al., 1981; Schneeweiß, 1995). Those problems are interdependent, i.e., the solution of one problem influences the solution (procedure) of another problem. For example, consider a hierarchical problem in which a top-level problem has to solve a base-level decision problem in one of its constraints. If the constraint must be checked multiple times while solving the top-level decision problem, even a heuristic's (relatively) fast computation time may become a bottleneck for the top-level problem. Instead, if the top-level decision problem does not need an elaborate solution to the base-level decision problem, a fast (approximate) anticipation without calculating the actual solution to the base-level decision problem could help to overcome this bottleneck. Nevertheless, the anticipation's accuracy may be crucial to the higher-level solutions performance. For instance, too "optimistic" anticipations could lead to decisions that violate the constraints of the base-level decision problem (and therefore also the top-level decision problem). Therefore, developing an accurate anticipation of the base-level decision problem is worthwhile to increase the chances of obtaining a feasible solution to the hierarchical problem.

A concrete manifestation of such a hierarchical decision problem is the interplay between the aforementioned PSBIJF and the *batch feasibility check*. Both proposed scheduling methods, the MILP formulation and the multi-start heuristic, use a simple batch feasibility check that is computationally very fast but can result in infeasible batches as it ignores the real geometrical properties of the shapes. For a "correct" batch feasibility check, we must know if a set of arbitrarily complex shapes can be placed on a metal sheet with a given height and width. The corresponding optimization problem is known as the *nesting* (or *strip packing*) problem and is NP-hard (Bennell & Oliveira, 2008; Fischetti & Luzzi, 2009). Even heuristics need several minutes for "complex" nesting problems, which is already unpractical for methods that must perform the feasibility check for every potential batch. The simple batch feasibility check applied so far in (Gahm et al., 2022b) sums up all area demands of the shape set and compares it to the total area of the metal sheet. This approach assumes all jobs can be perfectly nested without leaving any

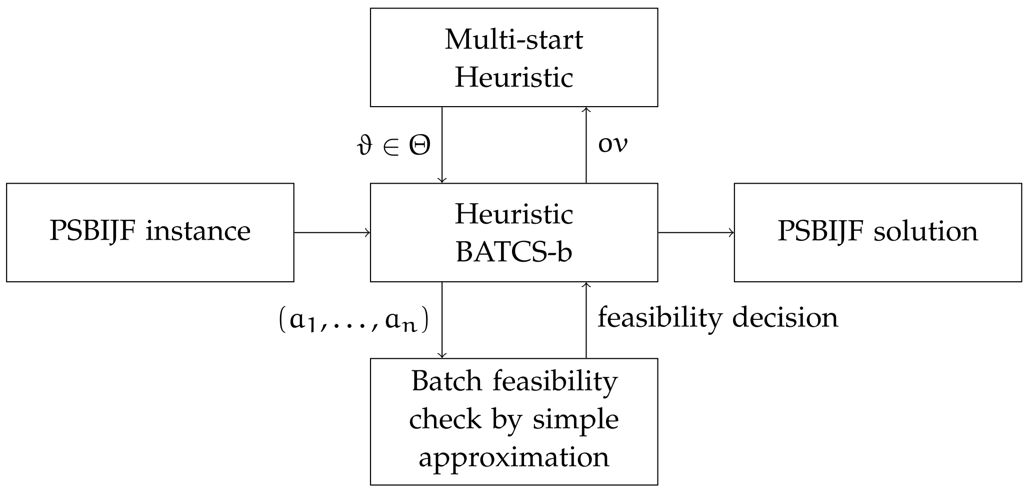


Figure 2: Procedure to solve the PSBIJF

unused area on the sheet. However, relaxing the nesting problem in such a strong way may lead to batching decisions that cannot be implemented when applying the schedule.

Figure 2 illustrates how the PSBIJF problem is solved according to the proposed approach in Gahm et al., 2022b. For each PSBIJF instance, the multi-start heuristic evaluates the objective value ov of every possible parameter configuration $\vartheta \in \Theta$ by executing the BATCS-b heuristic. In its batching decisions, the heuristic uses a sum of the job's net areas $(a_1, \ldots, a_n)$ to approximate whether the jobs fit onto the batch. Lastly, the multi-start heuristic returns the best schedule for the problem instance achieved by any parameter configuration ϑ in the parameter space Θ.

The question to be answered by the constituent elements of this doctoral thesis is how ML methods can be applied to increase solution quality and efficiency when solving the PSBIJF, a combinatorial optimization problem of emerging importance in the field of machine scheduling. Before answering this question, some basics of ML theory are necessary for a common understanding and are therefore discussed in the following section.

2.3 MACHINE LEARNING THEORY

This section briefly introduces basic ML theory concepts to ease the understanding of the constituent elements. These concepts help to understand why certain steps must be taken to determine an accurate ML model and belong to best practices (e.g., splitting test data, penalizing model complexity) nowadays (Goodfellow et al., 2016). Undoubtedly, giving a comprehensive overview of this vast field is difficult. For the interested reader, the following resources might be a good starting point:

Bishop, 2006; Hastie et al., 2009; Barber, 2012; Murphy, 2013; Shalev-Shwartz and Ben-David, 2014; Goodfellow et al., 2016; Mohri et al., 2018; MacKay, 2019; Russell and Norvig, 2021.

Depending on the time and domain of the application, the task of machine learning is or was defined with different motivations (Bishop, 2006). In fact, even the naming of the field was a challenge in its own right. The naming has changed over time: in its beginning, the more practical branch was known as "pattern recognition", and the theory behind machine learning as "statistical learning theory". With the upcoming introduction of bio-inspired methods, such as perceptrons (Rosenblatt, 1958), "artificial intelligence" has become a more popular name for this field. Artificial intelligence now embodies a vast field in computer science designated to mimic intelligent systems, and ML is a subfield that uses probabilistic models to achieve its goals (Russell & Norvig, 2021). Also, throughout the history of this field, its focus and methodology have shifted with the technologies available at that point in time.

In its foundations, early research focused on theoretical results applicable to asymptotic settings (e.g., infinitely many observations in the limit) and, with the uprise of powerful computing devices and massive data sets, shifted its focus (LeCun et al., 2015; Schmidhuber, 2015). It began as a question in statistics: "What must one know a priori about an unknown functional dependency in order to estimate it on the basis of observations?" (Vapnik, 2000). According to Vapnik, 2000, *statistical estimation* in the 1920s and 1930s (see e.g., Fisher, 1925 or Efron and Hastie, 2016) had very restrictive answers to that.

Starting in the 1960s, the thinking in this field shifted to a more practical viewpoint and addressed topics like small sample statistics. Also during this time, Vladimir Vapnik and Alexey Chervonenkis developed the Vapnik-Chervonenkis theory (VC-theory), which is very fundamental for research that followed in statistical learning theory (Vapnik & Chervonenkis, 1971). The ultimate goal of statistical learning theory is to formalize the principles that ML methods use to automate the model "learning" process based on observations. This theory leverages the knowledge about properties of the class of functions that represent the observations' functional dependency. Formally, let $x_i \in \mathcal{X}$ be an observation and $y_i \in \mathcal{Y}$ its observed label, then we believe that there exists a mapping $f : \mathcal{X} \to \mathcal{Y}$ that describes the dependency as $y_i = f(x_i) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ being the inherent noise in our observations (with unknown σ^2; see Bishop, 2006 for a detailed explanation of this connection). Assuming that the training data (observations) has some underlying "regularities" is key to this theory. Therefore, we desire to find functions that recognize and exploit some sort of "regularities" rather than "arbitrary" functions. Allowing arbitrary functions would lead to models that also encapture the noise in the training data, leading to poor performance on testing data – a phenomenon widely known as *overfitting* (Bishop, 2006).

Making assumptions is essential to obtain well-performing models. According to the *No Free Lunch* theorem in ML (see Wolpert, 1996), it is even impossible to predict future (test) data if we cannot make assumptions about the relation between training and test data. Both, the training and test data, are assumed to be sampled independently and from the same distribution (i.i.d) (Shalev-Shwartz & Ben-David, 2014). Being sampled from the identical distribution implies we can infer knowledge from the training about the underlying distribution of the observed phenomenon. The independence of the samples means that we have no restrictions on how to use the training data (e.g., no specific order or conditioning) and are able to utilize all the information we have. One important implication of both assumptions is that ML algorithms (may) increase their prediction performance with increasing training data size (Bousquet et al., 2004). But for a finite training data size, we can construct counterexamples such that an algorithm performs perfectly on the training data and "arbitrarily bad" on new data if it is allowed to select *any* function as a model. This is the phenomenon of overfitting in a very extreme form and can be overcome by restricting the set of possible functions (i.e., algorithm) that fulfill certain properties (e.g., regularity conditions for the set of functions or "simple" functions). However, if the set of functions is too restricted, the function cannot represent complex dependencies in the training data – this is the opposite of overfitting and known as *(inductive) bias*. So, selecting a promising class of functions is a sensitive task, and many different techniques are now available for ML researchers to control their model's "complexity". Furthermore, the fact that functions may perform well on training and badly on new, unseen data makes it necessary to split the data set: use the first split for training and assess the prediction performance on the second split.

To get a formal understanding of the abovementioned concepts, consider again the input space $\mathcal{X}$ and the output space $\mathcal{Y}$ (see Shalev-Shwartz and Ben-David, 2014 or Bousquet et al., 2004 for a much more detailed and formal explanation of what follows next). For convenience, we can restrict $\mathcal{Y}$ to the set $\{-1,\ 1\}$, meaning our inference problem is a classification problem. We assume that $(x, y) \in \mathcal{X} \times \mathcal{Y}$ are random variables with an unknown probability distribution $\mathbb{P}$. Our training data set consists of n i.i.d. drawn pairs of $(x_i,\ y_i)$ sampled from $\mathbb{P}$, and we want to find a mapping (i.e., the model) $h : \mathcal{X} \to \mathcal{Y}$ from a hypothesis class $\mathcal{H}$ (e.g., class of linear functions), which predicts the label y for x. To assess the prediction performance of h, we define the *risk* (also referred to as *error* or *loss*) of h as $R(h) := \mathbb{P}\,(h(x) \neq y)$, which is the risk of h misclassifying x. This notion allows us to decide which functions h must be preferred and to argue about functions that minimize the risk.

For our classification problem, we can define the regression function $\eta(x) := 2\,\mathbb{P}(Y = 1|X = x) - 1$ and the target function $t(x) = sgn\ \eta(x)$, which minimizes the risk: $R(t) = \inf_{h \in \mathcal{H}} R(h)$ as described in Bousquet et al., 2004. Such a risk-minimizing target function $t(x)$ is also called *Bayes classifier,* and its associated risk is called the *Bayes risk*. We have to find such a function t but, unfortunately, $\mathbb{P}$ is not known. Nevertheless, we can compute the *empirical risk* of a model h on a finite test data set with n data points:

$$R_n(h) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{h(x_i) \neq y_i} \tag{1}$$

as a proxy for the actual risk. However, since R_n is an approximation based on a finite sample, and the function is chosen from a function class "large enough", we could always find a function h that achieves empirical risk $R_n(h) = 0$ but maximizes the risk $R(h) = 1$. This is the problem of overfitting the model to the training data such that it performs poorly on unseen data. We can overcome this by restricting the hypothesis class from which h can be selected (e. g., h must be a linear function) or by penalizing "complexity" (which has different definitions depending on the analyzed subject) in the model itself. In the *empirical risk minimization* framework, we define a set of possible functions $\mathcal{H}$ and choose a function $h_n \in \mathcal{H}$ that minimizes the empirical risk on our training data of size n:

$$h_n = \arg\min_{h \in \mathcal{H}} R_n(g). \tag{2}$$

The caveat in this framework is to define a set $\mathcal{H}$ that is "rich" enough to contain the target function or approximate it without being too "rich" such that overfitting may occur. A good balance between both ends ensures good prediction performance and low error (i.e., bias-variance trade-off). *Structural risk minimization* is a paradigm that aims to control both, the risk and the size (or the complexity) of the model. Here, $d(h) \in \mathbb{N}$ describes the size of the model h (e. g., the number of adjustable weights in the model). The function $h_n \in \mathcal{H}$ minimizes the risk, and a penalizing term $pen(d(h))$ which increases with increasing size of h:

$$h_n = \arg\min_{h \in \mathcal{H}} R_n(h) + pen\,(d(h))\,. \tag{3}$$

If the penalizing term of h is the norm $||h||$ (multiplied by a constant), the penalizing term is also called regularizer. Typically, the 1- or 2-norms on the function properties (e.g., weights) are used in applications(Bishop, 2006).

Vapnik and Chervonenkis, 1971 nicely combined a concept of the function class $\mathcal{H}$'s complexity (at least for binary classification problems with $\mathcal{Y} = \{-1, 1\}$) and a bound to its risk in VC-Theory. Furthermore, their theory gives us a formal understanding of notions like "rich enough" or "complexity". For that purpose, we have to define the *VC dimension* of class $\mathcal{H}$. This is the maximal number of points it can "shatter", meaning the maximal number of points it can distinguish given any classification scenario. For example, in the space $\mathbb{R}^2$, the set of hyperplanes can distinguish three points with any given classification, but we can construct a counterexample with four points such that no hyperplane is able to distinguish them (see, for example, the *XOR-problem* Nitta, 2003), and hence, its VC-dimension is 3. Let us call $v_{\mathcal{H}}$ the VC dimension of $\mathcal{H}$, then, with probability at least $1 - \delta$, we get a risk lower bound for the empirical risk (see Bousquet et al., 2004[p. 182]):

$$\forall h \in \mathcal{H},\ R(h) \ \leqslant R_n(h) + 2\sqrt{2\frac{v_{\mathcal{H}} \log \frac{2en}{v_{\mathcal{H}}} + \log \frac{2}{\delta}}{n}}. \tag{4}$$

This may seem to be a result for a very restricted case (namely binary classification and for function classes that have VC dimension that can be "measured"), but there also exists research on more complex models (see Bartlett et al., 2019 for results on piecewise linear NNs). However, even for this simple case, the risk lower bound shows the following:

1. The size of the function class does not depend on the number of functions in that class but rather on their "geometry".
2. If the VC dimension h of a function class is finite, the empirical risk converges to the true risk with increasing training data size.

As the result of the risk lower bound suggests, increasing the test data size or reducing the complexity of the model can reduce this type of error. The error achieved by the model can be decomposed into two parts – approximation and estimation error (see Shalev-Shwartz and Ben-David, 2014[pp. 64–65]). The approximation error occurs due to inductive bias by selecting a class of too restrictive functions. It is the minimal risk achievable by functions in that class. The estimation error occurs because we can not measure the true risk but estimate it via the empirical risk. Increasing the complexity of $\mathcal{H}$ reduces the approximation error but might increase the estimation error due to overfitting. Conversely, reducing the complexity of $\mathcal{H}$ prevents overfitting, and thus, the estimated error is more accurate, but we introduce an approximation error. So, we are left with a sensitive task to find a balance between these two types of errors.

With these general concepts of ML in mind, the enhancement of existing and new scheduling methods by ML are discussed next.

2.4 MACHINE LEARNING ENHANCED SERIAL-BATCH SCHEDULING

This dissertation contributes to the literature with three different approaches of integrating ML models in scheduling. Figure 2 in section Section 2.2 shows how the scheduling problem PSBIJF is solved without integrated ML models. There, for each PSBIJF problem instance, the multi-start heuristic evaluates different parameter configurations for BATCS-b and, internally, approximates the feasibility of a batch by simply summing the jobs' area and comparing it to the maximum batch capacity. This is an easy-to-implement procedure but, as mentioned above, has potential for improvement.

Regarding parameter configurations, ML models can help to suggest parameter configurations based on previous executions without performing the costly parameter search at scheduling time. Of course, the generation (i.e., training) of the ML model itself is a computationally expensive task, but this phase is decoupled from the execution and is performed in advance. Likewise, ML methods also have the potential to provide more accurate batch feasibility approximations without causing a computational bottleneck. Furthermore, there are less apparent potentials when the problem is viewed from a higher level. As this thesis shows, selecting a suitable solution method for each instance individually is also a promising scenario to incorporate ML into the scheduling. Nevertheless, it is crucial to carefully ask the "right" questions to successfully integrate an ML model into scheduling.

Figure 3 depicts the integration of ML models according to this thesis's contributions A to D (indicated by the upper left boxes; light grey boxes indicate developed scheduling methods, and dark grey boxes ML integrations).

In detail, contribution A aims to approximately anticipate the solutions of a base-level problem in the context of hierarchical production planning, i.e., the top-level problem asks the ML model for predictions of a base-level problem and continues its computations based on these predictions. Figure 2 shows how the base-level decision problem, i.e., the batch feasibility check, is incorporated into the top-level decision problem by a simple approximation rule. Figure 3 depicts how the contribution of this dissertation alters the procedure by including an ML model in the feasibility check. Of course, the ML model's probabilistic predictions introduce uncertainty to the decisions made by the higher-level problem that must be considered by the (overall) scheduling method, resulting in a new challenge. Nevertheless, the tremendous speed-up (compared to solving the nesting problem) is critical for the top-level decision problem that must solve the base-level decision problem many

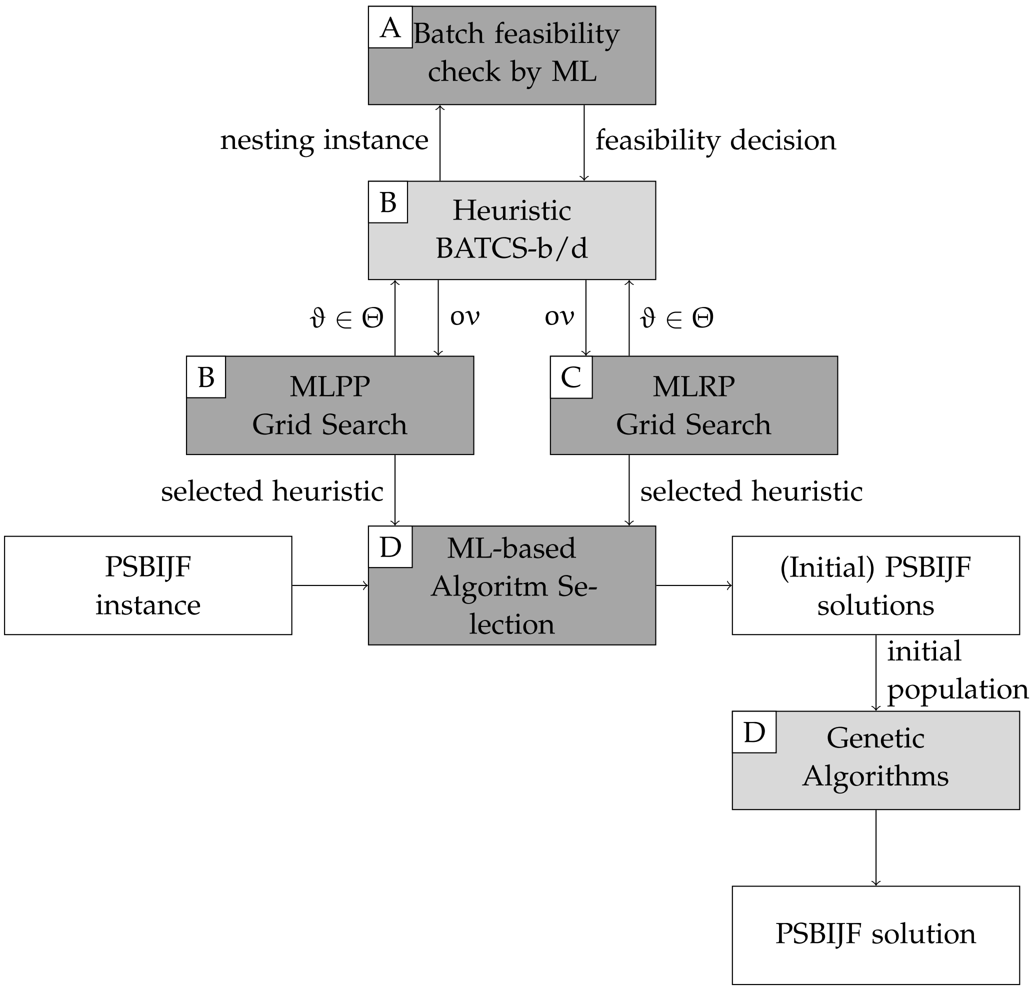


Figure 3: Machine Learning enhanced procedure for the PSBIJF

times, thus making it necessary to tackle this new challenge. Contribution A presents a framework to generate a suitable training data set, train and validate different ML models (i.e., different hypothesis classes), and include the learned model in the decision process of a hierarchical production planning problem from the metal-cutting industry.

The second approach to effectively apply ML models in machine scheduling is tuning a solution method's parameters, which is addressed in contributions B and C. As mentioned, many heuristics (or optimization algorithms in general) need parameters that impact their behavior during the solution calculation, which also influences the solution quality. A "good" parameter configuration depends on the problem instance, and therefore, finding suitable parameters for a problem instance states a new optimization problem. When no prior knowledge about the parameters is given, it is a common approach to discretize the parameter space Θ in each dimension and to combine each dimension as a Cartesian product (called "full grid"), then, to start the heuristic with different parameter settings for the problem instance, and to select the best-performing configuration. Even though this "brute-force" method is easy to implement, the efforts to search for reasonable parameters far exceed those needed to solve the problem. The inefficiency of such a "full grid search" limits its practical applicability. However, when prior knowledge about the performance of the parameter space for a problem instance is available, we can utilize this to identify suitable candidates for the parameter configuration.

There is a large body of research employing (statistical) models to tackle parameter tuning in general for optimization problems. Earlier approaches used less complex models to individually determine good parameter configurations for problem instances (e.g.,Kim et al., 1995). Since complex ML models are becoming more accessible, these models were utilized to determine suitable parameters. Nevertheless, using ML models to find suitable heuristic parameters has some caveats and, therefore, needs to be designed carefully for them to find suitable heuristic parameters. The first issue appears when multiple parameters have to be found for the heuristic. In this case, the ML model must be able to represent dependencies between these parameters. Second, an instance may have multiple parameter configurations leading to the best-known objective values. This means that the model must consider that multiple correct answers for the parameter configuration question exist. From the ML perspective, the uncertainty of the model's predictions must be considered when it is being applied to the parameter tuning. Otherwise, it will negatively impact the solution quality of the heuristic. Both contributions (B and C) present techniques to incorporate the uncertainty of the predictions in the parameter tuning process. In contribution B, the Machine Learning Parameter Prediction (MLPP) grid search first predicts a probably well-performing parameter configuration and searches in the region of that predicted configuration. In contribution C, the Machine Learning Ranking Prediction (MLRP) grid search creates a ranking based on the predicted objective values of parameter configurations and searches according to that ranking. Furthermore, Contribution B presents a refinement of BATCS-b, called BATCS-d, that, instead of controlling batch capacities, incorporates the "urgency" of jobs in a batch to stop filling a batch with jobs.

Contribution D utilizes ML models on a higher level: Contributions B and C present various approaches that use heuristics BATCS-b and BATCS-d and predict their parameters with different strategies. However, there is no clear winner among all problem instances, which leaves us with the question of which method to choose for a given problem instance. This problem is known as the Algorithm Selection Problem (ASP; see Rice, 1976) in the literature, and contribution D leverages the potential of ranking-based ML models to individually select a solution method for a given problem instance. The techniques developed in contributions B and C generate the data needed to train these models. Furthermore, it presents two Genetic Algorithms (GAs) to solve the PSBIJF, which are capable of using initial solutions obtained by the solution method determined by the ASP.

The next sections present contributions A-D in more detail.

3 CONSTITUENT ELEMENTS OF THIS DISSERTATION

3.1 CONTRIBUTION A

(Gahm et al., 2022a):

This contribution analyzes a specific aspect of the previously introduced serial-batch scheduling problem. One critical decision in this application is which jobs have to be processed together in batches (i.e., batching-decision), respecting two restrictions: i) only jobs with the same material requirements can be grouped into batches, and ii) the shapes may not overlap each other or the border of the metal sheets. So far, the second restriction implies a cutting and packing subproblem (referred to as *complex nesting*; see Wäscher et al., 2007 for detailed typology), which is NP-hard. Problems with such a dependency structure are widely studied in hierarchical production planning.

Figure 4 depicts in detail the components and their interactions of a hierarchical production planning problem. In the absence of an anticipated base-model (4), an essential characteristic of a hierarchical planning problem is that the top-level (1) asks the base-level (7) for a decision (6) on a stated problem and uses that answer (8) to control its further instructions. In our serial batch scheduling problem (i.e., the top-model), the complex nesting problem (i.e., the base-model) must be solved for every potential batch of interest, and the top-level must wait until the comlex nesting problem returns a solution to resume its computation. Depending on the solution method used to solve the serial-batch scheduling problem, the set of every potential batch of interest can be prohibitively large to be computed by a nesting algorithm. Therefore, to reduce the planning effort, this (or a similar) subproblem is included in a (strongly) relaxed and computationally easier version to the hierarchical planning problem (see e.g., Gahm et al., 2022b) by approximating the area demand of all shapes and comparing it to the area supply of the sheet (i.e., anticipated base-model). To finally "prove" the feasibility of batches in the schedule, the top-level can postpone solving the nesting problems until a solution to the serial-batching problem is found, and hence the final, much smaller set of nesting problems must be computed in (7).

This contribution compares three easy-to-calculate approximations of base-level solutions based on the area of the enclosing polygons, the convex hulls, and the minimal bounding rectangles of the geometrical

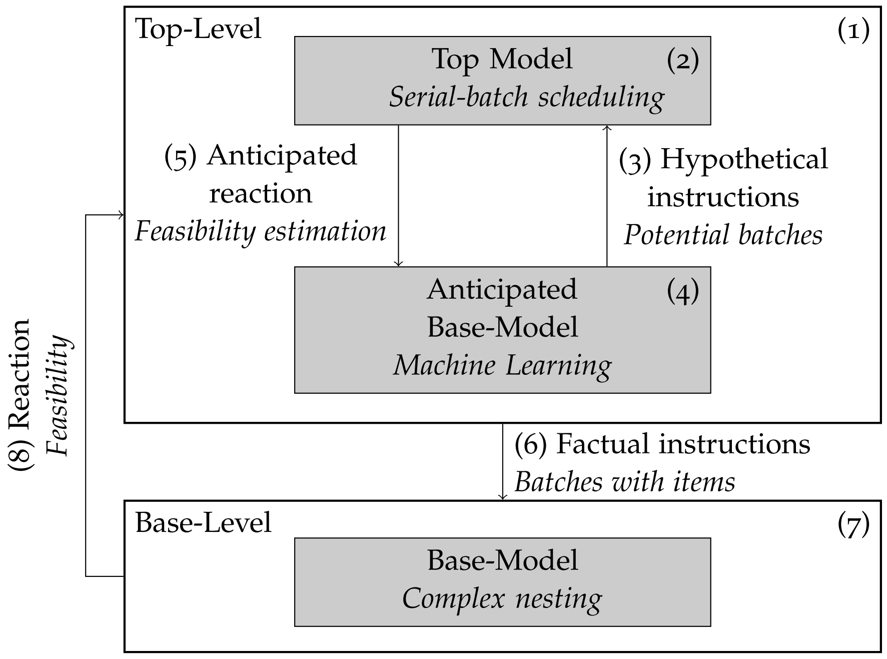


Figure 4: Hierarchical planning problem

shapes as reference values. These *simple approximations* sum up the respecting areas and divide the resulting sum by the width of the metal sheet to estimate the height needed to place all of the shapes on the metal sheet. A study with generated nesting problem instances shows that all three simple approximations perform poorly in estimating the actual height a nesting algorithm from the literature needs to place the geometrical shapes onto the metal sheet. This is troublesome for practitioners since false positive answers regarding the feasibility of a batch may imply schedules that cannot be implemented. So, improving the accuracy of the batch feasibility approximation is essential for successfully implementing the serial-batch scheduling problem. Contribution A proposes applying ML models as the anticipated base-model to approximate the nesting problem's solution, i.e., the height needed if the width of the metal sheet is fixed.

Developing an ML model requires a series of design choices and non-obvious decisions. All developing steps are elaborately presented in a framework to allow for reproducibility of the results. The provided data set for training plays a crucial part in obtaining a useful ML model. One aspect is to have enough data points – the estimation error decreases, and the risk lower bound gets tighter with increasing data size. Additionally, ML models adapt to the (assumed) i.i.d. data distributions and hence need problem instances complex enough to appear in real-world applications. Since no industrial data was available, we generated 88,200 nesting problem instances based on 6,000 shapes with different properties (e.g., convex, non-convex, or "complex" shapes, different sizes, different height to width ratios) and computed the needed height for these instances with a nesting algorithm to get the true output (labels). As ML models operate on numerical representations of the problem instances, we developed two feature vector representations based on instance characteristics.

Having assumptions about the data structure or the dependency between input and output is critical to develop a good model (see Section 2.3). Therefore, a wide range of ML models exists, each based on certain assumptions, representing different hypothesis classes $\mathcal{H}$. We selected 18 ML methods to evaluate their predictive performance (i.e., risk), covering major supervised ML types (e.g., linear, kernel-based, tree-based or ensemble models, or NNs). For the performance evaluation, we measured the Root Mean Squared Error (RMSE) between true nesting height and predicted height. Most ML methods have tunable hyper-parameters that, for example, control the model's complexity. After the hyper-parameter tuning phase, we evaluated every ML method's performance on a validation data set and selected the best three methods for a more exhaustive evaluation.

The final testing showed that the NN outperformed other regression methods and achieved an RMSE of 327 on the test data set, compared to the best-performing simple approximation method with an RMSE of 1,230. To better visualize the implications of these scores for the application case, we analyzed how many instances would have been underestimated with more than 10% of its actual height since the chances are high that such an underestimation could lead to infeasible solutions. The best simple approximation method underestimates the height by more than 10% in 10.1 % of the test instances and the NN in only 1.2% of the cases.

3.2 CONTRIBUTION B

(Uzunoglu et al., 2023a):

Contribution B shifts the focus to the serial-batch scheduling problem, defined as the top-level decision problem in the former contribution. It contributes to the literature in several ways. First, the refinement BATCS-d of the existing BATCS-b heuristic, designed to solve PSBIJF problem instances, is introduced. Second, an ML model augments the heuristics to improve the efficiency of finding appropriate parameters for the heuristic (by trading in solution quality). Last, post-optimization methods that improve the solution quality are introduced. The presented new heuristic builds upon the priority-based heuristic BATCS-b developed in Gahm et al., 2022b that assesses the "urgency" of jobs and batches and creates a schedule accordingly. Two parameters κ_1 and κ_2 control the weights for processing time and setup time in the assessment of a job's or batch's urgency. As filling the batch to its limit does not necessarily lead to good solutions, a third parameter b controls the maximal utilization of the batch, meaning that the heuristic stops filling a batch when a threshold batch utilization is reached. However, one downside of this utilization parameter b is that it is fixed for all batches of the problem instance and does not consider the overall urgency of a batch – if none of the jobs are due in a current batch, the heuristic would add further jobs without increasing the tardiness. To overcome this shortcoming, the new BATCS-d heuristic exchanges the static utilization limiting mechanism, which uses parameter b with a more "flexible" mechanism based on the urgency of a batch and is controlled by parameter d. As a computational study shows, the BATCS-d heuristic remarkably outperforms BATCS-b on average according to the performance metric *Mean Relative Improvement to the Worst* (MRIW).

Nevertheless, appropriate parameters κ_1, κ_2 and b (or d in BATCS-d) have to be set to achieve a reasonable solution quality. If no prior knowledge is available, a multi-start heuristic, i.e., performing the solution method multiple times with different parameter configurations, is often used to find good-performing parameters. The multi-start heuristic evaluates 1,771 parameter configurations for BATCS-b and 1,610 parameter configurations for BATCS-d. With this approach, however, the computation time for finding well-performing parameters far exceeds the solution time of the heuristic. For large problem instances (with up to 3,200 jobs), the multi-start heuristic needs several hours to find suitable parameters in this set of parameter configurations. To circumvent this computation-intensive parameter search, we present a ML-based approach to predict parameter configurations (see MLPP in Figure 3) that may lead to a good solution. We used 93,360 generated serial-batch scheduling instances and performed the multi-start heuristic to obtain the best-performing parameter configuration. As not many ML models are capable of representing the inherent interdependency of the parameter outputs, we chose the NN model for that task. Besides deciding the model type, other decision in regard to the ML pipeline like the pre-processing also influences the prediction performance. We included 18 pipeline configurations (i.e., they differ in dimensionality reduction methods, feature vector representations, and weights for the loss function) to the evaluation of the multi-target regression task. For problem instances with multiple parameter configurations leading to the same best (known) solution, we selected the "most central" parameter configuration as the representative label. As we can not access the heuristic during the learning phase, we used the Euclidean distance between the predicted configuration and best configuration as a surrogate loss function, assuming that configurations closer to the best configuration perform better when applied by the heuristic. In the hyper-parameter tuning phase, we evaluated 225 hyper-parameter settings on 40% of the data for the NNs to select the best settings for each pipeline configuration.

In order to compensate for the uncertainty in the ML model's prediction, we introduced a method that does not only consider the predicted parameter configuration but also a region next to the estimated configuration. Introducing a control parameter for the size of that region, gives the user the flexibility to trade-off between computation time and solution quality. Evaluated on 20% of the remaining problem instances, the MRIW values of the learning-augmented heuristics decrease by 1.66 percentage points on average but only need up to 10% of the initial computation time compared to the full grid search of BATCS-b or BATCS-d. If the solution quality is crucial in the application case, we can use the time savings of our ML-based parameter estimation to perform a post-optimization. Contribution B proposes a local search approach to improve solutions obtained by the ML-based parameter estimation method. The evaluation shows that by investing several minutes in the post-optimization, the solution quality substantially improves and even dominates the full grid search of both heuristic variants.

3.3 CONTRIBUTION C

(Uzunoglu et al., 2023b):

Contribution C tackles the problem of tuning the parameters for the BATCS-b heuristic for the serial-batch scheduling problem from another perspective. Assessing the performance of a parameter prediction by the distance to a best (known) parameter configuration assumes a "geometrical correlation" of parameter configurations and their solution quality when used in a heuristic. This becomes more of an issue if multiple parameter configurations may lead to the best solution, but we only use a single representative of the best solutions as a label. In this case, we would penalize our model for predictions close to a best parameter configuration that is not set as the representative label. As we present, this shortcoming can be circumvented by asking the ML model a different question. In this contribution, we develop ML models (i.e., NNs) that predict the objective value for a problem instance's numerical representation and a parameter configuration. The learned model can be leveraged to find a suitable parameter configuration for the problem instance by estimating multiple parameter configurations to an instance and selecting configurations that have "good" estimated objective values. Since those estimations are uncertain, we suggest creating a ranking based on the estimated objective values and applying that ranking for an Machine Learning Grid Search (called MLGS in contribution D and MLRP in this thesis; see in Figure 3) with different *ranking application strategies* to compensate for the uncertainty.

Incorporating the parameter configuration (κ_1, κ_2 and b for BATCS-b) into the feature vector handles the interdependency of the parameters naturally. Consequently, for every problem instance (93,360 in total), the number of possible data points corresponds to the number of parameter combinations (1,771 in total for our discretized intervals). This leads to a massive data set, making the hyper-parameter tuning of the ML model computationally intensive. Therefore, we propose to *subsample* a much lower number of parameter configurations per instance with different strategies and also evaluate how these strategies, combined with different feature vector representations, influence the accuracy of the prediction. The sampling strategies differ in how the samples are composed: the share of best (known) parameter configurations, the share of randomly selected parameter configurations, and whether or not the sample contains a "centralized" best (known) parameter configuration. After finding the best-performing NN hyper-parameters for every data pipeline, we validate which sampling and ranking application strategies perform best according to the MRIW metric.

The testing of our methods on 18,672 previously unseen problem instances (20% of all problem instances) reveals that MLGS only needs 10.8% of the initial computation time while only losing 2.46 percentage points of solution quality.

3.4 CONTRIBUTION D

(Uzunoglu et al., 2023c):

Contributions B and C proposed several solution methods (algorithms) to solve the PSBIJF. First, there are potentially two heuristics (BATCS-b and BATCS-d) differing on a lower level, namely when they stop filling batches. Second, both ML-based grid search variants, MLPP and MLRP, differ in their prediction strategy (predicting a single representative of best parameter configurations or a ranking of parameter configurations). Third, we examined different pipelines or even searching strategies for each ML-based grid search variant; each of them can be seen as an algorithm. However, analyzing the performance of these different algorithms does not reveal a clear best-performing algorithm – different algorithms dominate depending on the instance subclass. A situation like this raises the question of which algorithm to choose for a given problem instance. The Algorithm Selection Problem (ASP) deals with this question and is an emerging field that also applies ML. Many related publications utilize performance predictions as a basis for their decisions. In contrast to performance prediction methods, contribution D studies methods from a subfield of ML, learning-to-rank, for the ASP to find a well-performing algorithm. Learning-to-rank models are heavily used for predicting rankings in the literature on recommender systems and information retrieval. We evaluate the suitability of these and other modified learning approaches on the ASP. The results show that the learning-to-rank methods achieve good outcomes, but modifications of already existing learning algorithms perform similarly (even better for some cases).

Additionally, to further improve the solution quality, contribution D presents two Genetic Algorithms (GAs) that use the computed solution by the ASP's selected heuristic. The first GA represents the schedule in a more direct form allowing for better guidance of the population, but needs costly repair mechanisms to ensure the validity of the chromosomes after performing evolutionary mechanisms (e. g., mutation). In contrast, the second variant uses a random-keys-representation to encode schedules, which does not need repair mechanisms but makes it harder to stipulate desired solution properties. Both GAs need to set reasonable parameters to solve problem instances effectively. One of the parameters, for example, decides if the initial population should be generated by the selected heuristic (at the cost of its execution time) or created randomly (but therefore much faster). The parameter tuning results showed a substantially increased improvement when the selected heuristic generated the initial population. After setting the parameter of both GAs, we performed computational experiments on the test instance set.

These experiments indicated that, depending on the problem instance, both GAs achieve substantial performance improvements.

SCIENTIFIC CONTRIBUTIONS

CONTRIBUTION A

Applying machine learning for the anticipation of complex nesting solutions in hierarchical production planning

Christian Gahm, Aykut Uzunoglu, Stefan Wahl, Chantal Ganschinietz and Axel Tuma*

Published in the European Journal of Operational Research, 2022, 296(3), 819–836.
https://doi.org/10.1016/j.ejor.2021.04.006.
**Corresponding author*

Abstract:

In hierarchical production planning, the consideration of interdependencies between superior top-level decisions and subordinate base-level decisions is essential. In this respect, the anticipation of base-level reactions is highly recommended. In this paper, we consider an example from the metal-processing industry: a serial-batch scheduling problem constitutes the top-level problem and a complex nesting problem constitutes the base-level problem. The top-level scheduling decision includes a batching decision, i.e., the determination of a set of small items to be cut out of a large slide. Thus, to evaluate the feasibility of a batch, the base-level nesting problem must be solved. Because solving nesting problems is time consuming even when applying heuristics, it is troublesome to solve it multiple times during solving the top-level scheduling problem. Instead, we propose an approximative anticipation of base-level reactions by machine learning to approximate batch feasibility. To that, we present a prediction framework to identify the most promising machine learning method for the prediction (regression) task. For applying these methods, we propose new feature vectors describing the characteristics of complex nesting problem instances. For training, validation, and testing, we present a new instance generation procedure that uses a set of 6,000 convex, concave, and complex shapes to generate 88,200 nesting instances. The testing results show that an artificial neural network achieves the lowest expected loss (root mean squared error). Depending on further assumptions, we can report that the approximate anticipation

based on machine learning predictions leads to an appropriate batch feasibility decision for 98.8% of the nesting instances.

CONTRIBUTION B

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

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

Published in Computers & Operations Research, 2023, 151, 106122.
https://doi.org/10.1016/j.cor.2022.106122.

**Corresponding author*

Abstract:
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.

CONTRIBUTION C

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

Aykut Uzunoglu, Christian Gahm and Axel Tuma*
Published in Annals of Operations Research, 2023.
https://doi.org/10.1007/s10479-023-05541-w.
**Corresponding author*

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.

CONTRIBUTION D

Machine Learning based Algorithm Selection and Genetic Algorithms for serial-batch scheduling

Aykut Uzunoglu, Christian Gahm and Axel Tuma*

Submitted to Computers & Operations Research, 2023.

**Corresponding author*

Abstract:
Whenever combinatorial optimization problems cannot be solved by exact solution methods in reasonable time, tailor-made algorithms (heuristics, meta-heuristics) are developed. Often, these heuristics exploit structural properties and perform well on selected subsets of the problem space. For example, this is how the two best-known construction heuristics solve the scheduling problem investigated in this study (i.e., the scheduling of parallel serial-batch processing machines with incompatible job families, restricted batch capacities, arbitrary batch capacity demands, and sequence-dependent setup times). However, when the properties change, the performance of one algorithm might decrease, and another algorithm might have been the better choice. To resolve this issue, we propose using Machine Learning techniques to exploit the strengths of different algorithms and to select the probably best-performing algorithm for each problem instance individually. To that, we investigate a variety of methods from the "learning-to-rank" literature and propose several adaptations. Furthermore, because there is no algorithm for the considered scheduling problem that is capable to explore the entire solution space, we developed two Genetic Algorithms for the improvement of initial solutions computed by the selected algorithms. Here, we put special emphasis on ensuring that the solution representation (encoding) reflects the entire solution space and that the operators (e.g., for recombination and mutation) are appropriate to explore and exploit this space completely. Our computational experiments show an average increase of 36.73% in solution quality.

5 CONCLUSION

Machine Learning in combinatorial optimization is an ongoing, active field of research. This thesis presents different scientific contributions that use ML models on a complex machine scheduling problem. The following sections summarize the results and look forward to further questions of interest to both theory and practice.

5.1 KEY FINDINGS AND ADDED VALUE

This dissertation shows, based on a scheduling application case, how decision-making in Operations Research may benefit from including ML models in solution methods. All the contributions put emphasis on explaining all critical decisions that must be made in order to overcome common obstacles in ML and to obtain suitable ML models. For reproducibility reasons, the data sets in the contributions are made publicly available. Additionally, computational studies in each contribution have validated the applicability and usefulness of the proposed methods.

Contribution A first presents a serial-batch scheduling problem (top level) that has to solve a cutting and packing problem (base level) as one of its constraints, and both are NP-hard. These types of interdependencies are modeled in hierarchical production planning problems, and anticipated base-level models are commonly used to ease computations. This contribution uses an ML model to anticipate the answers to the nesting problem much more accurately than simple approximation methods. The increased accuracy in the anticipations has far-reaching implications for the application case – the number of false positive answers drastically decreases. Furthermore, the presented method is not only applicable to the presented application case but could be modified for other hierarchical production planning problems.

Contribution B focuses on the solution method that solves the serial-batch scheduling problem, defined as the top-level in the former contribution. It presents a refinement (BATCS-d) of an already existing heuristic (BATCS-b). Both heuristics need three parameters to be set, which hugely influences the solution quality. In the absence of prior knowledge about the dependency between parameters and problem instances, a grid search is a simple but time-consuming approach to find suitable parameters. Therefore, we translate the parameter-tuning problem to a regression problem and estimate parameter configurations for problem instances via ML models to augment our heuristics with these learned models. One challenge for the learning task is the fact that multiple parameter configurations could lead to the same objective value, which means that the model should actually output multiple predictions for a problem instance. We solve this by selecting the "most central" parameter configuration as the representative label. Again, we explain each decision to obtain our models in detail and publish our data set to ease the reproduction of our results. Furthermore, we introduce an additional search step based on the model's estimation to counteract the inherent uncertainty of predictions. A parameter for the search depth controls the trade-off between computation time and solution quality. Our computational studies show that the learning-augmented heuristics perform reasonably well while requiring only a fraction of the initial computation time, and by investing a short period of time in the post-optimization phase, our learning-augmented methods even outperform the computationally expensive full-grid search.

Contribution C deepens the analysis of the parameter search introduced in the previous contribution. Instead of estimating the parameter configuration directly, this contribution estimates the objective value achieved by the heuristic for a specific parameter configuration. An algorithm can then collect information on many different parameter configurations from the estimations to select a promising configuration. A significant methodological improvement is coupling the ML model's prediction with the heuristics performance. The loss function assesses, in this setting, the performance estimation of the heuristic rather than the geometrical "closeness" of the estimated configuration to the best (known) configuration. Also, this approach does not deal with the problem of having multiple valid estimations to an input. Because the feature vector now comprises the problem instance and the parameter configuration, the estimated performance is a unique output. However, combining the problem instance characteristics with different parameter configurations prohibitively increases the number of potential data points for the training. Therefore, this contribution invents subsampling strategies that immensely reduce the training effort. Additionally, it presents several methods to use the information on the estimated performance of parameter configurations to propose promising parameter configurations and simultaneously deal with the uncertainty in the estimated objective values.

Contribution D formulates the ASP on the developed algorithms in contributions B and C to choose the (probably) best performing algorithm for a PSBIJF instance. For that, it analyzes several state-of-the-art learning-to-rank algorithms and also introduces adaptations of existing ML algorithms to suggest an algorithm based on the features of a problem instance. The proposed learning-based algorithm selection increases the rate of selected best algorithms remarkably compared to a statically selected heuristic. Additionally, it presents two GAs that can use computed solutions of the former ASP as an initial population. Applying the GAs increases the solution quality by 36.73% on average.

5.2 OUTLOOK AND FUTURE RESEARCH

ML's breakthroughs have already had a tremendous impact on many different applications, so also on combinatorial optimization. To use ML models in combinatorial optimization, several integration points are proposed in the literature. For instance, to predict objective values or to estimate algorithm parameters. However, if model predictions are used in constraints, their lack of guarantee to correctly assess feasibility may depict a challenge for incorporating ML models. Consider, for example, the estimations used in Gahm et al., 2022b to anticipate complex nesting solutions. Underestimating the nesting height too much could lead to infeasible batches and schedules at the end. Future research could continue searching for methods that give users the flexibility to adjust the model's decisions according to their impact on the business case (e.g., by favoring models that overestimate if the uncertainty is high for a data point). A hot topic in ML is *quantifying an estimation's uncertainty* (see, for example, Gal and Ghahramani, 2016), i.e., how confident is the model that its classification is true or in which interval lies a regression output for a given significance level? Quantifying the uncertainty of an estimation could be valuable for improving decision-making in combinatorial optimization.

Contributions B and C show that besides asking for the "what", the "how" is also an important question to consider when integrating ML into combinatorial optimization. Both contributions aim to find suitable parameter configurations quickly but differ in how they formulate the regression question. Contribution C links the parameter configuration with the heuristic performance more naturally in its regression problem and hence allows for a more versatile application. For example, a GA could use performance estimations to define a diverse initial population. Lastly, the parameter tuning in contributions B and C searches in a discretized grid, as searching in a continuous parameter space is challenging. However, potential good candidates between the discrete parameters are missed. Contribution C showed that using randomly sampled configurations from discrete intervals performs well. Therefore, using randomly sampled configurations from a *continuous interval* to enable models to assess arbitrary queries would be a worthwhile future research topic.

Contribution D views the problem from a higher level and solves the ASP on the algorithms in contributions B and C and uses GAs for post-optimization of computed solutions. However, Kotthoff, 2016 describes solutions to the ASP that also allocate computational resources to algorithms. This is an interesting perspective on the situation in contribution D. Future research could, for example, analyze how they *allocate the computation time* for several interleaved algorithms (e.g., calculation of initial solutions and their post-optimization).

The contributions in this dissertation show that integrating ML models in solution methods for solving combinatorial optimization problems and particularly for machine scheduling unlocks a considerable potential for improvement in solution quality and efficiency. However, it requires a well-designed integration concept with a comprehensive analysis of what is needed from the ML model, and if existing ML models are capable of providing the desired answer. With the growing research in the intersection of both fields, even more potential and interconnections will be unveiled, driving us closer to the predicted "new era for combinatorial optimization algorithms" as stated in Bengio et al., 2021.

APPENDIX

A APPENDIX

A.1 CONTRIBUTION D

The following research article "Machine Learning based Algorithm Selection and Genetic Algorithms for serial-batch scheduling", referred to as contribution D, is under review in *Computers & Operations Research* (see Uzunoglu et al., 2023c). Due to the review process, the data repository was not linked in the document (see "*reference to be added*" in the text). The data is published under the following reference (Uzunoglu, 2023):
Uzunoglu, Aykut (2023), "Data set and models to select algorithms for a serial-batch scheduling problem", Mendeley Data, V1, doi:10.17632/4s2zfg6mg9.1

Highlights

Machine Learning based Algorithm Selection and Genetic Algorithms for serial-batch scheduling

Aykut Uzunoglu, Christian Gahm, Axel Tuma

- A serial-batch scheduling problem is the application case of our analysis.
- “Learning-to-rank” ML models select the best-performing heuristic.
- Two Genetic Algorithms are introduced with different encoding schemes.
- Both Genetic Algorithms can use solutions from the heuristics as initial populations.
- Our method substantially improves the solution quality.

Machine Learning based Algorithm Selection and Genetic Algorithms for serial-batch scheduling

Aykut Uzunoglu[a,*], Christian Gahm[a], Axel Tuma[a]

[a] *Business Administration Department, Chair of Production & Supply Chain Management, University of Augsburg, , Augsburg, 86159, , Germany*

Abstract

Whenever combinatorial optimization problems cannot be solved by exact solution methods in reasonable time, tailor-made algorithms (heuristics, meta-heuristics) are developed. Often, these heuristics exploit structural properties and perform well on selected subsets of the problem space. For example, this is how the two best-known construction heuristics solve the scheduling problem investigated in this study (i.e., the scheduling of parallel serial-batch processing machines with incompatible job families, restricted batch capacities, arbitrary batch capacity demands, and sequence-dependent setup times). However, when the properties change, the performance of one algorithm might decrease, and another algorithm might have been the better choice. To resolve this issue, we propose using Machine Learning techniques to exploit the strengths of different algorithms and to select the probably best-performing algorithm for each problem instance individually. To that, we investigate a variety of methods from the "learning-to-rank" literature and propose several adaptations. Furthermore, because there is no algorithm for the considered scheduling problem that is capable to explore the entire solution space, we developed two Genetic Algorithms for the improvement of initial solutions computed by the selected algorithms. Here, we put special emphasis on ensuring that the solution representation (encoding) reflects the entire solution space and that the operators (e.g., for recombination and mutation) are appropriate to explore and exploit this space completely. Our computational experiments show an average increase of 36.73% in solution quality.

Keywords: serial batching, total weighted tardiness, algorithm selection, genetic algorithm, machine learning

*Corresponding author

Email address: `aykut.uzunoglu@uni-a.de` (Aykut Uzunoglu)

1. Introduction

Whenever combinatorial optimization problems of industrial scale have to be solved, exact solution methods (algorithms in the narrower sense) need prohibitively long computation times due to the “combinatorial explosion” of the solution space (i.e., the set of all feasible solutions). Therefore, experts design heuristics or meta-heuristics (algorithms in a broader sense), incorporating knowledge about the specific problem to leverage a priori information about structures or properties of good solutions. These algorithms return good results for the subset of problem instances they were designed for, but fail to find good results for other subsets of the problem space comprising “all” problem instances of a specific problem. This is why, for most combinatorial problems, a variety of algorithms can be found in the literature. This situation also holds for the application case studied in this paper, a serial-batch scheduling problem often found in the metal processing industry (cf., e.g.,Helo et al. (2019)). It involves the decisions to group metal pieces into batches and to schedule the resulting batches, and can be summarized as the scheduling of parallel serial-batch processing machines with incompatible job families, restricted batch capacities, arbitrary batch capacity demands, and sequence-dependent setup times (named PSBIJF and classified as “P | if, crJ, sFS, sb | wT, F, Lex”-problem according to Gahm et al. (2022)). To solve the PSBIJF, several (learning-augmented) construction heuristics (LACH) can be found in literature (most recent ones in Uzunoglu et al. (2023a) and Uzunoglu et al. (2023b)), for which none of them clearly dominates the other.

The dominance of algorithms on a narrow subset of problem instances is more than a mere empirical observation but rather an underlying property of combinatorial optimization problems and their algorithms. In some sense, researchers refer to this phenomenon as the No Free Lunch theorem in optimization and search (NFL; see Wolpert and Macready (1997)). Like other “negative” theoretical results, such as Gödel’s incompleteness theorem in mathematics or Arrow’s impossibility theorem in social choice theory, NFL shows the theoretical boundaries of optimization, namely that no general algorithm can dominate other algorithms on the whole problem space or even all combinatorial optimization problems. Since NFL is not the focus of our paper, we refer the reader to Droste et al. (2002) for a more practical reframing of NFL, Ho and Pepyne (2002) for a version of finite and discrete input and output spaces, and Adam et al. (2019) for a review. Apart from the theoretical discussion, the NFL points towards an essential question for optimization problems: which algorithm to use when there is no clear winner?

Long before the formalization of NFL, Rice (1976) presented a framework that deals with the fact that algorithms dominate (in terms of performance) for certain problem subsets and get dominated by other algorithms for other problem subsets. In the presence of various algorithms for a problem, the so-called Algorithm Selection Problem (ASP) aims to find a selector that, depending on a given problem instance, chooses an algorithm performing best according to a performance metric. In its basic form, the ASP comprises a problem space P containing problem instances $i \in P$, an algorithm space A (finite or infinite), and a performance measure indicating the quality of an algorithm to solve a problem instance (e.g., objective value or computation time). Given that, the goal of the ASP is to find a selection function $S : P \rightarrow A$ mapping every problem instance to the algorithm that achieves the best value according to the performance measure. Rice proposes using a feature extraction step that converts a problem instance $i \in P$ to a feature representation $f(i) \in F$, in the feature space F, which is often $\mathbb{R}^m$. The framework introduced by Rice (1976) is very versatile and applicable to a broad range of problems and applications. Since then, many results have been published in Operations Research and other fields like "meta-learning" (a subfield of Machine Learning (ML)) that have successfully applied the idea of ASP (see Soares et al. 2004 or Feurer and Hutter 2019). Some results incorporate ML models in their selection procedure that predict an algorithm's performance metric (e.g., objective value or computation time) on a specific problem instance. Others even aim to design more sophisticated methods that create an "algorithm schedule" for a specific problem instance (e.g., Streeter et al. 2007 or Kadioglu et al. 2011). In such approaches, computational resources are not allocated to a single algorithm but are distributed among several algorithms according to their expected performance. Information from preceding algorithm runs, or even a pre-solving phase, can be used to make more intelligent decisions. Overall, the contributions to ASP have evolved massively, helping to improve solution quality by combining algorithms and adding their strengths. However, as Smith-Miles (2009) argues in her review on ASP and meta-learning, both research areas could benefit from each other, but those connections have been missed in the past. Algorithm selection could use meta-learning methods and ML models if the following requirements are fulfilled (Smith-Miles, 2009):

i A large collection of problem instances with various complexities is available.
ii Many diverse algorithms to solve the problem exist.
iii Performance metrics to evaluate the algorithm's performance are known.
iv Suitable features to describe the properties of problem instances can be computed.

With those requirements, the ASP can be stated as a learning problem: given the feature vector representation of the problem instances and objective values computed by the algorithms as training data, learn an ML model that predicts the best-performing algorithm. In this representation, the ML model corresponds to the function S in Rice's framework. In his review of algorithm selection applied to combinatorial optimization problems, Kotthoff (2016) emphasized the role of ML in *performance prediction* models. The ongoing progress suggests that "free lunches" are getting closer. With their growing prediction performance, ML models already play a significant role in the current literature and will gain importance in developing ASP solutions. They are capable of recognizing patterns between algorithms, problem instances, and their performance. Nevertheless, to achieve the most benefit from ML models, the question to be asked (i.e., what should be predicted) and the application of the model must be carefully designed.

In this paper, we propose using an ASP method to select the most appropriate algorithm for a single problem instance offline (i.e., before the solution process starts) and to use this algorithm to compute initial solutions to be improved by two Genetic Algorithms (GAs).

We make two major contributions to the literature:

– First, we thoroughly discuss and analyze ML models (from different research areas) and present several adaptations (e.g., an adapted loss function) for the ASP task at hand. In this context, we put special emphasis on the "question to ask" the ML model and on finding suitable models to answer it. As we show, "learning-to-rank" models are better suited for our ASP than pure performance predictions.

– Second, as no meta-heuristic has been developed so far for the PSBIJF, we present two GAs as improvement procedures. The two GAs mainly differ in their solution representation: the first one uses a single integer chromosome newly designed for the PSBIJF (but not restricted to it), and the second one, an adaptation from literature, combines an integer chromosome with a random keys chromosome and is enhanced by new recombination and mutation operators.

The structure of the paper is as follows. In Section 2, the PSBIJF is formally described, and existing algorithms for solving it are described. In Section 3, we discuss literature related to learning-to-rank models and GAs developed to solve problems similar to the PSBIJF. Section 4 presents the methods applied to solve the ASP, and Section 5 presents the two GAs. The basic experimental setup and the (hyper-)parameter tuning are described in Section 6 and Section 7, respectively.

The final experimental results are presented in Section 8. In the closing Section 9, we summarize the findings and give an outlook on future research topics.

2. Problem description (PSBIJF) and existing algorithms

The analyzed serial-batch scheduling problem, which we call PSBIJF (Parallel Serial-Batch scheduling with Incompatible Job Families), is described in this section. Additionally, we present existing algorithms in the literature that can solve the PSBIJF.

2.1. The PSBIJF

The basic task of the PSBIJF considered in this paper 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 scheduling (machine allocation and sequencing) of those batches on a set of m identical parallel machines ($M = \{M_l \,|\, l = 1, \ ..., \ m \in \mathbb{Z}^>\}$). The maximum batch capacity bc is identical for all machines, and the sum of the individual (arbitrary) batch capacity requirement cr_j of each job assigned to a batch must be lower. Generally, $cr_j \leq bc$ must hold. Furthermore, each job has individual weights w_j, processing times p_j, due dates d_j, and each job belongs to a job family f ($F = \{F_f \,|\, f = 2, \ ..., \ q \in \mathbb{Z}^>\}$), 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, batching 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 no more than 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), that jobs cannot be added or removed once processing of a batch has started (i.e., batch availability), and that the completion time of a job is the completion time of the batch to which a job is assigned (see Gahm et al. (2022) or Uzunoglu et al. (2023a) for more details).

The primary objective is the minimization of the total weighted tardiness. Additionally, we aim to minimize the total flow time whenever all jobs can be delivered in time (i.e., tardiness is equal to zero). This objective function is written as follows:

$$ov = \sum_{j \in J} w_j \cdot T_j + \frac{TF}{\tilde{C}_{\max} \cdot n \cdot 10} \tag{1}$$

with tardiness of job T_j and total flow time TF. The ordering of the objectives is achieved by defining the (constant) denominator to be greater than the nominator (i.e., the total flow time) in the second part. To assure that the denominator is greater than the TF, the upper bound of the maximum flow time of one job is estimated by the approximated makespan $\tilde{C}_{\max}$ and multiplied by the number of jobs. Because the makespan is approximated, we use a "safety" factor of 10.

2.2. Existing algorithms

For the first time, the previously defined PSBIJF was considered in Gahm et al. (2022). The authors developed a mixed-integer linear program (MILP), multi-start construction heuristics based on existing, adapted, and new priority rules, and a local search mechanism. Multi-starts are used to perform a grid-search with different heuristic parameter configurations. Their results show that the BATCS-b heuristic (originally abbreviated ATCS-BATCS(β)) with a "controlled batch utilization" outperforms all other approaches.

Based on these results, Uzunoglu et al. (2023b) developed a similar multi-start heuristic with "controlled batch urgency" (called BATCS-d) and used the same MILP and local search for improving initial solutions. Furthermore, to improve solution efficiency, the authors proposed learning-augmented heuristics using ML methods (i.e., Neural Networks (NNs)) to minimize the number of starts by predicting most suitable parameters for the heuristics. To that, not only the single predicted parameter configuration is used, but a reduced parameter grid is computed, and a parameter is introduced to control the size of the reduced grid (and thus, can be used to balance between solution quality and computation time). Their results show that the best variants with ML-reduced grids (BATCS-b-ML($PC_8, GS3$) and BATCS-b-ML($PC_6, GS3$)) are very competitive regarding solution quality and clearly outperform BATCS-b and BATCS-d in terms of computation time. The results also show that the "d"-variants generally outperform the "b"-variants but that for some problem instances, only a "b"-variant is capable to compute the best solution.

A similar approach to increase solution efficiency was proposed in Uzunoglu et al. (2023a). The authors also used ML to reduce the parameter grid searched by the multi-start heuristic BATCS-b but did not predict parameters. Instead, they used NNs to predict the performance of a certain parameter configuration and compute a ranking based on the predictions. After computing this ranking, which can be done efficiently due to the very low response times of NNs, several ranking application strategies are available to create a reduced parameter grid. Also, a parameter controls the size of the reduced grid. Their results show that their BATCS-b-MLGS variants are competitive regarding solution quality and clearly outperform BATCS-b in terms of computation time. Similar to the findings in Uzunoglu et al. (2023b), Uzunoglu et al. (2023a) observe that not one specific BATCS-b-MLGS variant performs best for all problem instances but that different ones achieve best results in relation to some problem instance characteristics. Therefore, we conclude that selecting the most promising algorithm among the best available for the PSBIJF before starting the solution process has the potential to improve the solution quality without increasing the computation time.

Table 1 lists the 13 most efficient algorithms from the literature. Note that we do not consider the algorithms presented in Uzunoglu et al. (2023b), which use improvement procedures since the ASP task considered here is to select the algorithm that efficiently computes initial solutions.

Table 1: Algorithms from the literature

Algorithm based on parameter predictions(see tables 21 and 22 in Uzunoglu et al. (2023b))	Algorithm based on ranking predictions (see table 6 in Uzunoglu et al. (2023a))
BATCS-b-MLPP($PC_4, GS\,3$)	BATCS-b-MLRP ([0,5,5]-AF, Bx)
BATCS-b-MLPP($PC_8, GS\,3$)	BATCS-b-MLRP ([0,5,5]-AF, B(9)-G)
BATCS-b-MLPP($PC_{10}, GS\,3$)	BATCS-b-MLRP ([1,2,7]-AF, Bx)
BATCS-d-MLPP($PC_1, GS\,3$)	BATCS-b-MLRP ([1,2,7]-AF, B(9)-G)
BATCS-d-MLPP($PC_4, GS\,3$)	BATCS-b-MLRP ([1,0,9]-AF, Bx)
BATCS-d-MLPP($PC_6, GS\,3$)	BATCS-b-MLRP ([1,0,9]-AF, B(9)-G)
BATCS-d-MLPP($PC_{10}, GS\,3$)	

Because Uzunoglu et al. (2023a) did not consider "d"-variants in their analysis, we performed a preliminary study to close this gap (see Table A.12 in Appendix A). Based on the results of this study, we added the four most robust algorithms (BATCS-d-MLRP ([1,2,7]-AF, B(5)-G), BATCS-d-MLRP ([1,2,7]-AF, B(9)-G), BATCS-d-MLRP ([1,4,5]-CF, B(5)-G), BATCS-d-MLRP ([1,4,5]-CF, B(9)-G)) to the algorithm space A (with $|A| = 17 =: n_A$). Note that the MLRP-variants in their original version use larger grids (compared to the MLPP-variants), resulting in higher computation times. To align both variants, we adjust the grid sizes for the

MLRP-variants in a way that all 17 considered algorithms perform similar grid searches. The following grid sizes are used (cf. Uzunoglu et al. (2023b)): 228 if $n < 100$, 76 if $n \in [100, 1000)$, and 20 if $n \geq 1000$. Note that for the “d”-variants, the grid sizes are 1 less because the set of considered “d”-values only contains 10 elements (the set of “b” values contains 11 values).

3. Related literature

3.1. Learning-to-rank literature

Concerning the problem at hand, we know that the four requirements i. to iv. (see Section 1) for using meta-learning methods are met. Therefore, we conclude that an offline trained ML model is suitable to fulfill the ASP task. For using ML models, one should carefully consider what level of information is necessary for algorithm selection, i.e., what the model’s response should be. For example, is it necessary to predict the objective value of an algorithm, or is an ordering of algorithms (i.e., ranking) sufficient to select an algorithm? The latter question might be easier to answer when the objective values differ immensely between problem instances (like it is the case for given PSBIJF). Note that a ranking suffices to select an algorithm with a fixed set of algorithms but is not applicable otherwise. Because such a ranking better reflects its application for the algorithm selection, it could also be seen as the “more natural” question to be asked. However, we will analyze both approaches in this paper.

For predicting objective values of algorithms, ML models and methods capable to perform regressions are required. As such ML models are widespread in literature, we are not going to discuss detail here but refer the reader to some standard literature (e.g., Murphy (2013) or Goodfellow et al. (2016)). In contrast, ML techniques returning a ranking are not so common and will be discussed in detail in the following.

As pointed out, a ranking of algorithms better reflects the application of the ML model output than a mere objective value prediction. ML models returning a ranking of objects are known as “learning-to-rank” models and are most often found in the field of recommender systems (e.g., recommending products to customers based on purchase history). Also, the terminologies “collaborative filtering” and “information retrieval” are used in the literature. For such problems, the ranking problem is stated as follows: given a query q_i (e.g., a user’s purchase history) and objects $x_1, \ldots, x_n$ (e.g., documents or products), find a ML model f that returns a ranking of objects x_a according to their relevance to the query. This assumes a numerical representation of the query q_i and the objects x_a. In the literature, the operation modes of the learning-to-rank models are defined to these three options:

- Pointwise ranking: every object gets a relevance score independent of other objects and is sorted according to the relevance score.
- Pairwise ranking: the model predicts which one to prefer for each pair of objects (the final ranking results from multiple preference relations predicted by the model).
- Listwise ranking: given a set of objects, the model outputs a permutation of the objects, considering the dependency in between.

Pointwise ranking computes a relevance score for each object and sorts accordingly. In this setting, standard loss functions, such as Mean Squared Error (MSE) (if different, possibly continuous, levels of relevance exist) or binary cross entropy (if only levels "relevant" and "not relevant" exist), can be used directly. However, they fail to capture the interdependency between objects related to a query (Freund et al., 2003), which is better incorporated in the last two options – pairwise and listwise approaches. A common drawback of pairwise ranking approaches is, however, that the learning objective minimizes the error in classifying preferences on pairs of objects rather than the error in ranking the objects (Cao et al., 2007).

Listwise ranking approaches can overcome this problem by calculating the loss function on the predicted ranked list of objects and comparing it to the ranked list given as ground truth instead of object pairs. However, a major challenge in this approach is to find an appropriate loss function that can be used in common learning mechanisms (e.g., the loss function must be differentiable for using gradient descent). Note that we refer to the lowest index in the ranking as highest position (indicating the probably best algorithm in the ASP) and the highest index in the ranking as lowest position (indicating the probably worst algorithm in the ASP). Järvelin and Kekäläinen (2000) proposed the metric Normalized Discounted Cumulative Gain (NDCG), taking into account that objects with high relevance in a higher position in the predicted ranking return a higher value (gain) for the user but objects in lower positions in the predicted ranking are less likely to be used. So, for example, misplacing the highest two objects in the ranking results in a higher loss than misplacing the lowest two objects. To define the NDCG metric, we first need a function $g\colon X \to \mathbb{R}$ returning the gain (relevance or value) of objects for a query and the cumulative gain function $cg : \mathbb{N} \to \mathbb{R}, p \mapsto \sum_{k=1}^{p} g(\sigma(k))$ that sums up the gains obtained by the ranking σ up to position p. Then, the cumulative gain is *discounted* according to the ranking position, for example, by dividing each gain in the sum by $\log_2(\cdot)$ of its rank: $dcg\colon p \mapsto \sum_{k=1}^{p} \frac{g(\sigma(k))}{\log_2(\sigma(k))}$. Other monotonically increasing transformations can be used for the discounting. Lastly, the achieved Discounted Cumulative Gain (DCG) is *normalized* by the highest DCG achievable by the optimal ranking (i.e., NDCG) to allow comparing predicted rankings between queries with different resulting rankings (Järvelin and Kekäläinen, 2002). The summation can also be truncated to a specific position k such that NDCG only considers the k highest ranked positions of the prediction (called NDCG(k)). As NDCG depends on the ranking σ, which returns the position of object x_a and hence is non-differentiable, the NDCG metric itself is non-differentiable and thus cannot be used in ML mechanisms relying on the loss function's gradient. Burges et al. (2006) circumvent the non-differentiability of the NDCG metric in their algorithm "LambdaRank" by defining a smooth approximative loss function. During training, it is known which properties the gradient should have to achieve a better ranking outcome. To derive an appropriate loss function, the goal is to choose the gradient of the desired loss function λ that fulfills this property. In their evaluation, they analyzed different λ functions and compared them against RankNet using the NDCG metric. Their analysis revealed that neural nets trained with LambdaRank clearly outperform a competitive pairwise approach (RankNet; see (Burges et al., 2005)) regarding accuracy (NDCG) and training time. Its boosted decision tree equivalent was named "LambdaMART" and published in Burges (2010). Also, with a focus on treating the ranking problem in a listwise manner, Cao et al. (2007) present "ListNet". ListNet is a neural network operating on probability distributions over permutations or the probability of objects being ranked on the top k positions (called top k probability). The computational experiments show the superiority of ListNet over other pairwise competitive algorithms on almost every test data set, concluding the advantage of listwise approaches over pairwise approaches.

Another line of research takes a more direct approach to find functions that are easier to optimize and behave like the metric of interest (e.g., NDCG). Some works try to optimize functions that are (upper) bounds to a ranking metric (see Chapelle et al. (2007) for a 'structured output' based approach and Xu et al. (2008) for a framework on upper bounds and applications). Another approach by Taylor et al. (2008) (called "SoftRank") tries to smooth the metric by assuming randomness and using a Gaussian distribution as a ranking score. Qin et al. (2010) proposed an approximation framework that reformulates metrics needing a sorting, meaning they define positions as indices (i.e., indexed by positions), to metrics that derive indices from their objects. Second, they approximate the position function with a logistic function based on the object's ranking score and apply their approximation in an optimization technique called “ApproxNDCG” (and “ApproxAP” for average precision metric), which also needs a hyper-parameter to be determined. They compared the accuracy of ApproxNDCG to SoftRank, ListNet, and others (but without including LambdaRank and LambdaMART in their analysis). They showed that ApproxNDCG outperforms all other methods on a variety of data sets.

The literature on learning-to-rank presents a rich body that differ in how they treat the dependency between objects: pointwise approaches neglect dependencies between objects, pairwise approaches consider preferences between pairs of objects, and listwise approaches work on the complete list of objects. Listwise approaches best reflect how the prediction is applied afterward, but loss functions of interest are not differentiable, making them not directly usable in gradient-based methods. Plenty of works attempt to overcome this issue, and the more recent approaches, such as LambdaRank/LambdaMART or ApproxNDCG, have shown to be superior in computational studies. In consequence, these are of special interest regarding the ASP task at hand.

3.2. Genetic algorithm related literature

The following literature review analyses publications using GAs for solving batch scheduling problems that are closely related to the problem at hand, i.e., batch scheduling problems with bounded batch capacities, batch availability, and incompatible job families. Using the knowledge base provided by Wahl et al., we identified 13 relevant articles using GAs (note that we also use their notation scheme to specify batch scheduling problems in the following). Since our goal is to design a GA for the PSBIJF, we put special emphasis on the analysis of the used evolutionary mechanisms (e.g., mutation or recombination). Another important aspect for each GA is whether the chromosome decoding procedure (or any other batching procedure) follows a ”full-batch-policy” (i.e., batches are always filled with jobs until batch capacity does not exceed) or not. As discussed in Gahm et al. (2022) and Uzunoglu et al. (2023b), forced full batches may lead to suboptimal decisions for the PSBIJF (but, of course, might have been appropriate for the originally addressed problem). Furthermore, our GAs are designed to improve the quality of pre-computed solutions. In order to represent these pre-computed solutions and not miss regions that could further improve them, we require our GAs to operate on the entire solution space. Two aspects that are contrary to this goal (even if they have other benefits) are: decomposition approaches (instead of solving all subproblems jointly) and exclusion of certain parts of the solution space. Therefore, we report if the GA solves the complete problem or a decomposition approach is used, and if the solution methods cover the entire solution space of the problem. In Table 2, the developed GAs of the relevant literature are summarized according to the discussed aspects, if a full-batch-policy was used (F) or not (L), if the entire problem was solved by the GA (C) or a decomposition approach was used (D), the type of representation/chromosomes (bin := binary, int := integer; rk := random key, and obj := objects), and the information represented by the chromosomes.

Table 2: Overview of related GA developments

Reference	Problem specification	(F) or (L)	(C) or (D)	Chromosome type(s)	Chromosome information (size)
Balasubramanian et al. (2004)	P | pF, if, cr1, pb | wT'	F	D	int	Job to batch assignment (n)
		F	D	int	Batch to machine assignment (n)
Koh et al. (2004)	P | pF, if, crJ, pb | C, wC, Cmax'	F	C	rk+rk	Job to batch assignment + batch to machine assignment with batch sequence ($2n$)
		F	D	rk	Job to batch assignment (n)
Koh et al. (2005)	S | pF, if, crJ, pb | C, wC, Cmax'	F	D	rk	Job to batch assignment (n)
Mönch et al. (2005)	P | pF, rJ, if, cr1, pb | wT'	L	D	int	Job to batch assignment (n)
		L	D	int	Batch to machine assignment (n)
Malve and Uzsoy (2007)	P | rJ, if, cr1, pb | Lmax	F	C	rk	Job sequence (n)
Mönch et al. (2007)	P | rJ, net, if, cr1, sFS, elig, pb | wT	L	D	int	Batch to machine assignment (n)

Dauzère-Pérès and Mönch (2013)	S | pF, dJ, if, cr1, pb | wU	F	C	rk	Job sequence (n)
Jia et al. (2013)	P | pF, rJ, re, if, cr1, maxL, pb, bLb, on | wT, TP, cIh	F	D	int	Batch to machine assignment (n)
Castillo and Gazmuri (2015)	HJ | if, crJ, sFMS, cb, sb, bF, bLb | Cmax	L	C	objects	Batch sequence, size, machine assignment, family (n)
Huynh and Chien (2018)	P | pF, dlJ, if, crJ, sFS, pb | Cmax	L	C	int+rk	Job to batch assignment + batch to machine assignment with batch sequence ($2n$)
Huang et al. (2020)	S, aFlex | rJ, if, cr1, pb | Cmax'	F	D	rk+ binary	Job sequence + preventive maintenance ($2n$)
Kim et al. (2021)	S, aFlex | pF, pDet, if, crJ, pb | Cmax	F	D	rk	Job to batch assignment (n)
		F	D	rk	Batch sequence (up to n)
		F	D	rk+rk	Job to batch assignment + Batch sequence (up to $2n$)
Wu et al. (2022)	HJ, aFlex | pF, rO, dlO, re, if, cr1, elig, pb, bFM | TP'	L	C	objects	Job to batch assignment, batch to machine assignment, and batch sequence (up to n)

All approaches not using a full-batch-policy are of greatest interest and, therefore, analyzed in greater detail in the following.

Mönch et al. (2005) proposed using GAs to solve the "P | pF, rJ, if, cr1, pb | wT"-problem. They used the same two decomposition approaches first presented in Balasubramanian et al. (2004) but combined the GAs with other heuristics for batching and sequencing. Their modified ATC dispatching rule considered multiple batch combinations, and thus, also batches with "free" capacity are considered (no full-batch-policy). However, due to the decomposition approach and the applied dispatching rules for batching and sequencing, not the entire solution space is covered. In their GAs' implementations, the roulette wheel selection mechanisms and operators (one-point crossover and a flip mutation) were the same as in Balasubramanian et al. (2004).

To solve a complex job shop scheduling problem,Mönch et al. (2007) proposed a decomposition approach, which needs to solve the batch-scheduling sub-problem "P | rJ, net, if, cr1, sFS, elig, pb | wT". For solving this sub-problem, a GA allocated and sequenced already formed batches to and on machines. To that, the integer chromosome represents a batch's machine allocation, and the sequence of genes (representing batches) in the chromosome is equal to the sequence of batches on the allocated machine. The GA used a one-point crossover, flip mutation, and overlapping populations as evolutionary mechanisms. Computations terminated after reaching a given number of generations. The batching procedure considered multiple batch combinations and thus did not aim at full batches, but due to the decomposition approach, not the entire solution space is covered.

Castillo and Gazmuri (2015) proposed three GAs with different types of crossovers for solving the "HJ | if, crJ, sFMS, cb, sb, bF, bLb | Cmax"-problem. For solution representation, the authors used an ordered set of batches with additional information such as batch size and assigned machine. Three crossover mechanisms were developed: "edge recombination-based crossover", "batch position-based crossover" (similar to a one-point crossover), and "guided mutation crossover" (a local search around the clone parent, towards the guide parent). For mutation, the authors proposed four mechanisms that partially use local search to keep all chromosomes feasible: "mutate amount of batches", "mutate batch sizes", "mutate machine assignments", and "mutate batch sequence" (i.e., batch swapping). The experiments showed that the guided mutation crossover (with the integrated local search) had the fastest convergence of all three crossovers. The applied chromosome representation allows any batch size, the complete problem is solved, and the entire solution space is covered. However, solution representation and recombination and mutation operators are highly problem-specific.

For the "P | pF, dlJ, if, crJ, sFS, pb | Cmax"-problem, Huynh and Chien (2018) proposed a multi-subpopulation GA combined with heuristics. Because our second GA uses the same solution representation and similar operators, we are not going into detail here but in Section 5.2. The multi-subpopulation GA used three parallel subpopulations that evolved independently and were coordinated at certain points to prevent any single subpopulation from converging too quickly or slowly. To coordinate the three subpopulations, new subpopulations containing new chromosomes were created after a certain number of generations by interchanging a given number of best solutions. The results showed that the multi-subpopulation GA outperformed the conventional GAs by about 5% in terms of solution quality, while the CPU time was about the same. Unfortunately, the authors did not report results without integrating the two local search heuristics. Therefore, it is not possible to isolate their multi-subpopulation-strategy's contribution to the result. However, since the representation by two chromosomes allows arbitrary batch sizes and the complete problem is solved, the entire solution space for the problem at hand is covered, making their approach interesting for solving the PSBIJF.

To solve the "HJ, aFlex | pF, rO, dlO, re, if, cr1, elig, pb, bFM | TP"-problem, Wu et al. (2022) developed a GA where each chromosome represented a sequence of "batch objects" defining the machine (assigned tool), the job family (recipe), the set of assigned operations, and the start time of the batch. The authors developed several problem-specific properties to reduce the solution space for the GA. These properties were also used by the procedure to randomly generate the initial population. Based on a fitness-proportional parent selection, the offspring were generated through representation-specific crossover and mutation operations. The authors explicitly stated that they did not follow a full-batch-policy because smaller batches may lead to better schedules. The proposed GA covers the entire solution space of the problem considered here. However, due to the large difference regarding the basic problem settings compared to the PSBIJF and the GA's high problem specificity, particularly because of the used properties, we do not consider its application to be expedient.

4. Algorithm selection by learning-to-rank

As the more recent literature in algorithm selection suggests, ML models can be powerful tools to select good algorithms for a problem instance to solve. Empirical hardness models or models to predict the performance of an algorithm and problem instance are popular choices. However, predicting the actual performance value is unnecessary but a mere ordering of algorithms is sufficient for algorithm selection. Furthermore, evaluating the performance of an ML model for algorithm selection based on the error (e.g., the MSE) between the actual and predicted algorithm performance does not coincide with its later use. Loosely speaking, predicting an algorithm's (continuous) objective values is more challenging than predicting the algorithm's performance on an ordinal scale (e.g., ”good” or ”mediocre” performance) or a listing according to the performance (which implies the latter two cases). This becomes particularly important when objective values can vary by several orders of magnitude from problem instance to problem instance, as is the case for the PSBIJF with the total weighted tardiness objective. In consequence, learning-to-rank methods seem the better choice. Nevertheless, one should be aware that many learning-to-rank algorithms were developed with use cases very different from algorithm selection. They stem from applications like recommender systems or search engines with vast sets of objects explained by numerical feature vectors and deal with problems like bias in human-generated data. Therefore, thoroughly analyzing the applicability of different learning-to-rank methods is essential to find the most suitable technique for our ASP.

Because NN are commonly used by recent ranking methods (and regression tasks in general) and Gradient-Boosted Decision Trees (GBDT) have shown their superior performance as learning-to-rank methods, we use them as basic ML model types in our ASP methods.

In the context of ranking ML tasks like the ASP, the way of labeling the training data plays an important role. Therefore, we use several "labeling strategies" in our analysis. In the first strategy, called "BIN", all algorithms from A that have computed the best solution for a problem instance are indicated by 1 and otherwise by 0. That means, a sample for a problem instance has n_A binary labels indicating the performances of n_A algorithms. In strategy "OV", n_A real-value labels depict the objective values computed by each algorithm (ov_a) for a problem instance, and in "OVN", the objective values have been normalized on the interval $[0, 1]$ with $ov_a = \frac{ov_a - ov_{min}}{ov_{max} - ov_{min}}$ (regarding a single problem instance).
Furthermore, we use the labeling strategy "RANK", which has $n_A = |A|$ labels with an integer from the interval $\{1, ..., n_A\}$. Here n_A marks the best algorithm and 1 the worst. If, for example, two algorithms achieved the best objective values, both are labeled with 17, and the next is labeled with 15. We also use a version of RANK with scaled labels from $\{\frac{1}{n_A}, \frac{2}{n_A}, \ldots, 1\}$, called RANK-SC, to analyze the impact of scaling on the training strategies. Here, 1 marks the best algorithm and $\frac{1}{n_A}$ the worst.

In addition to the labeling strategy, the loss function used during training, validation, and testing is essential. Therefore, we investigate different loss functions that are coupled to the labeling strategy and model type (i.e., not all loss functions are applicable to all labeling strategies or model types). For the labeling strategy BIN, the loss function Binary Cross Entropy (BCE), and for the labeling strategies OV, N-OV, and RANK-SC the MSE are used. A newly developed loss function is called "importance weighted MSE" (iwMSE(α)). The idea of this loss function is to emphasize better (higher) ranked algorithms, so in a sense, the ML model also has to understand which algorithms should be ranked higher. It is basically the MSE with a weight depending on the "true" ranking y_a of algorithm

a: $\text{iwMSE}(y, \hat{y}) = \frac{1}{n_A} \sum_{i=1}^{n_A} (y_a - \hat{y}_a)^2 (\alpha y_a + 1)$ (with y_a representing the label and $\hat{y}_a$ the predicted ranking score). We use the parameter $\alpha \in \mathbb{R}^{>0}$ to control the effect of the weighting. Of course, this biases a model to predict higher ranking scores $\hat{y}_a$ since missing higher-ranked algorithms is punished stronger than missing lower-ranked algorithms. Therefore, the control parameter should not be exaggerated and be tuned with caution. Theoretically, an "optimal" model would not mind this effect, but approximation and estimation errors (and the fact that gradient descent could converge in a local optimum) hinder us from finding such models in practice. In our experiments, we analyze the impacts for $\alpha \in \{1, 2, 3\}$.

The following "loss functions" also adapt the learning process (algorithm) and therefore are not applicable to both model types (see Section 3.1 for further details): ApproxNDCG is a loss function particularly developed for NNs (Qin et al., 2010), LambdaRankNDCG is the learning method published in Burges et al. (2006) for the NDCG loss, and LambdaRankNDCG(1) is its truncated version to rank 1 only. For GBDTs and the labeling strategy BIN, the LambdaBinary trains according to the mean average precision Donmez et al. (2009). RankPairwise uses a GBDT-adaptation of the pairwise loss function RankNet developed by Burges et al. (2005), and LambdaMART is the GBDT-equivalent to LambdaRankNDCG presented in Burges (2010). All appropriate ML model types, labeling strategy, and loss function combinations forming ASP-models are summarized in Table 4 in Section 7.1. Note that all the presented ML model type, labeling strategy, and loss function combinations can be considered as listwise ranking approaches as we have a finite and fixed object space (algorithm space).

For the training, validation, and testing of the ASP-models, we use a large instance set from the literature (see Section 6.1) and solve each of the 71,040 instances with the 17 available algorithms. This leads to a set of 1,207,680 data points when each problem instance and algorithm combination is processed individually.

To convert a problem instance $i \in P$ into a feature representation, we use the AF-vector (aggregated feature-vector) proposed in Uzunoglu et al. (2023b), as it has shown to be suitable for adequately representing the properties of the PSBIJF problem instance.

Since with the objective function in Eq. (1) also a metric to evaluate an algorithm's performance exists, all four requirements to use ML models (see i. to iv. in Section 1) for solving the ASP Algorithm are fulfilled.

5. The Genetic Algorithms

To solve the present serial-batch scheduling problem, we propose to improve the solutions computed by the LACH with two GAs that mainly differ in their problem representation. The first one uses an integer representation where each job is assigned a batch-position (representing a specific position on one of the machines). We abbreviate this approach as GA-J2P (job-to-position assignment). In the language of genetics, this means that a phenotype (schedule) is encoded by a genotype (individual) consisting of a single chromosome of n genes (one for each job), and the allele is an integer value representing one of the given batch-positions (for definitions and details of GA-related terms used in this paper see Eiben and Smith (2015)). The second GA uses a genotype consisting of two chromosomes: The first chromosome consists of n integer genes representing the job-to-batch assignment, and the second chromosome consists of n real-valued chromosomes representing the batch-to-machine assignment and the sequence of batches on a machine. This one is called GA-J2BRK (job-to-batch assignment with random keys).

For both GAs, we implemented the following basic GA scheme (cf. Eiben and Smith (2015)) with the parameters "population size" (μ), "initial population composition" (ipc), "elitism rate" (esr; fraction of best individuals that are certain to be transferred to the next generation), "survival rate" (svr; fraction of individuals that survive, i.e., that are transferred unaltered to the next generation), "survivor selection mechanism" (svs; how individuals are selected for survival), "parent selection mechanism" (pas; how individuals are selected to compute offspring individuals), "recombination probability" (P_{rec}; probability that two parents are recombined into two offspring individuals), "mutation probability" (P_{mut}; probability that a gene of an offspring individual is mutated), and the "termination condition setting" (tcs).

The GA-specific components are described in the following sections. However, we use the same termination criterion "maximum execution times in seconds" for both GAs. This value is defined in relation to the number of jobs of an instance, as this is the major characteristic influencing the computation time. During different stages of the development and final testing of our GAs, we would like to use different settings to keep computation times manageable. Therefore, two settings are defined according to Table 3.

Table 3: Termination setting defining maximum GA runtimes in seconds [s]

Setting	$n \leq 600$	$n \in (600, 1200]$	$n \in (1200, 2, 400]$	$n > 2400$
TS1	30	60	90	120
TS2	90	180	270	360

5.1. GA-J2P

The basic idea of GA-J2P is the genotype representation of a schedule by a single chromosome containing the complete information without using any heuristics for batching or scheduling. The proposed representation can model the complete solution space, which is particularly important with respect to the problem characteristics combination of serial batching, arbitrary batch capacity requirements, and weighted tardiness. At the same time, we wanted to avoid an overly specific representation by complex genes (see. e.g., Castillo and Gazmuri (2015) and Wu et al. (2022)) to make the proposed GA applicable to a wider range of batch scheduling problems.

5.1.1. Genotype representation and decoding

The genotype consists of a single integer chromosome with one gene for each of the n jobs (gene indices are also job indices j), and the allele values represent batch-positions. Hereby, batch-positions simultaneously encode the machine and the sequence of batches on a machine, and therefore, each batch-position simultaneously represents one batch.
This representation idea is very similar to the main decision variable (X) used in the mixed-integer linear program presented in Gahm et al. (2022). In general, the number of batch-positions on each machine must not be less than n. However, since we observed that the number of batches created in the initial solutions is much smaller than this value and Gahm et al. (2022) report a similar observation (see their table 14), we introduce the parameter "batch-positions per machine" bpm) to control the total number of batch-positions. Here, we assume that smaller bpm values are beneficial for the solution process, but it is important to use values that do not restrict the solution space in a way that prevents good or optimum solutions. The effectiveness of different bpm values will be analyzed in detail in the experimental section. Batch-positions are numbered as follows: $bp = 1$ for the first batch on machine $i = 1$, bp=2 for the second batch on machine i=1, $bp = bpm$ for the last batch on machine $i = 1$, $bp = (i - 1)bpm + 1$ for the first batch on machine $i = 2$, $bp = (i - 1)bpm + 2$ for the second batch on machine $i = 2$, $bp = (i - 1)bpm + bpm$ for the last batch on machine $i = 2$, and so on. Fig. 1 illustrates in part a) the genotype chromosome assigning a batch-position to each job and in part b) the resulting phenotype schedule after decoding, i.e.,

Algorithm 1 General GA scheme

```
$pop(0)\{\ \}$ := getInitialPopulation($\mu$, $ipc$)
evaluate( $pop(0)\{\ \}$ )                          ▷ calculate fitness of all initial individuals
Do
    $g := g + 1$                                     ▷ increment generation
    $s\{\ \}$ := selectSurvivor($pop(g-1)\{\ \}, esr, svr, svs$)   ▷ survivors are directly transferred to the next generation
    $p\{\ \}$ := selectParents( $pop(g-1)\{\ \}, 1 - svr, pas$)    ▷ for recombination
    $o\{\ \}$ := recombine($p\{\ \}, P_{rec}$)            ▷ recombination of two parents
    $m\{\ \}$ := mutate($o\{\ \}, P_{mut}$)               ▷ mutation of the offspring
    evaluate($m\{\ \}$)                                ▷ calculate fitness of all offspring individuals
    $pop(g)\{\ \} := s\{\ \} + m\{\ \}$
Until $tcs$ is satisfied
```

Figure 1: Example genotype a) and its phenotype b) in GA-J2P

a)

$f=$	*X*	*Y*	*X*	*Z*	*Y*	*Z*	*Z*	*X*	*X*	*Y*	*X*
$j=$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
Chromosome A	2	5	2	3	8	1	3	6	6	7	9

b)

Batch per machine

	1	2	3	4	bpm=5
i=1	$\boxed{1}$ *6*	$\boxed{2}$ *1, 3*	$\boxed{3}$ *4, 7*	$\boxed{4}$	$\boxed{5}$ *2*
i=2	$\boxed{6}$ *8, 9*	$\boxed{7}$ *10*	$\boxed{8}$ *5*	$\boxed{9}$ *11*	$\boxed{10}$

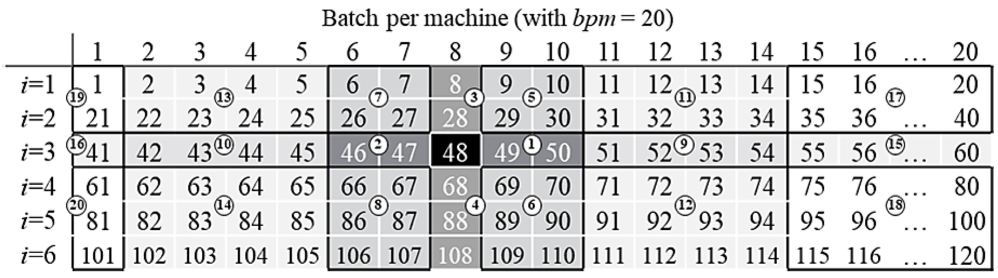
after assigning jobs (italic) to the batch-positions (in the squares) on the machines. Note that we do not have any ordering or sorting (e.g., by job families) in the integer chromosome to avoid any positional bias (see the associated families f per job in Fig. 1 a)).

The decoding follows the gene sequence of the integer chromosome and adds one job after the other to the corresponding batches: for example, in Fig. 1 b), job 1 to batch-position 2 → (then) job 2 to batch-position 5 → job 3 to batch-position 2 and so on. Note that empty batches (e.g., batch-position 4) are just ignored during the evaluation of the schedule.

5.1.2. Close and least impact insertion procedure (CLIP)

Several operations in GA-J2P (e.g., random solution generation or recombination) require the insertion of a job into "new" batch-positions if the desired batch-position is already occupied and an insertion is not feasible (due to job family or

Figure 2: Sections in the CLIP mechanism



Batch per machine (with *bpm* = 20)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	…	20
i=1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		20
i=2	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	…	40
i=3	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	…	60
i=4	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	…	80
i=5	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	…	100
i=6	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	…	120

batch capacity requirements). In this case, our proposed insertion procedure aims to insert the jobs into batch-positions closest to the initially desired batch-position (e.g., near on the same machine or at a similar position on a different machine) and that have the least impact on the overall schedule. The latter aspect is important to allow the transfer of "good" partial schedules from one genotype to the other. To achieve this insertion in an efficient manner, we define 20 insertion sections, as exemplarily depicted in Fig. 2.

For the definition of these sections, we use several bounds in relation to *bpm*: sections 1 to 8 are defined by $lb_1 = 1$ and $ub_1 = 0.1 \cdot bpm$, sections 9 to 14 are defined by $lb_2 = ub_1 + 1$ and $ub_2 = 3 \cdot ub_1$, and sections 15 to 20 are defined by $lb_3 = ub_2 + 1$ and $ub_3 = bpm$. Given these definitions, Fig. 2 illustrates the procedure to find a new batch-position for all the jobs with the desired batch-position 48: we first try to insert the job in a batch-position in section 1 (49→50), then in a batch-position in section 2 (47→46), then in a batch-position in section 3 (28→8), then in a batch-position in section 4 (68 → 88 → 108), then in a batch-position in section 5 (29 → 9 → 3 → 10), then in a batch-position in section 6 (69 → 89 → 109 → 70 → 90 → 110), then in a batch-position in section 7 (27 → 7 → 26 → 6), then in a batch-position in section 8 (67 → 87 → 107 → 66 → 86 → 106), and so on.

Note that the sections 3, 5, 7, 11, 13, 17, and 19 contain the corresponding batch-positions from the current machine (here 3) to machine 1 ($i = 1$) and that the sections 4, 6, 8, 12, 14, 18, and 20 contain the corresponding batch-positions from the current machine (here 3) to machine $i = m$.

5.1.3. Initial population composition

To generate the initial population for GA-J2P, we use three approaches controlled by the parameter "initial population composition" (*ipc*): the first approach uses only randomly generated solutions (*ipc* = *ran*), the second one uses 30% initial solutions computed by the LACH and 70% randomly generated solutions (*ipc* = *icr*), and the third one uses 30% initial solutions computed by the LACH and 70% randomly modified initial solutions (*ipc* = *iri*; by the recombination and mutation variation operators described below). The latter two approaches significantly differ in the diversity of the initial population, and we will investigate the effect on the solution process in our experimental study. Depending on the population size and since the learning-augmented heuristics provide only a limited number of solutions related to the number of jobs (i.e., 227 or 228 if $n<100$, 75 or 76 if $n \in [100, 1000)$, and 19 or 20 if $n>1000$; cf. Section 2.2), the number of available initial solutions may not be sufficient with respect to the given proportion. In this case, additional random solutions are added to the initial population according to the population size μ. Because of the limited number of available initial solutions, we are not going to tune the fraction of initial solutions during parameter tuning.

For the fully randomly generated solutions, we draw batch-positions from $\{1, \ldots, m \cdot bpm\}$ for each job (note that if not stated otherwise, random numbers are always drawn from restricted uniform distributions). To ensure feasible solutions in the initial population, we decode all generated genotypes after generation. If we identify an infeasible batch in terms of incompatible job family or batch capacity during decoding, we store the job in an additional list instead of adding it to the batch. After adding all jobs to the schedule or the list, we sort the list by non-decreasing batch positions and add the jobs to the schedule in that order using CLIP.

For generating randomly modified initial solutions, we randomly pick two of the initial solutions, apply all five mutation operators to them, use all two recombination operators to generate four offspring, and then apply all five mutation operators to the offspring. This leads to $10 + 4 + (4 \cdot 10) = 54$ randomized initial solutions. These actions are repeated with a gradually increasing chance of mutation until the necessary quantity of initial solutions is reached. The probability of mutation begins at .02 and rises by .005 per cycle. As all operators compute feasible solutions, an additional feasibility checking is not required here.

5.1.4. Elitism and survivor selection

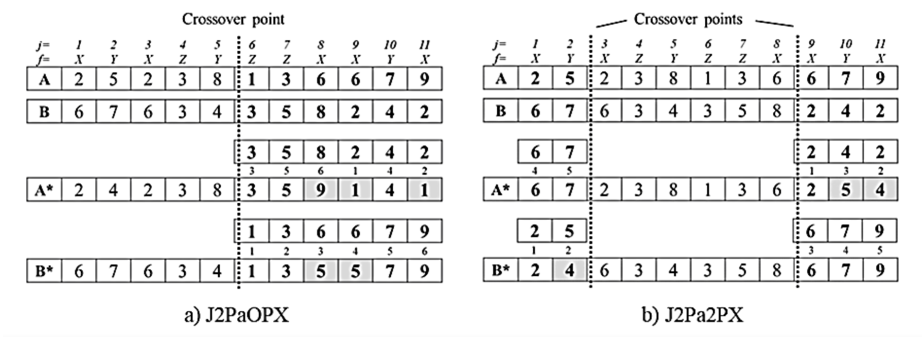
Since GA-J2P is basically designed to improve initial solutions, we use an elitism mechanism that directly transfers a certain integer number ε of best genotypes to the next population. This number is controlled by the parameter "elitism rate" *esr* and defines the number of transferred best genotypes as $\varepsilon = \lfloor esr \cdot \mu \rfloor$. For the same reason, we follow a steady-state population management model, i.e., we do not replace the entire population in each generation, but only a part of it (cf., Eiben and Smith (2015)). In literature, the number of individuals replaced by its offspring is named λ, and thus, the number of surviving individuals is $\mu - \lambda$. As the number of surviving genotypes is controlled by the "survival rate" parameter svr, $\lambda = \mu - \lfloor svr \cdot \mu \rfloor$. To preserve the diversity of populations, not the best $\mu - \lambda$ genotypes are selected but more advanced selection mechanisms, controlled by the "survivor selection mechanism" parameter svs are applied. Here, we will study the fitness-based selection mechanisms "Tournament Selection" (TOS) and "Stochastic-universal Sampling" (SUS). In TOS, a genotype wins a "tournament" if its fitness is greater than the fitness of the other $s-1$ competing genotypes, whereat the competing genotypes are drawn randomly. To vary selection pressure, one can adjust the tournament size s. When s is larger, genotypes with lower fitness have a reduced chance of surviving. Note that the worst genotype never survives, and the fittest genotype is the winner of every tournament in which it competes. The tournament selector is commonly used in practice due to its lack of stochastic noise in comparison to fitness proportional selectors. In contrast, SUS chooses individuals based on a given probability (related to the fitness) to minimize the chance of fluctuations. It can be seen as a form of a roulette wheel game with evenly spaced points that we spin. SUS uses a single random value to select individuals at equally spaced intervals. This fitness-based selection method grants a better chance of the selection of weaker individuals, thus reducing the "unfairness" associated with other fitness-based selection methods. Because both survivor selection mechanisms are intended to preserve diversity, we use a tournament size of two ($s = 2$) in all experiments. The parameters esr,svr, and svs will be tuned later.

5.1.5. Parent selection and recombination

To randomly select the parents that are used for offspring generation, we also study the two selection mechanisms TOS (with $s = 2$) and SUS from Section 5.1.4. The parameter "parent selection mechanism" pas is used to differentiate between them.

For the recombination of two genotypes, we adapted two standard operators from the literature: one-point crossover and two-point crossover. The first operator, "J2P adapted one-point crossover" (J2PaOPX), begins with randomly selecting a crossover point from $\{2, \ldots, m \cdot bpm - 1\}$ (e.g., 6 in Fig. 3 a)). This point partitions both the chromosomes A and B into head and tail (see Fig. 3 a)). The J2P-specific adaptation takes place with the exchanging of the tails. Here, the batch-positions are transferred by CLIP in non-decreasing order of batch-positions and job indices (see the small numbers in Fig. 3 a)) to the two offspring A* and B*, respectively. Note that all following examples assume $m = 1$, $bc = 2$ (jobs), and $bpm = 11$. Furthermore, actualized batch-positions are marked bold, and that

Figure 3: Example one-point a) and two-point crossover recombinations



gray shaded fields signal batch-position updates by CLIP.

The second recombination operator "J2P adapted two-point crossover" (J2Pa-2PX) follows the same procedure but uses two randomly selected crossover points from $\{2, \ldots, m \cdot bpm\}$ (e.g., 3 and 9 in Fig. 3 b)) and first transfers the middle parts and then CLIP inserts head and tail in non-decreasing order of batch-positions and job indices (see Fig. 3 b)).

Both recombination operators have individual recombination probabilities (P^{J2P}_{aOPX} and P^{J2P}_{a2PX}) that must be tuned, whereby a value of 0 indicates that the operator is not in use (as also for the following mutation operators).

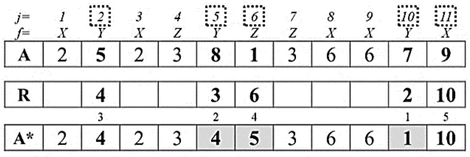
5.1.6. Mutation

Mutation operators have two distinct (conflicting) roles in the evolution process of a GA: Exploitation, i.e., intensification of the search in promising regions of the solution space by making small changes, and exploration, i.e., maintaining population diversity to prevent a premature convergence to a local optimum. In this context, we developed five operators for mutating the offspring resulting from recombination: "J2P random resetting" (J2Prr), "J2P batch swap" (J2Pbsw), "J2P job swap" (J2Pjsw), "J2P batch insert" (J2Pbin), and "J2P job insert" (J2Pjin).

Random resetting is a standard mutation operator for integer chromosomes that mutates each gene independently (with a given probability P^{J2P}_{rr}) and draws a new allele value (batch-position) from $\{1, \ldots, m \cdot bpm\}$. The new batch positions are added by CLIP in non-decreasing order of batch-positions and job indices.

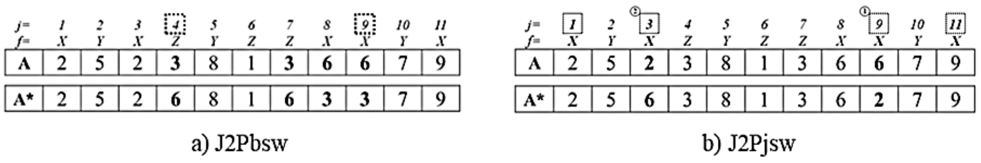
Fig. 4 shows an example where the jobs 2, 5, 6, 10, and 11 have been selected, new randomly chosen batch-positions have been drawn, and added to the new chromosome A* using CLIP. The number of mutated genes in a population can be approximated by $\mu \cdot n \cdot P^{J2P}_{rr}$ (where n represents the number of genes in the chromosome). Therefore, J2Prr combined with a high mutation probability

Figure 4: Example random resetting mutation



j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
A	2	5	2	3	8	1	3	6	6	7	9
R		4			3	6				2	10
		3			2	4				1	5
A*	2	4	2	3	4	5	3	6	6	1	10

Figure 5: Example batch swap mutation



j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
A	2	5	2	3	8	1	3	6	6	7	9
A*	2	5	2	6	8	1	6	3	3	7	9

a) J2Pbsw

j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
A	2	5	2	3	8	1	3	6	6	7	9
A*	2	5	6	3	8	1	3	6	2	7	9

b) J2Pjsw

P_{rr}^{J2P}can be used to maintain population diversity as many jobs to batch-positions assignments may be affected, and batches may be "opened" or "closed".

The mutation operator J2Pbsw swaps two complete batches with arbitrary families. This is done by randomly selecting two genes with probability P_{bsw}^{J2P} and swapping batch-positions for all jobs assigned to one of the batch-positions (see Fig. 5 a)). Since both batches (batch-positions 3 and 6, respectively) were feasible in terms of job family and batch capacity, the swapping must be feasible.

In contrast, infeasibilities may occur when J2Pjsw tries to swap two jobs between batches (by interchanging the batch-positions). Therefore, we first randomly select a single gene with probability P_{jsw}^{J2P} (indicated by 1 in Fig. 5 b)), then determine all genes with jobs of the same family (e.g., 1, 3, and 11 in Fig. 5 b)) and that are not in the selected batch-position (since otherwise the probability of swapping jobs from batches with many jobs is lower compared to other batches). Then, we randomly select one of these genes (indicated by 2 in Fig. 5 b)). If swapping is feasible in terms of batch capacity, the batch-positions are interchanged, otherwise, the chromosome remains unchanged. In general, J2Pbsw has a greater impact on a chromosome (because multiple jobs change their batch-position) than J2Pjsw, which only changes the batch-position of two jobs. In both cases, the number of batches remains the same.

The mutation operator J2Pbin inserts a complete batch at a different batch-position. First, a single gene j is randomly selected with probability P_{bin}^{J2P} , and the assigned batch-position bpA is determined (e.g., $j = 9$ with $bpA = 6$; cf., Fig. 6 a)). Then, the new batch-position bpB is drawn from $\{1, m \cdot bpm\}$. If bpB is empty (i.e., no job is assigned to it), all jobs at bpA are inserted into bpB

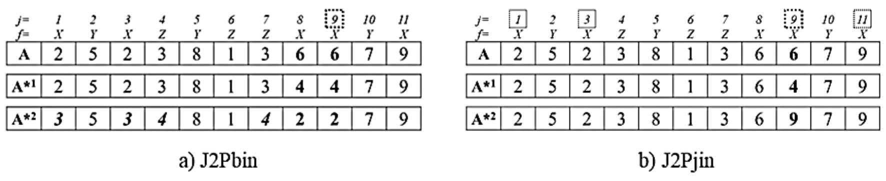


Figure 6: Example batch insert a) and job insert b) mutatios

j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
A	2	5	2	3	8	1	3	**6**	**6**	7	9
A*1	2	5	2	3	8	1	3	**4**	**4**	7	9
A*2	*3*	5	*3*	*4*	8	1	*4*	**2**	**2**	7	9

a) J2Pbin

j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
A	2	5	2	3	8	1	3	6	**6**	7	9
A*1	2	5	2	3	8	1	3	6	**4**	7	9
A*2	2	5	2	3	8	1	3	6	**9**	7	9

b) J2Pjin

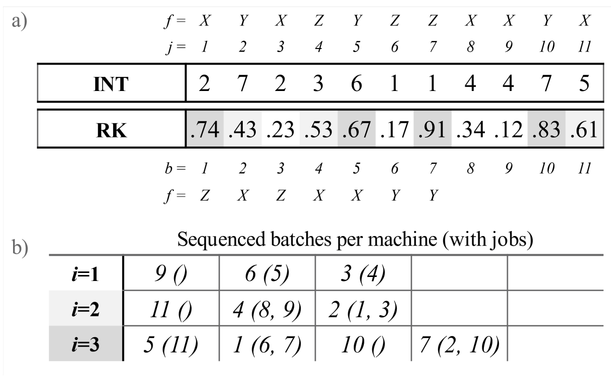
by updating the corresponding genes (see, for example, chromosome A^{*1} with $bpB = 4$). If bpB is not empty (e.g., $bpB = 2$), all jobs at bpA are also inserted into bpB, and the batch-positions of jobs formerly assigned to bpB are inserted into $bpB+1$. If $bpB+1$ is not empty, its jobs are moved to $bpB+2$ and so on (see chromosome A^{*2} in Fig. 6 a); additionally, updated batch-positions are marked in italics). If $bpB + x = m \cdot bpm$ holds, the "search" for empty batch-positions starts with batch-position 1 and ends at the latest when bpA is reached (which must now be empty). This operator can have a high impact on a chromosome if many batch-positions are occupied because many jobs may change their batch-position.

The fifth mutation operator J2Pjin extracts a single job from one batch and inserts it into another batch that can already contain jobs (of the same family) or is empty. Again, we first randomly select a single gene j with probability P_{jin}^{J2P}, and the associated job family jfA and the assigned batch-position bpA is derived from that gene (e.g., $j = 9$ with $jfA = X$ and $bpA = 6$; cf., Fig. 6 b)). Then, all batch-positions associated with jfA and different from bpA are determined and combined with all empty batch-positions (e.g., 2 and 9 combined with 4, 10, and 11). From this set of batch-positions, the new batch-position bpB is randomly selected. If bpB is empty (e.g., $bpB = 4$), the batch-position of j is updated (cf., chromosome A^{*1} in Fig. 6 b)). If bpB is not empty, a capacity feasibility check is required. If the check is positive, job j is inserted into bpB (see chromosome A^{*2} in Fig. 6 b)); otherwise, the chromosome remains unchanged. The impact of this operator on a chromosome is relatively low, as only a single job changes its batch-position. However, it can "open" a new batch or "close" a batch (if j was the only job in batch bpA and is inserted into a batch already containing jobs).

5.2. GA-J2BRK

The schedule representation of GA-J2BRK bases by two chromosomes was also used by Huynh and Chien (2018) for a problem very similar to the PSBIJF. However, we propose several adaptations and enhancements.

Figure 7: Example genotype a) and its phenotype b) in GA-J2BRK



a)

	f =	*X*	*Y*	*X*	*Z*	*Y*	*Z*	*Z*	*X*	*X*	*Y*	*X*
	j =	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
INT		2	7	2	3	6	1	1	4	4	7	5
RK		.74	.43	.23	.53	.67	.17	.91	.34	.12	.83	.61
	b =	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*
	f =	*Z*	*X*	*Z*	*X*	*X*	*Y*	*Y*				

b)

Sequenced batches per machine (with jobs)

***i*=1**	*9 ()*	*6 (5)*	*3 (4)*		
***i*=2**	*11 ()*	*4 (8, 9)*	*2 (1, 3)*		
***i*=3**	*5 (11)*	*1 (6, 7)*	*10 ()*	*7 (2, 10)*	

5.2.1. Genotype representation and decoding

The first chromosome (INT) of the J2BRK genotype consists of n integer genes with one gene for each of the n jobs (gene indices are therefore simultaneously job indices j), and the alleles define the job-to-batch assignment by batch indices. In contrast to Huynh and Chien (2018), we do not group jobs by job families to avoid positional biases. The second chromosome (RK) consists of n real-valued genes (one for each possible batch), and the allele represents random keys from $[0, 1]$ indicating the batch-to-machine assignment and the sequence of batches on a machine. Fig. 7 shows the two chromosomes in part a).

The batch-to-machine assignment is decoded by dividing $[0, 1]$ into m equal ranges (e.g., for m=3: $[0, {}^1\!/\!_m)$, $[{}^1\!/\!_m, {}^2\!/\!_m)$, and $[{}^2\!/\!_m, 1]$) and batches with random keys in the first range are assigned to machine one, batches with random keys in the second range are assigned to machine two, and so on. The sequencing of batches on the machines is done by sorting the batches by their random keys. To avoid that small random key changes leading to machine assignment changes also completely changes the position of the changed batch and all other batches on the "new" machine, we change the batch sorting according to the machine index: On machines with an odd index, batches are sequencing in non-decreasing order of their random keys, whereas on machines with an even index, batches are sequencing in non-increasing order of their random keys. The decoding of the example is shown in Fig. 7 part b).

5.2.2. Initial population composition

To generate the initial population for GA-J2BRK, we use the same three approaches as before: $ipc = ran, icr$, or iri.

To generate completely random initial solutions, we first randomly draw n

integers from $\{1, \ldots, n\}$ for chromosome INT and n random keys from $[0, 1]$ for chromosome RK. As this will likely result in infeasible solutions regarding job families and batch capacities, we need a repair mechanism. For repairing, we check if the insertion of a job into a batch is feasible, and if not, the corresponding allele in RK is incremented by one as long as insertion becomes feasible. Since this may lead to a "chain" of allele updates, the resulting genotype is very different from the initial one. However, as we just want to compute random initial solutions here, it does not matter.

The procedure for generating randomized initial solutions remains the same as before. We apply mutation and recombination operators iteratively, gradually increasing the mutation probabilities.

5.2.3. Elitism and survivor selection

For elitism and survivor selection, we use the same mechanisms as for GA-J2P with the control parameters esr, svr, and svs to be tuned for GA-J2BRK.

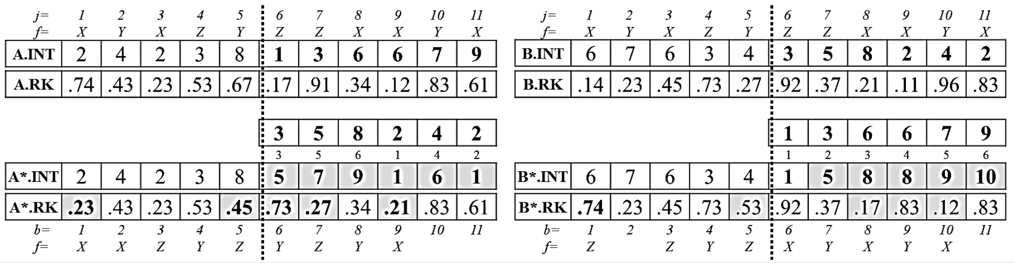
5.2.4. Parent selection and recombination

To select the parents that are used for offspring generation, we also study the two selection mechanisms TOS (with $s = 2$) and SUS, controlled by the parameter pas.

For recombing genotypes A and B, Huynh and Chien (2018) used a job family-related approach. They selected all genes of a randomly selected job family in genotype A (B), transferred these genes to offspring A* (B*), and transferred the genes of all other job families from B (A) to A* (B*). For the random key chromosome, they used a one-point crossover. In contrast to this job family-related approach, we propose using adapted standard recombination mechanisms: "J2BRK adapted one-point crossover" (J2BRKaOPX) and "J2BRK adapted two-point crossover" (J2BRKa2PX).

J2BRKaOPX starts with the random selection of the crossover point from $\{1, \ldots, n\}$ to partition both chromosomes of genotypes A and B into heads and tails (e.g., 6 in Section 5.2.4). First, the heads of the INT chromosomes and the complete RK chromosomes are transferred to the offsprings A* and B*. Next, the tail of both chromosomes of B (A) are transferred to A* (B*) in non-decreasing order of assigned batches (already existing random keys are overwritten). If a batch index to be transferred is already in use, the next unused batch index is used, and the random key is set accordingly. Note that this procedure does not affect job-to-batch and batch-to-machine assignments, but only the batch sequence on a machine might change. Thus, J2BRKaOPX is able to preserve promising parts of both parent genotypes. In Section 5.2.4, allele values resulting from tail transfers

Figure 8: Example recombination in GA-J2BRK



j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
A.INT	2	4	2	3	8	**1**	**3**	**6**	**6**	**7**	**9**
A.RK	.74	.43	.23	.53	.67	.17	.91	.34	.12	.83	.61
						3	**5**	**8**	**2**	**4**	**2**
						3	5	6	1	4	2
A*.INT	2	4	2	3	8	**5**	**7**	**9**	**1**	**6**	**1**
A*.RK	**.23**	.43	.23	.53	**.45**	**.73**	**.27**	.34	**.21**	.83	.61
b=	1	2	3	4	5	6	7	8	9	10	11
f=	X	X	Z	Y	Z	Y	Z	Y	X		

j=	1	2	3	4	5	6	7	8	9	10	11
f=	X	Y	X	Z	Y	Z	Z	X	X	Y	X
B.INT	6	7	6	3	4	**3**	**5**	**8**	**2**	**4**	**2**
B.RK	.14	.23	.45	.73	.27	.92	.37	.21	.11	.96	.83
						1	**3**	**6**	**6**	**7**	**9**
						1	2	3	4	5	6
B*.INT	6	7	6	3	4	**1**	**5**	**8**	**8**	**9**	**10**
B*.RK	**.74**	.23	.45	.73	.53	.92	.37	.17	.83	.12	.83
b=	1	2	3	4	5	6	7	8	9	10	11
f=	Z		Z	Y	Z	X	Y	X	Y	X	

are marked in bold, and gray shaded fields indicate values affected by the batch renumbering.

The J2BRKa2PX recombination operator follows the same procedure except using two randomly selected crossover points from $\{1, \ldots, n\}$, first transfers the middle parts, and then completes the offspring genotypes accordingly.

5.2.5. *Mutation*

For mutation, we propose using three standard operators, "J2BRK random resetting" (J2BRKrr), "J2BRK uniform" (J2BRKuni), and "J2BRK Gaussian" (J2BRKgau).

J2BRKrr is an adaptation of the standard random resetting mutation operator for integer chromosomes. It mutates each gene with a given probability P_{rr}^{J2BRK} by drawing new allele values for the INT chromosome from $\{1, \ldots, n\}$. If one of the drawn batch indices is already in use, renumbering takes place. The random keys of chromosome RK remain unchanged.

In contrast, J2BRKuni (also used by Huynh and Chien (2018)) and J2BRKgau keep job-to-batch assignments unchanged, but random keys change. They mutate each gene with probability P_{uni}^{J2BRK} and P_{gau}^{J2BRK}, respectively. As the names suggest, they differ in the distribution new allele values are drawn from: J2BRKuni draws from a uniform distribution restricted by $[0, 1]$, whereas J2BRKgau draws from a Gaussian (normal) distribution with the mean equal to the old random key and a standard deviation of $\frac{0.25}{m}$. The adjustment by m is made to make the probability that this mutation operator leads to a machine swap of a batch independent of the number of machines m. Obviously, the drawn values must be clamped to $[0, 1]$. The INT chromosome remains unchanged when J2BRKuni and J2BRKgau are applied.

In addition to these standard mutation operators, we developed four problem-specific mutation operators similar to those for GA-J2P: "J2BRK batch swap" (J2BRKbsw), "J2BRK job swap" (J2BRKjsw), "J2BRK batch insert" (J2BRKbin), and “J2BRK job insert” (J2BRKjin).

J2BRKbsw swaps two complete batches of arbitrary job families by randomly selecting two genes (with probability P_{bsw}^{J2BRK}) and swapping the batch indices for all jobs assigned to one of these indices. Since both batches were feasible in terms of job family and batch capacity, the swapping must be feasible. RK remains unchanged.

As for J2Pjsw (cf., Fig. 5 b)), J2BRKjsw first randomly selects a single gene from chromosome INT with probability P_{jsw}^{J2BRK}, then determines all genes from INT with jobs of the same family and that are not in the same batch, and finally randomly selects one of these genes. If swapping the two jobs between the assigned batches is feasible in terms of batch capacity, the batch indices are swapped, otherwise, the chromosome remains unchanged. Random keys are always left unchanged. A similar mutation operator is also used by Huynh and Chien (2018).

The mutation operator J2BRKbin inserts a complete batch at a new position. To this, a gene from INT is selected with probability P_{bin}^{J2BRK}, and the assigned batch bA is determined. Then, the new batch bB is drawn from $[1, n]$. If bB equals bA, no insertion is done. If batch bB is empty, all jobs are inserted into bB by updating the corresponding genes of chromosome INT. If bB is not empty, batch bA is inserted before bB by setting the random key of bA to the random key of bB decremented by a very small number. This operator can be very influential as a new batch may be created, and batch positions on a machine may change.

J2BRKjin extracts a single job from one batch and inserts it into another batch that can already contain jobs (of the same family) or is empty. We first randomly select a gene (job) j from INT (with probability P_{jin}^{J2BRK}) and determine its job family jfA. Then, all batches also containing jobs of family jfA are combined with all empty batches. From this set of batches, the new batch bB is randomly selected. If bB is empty, the batch index of j is updated to bB. If bB is not empty, a capacity feasibility check is required. If it is positive, job j is inserted into bB. Otherwise, the chromosome INT remains unchanged (RK remains unchanged in any case). The effect of this operator on INT is relatively small because only a single job-to-batch assignment is changed. However, it can ”open” a new batch or ”close” a batch if j was the only job in batch bA and is inserted into a batch that already contains jobs.

5.3. *Task parallel implementation*

As the task parallel implementation of the learning augmented heuristics proposed in Uzunoglu et al. (2023b) resulted in a remarkable speedup, we follow this approach not only in the implementation of the heuristics computing initial solutions but also for our GAs. Similar to the multi-start of the heuristics, we start several evolution engines in parallel threads. This means that several identically parametrized GA are executed, and the best result of all of them is returned. As the results of Uzunoglu et al. (2023b) showed that the greatest efficiency gains are achieved with four parallel threads, we also use four threads in our experiments.

6. Experimental setup

To perform training, validation (hyper-parameter tuning), GA parameter tuning, and testing, we need to prepare the required data and ML models. Note that all used data sets are either already available for download or are added to a new data set on Mendeley Data (*reference to be added*).

6.1. Problem instances and data sets

It is vital to have a sufficiently large set of problem instances to train well-performing ML models with good generalization and prediction accuracy. Furthermore, to validate models' applicability to real-world problems, the instances should also be realistic, diverse (i.e., representing different scenarios), and present a challenging learning task. The data set provided by Gahm fulfills these requirements. It contains three sets of instances that differ in the number of jobs n and machines m. As we are interested realistic-large problem instances, we use the sets L (containing 57,600 large instances with $n \in \{100, 200, 400\}$ and $m \in \{1, 3, 4, 5, 10\}$) and XL (containing 13,440 large instances with $n \in \{800,$ $1{,}600,$ $3{,}200\}$ and $m \in \{5, 10, 20\}$) to define the problem space P for this paper ($|P| = 71,040$). All generated instances base on nine attributes (like n or m), and for each attribute combination, five instances are randomly generated (thus, each instance has a marker $s \in \{1, 2, \ldots, 5\}$). Based on the marker, we can define different data sets for training, validation (hyper-parameter tuning), GA parameter tuning, and testing. To represent these subsets, we use the notation D_s (e.g., $D_{1,2}$ consists of all instances with the markers 1 and 2).

6.2. Model training for the learning augmented heuristics

Since all 17 algorithms from algorithm space A use ML models to predict parameters in one way or another, we must train these models. Because hyper-parameter tuning and validation were already done in the referenced publications, we can use $D_{1,2,3,4}$ for training the models. For future use, we provide all 17 models (and scaling data) for download (*reference to be added*).

6.3. Performance metric

Comparing the performance of algorithms relative to the best solution could be misleading if objective values tend to get close (or equal to) zero (like for the weighted tardiness criterion of the PSBIJF). In these cases, small absolute deviations to zero result in huge relative deviations, which may distort the analysis even with a large number of experiments. To get a robust metric for our analysis, we use the key figure Mean Relative Improvement to the Worst (MRIW; cf., Valente and Schaller (2012) and Gahm et al. (2022)). The relative improvement to the worst objective value for a problem instance $RIW_{a,i}$ is based on the following definitions: let $A' \subseteq A$ be the (sub)set of all algorithms to be analyzed, let $ov_{a,i}$ be the objective value computed by algorithm $a \in A'$ for problem instance $i \in P'$ (with $P' \subseteq P$), and let $ov_i^{worst} = \max_{a \in A'} ov_{a,i}$ be the worst achieved objective value by one of the algorithms (note that ov_i^{worst} is not an intermediate solution or the worst candidate objective value observed during the execution of an algorithm but the worst final solution of one of the considered algorithms). With these definitions, we define the relative improvement to the worst for algorithm a and instance i as:

$$RIW_{a,i} = \frac{(ov_i^{worst} - ov_{a,i})}{ov_i^{worst}}$$

if $ov_i^{worst} > 0$ and otherwise, $RIW_{a,i} = 0$. Aggregating these values using the mean over all instances of interest (P') for an algorithm a gives us the $MRIW_a$. Here, it is fundamentally important that $MRIW_a$-values can only be compared to each other if the same (sub)set of algorithms A' (and instances) is used for their computation.

6.4. Implementation

The training, hyper-parameter tuning, validation, and testing of all ML models is implemented in Python and uses “Keras” (keras.io), “TensorFlow” and “TensorFlow Ranking (tensorflow.org), “scikit-learn” (Pedregosa et al., 2011), and XGBoost (xgboost.ai). All heuristics, the GAs, and a tool for the management of the experiments are implemented in Java 10. For the GA implementations, we use the Jenetics framework (jenetics.io), and for task-parallel execution of the learning-augmented heuristics and the GAs, we use the “Parallel Java 2 PJ2” API (available at `http://jimihford.github.io/pj2/`).

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.

7. Parameter tuning

7.1. ASP model hyper-parameter tuning

The hyper-parameter tuning of the ASP models uses the dataset $D_{1,2}$ for training and D_3 for validation (i.e., to determine the most suitable one). For the 15 NN-based ASP models, we use NNs with three hidden layers and evaluated the following hyper-parameters: 256, 512, and 1024 neurons for all three layers independently, the dropout rates .1, .2, and .3, and the L2 weight regularization rates .001, .002, and .003. In total, 243 ($= 3^3 \cdot 3{\cdot}3$) hyper-parameter combinations have to be evaluated to determine the best configuration for each NN-based ASP model. For the 14 GBDT-based ASP models, we evaluated the following hyper-parameters: number of estimators 250, 350, and 450, learning rates .01 and .001, maximum depth 8, 9, and 10, minimal child weights 1 and 2, subsampling rates .5 and .7, column sampling rates .5 and .7. In total, 144 ($= 3 \cdot 2 \cdot 3 \cdot 2 \cdot 2 \cdot 2$) hyper-parameter combinations have to be evaluated.

Because the ASP models use different loss functions, we cannot use them for the validation of the most suitable ASP model. Therefore, we introduce the metric "hit rate", quantifying the ratio of how often the model's proposed (best) algorithm was in fact one of the best algorithms for a problem instance. Table 4 depicts the best hit scores for every model, labeling strategy, and loss function after tuning the hyper-parameters (see Appendix B Table B.13 and Table B.14 for best hyper-parameters for each ML model, labeling strategy, and loss function combination).

Table 4: Hit rates by ML model, labeling strategies, and loss functions

ML model type	**Labeling strategy**	**Loss function**	**Hit rate[%]**	**MRIW[%]**
	BIN	BCE	70.16	40.33
	OV	MSE	57.09	36.38
	N-OV	MSE	64.92	39.61
		MSE	66.31	39.89
		iwMSE(1)	68.94	40.31
		iwMSE(2)	68.98	40.39
	RANK	iwMSE(3)	61.77	38.98
NN		ApproxNDCG	68.28	39.45

		LambdaRankNDCG	67.46	39.27
		LambdaRankNDCG(1)	67.91	39.34
	RANK-SC	MSE	65.13	39.71
		iwMSE(1)	65.70	39.72
		iwMSE(2)	68.00	40.04
		iwMSE(3)	64.23	38.46
		ApproxNDCG	64.75	39.34
GBDT	BIN	BCE	73.44	41.14
	BIN	LambdaBinary	69.93	40.59
	OV	MSE	67.37	39.84
	N-OV	MSE	71.90	41.22
	RANK	MSE	72.99	41.12
		iwMSE(1)	73.11	41.10
		iwMSE(2)	73.07	41.07
		iwMSE(3)	72.99	41.08
		RankPairwise	71.13	40.77
		LambdaMARTNDCG	70.26	40.57
	RANK-SC	MSE	72.99	41.12
		iwMSE(1)	73.08	41.13
		iwMSE(2)	73.18	41.15
		iwMSE(3)	73.24	41.14

The results in Table 4 show that GBDT as ML model type generally outperforms NN, which is not surprising since GBDT is known to outperform other models on tabular data (in contrast to image data or text documents; Shwartz-Ziv and Armon (2022)). For both model types, representing the ranking problem as a binary classification achieves the highest hit scores. It is also evident that using (pure) objective value labeling strategies (see OV rows in Table 4) perform worse than all other labeling strategies for NN and GBDT, respectively. Consequently, we conclude that asking ML models for a ranking of algorithms is better than asking for objective values for each of the algorithms in A. For the NN, the two best-performing settings use the labeling strategies BIN with training strategy BCE and RANK combined with the custom loss function iwMSE(α). Interestingly, the custom loss function outperforms MSE, and the scaling factor α seems to play an important role as the hit rate decreases from 68.98% to 61.77% when α changes from 2 to 3. This indicates that tuning the α-value is important. For GBDT, the two best-performing settings are BINARY labels with BCE and

RANK-SC labels with the custom loss function iwMSE(α). In contrast to the NN, the GBDT seems to be less sensitive to changes in the α-value. However, tuning the α-value is still important. Overall, the GBDT model type achieves more consistent hit rates throughout the different methods. The results also show us that the scaled RANK version RANK-SC performs better for GBDT, while this is not the case for NN. Somewhat surprisingly, the "simple" binary encoding BIN combined with BCE for both GBDT and NN outperforms more sophisticated alternatives (such as LambdaRankNDCG or ApproxNDCG) with respect to the ASP at hand. The only loss function that achieves a similar performance is the newly introduced iwMSE(α) function. An advantage of iwMSE(α) is its less complex implementation compared to the more sophisticated ones. However, further studies have to examine if iwMSE(α) is generally suitable for ranking predictions.

The best determined hyper-parameters for the four best ASP models show that both model types prefer higher capacity models in almost every experiment. To achieve a maximum prediction performance, we performed a second hyper-parameter tuning on the best four ASP models with extended capacities but omitted certain hyper-parameters for the GBDT since they did not lead to accurate models (e.g., 250 and 350 as number of estimators or 8 as maximum depth). For GBDT models we define the hyper-parameter space as: 450 or 550 for number of estimators, 9 or 10 for maximal depth, .001 or .01 for the learning rate, 1 or 2 for minimum child weight, and .5 or .7 for subsampling rates for data points and columns. For the NN we increase the number of neurons in the hidden layers (all three hidden layers with 256, 512, 1024, or 2048 neurons, dropout rate of .1 or .2, and regularization rates of .001, .002, or .003). To further improve prediction accuracy, in the second hyper-parameter tuning, training is performed the data set $D_{1,2,3}$ and validation on D_4.

Table 5: Results of second hyper-parameter tuning

ML model type	**Labeling strategy**	**Loss function**	**Hit rate[%]**	**MRIW[%]**
NN	BIN	BCE	64.75	40.95
	RANK	iwMSE(2)	61.87	40.54
GBDT	BIN	BCE	67.67	41.56
	RANK-SC	iwMSE(3)	66.57	41.43

Table 5 shows that overall the hit rate in the second hyper-parameter tuning slightly decreases due to the new validation data set D_4. Nevertheless, the results confirm that GBDT-RANK-SC-iwMSE(3) is competitive and that GBDT-BIN-BCE outperforms the other ASP models (see Appendix C Table C.15 and Table C.16 for best-performing hyper-parameters). In consequence, *GBDT-BIN-BCE* is used to select the learning-augmented heuristics for initial solution computation in all following experiments.

7.2. GA-J2P parameter tuning

Because of the large number of parameters and values for GA-2JP, we perform a two-step tuning to keep experiments at a manageable level. For the parameter tuning of GA-J2P (and also GA-J2BRK), we use a reduced set D_{PT} of 30 instances, i.e., five randomly selected instances for each $n \in \{100, 200, 400, 800, 1600, 3200\}$ from D_4.

In parameter tuning step one, we are interested in the general behavior of GA-J2P, for example, regarding the initial population composition or the effect of the elitism mechanism. The parameters and their investigated values, as listed in Table 6, are used to set up 51,840 experiments in the study (1,728 parameter configurations · 30 problem instances from D_{PT}). For termination, we use setting TS1 (see Table 3). Because the MRIW of GA-J2P without initial ($ipc = ran$) solutions is so much worse compared to the other ones (33.8% vs. 93.7% and 93.7%), we remove these results from further considerations (and MRIW computations) to make the differences between parameter values more traceable. To simplify readings, we introduce $P^{J2P}_{cust.}$ for representing identical probabilities for P^{J2P}_{bsw}, P^{J2P}_{jsw}, P^{J2P}_{bin}, and P^{J2P}_{jin}. In Table 6, bold values indicate values to be further investigated in step two, and bold and underlined values are fixed based on the results.

Table 6: Parameters and results of study GA-J2P-PT1

Parameters and values		# values	MRIW [%]		
bpm	n , **1.2(n/m)**	2	19.95	20.88	
ipc	(ran ,) icr , **<u>iri</u>**	3		19.98	20.85
μ	**200** , 400	2	20.60	20.23	
esr	0 , **<u>.05</u>**	2	19.26	21.57	
svr	**0.2** , **0.4**	2	19.88	20.94	
svs	TOS(2) , **<u>SUS</u>**	2	20.40	20.43	
pas	TOS(2) , **<u>SUS</u>**	2	20.24	20.59	

$rec. \begin{pmatrix} P^{J2P}_{aOPX} \\ P^{J2P}_{a2PX} \end{pmatrix}$ $\begin{pmatrix} .2 \\ .0 \end{pmatrix}, \begin{pmatrix} \underline{.0} \\ \mathbf{.2} \end{pmatrix}, \begin{pmatrix} .2 \\ .2 \end{pmatrix}$	3	20.64	21.08	19.53
$mut. \begin{pmatrix} P^{J2P}_{rr} \\ P^{J2P}_{cust} \end{pmatrix}$ $\begin{pmatrix} .02 \\ .00 \end{pmatrix}, \begin{pmatrix} \underline{0} \\ \mathbf{.02} \end{pmatrix}, \begin{pmatrix} .02 \\ .02 \end{pmatrix}$	3	17.15	22.25	21.85
Tot. # of parameter configurations	1,728			

The results in Table 6 show that reducing the number of batch positions has a remarkable effect on the solution quality, and thus, we conclude that different parameter values are worth exploring in the second parameter tuning step. As already mentioned, the standalone execution of GA-J2P without initial solutions is so much worse that we do not investigate this approach further. Furthermore, since the randomized initial solutions outperform initial populations with fully randomized solutions, we fix the parameter ipc to iri. We also fix the parameters esr to .05 as larger values are not used in literature and therefore perceived as unfavorable, and svs to SUS and pas to SUS as results are almost similar. Because smaller population sizes (leading to more generations) seem to be favorable, we will explore different settings for μ in the next tuning step. Regarding recombination, we see that J2Pa2PX outperforms J2PaOPX, and thus, we only tune P^{J2P}_{a2PX} in the second step. For mutation, we can see that the problem-specific mutation operators J2Pbsw, J2Pjsw, J2Pbin, and J2Pjin perform best when solely applied. As the survival rate (svr) is directly related to recombination and mutation probabilities, we investigate both esr values in step two.

In the second parameter tuning study (GA-J2P-PT2), we use 540 parameter configurations to define 16,200 experiments (see Table 7). To ensure a feasible transfer of initial solutions to the genotype representation, we must guarantee that the number of batches on each machine is not smaller than in any initial solution. Therefore, we calculate the maximum value of used batches per machine of all solutions bpm^{MAX}_{m} and adjust the finally available batches-positions per machine accordingly. In addition, we use this value to define two additional settings.

Table 7: Parameters and results of study GA-J2P-PT2

Parameters and values	# values	MRIW [%]

bpm	$\underline{\mathbf{max\{n, bpm_m^{MAX}\}}}$, $\max\{1.1n/m, bpm_m^{MAX}\}$, $\max\{1.2n/m, bpm_m^{MAX}\}$, $1.1bpm_m^{MAX}$, $1.2bpm_m^{MAX}$	5	8.07	7.62	7.91	7.88	7.75	
μ	$\underline{\mathbf{150}}$, 200 , 250	3	7.92	7.89	7.72			
svr	.2 , $\underline{\mathbf{.4}}$	2	7.43	8.27				
P_{a2PX}^{J2P}	$\underline{\mathbf{.2}}$, .3 , .4	3	9.44	7.76	6.33			
P_{cust}^{J2P}	.02 , .04 , $\underline{\mathbf{.08}}$, .12 , .16 , .20	6	6.91	7.68	8.42	8.30	8.08	7.69
Tot. # of parameter configurations:		540						

Based on the results of the parameter tuning studies GA-J2P-PT1 and GA-J2P-PT2, we fix the parameters of GA-J2P as follows: $bpm = max\{n/m, bpm_m^{MAX}\}$, $ipc = iri$, esr=.05, svr=.4, svs=SUS, pas=SUS, P_{a2PX}^{J2P} = .2, and $P_{bsw}^{J2P} = P_{jsw}^{J2P} = P_{bin}^{J2P} = P_{jin}^{J2P}$ = .08 (all other operator probabilities are set to 0).

7.3. *GA-J2BRK parameter tuning*

We also perform a two-step parameter tuning for GA-J2BRK on instance set D_{PT}. The parameters and their investigated values, as listed in Table 8, are used to set up 60,480 experiments in study GA-J2BRK-PT1. For termination, we use setting TS1 (see Table 3). The version of GA-J2BRK without initial solutions ($ipc = ran$) is much worse than the others (22.7% vs. 89.1% and 85.1%), and therefore, we remove the corresponding results from further considerations and MRIW computations. In Table 8, bold values indicate values to be further investigated in step two, and bold and underlined values are fixed based on the results. To simplify readings, we introduce P_{cust}^{J2BRK} for representing identical probabilities for P_{bsw}^{J2BRK}, P_{jsw}^{J2BRK}, P_{bin}^{J2BRK}, and P_{jin}^{J2BRK}. Similarly, $P_{uni,gau}^{J2BRK}$ represents identical probabilities for P_{uni}^{J2BRK} and P_{gau}^{J2BRK}.

Table 8: Parameters and results of study GA-J2BRK-PT1

Parameters and values		# values	MRIW [%]		
ipc	(cr ,) icr , $\underline{\mathbf{iri}}$	3		23.59	24.15
μ	200 , **400**	2	23.82	23.92	
esr	0 , $\underline{\mathbf{.05}}$	2	23.00	24.74	

svr	.2 , <u>**0.4**</u>	2	23.48	24.26	
svs	TOS(2) , <u>**SUS**</u>	2	23.80	23.94	
pas	<u>**TOS(2)**</u> , SUS	2	23.91	23.83	
$\begin{pmatrix} P^{J2BRK}_{aOPX} \\ P^{J2BRK}_{a2PX} \end{pmatrix}$	$\begin{pmatrix} .2 \\ 0 \end{pmatrix}, \begin{pmatrix} \underline{\mathbf{0}} \\ \underline{\mathbf{.2}} \end{pmatrix}, \begin{pmatrix} .2 \\ .2 \end{pmatrix}$	3	23.61	24.33	23.67
$\begin{pmatrix} P^{J2BRK}_{rr} \\ P^{J2BRK}_{cust} \\ P^{J2BRK}_{uni,gau} \end{pmatrix}$	$\begin{pmatrix} .02 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \underline{.02} \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ \underline{\mathbf{.02}} \end{pmatrix}$	3	23.89	23.94	24.03
	$\begin{pmatrix} .02 \\ .02 \\ 0 \end{pmatrix}, \begin{pmatrix} .02 \\ .02 \\ 0 \end{pmatrix}$	2	23.79	23.88	
	$\begin{pmatrix} 0 \\ .02 \\ .02 \end{pmatrix}, \begin{pmatrix} .02 \\ .02 \\ .02 \end{pmatrix}$	2	23.92	23.64	
Tot. # of parameter configurations:		2,016			

The results in Table 8 support the previous observation that randomized initial solutions are preferable compared to completely randomly generated solutions for initial population composition. In contrast to GA-J2P, GA-J2BRK performs better with larger populations and μ will be tuned in the second step. For the parameters *esr*, *svr*, *svs*, and *pas*, the results are very similar to those from GA-J2P, and thus, we proceed as before. The recombination by J2BRKa2PX leads to better results compared to J2BRKaOPX, accordingly, we tune the corresponding probability P^{J2BRK}_{a2PX} in the second step. The results for the mutation operator are not as clear as for GA-J2P because the MRIWs are close together. However, as the setting with P^{J2BRK}_{rr}=.00, $P^{J2BRK}_{cust.}$=.02, and $P^{J2BRK}_{uni,gau}$=0 (23.94%) and P^{J2BRK}_{rr}=0, $P^{J2BRK}_{cust.}$=0, and $P^{J2BRK}_{uni,gau}$=.02 (24.03%) performs best, we are going to optimize $P^{J2BRK}_{cust.}$ and $P^{J2BRK}_{uni,gau}$ independently of each other in tuning step two. The investigated parameter values are summarized in Table 9.

Table 9: Parameters and results of study GA-J2BRK-PT2

Parameters and values		# values	MRIW [%]				
μ	300 , <u>**350**</u> , 400 , 450	4	1.29	1.31	1.28	1.26	1.29
svr	.2 , <u>**.4**</u>	2	1.26	1.31			
P^{J2BRK}_{aOPX}	<u>**.2**</u> , .3 , .4	3	1.36	1.3	1.2		

mut.	P^{J2BRK}_{cust} = .02, .04 , .08 , .12 , .16 (with P^{J2BRK}_{rr} =0, $P^{J2BRK}_{uni,gau}$=0)	10	1.27	1.3	1.23	1.27	1.24
	$P^{J2BRK}_{uni,gau}$ =$\underline{\mathbf{.02}}$, .04 , .08 , .12 , .16 (with P^{J2BRK}_{rr} =0, P^{J2BRK}_{cust}=0)		1.37	1.31	1.29	1.31	1.26
Tot. # of parameter configurations:		240					

The results in Table 9 clearly indicate most suitable parameters. The better performance of $P^{J2BRK}_{uni,gau} = .02$ compared to all other mutation settings can be traced back to be the most efficient one and leads to the highest number of generations in the given time limit. Based on the overall results of the parameter tuning studies GA-J2BRK-PT1 and GA-J2BRK-PT2, we fix the parameters of GA-J2BRK as follows: $ipc = iri$, $esr = .05$, $svr = .4$, $svs = SUS$, $pas = TOS$, P^{J2BRK}_{a2PX}=.2, and $P^{J2BRK}_{uni,gau}$=.02 (all other operator probabilities are set to zero).

8. Experimental results

For the final testing phase of our developed algorithms, we reserved D_5 (20% of the data), which was not used in either of the previous training or tuning phases, for assessing the solution quality on unseen problem instances. We use the MRIW metric to compare four solution methods:

- **BATCS-d-MLPP(PC$_6$, GS3)** has the highest MRIW score among the 17 solution methods (see Section 2.2) on $D_{1,2,3,4}$. We use this solution method as a baseline to compare the effectiveness of our algorithm selection method.

- **AlgSel(GBDT-BIN-BCE)** has the highest hit rate according to our analysis in Section 7.1. We trained the model on $D_{1,2,3,4}$ to predict the best-performing solution method for each problem instance from Section 2.2 and used that algorithm to solve it.

- **GA-J2BRK** is the GA with the parameters from Section 7.3.

- **GA-J2P** is the GA with the parameters from Section 7.2.

Both GAs used the AlgSel(GBDT-BIN-BCE) to create their initial population and terminated their computations after reaching the time limits as defined in TS2 (see Table 3). So, our algorithm set A' consists of these four solution methods for the computation of the MRIW. Table 10 presents the MRIWs of all four methods grouped according to the number of jobs (n) and number of machines (m). Certain (n, m) groups do not exist in the used data set and hence are left blank in the table.

Table 10: MRIWs [%] per solution method, number of jobs, and machines.

n=	100	200	400	800	1,600	3,200	MEAN
BATCS-d-MLPP (PC$_6$, GS3)							
m=1	2.02	0.85	1.76				1.54
m=3	2.55	2.29	1.78				2.21
m=4	2.60	2.47	1.80				2.29
m=5	2.19	2.24	2.05	1.63			2.05
m=10	2.33	2.32	2.61	2.45	1.95		2.37
m=20				3.13	4.25	3.48	3.56
MEAN	2.34	2.03	2.00	2.40	3.10	3.48	2.23
AlgSel(GBDT-BIN-BCE)							
m=1	6.71	6.60	6.92				6.74
m=3	4.62	5.55	4.93				5.03
m=4	5.99	6.53	5.72				6.08
m=5	5.92	5.96	5.25	4.51			5.47
m=10	9.48	9.22	8.01	5.68	5.63		7.95
m=20				9.79	5.25	3.65	7.25
MEAN	6.54	6.77	6.17	6.66	5.44	3.65	6.42
GA-J2BRK							
m=1	11.52	9.94	10.08				10.51
m=3	13.68	11.94	9.61				11.74
m=4	15.87	14.40	10.79				13.69
m=5	17.19	14.46	12.15	8.45			13.37
m=10	22.22	20.38	16.18	11.87	10.94		17.21
m=20				18.50	12.57	4.46	14.18
MEAN	16.09	14.23	11.76	12.94	11.75	4.46	13.64
GA-J2P							
m=1	47.40	50.33	42.21				46.65
m=3	41.85	40.41	30.46				37.57
m=4	43.74	40.54	28.84				37.71

m=5	45.44	39.13	28.08	22.02			34.44
m=10	52.19	45.47	32.23	21.25	18.48		36.48
m=20				25.20	16.38	9.90	19.71
MEAN	46.12	43.18	32.36	22.82	17.43	9.90	36.73

Our ML-based algorithm selection method outperforms the "statically" chosen best solution method in every group (n, m). Averaged over all instances, it achieves an MRIW of 6.42 % compared to the static best solution method's MRIW of 2.23%. The performance improvement is higher for instances with fewer jobs and decreases for very large instances, but nonetheless exists. This justifies the efforts of training a model to rank solution methods if several are available for the problem at hand.

Both GAs create their initial population using the AlgSel(GBDT-BIN-BCE) and hence achieve, as expected, substantially higher performances in each group (n, m) than the algorithm selection method without post-optimization. Interestingly, a clear winner exists between both GAs. GA-J2P immensely outperforms GA-J2BRK in each group. The greatest difference in performance improvement between both GAs can be observed for small instances (e.g., $n = 100$ or $n = 200$). Across the different machine settings (m), both GAs performed (rather) similarly. One exception to that is the performance of GA-J2P for $m = 20$. From $m = 10$ to $m = 20$, the MRIW drops from 36.48% to 19.71%. However, this effect is due to the fact that large instances with $n = 3200$ only exist for $m = 20$.

Table 11: Results of GA-J2P by instance characterisics

Instance characteristic		MRIWs [%]					
n	100, 200, 400, 800, 1600, 3200	46.12	43.18	32.36	22.82	17.43	9.90
m	1, 3, 4, 5, 10, 20	46.65	37.57	37.71	34.44	36.48	19.71
q	3, 5, 10, 20, 40	35.98	38.53	39.49	39.62	16.23	
jtfam	ND, UD	36.65	36.81				
crs	1, 2, 3, 4	46.74	37.91	31.74	30.53		
st	AR, AE, SE	36.38	37.63	36.19			
eta	0.25, 0.75	37.51	35.96				
tf	0.3, 0.6	38.67	34.79				
rdd	0.5, 2.5	39.17	34.30				

Table 11 shows the MRIW grouped for each instance characteristic (see Gahm et al. (2022) or Uzunoglu et al. (2023b) for detailed description of instance characteristics). The job to family assignment mode ($jtfam$), setup time allocation (st), setup time severity(eta), tightness factor (tf), and due date range factor (rdd) seem to have minor effects on the performance of the GA. The capacity requirement scenarios (crs) define the ranges of a job's capacity demand and have a noticeable effect on the MRIW. Also, the MRIW for the number of job families (q) drops immensely for $q = 40$, but, again, this number of job families only exists for $n = 3200$. Due to its coupling to the job size, it is unclear how much of this effect should be attributed to the characteristic itself. An explanation for the trend of decreasing improvement with increasing job size might be the coupling between job size and the "complexity" of the problem: large instances need more time for their mutation and repair mechanisms, and therefore, GAs can search for solution improvements for these instances less intensely. However, this situation needs a deeper understanding of its causes and has to be analyzed more nuancedly in future research.

9. Conclusions and further research directions

In this paper, we presented two learning-based contributions to both ends of using a heuristic to solve a complex scheduling problem. First, we addressed an issue appearing before using a heuristic – selecting the most suitable method. The Algorithm Selection Problem is an active topic in research and has come a long way since its initial formulation in 1976 by Rice. Recent contributions incorporate ML models, for example, by predicting the performance of an algorithm on a given instance. In our approach, we used learning-to-rank methods from different ML research fields like information retrieval and recommender systems that best suit the needs of the algorithm selection task. Our computational results show that asking "the right questions" does matter: Asking for the (probably) best algorithm (or a ranking of algorithms) instead of asking for a performance prediction results in a hit rate. Besides this finding, we can confirm that GBDTs perform better on the tabular data of our ASP than NNs, and we can report that the importance-weighted MSE with weighting parameter α (iwMSE(α)) is competitive and worth to be studied in more detail in the future. Overall, the application of the ASP model GBDT-BIN-BCE to dynamically select an algorithm for solving an instance has outperformed the static selection of the best algorithm (BATCS-d-MLPP(PC_6, *GS3*)) (MRIW of 6.42% vs. 2.23%).

The second major contribution of this paper addressed what happens after solving the problem instance with a selected construction heuristic – improving the initial solution(s). To that, we presented the two GAs GA-J2P and GAJ2BRK with different representations (encodings). The random keys approach (GAJ2BRK) represents feasible solutions with every representation and, therefore, does not need any costly repair mechanism during its computation like the GA-J2P approach. However, a great advantage of GA-J2P is that we can easily include knowledge about reasonable solutions. For example, we know the number of used batches (i.e., batch positions per machine) from the initial solutions and use it to limit the number of available batches. As the experimental results have shown, the GA-J2P approach achieves significantly higher improvements when compared to GA-J2BRK. Nonetheless, the improvements decrease with increasing instance size and lead to the question of how to utilize the potential of our solution method for such problem instances.

In conclusion, we demonstrated in this paper how to integrate several parts in the decision-making to get better solutions quicker. We strongly believe that having a more holistic view on the solution procedure(s) will lead to very fruitful results for practical application.

Furthermore, we advocate shifting the goal of ASP from a mere, isolated selection of the best algorithm to decisions involving the allocation of computational resources, interleaving of algorithms, or use of information gathered online during the solving. Kotthoff (2016) presented an interesting survey on algorithm selection and referred to the topic in a broader sense with the term algorithm portfolio. The algorithm portfolio technique outputs several (probably) good algorithms for a problem instance to achieve more robust results. Their online variants may even change the currently used algorithm if the solution quality misses the expectation to mitigate wrong decisions made at the beginning. One next challenge for our solution approach(es) would be to include every part of our solution concept into the ASP. This could help us to use the potentials of the GA(s) even for the largest instances. Fundamental questions implied by that would be: how much time should we invest in finding the initial population compared to the GA? Or, which GA-variant to choose, and which parameter configuration to select, could be included in the ASP. Taking that idea even further, information from a pre-solving phase of several GA variants could be used to further improve the solution quality. These ideas and questions will definitely need sophisticated ML models tailored to give the right answers. Researchers must, however, carefully consider how to incorporate the feedback from these ML models into their decision-making.

Appendix A. Preliminary study of “d-MLRP” variants

Table A.12: Performance of “d-MLRP” variants by pipeline configuration and ranking application strategy

Pipeline configuration	**Ranking application strategy**						
	B1	*B1-G*	*Bx*	*B(3)-G*	*B(5)-G*	*B(9)-G*	Mean
[1,9,0]-CF	34,42	76,41	76,75	76,86	76,86	76,90	69,70
[1,9,0]-AF	34,68	76,24	76,44	76,42	76,44	76,54	69,46
[0,5,5]-CF	51,82	77,38	77,55	77,59	77,64	77,46	73,24
[0,5,5]-AF	54,80	77,43	77,55	77,53	77,55	77,56	73,74
[1,4,5]-CF	56,80	77,55	77,62	77,53	**77,63**	**77,55**	**74,12**
[1,4,5]-AF	53,23	77,39	77,73	77,56	77,58	77,66	73,52
[0,2,8]-CF	52,56	77,02	77,46	77,38	77,44	77,53	73,23
[0,2,8]-AF	55,07	77,43	77,52	77,49	77,47	77,51	73,75
[1,2,7]-CF	56,05	77,15	77,33	77,30	77,38	77,34	73,76
[1,2,7]-AF	57,99	77,40	77,74	77,72	**77,69**	**77,73**	**74,38**
[0,1,9]-CF	46,56	76,95	76,90	77,14	77,11	76,99	71,94
[0,1 9]-AF	51,68	77,07	77,12	77,08	77,03	77,13	72,85
[1,0,9]-CF	49,32	77,06	77,00	77,06	77,02	76,99	72,41
[1,0,9]-AF	52,50	77,24	77,43	77,34	77,52	77,42	73,24
[0,0,10]-CF	47,44	76,74	75,97	76,74	76,72	76,52	71,69
[0,0,10]-AF	47,81	77,07	76,84	77,26	77,29	77,32	72,26
	52.87	67.94	68.75	68.55	**68.71**	**68.76**	

Appendix B. Best-performing hyper-paramters for tuning phase 1

Table B.13: Best-performing hyper-parameter for NN tuning phase 1

Labeling strategy	**Loss function**	**layer 1**	**layer 2**	**layer 3**	**dropout rate**	**reg rate**
BIN	BCE	1024	1024	1024	.1	.01
OV	MSE	256	1024	256	.3	.001
N-OV	MSE	1024	1024	512	.1	.01
RANK	MSE	1024	1024	512	.1	.001
	iwMSE(1)	1024	1024	512	.1	.001
	iwMSE(3)	1024	512	512	.1	.002
	iwMSE(3)	1024	1024	256	.1	.002
	ApproxNDCG	1024	1024	256	.1	.002
	LambdaRankNDCG	512	1024	1024	.1	.002
	LambdaRankNDCG(1)	1024	256	256	.1	.002
RANK-SC	MSE	1024	1024	1024	.1	.001
	iwMSE(1)	1024	1024	1024	.1	.002
	iwMSE(2)	512	512	1024	.1	.01
	iwMSE(3)	1024	512	512	.1	.001
	ApproxNDCG	1024	256	512	.1	.002

Table B.14: Best-performing hyper-parameter for GBDT tuning phase 1

Labeling strategy	**Loss function**	**number estimators**	**learning rate**	**maximum depth**	**min child weight**	**sub-sampling rate**	**column sample**
BIN	BCE	450	.01	9	1	.7	.5
BIN	LambdaBinary	350	.1	8	2	.7	.5
OV	MSE	450	.01	10	2	.7	.7
N-OV	MSE	450	.01	10	1	.7	.7
RANK	MSE	350	.01	10	1	.7	.7
	iwMSE(1)	450	.01	10	2	.7	.7

	iwMSE(2)	450	.01	10	2	.7	.7
	iwMSE(3)	350	.01	10	2	.7	.7
	RankPairwise	450	.1	9	2	.7	.5
	LambdaMARTNDCG	250	.1	9	1	.7	.5
RANK-SC	MSE	350	.01	10	1	.7	.7
	iwMSE(1)	450	.01	10	1	.7	.5
	iwMSE(2)	450	.01	9	2	.7	.5
	iwMSE(3)	450	.01	10	1	.7	.5

Appendix C. Best-performing hyper-paramters for tuning phase 2

Table C.15: Best performing hyper-parameters for NN phase 2

Labeling strategy	**Loss function**	**layer 1**	**layer 2**	**layer 3**	**dropout rate**	**reg rate**
BIN	BCE	2048	256	2048	.1	.001
RANK	iwMSE(2)	2048	2048	2048	.1	.002

Table C.16: Best-performing hyperparameters for GBDT phase 2

Labeling strategy	**Loss function**	**number estimators**	**learning rate**	**maximum depth**	**min child weight**	**sub-sampling rate**	**column sample**
BIN	BCE	550	.01	10	2	.7	.5
RANK	iwMSE(2)	550	.01	10	2	.7	.5

References

Adam, S.P., Alexandropoulos, S.A.N., Pardalos, P.M., Vrahatis, M.N., 2019. No Free Lunch Theorem: A Review, in: Demetriou, I.C., Pardalos, P.M. (Eds.), Approximation and Optimization. Springer International Publishing, Cham. volume 145 of *Springer Optimization and Its Applications*, pp. 57–82. doi:DOI:10.1007/978-3-030-12767-1_5.

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. International Journal of Production Research 42, 1621–1638. doi:DOI:10.1080/00207540310001636994.

Burges, C., Ragno, R., Le, Q., 2006. Learning to Rank with Nonsmooth Cost Functions, in: B. Schölkopf, J. Platt, T. Hoffman (Eds.), Advances in Neural Information Processing Systems, MIT Press. URL: https://proceedings.neurips.cc/paper_files/paper/2006/file/af44c4c56f385c43f2529f9b1b018f6a-Paper.pdf.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G., 2005. Learning to rank using gradient descent, in: Dzeroski, S. (Ed.), Proceedings of the 22nd international conference on Machine learning - ICML '05, ACM Press, New York, New York, USA. pp. 89–96. doi:DOI:10.1145/1102351.1102363.

Burges, C.J.C., 2010. From ranknet to lambdarank to lambdamart: An overview. Learning 11, 81.

Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H., 2007. Learning to rank: from pairwise approach to listwise approach, in: Ghahramani, Z. (Ed.), ICML 2007, ACM, New York. pp. 129–136. doi:DOI:10.1145/1273496.1273513.

Castillo, F., Gazmuri, P., 2015. Genetic algorithms for batch sizing and production scheduling. The International Journal of Advanced Manufacturing Technology 77, 261–280. doi:DOI:10.1007/s00170-014-6456-5.

Chapelle, O., Le, Q., Smola, A., 2007. Large margin optimization of ranking measures, in: NIPS workshop: Machine learning for Web search.

Dauzère-Pérès, S., Mönch, L., 2013. Scheduling jobs on a single batch processing machine with incompatible job families and weighted number of tardy jobs objective. Computers & Operations Research 40, 1224–1233. doi:DOI:10.1016/j.cor.2012.12.012.

Donmez, P., Svore, K.M., Burges, C.J., 2009. On the local optimality of LambdaRank, in: Allan, J., Aslam, J., Sanderson, M., Zhai, C., Zobel, J. (Eds.), Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA. pp. 460–467. doi:DOI:10.1145/1571941.1572021.

Droste, S., Jansen, T., Wegener, I., 2002. Optimization with randomized search heuristics—the (A)NFL theorem, realistic scenarios, and difficult functions. Theoretical Computer Science 287, 131–144. doi:DOI:10.1016/S0304-3975(02)00094-4.

Eiben, A.E., Smith, J.E., 2015. Introduction to evolutionary computing. Natural computing series. second edition ed., Springer, Berlin and Heidelberg and New York and Dordrecht and London.

Feurer, M., Hutter, F., 2019. Hyperparameter Optimization. Springer International Publishing. p. 3–33. URL: http://dx.doi.org/10.1007/978-3-030-05318-5_1, doi:DOI:10.1007/978-3-030-05318-5_1.

Freund, Y., Iyer, R., Schapire, R.E., Singer, Y., 2003. An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research 4, 933–969.

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

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, 5131–5154. doi:DOI:10.1080/00207543.2021.1951446.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. Adaptive computation and machine learning, The MIT Press, Cambridge, Massachusetts and London, England.

Helo, P., Phuong, D., Hao, Y., 2019. Cloud manufacturing – Scheduling as a service for sheet metal manufacturing. Computers & Operations Research 110, 208–219. doi:DOI:10.1016/j.cor.2018.06.002.

Ho, Y.C., Pepyne, D.L., 2002. Simple Explanation of the No-Free-Lunch Theorem and Its Implications. Journal of Optimization Theory and Applications 115, 549–570. doi:DOI:10.1023/A:1021251113462.

Huang, J., Wang, L., Jiang, Z., 2020. A method combining rules with genetic algorithm for minimizing makespan on a batch processing machine with preventive maintenance. International Journal of Production Research 58, 4086–4102. doi:DOI:10.1080/00207543.2019.1641643.

Huynh, N.T., Chien, C.F., 2018. A hybrid multi-subpopulation genetic algorithm for textile batch dyeing scheduling and an empirical study. Computers & Industrial Engineering 125, 615–627. doi:DOI:10.1016/j.cie.2018.01.005.

Järvelin, K., Kekäläinen, J., 2000. IR evaluation methods for retrieving highly relevant documents, in: Belkin, N.J., Ingwersen, P., Leong, M.K. (Eds.), SIGIR 2000, ACM Press, New York. pp. 41–48. doi:DOI:10.1145/345508.345545.

Järvelin, K., Kekäläinen, J., 2002. Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems 20, 422–446. doi:DOI:10.1145/582415.582418.

Jia, W., Jiang, Z., Li, Y., 2013. Closed loop control-based real-time dispatching heuristic on parallel batch machines with incompatible job families and dynamic arrivals. International Journal of Production Research 51, 4570–4584. doi:DOI:10.1080/00207543.2013.774505.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M., 2011. Algorithm selection and scheduling, in: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, Springer-Verlag, Berlin, Heidelberg. p. 454–469. doi:DOI:10.5555/2041160.2041198.

Kim, Y.J., Jang, J.W., Kim, D.S., Kim, B.S., 2021. Batch loading and scheduling problem with processing time deterioration and rate-modifying activities. International Journal of Production Research , 1–21doi:DOI:10.1080/00207543.2020.1866783.

Koh, S.G., Koo, P.H., Ha, J.W., Lee, W.S., 2004. Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. International Journal of Production Research 42, 4091–4107. doi:DOI:10.1080/00207540410001704041.

Koh, S.G., Koo, P.H., Kim, D.C., Hur, W.S., 2005. Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families. International Journal of Production Economics 98, 81–96. doi:DOI:10.1016/j.ijpe.2004.10.001.

Kotthoff, L., 2016. Algorithm Selection for Combinatorial Search Problems: A Survey, in: Bessiere, C., de Raedt, L., Kotthoff, L., Nijssen, S., O'Sullivan, B., Pedreschi, D. (Eds.), Data Mining and Constraint Programming. Springer International Publishing and Imprint and Springer, Cham. volume 10101 of *LNCS sublibrary. SL 7, Artificial intelligence*, pp. 149–190. doi:DOI:10.1007/978-3-319-50137-6{\textunderscore}7.

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. Computers & Operations Research 34, 3016–3028. doi:DOI:10.1016/j.cor.2005.11.011.

Mönch, L., Balasubramanian, H., Fowler, J.W., Pfund, M.E., 2005. Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times. Computers & Operations Research 32, 2731–2750. doi:DOI:10.1016/j.cor.2004.04.001.

Mönch, L., Schabacker, R., Pabst, D., Fowler, J.W., 2007. Genetic algorithm-based subproblem solution procedures for a modified shifting bottleneck heuristic for complex job shops. European Journal of Operational Research 177, 2100–2118. doi:DOI:10.1016/j.ejor.2005.12.020.

Murphy, K.P., 2013. Machine learning: A probabilistic perspective. Adaptive computation and machine learning series. 4. print. (fixed many typos) ed., MIT Press, Cambridge, Mass. URL: http://www.cs.ubc.ca/%7Emurphyk/MLbook/.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Édouard Duchesnay, 2011. Scikit-learn: Machine learning in python. Journal of Machine Learning Research 12, 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

Qin, T., Liu, T.Y., Li, H., 2010. A general approximation framework for direct optimization of information retrieval measures. Information Retrieval 13, 375–397. doi:DOI:10.1007/s10791-009-9124-x.

Rice, J.R., 1976. The Algorithm Selection Problem, in: Advances in Computers Volume 15. Elsevier. volume 15 of *Advances in Computers*, pp. 65–118. doi:DOI:10.1016/S0065-2458(08)60520-3.

Shwartz-Ziv, R., Armon, A., 2022. Tabular data: Deep learning is not all you need. Information Fusion 81, 84–90. doi:DOI:10.1016/j.inffus.2021.11.011.

Smith-Miles, K.A., 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. ACM Computing Surveys 41, 1–25. doi:DOI:10.1145/1456650.1456656.

Soares, C., Brazdil, P.B., Kuba, P., 2004. A meta-learning method to select the kernel width in support vector regression. Machine Learning 54, 195–209. URL: http://dx.doi.org/10.1023/B:MACH.0000015879.28004.9b, doi:DOI:10.1023/b:mach.0000015879.28004.9b.

Streeter, M.J., Golovin, D., Smith, S.F., 2007. Combining multiple heuristics online, in: AAAI Conference on Artificial Intelligence. URL: https://api.semanticscholar.org/CorpusID:7982549.

Taylor, M., Guiver, J., Robertson, S., Minka, T., 2008. SoftRank, in: Najork, M., Broder, A., Chakrabarti, S. (Eds.), Proceedings of the international conference on Web search and web data mining - WSDM '08, ACM Press, New York, New York, USA. p. 77. doi:DOI:10.1145/1341531.1341544.

Uzunoglu, A., Gahm, C., Tuma, A., 2023a. A machine learning enhanced multi-start heuristic to efficiently solve a serial-batch scheduling problem. Annals of Operations Research doi:DOI:10.1007/s10479-023-05541-w.

Uzunoglu, A., Gahm, C., Wahl, S., Tuma, A., 2023b. Learning-augmented heuristics for scheduling parallel serial-batch processing machines. Computers & Operations Research 151, 106122. doi:DOI:10.1016/j.cor.2022.106122.

Valente, J.M., Schaller, J.E., 2012. Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. Computers & Operations Research 39, 2223–2231. doi:DOI:10.1016/j.cor.2011.11.005.

Wahl, S., Gahm, C., Tuma, A., . Knowledge base for batch-processing machine scheduling research. doi:DOI:10.17632/7cv58py5hk.3.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1, 67–82. doi:DOI:10.1109/4235.585893.

Wu, K., Huang, E., Wang, M., Zheng, M., 2022. Job scheduling of diffusion furnaces in semiconductor fabrication facilities. European Journal of Operational Research 301, 141–152.

doi:DOI:10.1016/j.ejor.2021.09.044.

Xu, J., Liu, T.Y., Lu, M., Li, H., Ma, W.Y., 2008. Directly optimizing evaluation measures in learning to rank, in: Chua, T.S., Leong, M.K., Myaeng, S.H., Oard, D.W., Sebastiani, F. (Eds.), Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, ACM, New York, NY, USA. pp. 107–114. doi:DOI:10.1145/1390334.1390355.

BIBLIOGRAPHY

Balasubramanian, Hari, Lars Mönch, John Fowler, and Michele Pfund (2004). “Genetic algorithm based scheduling of parallel batch machines with incompatible job families to minimize total weighted tardiness.” In: *International Journal of Production Research* 42.8, pp. 1621–1638. ISSN: 0020-7543. DOI: 10.1080/00207540310001636994.

Barber, David (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press. ISBN: 9780521518147. DOI: 10.1017/CBO9780511804779.

Bartlett, Peter L., Nick Harvey, Christopher Liaw, and Abbas Mehrabian (2019). “Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks.” In: *The Journal of Machine Learning Research* 20.1, pp. 2285–2301.

Bengio, Yoshua, Andrea Lodi, and Antoine Prouvost (2021). “Machine learning for combinatorial optimization: A methodological tour d'horizon.” In: *European Journal of Operational Research* 290.2, pp. 405–421. ISSN: 03772217. DOI: 10.1016/j.ejor.2020.07.063.

Bennell, Julia A. and Jose F. Oliveira (2008). “The geometry of nesting problems: A tutorial.” In: *European Journal of Operational Research* 184.2, pp. 397–415. ISSN: 03772217. DOI: 10.1016/j.ejor.2006.11.038.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York: Springer. ISBN: 978-0-387-31073-2.

Bitran, Gabriel R., Elizabeth A. Haas, and Arnoldo C. Hax (1981). “Hierarchical Production Planning: A Single Stage System.” In: *Operations Research* 29.4, pp. 717–743. ISSN: 0030-364X. DOI: 10.1287/opre.29.4.717.

Bousquet, Olivier, Stéphane Boucheron, and Gábor Lugosi (2004). “Introduction to Statistical Learning Theory.” In: *Advanced Lectures on Machine Learning*. Ed. by Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch. Vol. 3176. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 169–207. ISBN: 978-3-540-23122-6. DOI: 10.1007/978-3-540-28650-9_8.

Efron, Bradley and Trevor Hastie (2016). *Computer age statistical inference*. Institute of Mathematical Statistics monographs. New York, NY, USA: Cambridge University Press. ISBN: 9781107149892.

Fischetti, Matteo and Ivan Luzzi (2009). “Mixed-integer programming models for nesting problems.” In: *Journal of Heuristics* 15.3, pp. 201–226. ISSN: 13811231. DOI: 10.1007/s10732-008-9088-9.

Fisher, R. A. (1925). "Theory of Statistical Estimation." In: *Mathematical Proceedings of the Cambridge Philosophical Society* 22.5, pp. 700–725. ISSN: 0305-0041. DOI: 10.1017/S0305004100009580.

Gahm, Christian, Aykut Uzunoglu, Stefan Wahl, Chantal Ganschinietz, and Axel Tuma (2022a). "Applying machine learning for the anticipation of complex nesting solutions in hierarchical production planning." In: *European Journal of Operational Research* 296.3, pp. 819–836. ISSN: 03772217. DOI: 10.1016/j.ejor.2021.04.006.

Gahm, Christian, Stefan Wahl, and Axel Tuma (2022b). "Scheduling parallel serial-batch processing machines with incompatible job families, sequence-dependent setup times and arbitrary sizes." In: *International Journal of Production Research* 60.17, pp. 5131–5154. ISSN: 0020-7543. DOI: 10.1080/00207543.2021.1951446.

Gal, Yarin and Zoubin Ghahramani (2016). "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." In: *international conference on machine learning*, pp. 1050–1059.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Gu, Jiuxiang et al. (2018). "Recent advances in convolutional neural networks." In: *Pattern Recognition* 77, pp. 354–377. ISSN: 00313203. DOI: 10.1016/j.patcog.2017.10.013.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning*. New York, NY: Springer New York. ISBN: 978-0-387-84857-0. DOI: 10.1007/978-0-387-84858-7.

Hax, Arnoldo C. and Harlan C. Meal (1973). "Hierarchical integration of production planning and scheduling." In: *Sloan working papers no.656-73*.

Helo, Petri, Duy Phuong, and Yuqiuge Hao (2019). "Cloud manufacturing – Scheduling as a service for sheet metal manufacturing." In: *Computers & Operations Research* 110, pp. 208–219. ISSN: 03050548. DOI: 10.1016/j.cor.2018.06.002.

Jordan, M. I. and T. M. Mitchell (2015). "Machine learning: Trends, perspectives, and prospects." In: *Science (New York, N.Y.)* 349.6245, pp. 255–260. DOI: 10.1126/science.aaa8415.

Kim, Soo-Young, Young-Hoon Lee, and Dillip Agnihotri (1995). "A hybrid approach to sequencing jobs using heuristic rules and neural networks." In: *Production Planning & Control* 6.5, pp. 445–454. ISSN: 0953-7287. DOI: 10.1080/09537289508930302.

Kotthoff, Lars (2016). "Algorithm Selection for Combinatorial Search Problems: A Survey." In: *Data Mining and Constraint Programming*. Ed. by Christian Bessiere, Luc de Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi. Vol. 10101. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 149–190. ISBN: 978-3-319-50136-9. DOI: 10.1007/978-3-319-50137-6_7.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning." In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

MacKay, David J. C. (2019). *Information theory, inference, and learning algorithms*. Cambridge: Cambridge University Press. ISBN: 0521642981.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar (2018). *Foundations of machine learning*. Second edition. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press. ISBN: 9780262039406.

Murphy, Kevin P. (2013). *Machine learning: A probabilistic perspective*. 4. print. (fixed many typos). Adaptive computation and machine learning series. Cambridge, Mass.: MIT Press. ISBN: 0262018020.

Nitta, Tohru (2003). "Solving the XOR problem and the detection of symmetry using a single complex-valued neuron." In: *Neural networks : the official journal of the International Neural Network Society* 16.8, pp. 1101–1105. DOI: 10.1016/S0893-6080(03)00168-0.

Pandey, Mohit, Michael Fernandez, Francesco Gentile, Olexandr Isayev, Alexander Tropsha, Abraham C. Stern, and Artem Cherkasov (2022). "The transformational role of GPU computing and deep learning in drug discovery." In: *Nature Machine Intelligence* 4.3, pp. 211–221. DOI: 10.1038/s42256-022-00463-x.

Rardin, Ronald L. and Reha Uzsoy (2001). "Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial." In: *Journal of Heuristics* 7.3, pp. 261–304. ISSN: 13811231. DOI: 10.1023/A:1011319115230.

Rice, John R. (1976). "The Algorithm Selection Problem." In: *Advances in Computers Volume 15*. Vol. 15. Advances in Computers. Elsevier, pp. 65–118. ISBN: 9780120121151. DOI: 10.1016/S0065-2458(08)60520-3.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, pp. 386–408. ISSN: 0033-295X. DOI: 10.1037/h0042519.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors." In: *Nature* 323.6088, pp. 533–536. ISSN: 0028-0836. DOI: 10.1038/323533a0.

Russell, Stuart and Peter Norvig (2021). *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Education Limited. ISBN: 1-292-40113-3.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: an overview." In: *Neural networks : the official journal of the International Neural Network Society* 61, pp. 85–117. DOI: 10.1016/j.neunet.2014.09.003.

Schneeweiß, Christoph (1995). “Hierarchical structures in organisations: A conceptual framework.” In: *European Journal of Operational Research* 86.1, pp. 4–31. ISSN: 03772217. DOI: 10.1016/0377-2217(95)00058-X.

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding Machine Learning*. Cambridge University Press. ISBN: 9781107057135. DOI: 10.1017/CBO9781107298019.

Shen, Liji and Udo Buscher (2012). “Solving the serial batching problem in job shop manufacturing systems.” In: *European Journal of Operational Research* 221.1, pp. 14–26. ISSN: 03772217. DOI: 10.1016/j.ejor.2012.03.001.

Uzunoglu, Aykut, Christian Gahm, Stefan Wahl, and Axel Tuma (2023a). “Learning-augmented heuristics for scheduling parallel serial-batch processing machines.” In: *Computers & Operations Research* 151, p. 106122. ISSN: 03050548. DOI: 10.1016/j.cor.2022.106122.

Uzunoglu, Aykut, Christian Gahm, and Axel Tuma (2023b). “A machine learning enhanced multi-start heuristic to efficiently solve a serial-batch scheduling problem.” In: *Annals of Operations Research*. ISSN: 0254-5330. DOI: 10.1007/s10479-023-05541-w.

Uzunoglu, Aykut, Christian Gahm, and Axel Tuma (2023c). “Machine Learning based Algorithm Selection and Genetic Algorithms for serial-batch scheduling.” submitted to: *Computers & Operations Research*.

Uzunoglu, Aykut (2023). *Data set and models to select algorithms for a serial-batch scheduling problem*. DOI: 10.17632/4s2zfg6mg9.1.

Vapnik, V. N. and A. Ya. Chervonenkis (1971). “On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities.” In: *Theory of Probability & Its Applications* 16.2, pp. 264–280. ISSN: 0040-585X. DOI: 10.1137/1116025.

Vapnik, Vladimir N. (2000). *The Nature of Statistical Learning Theory*. New York, NY: Springer New York. ISBN: 978-1-4419-3160-3. DOI: 10.1007/978-1-4757-3264-1.

Vepsalainen, Ari P. J. and Thomas E. Morton (1987). “Priority Rules for Job Shops with Weighted Tardiness Costs.” In: *Management Science* 33.8, pp. 1035–1047. ISSN: 0025-1909. DOI: 10.1287/mnsc.33.8.1035.

Wäscher, Gerhard, Heike Haußner, and Holger Schumann (2007). “An improved typology of cutting and packing problems.” In: *European Journal of Operational Research* 183.3, pp. 1109–1130. ISSN: 03772217. DOI: 10.1016/j.ejor.2005.12.047.

Wolpert, David H. (1996). “The Lack of A Priori Distinctions Between Learning Algorithms.” In: *Neural Computation* 8.7, pp. 1341–1390. ISSN: 0899-7667. DOI: 10.1162/neco.1996.8.7.1341.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*.