OR Spectrum (2019) 41:871–893
https://doi.org/10.1007/s00291-019-00567-8

REGULAR ARTICLE

A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking

Frank Benda[1,2] · Roland Braune[3] · Karl F. Doerner[3,4] · Richard F. Hartl[3]

Received: 31 December 2017 / Accepted: 5 November 2019 / Published online: 18 November 2019

Abstract

In proposing a machine learning approach for a flow shop scheduling problem with alternative resources, sequence-dependent setup times, and blocking, this paper seeks to generate a tree-based priority rule in terms of a well-performing decision tree (DT) for dispatching jobs. Furthermore, generating a generic DT and RF that yields competitive results for instance scenarios that structurally differ from the training instances was another goal of our research. The proposed DT relies on high quality solutions, obtained using a constraint programming (CP) formulation. Novel aspects include a unified representation of job sequencing and machine assignment decisions, as well as the generation of random forests (RF) to counteract overfitting behaviour. To show the performance of the proposed approaches, different instance scenarios for two objectives (makespan and total tardiness minimisation) were implemented, based on randomised problem data. The background of this approach is a real-world physical system of an industrial partner that represents a typical shop floor for many production processes, such as furniture and window construction. The results of a comparison of the DT and RF approach with two priority dispatching rules, the original CP solutions and tight lower bounds retrieved from a strengthened mixed-integer programming (MIP) formulation show that the proposed machine learning approach performs well in most instance sets for the makespan objective and in all sets for the total tardiness objective.

Keywords Machine learning · Flow shop scheduling · Decision trees · Random forests · Constraint programming

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00291-019-00567-8) contains supplementary material, which is available to authorized users.

✉ Frank Benda
frank.benda@univie.ac.at

Extended author information available on the last page of the article

1 Introduction

With the emergence of topics related with smart manufacturing and data analytics, established machine learning approaches may gain new life to achieve deeper insights into production processes (Wuest et al. 2016). Machine learning approaches can be used to capture complex processing environments in a way such that scheduling policies, in particular dispatching rules, can be derived.

In this context, we propose a machine learning approach that is used to generate a tree-based priority dispatching rule for material movement decisions. The subject of this paper is a problem setting based on a real-world physical system of our industrial partner—a business consultancy located in Vienna. In essence, it can be classified as a hybrid flow shop scheduling problem with alternative resources, sequence-dependent setup times, limited intermediate buffers, and blocking. A transport resource with limited capacity is also part of the model configuration.

We intend to show that our machine learning approach performs well in such scheduling problems, with makespan and total tardiness minimisation as objectives, using solution information obtained from optimisation runs of a constraint programming (CP) solver that can provide high quality, feasible solutions. These solutions are then read-in with the help of a deterministic shop floor simulator and transformed into training examples. Finally, the training examples are used to build a decision tree (DT). The training examples consist of pairwise comparisons of all possible material movements, attributes, and the corresponding classification. To calculate the classification error, we build the tree on a training set and test it on both the training and a test set. We use a cross-validation procedure to estimate the off-training-set error rate of the tree. The DT acts as a classifier, indicating whether a possible job movement should be conducted or not.

Prior literature offers different approaches to exploit the training examples for building a DT. Shahzad and Mebarki (2012) generate the training examples using an optimisation module that solves instances based on tabu search. Olafsson and Li (2010) transform dispatching lists, created by a simulated scheduler in combination with a weighted earliest due date rule (EDD), into a training set.

To the best of our knowledge, the combination of our flow shop scheduling problem characteristics has not yet been discussed in the literature. Flow shops with limited intermediate buffer in general are considered by Brucker et al. (2003) and Leisten (1990). Hall and Sriskandarajah (1996) describe blocking as a lack of storage, noting that the flow shop problem with a finite buffer is a common scheduling problem. Mascis and Pacciarelli (2002) introduce blocking caused by a processed job waiting to be moved to the next machine. In our case, we consider limited buffer both before and after the processing slot of a machine. The limited storage space in front of (and behind) the machines prohibits accumulating arbitrary amounts of material between processing stages and can thus lead to blocking behaviour if all the available buffer on a stage has been depleted. Ruiz et al. (2005) describe sequence-dependent setup times caused by switching between two different job types on a processing slot. In comparison with these approaches, we deal with greater complexity regarding the shop floor and order configuration.

For machine learning in similar environments, Doh et al. (2014) suggest a decision tree-based approach to select a priority rule combination, depending on the current status quo on the shop floor. For our configuration, such an approach is not suitable because the goal is to develop a priority rule instead of choosing one.

The contribution of this paper is threefold. First, we extract information from solutions provided by a CP solver, based on an appropriate formulation of the problem, to generate the training data. Second, the generated dispatching rule directly combines the job sequencing with the machine assignment in one step. The decision involves not only the next job to be moved but also onto which machine the job should be transported to. This approach allows for more complex decision making. Third, we embed our DT in a random forest (RF) approach to counteract overfitting.

The content of the paper is organised as follows: Section 2 contains the problem statement in detail. In Sect. 3, we describe how DTs and RFs are actually built and applied to the problem at hand. The configuration and generation of test instances finally used in the computational experiments are presented in Sect. 4. The computational results for the DT and RF approach are summarised and discussed in Sect. 5. Finally, the discussion in Sect. 6 briefly reviews the contributions and offers suggestions for future research.

2 Problem statement

We implemented an abstract model based on the basic shop floor configuration of the industrial partner, which is depicted exemplarily in Fig. 1 for an exemplary configuration of the shop floor. Three different type of jobs have to be processed on every processing stage starting at the raw material stage. The processing stages consist of one or more parallel identical or unrelated machines (depending on the scenario). After having passed the last processing stage, all the jobs are collected in a box. As soon as all the jobs of an order are finished, this order is marked as executed. The order constraint is important for the total tardiness objective function.

The machines are equipped with buffer slots, a processing slot, and a transport slot. For every processed job that is situated on a transport slot of a machine, a crane movement has to be initiated. The crane as a limited transport resource is able to move one job at a time from a transport slot or raw material stage to a buffer slot of a subsequent stage. A job transport between two machines can only be conducted by picking up the job at a transport slot and moving it to the first buffer slot of the subsequent machine. Overtaking other jobs on slots of that machine is not allowed. Blocking might occur if all buffer slots of each machine within a stage are fully occupied, which also would create blocking on machines in preceding stages.

Finally, job processing is subject to sequence-dependent setup times. Depending on the type of the preceding job, additional work might be necessary to set up the machine for the next job. It is assumed here that the setup activity can already start *before* the next job has arrived at the machine.

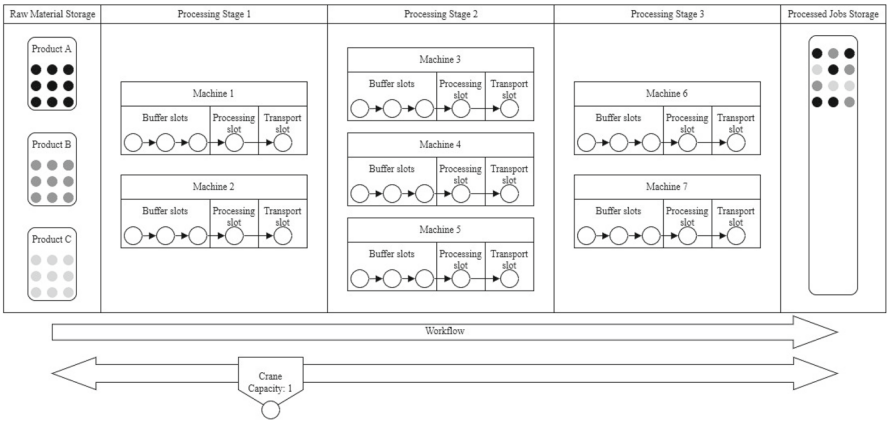

Fig. 1 Exemplary configuration of the shop floor

Table 1 List of attributes

Abbreviation	Description	Category
NP	Next processing time	Job
ST	Setup time on next machine	
B[a]	Does job block buffer in front of processing slot?	
S	Number of occupied processing and buffer slots on next machine	Machine
MR	Machine workload on current machine	
MN	Machine workload on next machine	
C	Crane position in relation to job position	Transport
T	Transport time current machine to next machine	
NT[a]	Is there a job on next machine on transport slot available?	
RDD	Time left until due date is reached	Tardiness
ADD	Absolute due date	
CDDR	Critical ratio: (due date - current time) / total shop time left	

[a]Binary attribute values

3 Solution approach

For the problem at hand, we propose a machine learning approach for building a DT that can be applied as a dispatching rule for the flow shop problem, described in Sect. 2. A DT is a classifier for solving classification problems and is built using training examples. Those training examples consist of pairwise comparisons of available material movements, capturing the status quo situation of the shop floor at the moment of decision. The status quo situation is used to describe the training examples with the help of attributes. Those attributes, in turn, are an important part for training and applying the DT. The process of attribute selection, DT generation, and application

is described in this section. Furthermore, a method—critical path approach—to filter the training examples used to build the DT is demonstrated.

3.1 Attribute selection

Several test runs were conducted to determine a comprehensive list of attributes related to the current state of jobs, machines, transport resource, and the prospective due date adherence. After these test runs, 12 of them have finally been chosen, as listed in Table 1. Those tests served to analyse factors, such as the crane transport behaviour, the job assignment to the parallel machines, comparing the workload ratios of all machines for each instance within every status quo of the production process, and blocking behaviour with its effect on preceding stages.

3.2 Building the decision tree using the C4.5 algorithm

Preliminary experiments revealed that the CP formulation (Appendix B) delivers good upper bounds, much faster than the MIP described in Appendix A, distributed as online supplementary material. The schedules created by the CP solver are read-in by a deterministic shop floor simulator, as described in Sect. 3.3. The training data are then handed over to the C4.5 DT builder algorithm (`Accord.NET Machine Learning Framework` Version 3.8.0).

The C4.5 algorithm (DT builder algorithm) is an advancement of the original Iterative Dichotomiser 3 (ID3) developed by Quinlan (1986). The DT consists of decision nodes, branches, and leaves. Each node represents a true/false decision, leading to exactly two branches. To start, the root node of the DT is determined by selecting the attribute that separates the training examples as good as possible into two groups. The criteria for comparing the goodness of such a split is the resulting information gain. A branch gets created for each part of the training examples, and the algorithm proceeds recursively. As soon as no further information gain results from any of the attributes, a leaf is created with the corresponding classification (Quinlan 1993).

In Fig. 2, an exemplary DT is shown. The goal is to determine whether material movement option 1 is preferred over option 2. The classification is formulated as "Prefer option 1?" within a pairwise comparison ("Yes" or "No") (Olafsson and Li 2010). We start at the root of the tree. The first comparison between the two options is their transport time to the next machine and whether the difference is greater than 0.5. If not, we already receive the classification "No", and otherwise, we follow the branch to the next attribute—the difference in processing times on the next machines. For every decision node, the attribute value that defines the split for this node determines the next branch to follow. The leaf finally contains the classification for the test example.

3.3 Training the DT from CP solution information using a shop floor simulator

The solution of every problem instance is read-in by a simulator which is, in essence, a customised list scheduling algorithm. The entire production sequence is simulated

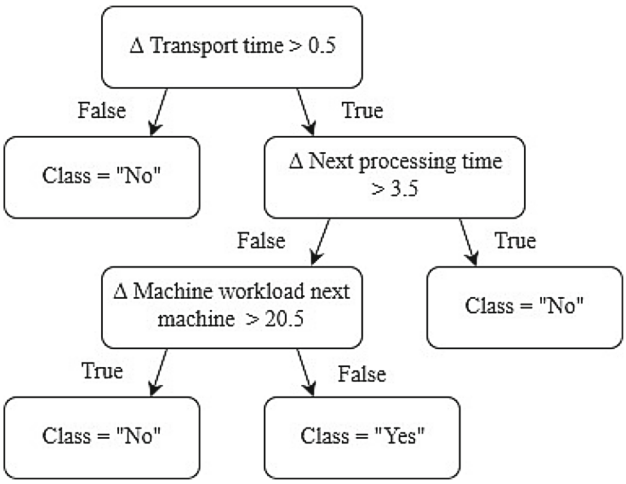

Fig. 2 Decision tree example

according to information from the CP schedule. Whenever there is more than one job available, or one job can be put onto more than one machine, the solution of the CP result for this decision is read-in. At each of the decision points, we collect the attribute values for building the training examples. The format of the information consists of three different parts: pairwise comparisons of possible material movements, attributes, and class labels.

Li and Olafsson (2005) propose pairwise comparisons that are the first component of the training data. The basic idea is to map sequencing decisions to precedence information between any two jobs that are currently schedulable. Assume that three jobs can be scheduled on a machine at the same time and a (precomputed) schedule gives (3,1,2) as the best sequence to resolve this resource contention situation. Then the resulting precedences that fully describe this sequence are $3 \rightarrow 1$, $3 \rightarrow 2$, and $1 \rightarrow 2$. Hence, the DT approach attempts to derive this kind of precedence information for any two jobs that are ready to be scheduled.

The novelty that we introduce is to combine the decision about which job to choose with the decision onto which machine it should go next. For example, the possibility J2M3 is shorthand for "put job 2 onto machine 3". For every decision within the instances solved by CP, the selected possibility out of all other possibilities is marked as "chosen", and all the others are marked "not chosen". Every possibility then becomes an option, ranked by job number and machine number in numerical order. Whether the "chosen" option is option 1 or 2 (within a pair of jobs) depends on its rank within the numerical order. In the next step, every possibility marked as "not chosen" gets compared with the one marked "chosen", with the help of attributes, so it results in pairs of options.

Because the description of the classification is "Choose option 1?" the target classes of the DT are binary and labelled "No" (0) or "Yes" (1). Every pairwise comparison trains the decision tree. Every comparison contains two options, and the classification of either option 1 ("Choose option 1?" Classification: "Yes") or option 2 ("Choose

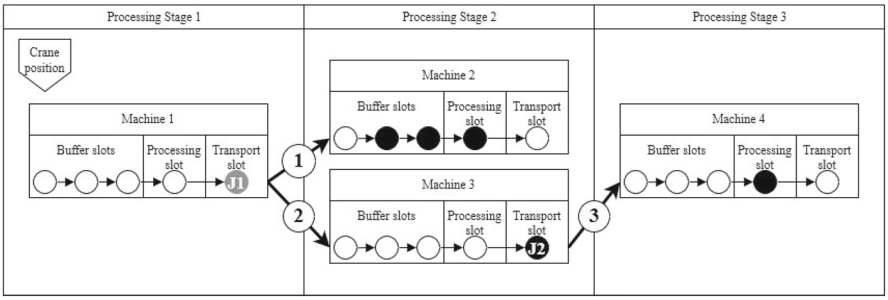

Fig. 3 Example: status quo on shop floor with possible job movements

Table 2 Example: pairwise comparisons for CP result "Choose J1M3"

Pairwise comparisons		Attributes (Δ)									Classification
Option 1	Option 2	NP	C	MR	MN	T	ST	S	B	NT	Choose option 1?
J1M2	**J1M3**	3	0	0	56	0	0	3	0	−1	No
J1M3	J2M4	−1	−1	0	−6	−2	12	−1	0	1	Yes

option 1?" Classification: "No") is the one preferred by the result of the CP. Comparing two options that are not chosen does not lead to a definite classification for "Choose option 1?". So we skip those pairwise comparisons. The last step combines the pairwise comparisons and their classification with status quo information about the shop floor at the moment of the decision.

This status quo is captured by the attributes, as described above. As the goal is to compare two options, each with its own attribute values, we calculate the difference for each kind of attribute and use this quantity as a branching criterion in a DT, instead of absolute values.

To demonstrate how we capture the status quo of the shop floor, we offer an example, as summarised in Fig. 3. There are three possible material movements: ① Put job 1 onto machine 2 (J1M2), ② Put job 1 onto machine 3 (J1M3), ③, Put job 2 onto machine 4 (J2M4). Assume that the CP result is to choose J1M3, such that J1M3 is marked as "chosen". This leads to two different pairwise comparisons: J1M2 with J1M3 and J1M3 with J2M4, as listed in Table reftab:ClassificationCPExample. The options are ranked as first or second, according to the numerical order. In the next step, for each attribute, we compare the values of option 1 and option 2 by calculating the difference, that is, value option 1 minus value option 2. If, for the next processing time of job 1 on machine 2 (J1M2) is 26 time units, whereas the processing time of job 1 on machine 3 (J1M3) is 23 time units, the outcome is 3 (see NP-column, Table 2). The classification results in a "No" for the first row and a "Yes" for the second row, referring to whether option 1 is the "chosen" option: "Choose option 1?" In the next step, the training examples enter the DT builder algorithm, as described in Sect. refsec2:BuildDecisionTree.

Table 3 Example: pairwise comparisons for applying a DT

Pairwise comparisons		Attributes (Δ)									Classification
Option 1	Option 2	NP	C	MR	MN	T	ST	S	B	NT	Choose option 1?
J1M2	J1M3	3	0	0	56	0	0	3	0	−1	No
J1M2	J2M4	2	−1	0	50	−2	12	2	0	0	No
J1M3	J2M4	−1	−1	0	−6	−2	12	−1	0	1	Yes

Finally, the classes determined by the DT must be transformed into a precise decision, about where to move the crane to transport the corresponding job to the next machine. Therefore, a yes–no voting was implemented, as described next.

3.4 Applying the trained DT: the yes–no voting procedure

The output of a DT is classes, in terms of whether an option should be preferred or not. But the classifications of the pairwise comparisons still do not provide enough information to decide which job-machine possibility should be selected. Therefore, the classifications have to be transformed into a precise material movement decision. The example in Sect. 3.3 illustrates the transformation.

The status quo on the shop floor is the same as in Fig. 3. However, the next material movement is not yet known. The DT is used to decide which of the possible material movements should be carried out next. Therefore, the output of the DT is the classification of each pairwise comparison of all possible material movements, as listed in Table 3.

After finishing the classification process for each pairwise comparison, it is still unknown which possibility to select. This is accomplished by a yes–no voting scheme (Li and Olafsson 2005). It counts the amount of "Yes" and "No" responses for each possibility by transforming the classes in Table 3 into votes. For example, a "No" is added for option 1 (J1M2), and a "Yes" is added for option 2 (J1M3). We proceed analogously with all the pairwise comparisons and get the yes–no voting results in Table 4. With the help of a majority voting, the option with the highest amount of "Yes" is chosen as the next material movement. In this example, possibility J1M3 receives the most "Yes" votes, so job 1 gets moved to machine 3. If the score is even, a possibility that outranks another in the numerical order will be chosen. Note that it is sufficient to make a sequencing decision with regard to the *next* job to be scheduled only. Therefore, it is not necessary to establish a complete sequence of all available jobs.

3.5 Counteracting overfitting using a RF approach

To prevent the DT from overfitting the data, one effective technique is pruning (Quinlan 1993). However, preliminary tests with pruning indicated only limited success, without achieving the expected effect of more accurate trees or a decrease in the overall makespan. Therefore, we decided to employ RFs as a more advanced but still

Table 4 Example: yes–no voting for the possible material movements

Option 1	Option 2	Choose option 1?	J1M2		J1M3		J2M4	
			Yes	No	Yes	No	Yes	No
J1M2	J1M3	No		1	1			
J1M2	J2M4	No		1			1	
J1M3	J2M4	Yes			1			1
		Sum		2	2		1	1

easy-to-use approach for this purpose. Ho (1995) introduced the use of multiple trees as classifiers. A forest is generated randomly with the objective of differentiating the DTs. Generating a RF involves four main steps with two random features (Breiman 2001):

1. Set the number n of trees to be generated.
2. Set the sample ratio s of how many training examples out of the training data will be selected at most to train each tree, which represents the first source of randomness.
3. Set the maximum number of attributes a out of all attributes A that can be used to generate each tree, which is the second source of randomness. For every node of a tree, a number of $a \ll A$ attributes are taken into consideration for the split.
4. Fully build the trees without pruning them.

Figure 4 presents a simplified version of a RF. In accordance with a single DT, test examples are handed over to the RF. Each tree within the RF determines a classification for this example by following the branches until a leaf is reached. In contrast with the DT approach though, the process involves all of the generated trees. This results in n potentially different classifications. The most voted class is the final classification. However, finding a good setting of randomisation parameters so that the RF achieves an improvement over a single DT remains difficult. Therefore, we conducted a series of tests to test and tune the RF parameters, as described in detail in Sect. 4.

3.6 Example selection based on critical paths

In the basic setting, both the DT and the RF are fitted, based on available examples from the respective training set. The RFs inherently rely on drawing sub-samples of the training set and thus already perform a special kind of example selection, namely, a strictly randomised one. The principle of bootstrap aggregation, also referred to as bagging, should mitigate the well-known variance issues with standard decision trees, which emerge in the form of high sensitivity even to very slight changes in the training data and a proneness to overfitting behaviour. Bagging represents a so-called meta-algorithm or ensemble method in machine learning because it combines two techniques: bootstrapping (referring to the random sub-sampling part) and aggregation by averaging the output of multiple individual classifiers (DTs in our case). However,

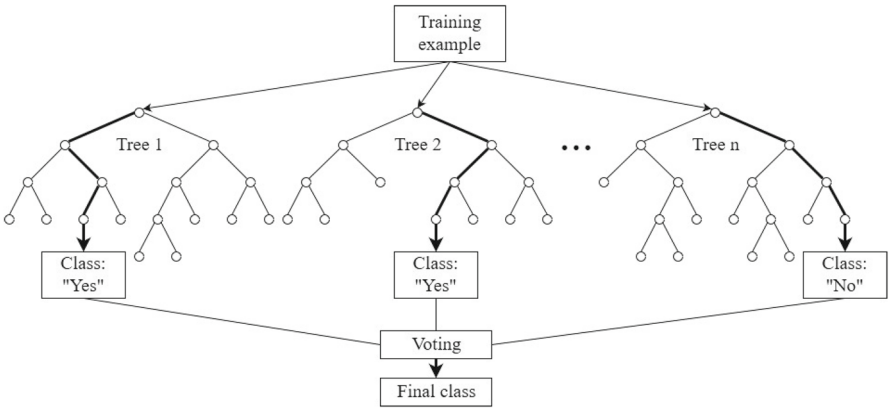

Fig. 4 Simplified RF example

a systematic and goal-oriented selection of examples cannot be achieved this way, since such an approach would have to be problem dependent. In the context of the scheduling problem, we study that not every decision made during the chronological scheduling has a direct impact on schedule performance. Consequently, it would be desirable to identify only relevant decisions and training examples before fitting a DT or a RF.

With a graph representation of a schedule (Balas 1969), we can use critical paths for this purpose. It identifies jobs or operations that cannot be delayed without immediately deteriorating the objective function value. This principle is most easily applied in the makespan context but also can be extended to sum-based objectives like total tardiness (Braune et al. 2013). We use this concept to filter training examples as follows: add only those pairwise comparisons that involve at least one job that is critical in the CP schedule. The criterion should omit decisions that do not have a direct impact on the objective value before they enter the tree fitting stage. In this way, we seek to prevent potentially irrelevant information from "disturbing" and misleading the tree generation process.

4 Experiments

In this section, we introduce the variety of shop floor configurations and the different instance scenarios used in the experimental study regarding the two objectives, makespan and tardiness.

4.1 Configuration of the instance scenarios in the makespan experiments

In the first step, the objective was to minimise the overall makespan. For this purpose, a set of three instance scenarios was defined to determine the effect of machine blocking and a high degree of capacity utilisation on the transport resource (crane blocking), as well as a mixture of both aspects, as indicated in Table 5. The machine blocking (MB)

Table 5 Configuration of the instance scenarios used in the makespan experiments

	MB scenario	CB scenario	Mixed scenario
Products	3	3	3
Processing stages	3–5	3–5	3–5
Parallel machines	1–3	**2–3**	1–3
Buffer slots	1–3	1–3	1–3
Processing time	20–34	20-34	20–34
Setup time	8–13	8–13	8–13

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

Table 6 Job load sets for the instance scenarios used in the makespan experiments

	Low load	Medium load	High load
Jobs per order	2–3	4–11	7–12
Orders	3–5	3–4	3–6
Jobs in total	6–15	12–44	21–72
Instances	80	80	80

was enforced by a bottleneck stage with at most one machine. For the crane blocking (CB) instances, each processing stage contains at least two parallel machines. On these parallel machines, the processing times for the same job type might differ (unrelated parallel machines). Each instance scenario is split according to the three different levels of job loads, as listed in Table 6. Note that the given number ranges in Tables 5 and 6 for jobs, orders, machines, processing times, etc., are just the quantities used for the experiments in this paper. Our proposed approach is of course able to handle arbitrary configurations, as long as they are structurally equivalent to the described shop floor setting.

For every instance scenario of the makespan objective function, 80 instances were randomly generated—according to the configurations in Tables 5 and 6. After training the tree, it gets evaluated with a cross-validation procedure. To perform a cross-validation, the 80 instances of a job load set form the basic data set. In each iteration of the procedure, this set is split into a training set of 70 training instances to build the DT and 10 test instances. Therefore, with eight runs, every subset becomes the test set exactly once—an eightfold cross-validation (Blockeel and Struyf 2003). The DT is built from the training set, and then the test set is used to evaluate the deviation of the calculated results from the CP results that serve as the baseline for all comparisons.

In addition to comparing the DT and RF approach to the CP formulation, we assessed their performance in relation to simple priority dispatching rules, for which the basic principle is to choose the job with the shortest processing time (Panwalkar and Iskander 1977). Priority rule 1 (PR1), extended to take setup time into consideration, selects the job with the shortest processing time on the next machine, combined with the inevitable setup time on that machine. Priority rule 2 (PR 2) adds the machine workload to PR1. Although PR2 includes PR1, we still consider PR1 in the results, so

Table 7 Configuration of the instance scenarios used in the total tardiness experiments

Scenario:	Bottleneck stages	Restricted buffer slots	Fully equipped shop floor
Products	3	3	3
Processing stages	4–7	4–7	4–7
Parallel machines	1–4	3–5	4–6
Buffer slots	2–4	1	5–8
Processing time	10–50	10-20	10–50
Setup time	3–10	3–10	3–10
Jobs per order	10	10	10
Orders	7	7	7
Instances	80	80	80

that we could analyse the effect of the overall makespan improvement adding machine workload to the priority rule. The comprehensive comparison of all approaches therefore includes the CP results compared with the DT approach, RF approach, PR1 without machine workloads, and PR2 including machine workloads.

Several test runs determine the parameters for the DTs and RF. Test runs for the DT approach were mainly concerned with varying the height of the tree. Heights of at least 3 to at most 50 were tested to determine their impact on the overall makespan.

4.2 Configuration of the instance scenarios in the total tardiness experiments

In the second step, a new objective function was introduced: minimising the overall tardiness of all orders. The goal was to generate new instance scenarios with only slight blocking behaviour and a fixed number of jobs per order. The configuration is listed in Table 7.

The first instance scenario—bottleneck stages (BS)—simulates moderate machine blocking behaviour without influencing the production process too much. In the second scenario—restricted buffer slots (RB)—blocking does not only occur due to only one machine in a processing stage but rather due to only one buffer slot on every single machine on all the stages. The slightly reduced processing times increase this effect. Finally, the third scenario—fully equipped shop floor (FE)—exemplifies a shop floor with nearly no buffer constraints.

Again, 80 instances were generated randomly for each scenario. The main difference between the makespan and the tardiness instances is how we set the processing times of same job types on parallel machines. To calculate due dates, the processing times of a job type on parallel machines were set to the same values (identical parallel machines), whereas in the makespan instances these processing times might differ (unrelated parallel machines).

The due date for an order was set deterministically by adding up the processing times on each processing stage for each job, plus its transport time. The resulting value was then multiplied by a factor of 1.3, which is a common procedure for tardiness job shop scheduling problems (Braune et al. 2013). Furthermore, jobs of the same order are not

necessarily processed in a row but probably simultaneously on parallel machines. To take this effect into account, the processing time of that job on each stage is divided by the total number of machines on that stage for the due date calculation. Reflecting the new total tardiness objective, PR1 was replaced by a due date specific dispatching rule, earliest due date (EDD). Instead of the makespan-related PR2, we also chose the well-established apparent tardiness cost (ATC) dispatching rule (Vepsalainen and Morton 1987). For both rules, the workload of possible next machines is also considered.

5 Computational results

Several test runs were conducted to determine a well-performing setup of parameters for both the instance generation and the DT and RF approach for both objectives. Those tests were implemented in C# on a workstation with Windows 10, 64 bit, an i7-4790 CPU core with 3.6 GHz, and 8 GB of RAM. Building the tree on this setup took less than a second. The IBM ILOG CPLEX and the IBM ILOG CP Optimizer were used to solve the MIP and CP formulation. The CP solution quality rating for the instance scenarios by running the MIP formulation ("Appendix A") is discussed in "Appendix C", distributed as online supplementary material. In this section, we present the best performing DT and RF configurations and discuss their results.

5.1 Results for the makespan objective with unrelated parallel machines

The results for the makespan objective are listed in Table 8. In interpreting these findings, we caution that in the low job load set, the number of pairwise comparisons is much lower than for the medium and high job load sets. For example, there are about 6800 training examples in the mixed scenario with a low job load for the cross-validation. In the medium job load set, there are about 212,000 training examples, whereas in the high job load set, about 557,000 training examples are generated. The limited training data result in better results for the PR2 than the DT and the RF approach in all low job load sets.

For every instance of the ten test instances, we calculate the deviation and consider it as a factor by which the CP makespan would have to be multiplied to match the objective function value of the DT schedule. The overall deviation then is determined using the arithmetic mean of all the results for that eightfold cross-validation for each approach. For example, a value of 1.151 would imply that the deviation of the corresponding approach from the CP result is 15.1% on average for all instances of that job load set. Furthermore, we compare the DT and RF results against those obtained with straightforward priority dispatching rules.

The configurations in these tables indicate the height of a DT for the DT approach and the number of trees/sample ratio/attribute ratio for the RF approach. The height of the trees was set at eight for the RF approach, because DTs with that configuration performed well in several test runs. To configure the other three parameters, number of trees, sample ratio, and setting of the maximum number of attributes, as listed in Sect. 3.5, we started with randomly chosen parameters of 50 trees, 0.5 sample ratio,

Table 8 Cross-validation results for the instance scenarios with makespan objective and unrelated parallel machines

Approach	Configuration	Instance scenario								
		Machine bottleneck Job load set			Crane bottleneck			Mixed		
		Low	Med	High	Low	Med	High	Low	Med	High
CP		1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
PR1		1.192	1.177	1.203	1.294	1.337	1.325	1.231	1.276	1.271
PR2		**1.151**	1.164	1.185	**1.233**	1.308	1.295	**1.181**	1.247	1.254
DT	4	1.387	1.163	**1.110**	1.299	1.378	1.158	1.350	1.336	1.230
	5	1.387	1.158	**1.110**	1.297	1.377	1.160	1.354	1.300	1.163
	8	1.182	**1.134**	1.127	1.280	1.264	1.184	1.257	1.252	1.173
	20	1.187	1.170	1.139	1.271	1.257	1.161	1.256	1.250	1.164
RF	50/0.75/0.5	1.176	1.146	1.156	1.238	1.215	**1.132**	1.224	1.194	**1.130**
	50/0.5/0.5	1.182	1.147	1.155	1.242	**1.213**	1.134	1.226	**1.189**	1.142

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

and 0.5 for the attributes ratio. The test runs showed that increasing or decreasing the amount of trees did not have a significantly positive effect on reducing the overall makespan. Therefore, only the sample and attribute ratio were varied, using step sizes of 0.1. The best performing configurations are presented in Table 8.

The results differ for the medium and high job load sets versus the low job load sets. In the latter case, the PR2 cannot keep up with the DT approach and RF approach: both perform better in every job load set. Furthermore, for most of the job load sets, the RF performs better than a single DT and succeeds in "generalising" the learned target concept better.

After the cross-validation, the next step is to choose a training set out of the job load sets and instance scenarios to build a generic DT. For this purpose, we chose the best performing DT (machine bottleneck scenario, high job load set, height 5) and RF (mixed scenario, high job load set, 50 trees, 0.75 sample ratio, 0.5 attribute ratio), then applied these trees to all the other sets. The trees were trained on a particular training set, so by testing on different sets, we can analyse their extrapolation capabilities. The results in Table 9 indicate that the generic DT and RF perform well in a medium and high job load set but, again, not as well in a low job load set, relative to PR1 and PR2.

5.2 Results for the total tardiness objective with identical parallel machines

The test procedure for the eightfold cross-validation test runs for the tardiness instances was analogous to that for the makespan instances. The results of the best performing DT for each scenario are shown in Table 10. Here, the benchmark result is the tardiness for each order within an instance achieved with the CP formulation. Again, the deviation is calculated and reported as a factor by which CP tardiness would have to be multiplied to

Table 9 Results for the generic DT and RF for the instance scenarios with makespan objective and unrelated parallel machines

Approach	Instance scenario	Job load set	Instance scenario								
			Machine bottleneck			Crane bottleneck			Mixed		
			Job load set								
			Low	Med	High	Low	Med	High	Low	Med	High
CP			1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
PR1			1.192	1.177	1.203	1.294	1.337	1.325	1.231	1.276	1.271
PR2			**1.151**	1.164	1.185	**1.233**	1.308	1.295	**1.181**	1.247	1.254
DT	MB	High	1.212	**1.131**	–	1.301	**1.201**	1.176	1.254	**1.184**	**1.144**
RF	Mixed	High	1.171	1.146	**1.154**	1.240	1.216	**1.133**	1.225	1.195	–

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

Table 10 Cross-validation results for the instance scenarios with total tardiness objective and identical parallel machines

Instance set	DT depth	Training examples	CP	EDD	ATC	DT
Bottleneck stages	8	All	1.000	1.693	2.066	1.229
		Critical				**1.191**
Restricted buffer slots	8	All	1.000	1.389	1.837	**1.096**
		Critical				1.249
Fully equipped shop floor	8	All	1.000	2.315	1.779	2.264
		Critical				**1.318**

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

match the objective function value of the DT or priority rule schedules. Both example selection approaches are listed in Table 10 (critical path approach and making use of all training examples—defined as "all"). We do not report lower bound information in this context, because neither the MIP nor the CP solver can provide sufficiently tight bounds to be used for a meaningful comparison.

The common priority rule performances can be undercut in these scenarios. The critical path approach actually leads to a tardiness reduction in two of three instance scenarios. In scenarios with a huge number of possible material movements, the critical path approach improves the overall result by skipping possibly irrelevant training examples. However, this outcome does not hold for the restricted buffer slot scenario, in which fewer possible material movements occur at the same time, so that more training examples seem even more in favour of the DT performance. With many more material movement options at a time compared with the makespan instances, the EDD rule can handle the total tardiness minimisation quite well, in that the processing times on parallel machines for the same job types are equal. Surprisingly, the simple concept

Table 11 Cross-validation results for the makespan instance scenarios with total tardiness objective and identical parallel machines

Instance set	CP	EDD	ATC	DT Critical	DT All	RF
MB high	1.000	1.258	1.538	1.109	**1.050**	1.061
MB med	1.000	2.158	2.428	**1.371**	1.448	1.434
MB low	1.000	1.446	1.499	**1.417**	1.570	1.511
CB high	1.000	1.651	1.927	1.532	**1.507**	1.543
CB med	1.000	1.881	2.148	1.675	1.663	**1.577**
CB low	1.000	2.165	2.169	2.082	1.773	**1.569**
Mixed high	1.000	1.430	1.674	1.377	1.399	**1.372**
Mixed med	1.000	2.306	2.167	1.557	2.514	**1.546**
Mixed low	1.000	1.805	1.841	1.765	1.884	**1.761**

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

of the EDD rule seems to work better than the more sophisticated ATC dispatching rule in this particular shop floor setting.

Thus, the CP formulation can be applied as a source for training examples in a total tardiness objective configuration. The decision trees and random forests built on the basis of the corresponding CP solutions yield competitive results and outperform the EDD and ATC dispatching rules.

5.3 Results for applying the total tardiness objective function to the makespan instance scenarios with identical and unrelated parallel machines

Noting that the DT approach for the total tardiness objective function worked well, we applied it to modified versions of the instances used for the makespan experiments, as introduced in Sect. 4.1 (see Tables 5 and 6). These modifications involved applying the same processing times on parallel machines and the critical path approach.

As shown in Table 11, building the DT using the critical path approach leads to competitive results in the machine bottleneck scenarios with medium and low job loads. The DT with all training examples achieves good performances in the high load instance sets. However, the RF approach can reduce tardiness in nearly all scenarios—yielding the best performance in 5 of 9 instance scenarios and good results in the remaining 4. Thus, it seems more advantageous to make use of the RF approach for instance scenario sets with the total tardiness objective.

In the next step, we calculated the due dates for the makespan instances with unrelated machines, as described in Sect. 4.2, to test the DT and RF performance on an instance scenario set with total tardiness as the objective and unrelated parallel machines. This machine setting, with differing processing times on parallel machines, complicated the determination of the due dates, as well as the calculation of the remaining work for chronological scheduling.

Table 12 Cross-validation results for the makespan instance scenarios with total tardiness objective and unrelated parallel machines

Instance set	CP	EDD	ATC	DT All, depth 8	RF
MB high	1.000	1.619	1.964	1.351	**1.099**
CB high	1.000	1.477	1.742	**1.255**	1.397
Mixed high	1.000	1.373	1.685	1.288	**1.134**

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

Table 13 Results for the generic trees

Objective	Machines	Training examples	EDD/PR1	ATC/PR2	DT All, depth 8	RF
Tardiness	Identical	All	1.667	1.388	1.385	**1.204**
Tardiness	Unrelated	All	1.706	1.984	**1.316**	1.490
Makespan	Unrelated	All	1.289	1.268	**1.126**	1.303

Bold values indicate best values or hint the differences between the approaches for better readability and interpretability

Due to the amount of jobs on the shop floor and the resulting number of decisions, we focus only on the high job load scenarios for these calculations. The results in Table 12 show that the RF outperforms the other approaches in 2 of 3 instance scenarios; for the CB, high scenario does the DT perform better. The performance of both the DT and RF approaches for this scenario and shop floor configuration thus are quite competitive.

5.4 Applying a generic DT and RF on different instance scenarios

Finally, we compared the varying approaches in terms of their ability to apply a generic DT and RF as a dispatching rule to other instance scenarios. Three different configuration–objective combinations were chosen exemplarily: total tardiness objective with identical parallel machines, total tardiness objective with unrelated parallel machines, and makespan objective with unrelated parallel machines. To test whether our machine learning approach can reduce the makespan or total tardiness, we selected 27 instances out of the pertinent MB and CB scenarios with high job load, as well as 26 instances of the mixed scenario with a high job load. Again, we conducted an eightfold cross-validation—analogously to the test runs described above.

In these scenario settings, the DT and RF yield better results than the corresponding dispatching rules, as indicated in Table 13. For the total tardiness objective with identical machines, the RF performs best. In scenarios with unrelated parallel machines, the DT outperforms the RF, in accordance with our observation that the DT and RF can be applied as a generic tree dispatching rule that performs well in instance scenarios, with various shop floor configurations and objectives.

Table 14 Binary classification results (before yes–no voting) for the DT with makespan objective and unrelated parallel machines, and the tardiness objective with identical parallel machines

Instance set	CB high				MB high				Mixed high			
Training examples	Critical		All		Critical		All		Critical		All	
DT depth	8 (%)	50 (%)	8 (%)	50 (%)	8 (%)	50 (%)	8 (%)	50 (%)	8 (%)	50 (%)	8 (%)	50 (%)
Makespan objective	84	87	88	89	82	81	85	82	85	88	85	88
Tardiness objective	88	89	81	79	83	84	74	72	87	87	79	78

5.5 Evaluation of the classification accuracy performance

To assess the classification performance of the generated decision trees on a somewhat lower level, we performed an accuracy analysis, both at the level of binary classifications and in terms of the proportion of job/machine pairs that have been chosen in exact correspondence with the CP solution. The goal is to gain deeper insights into the behaviour of the DT as a pure classifier in this specific problem setting. It also allows for a direct comparison of the DT with other popular classification approaches known from machine learning.

The binary classification problem considered in this context boils down to fixing the relative order between two job/machine pairs, as outlined in Sect. 3. More precisely, the target value equals 1 if the pair J1M1 is preferable to J2M2, and 0 otherwise. For the associated analysis, we split the set of 80 problem instances of each scenario (see Table 5) into a training and a test set, containing 50 and 30 instances, respectively. To retrieve a sufficiently large amount of pairwise comparisons, we only include the high load versions of those scenarios.

Table 14 summarises the accuracy results for the makespan and tardiness types of scenarios. The reported percentage values correspond to the proportion of times the binary value predicted by the DT coincides with the target values, as provided by the CP solution. We only used pairwise comparisons if at least one of the two options (J1M1 or J2M2) actually was chosen by the CP to be scheduled next at each decision point.

Prediction accuracies of notably above 80% can be achieved in most of the cases. We particularly note the strong influence of the example selection mechanism on classifier performance. For makespan instances with unrelated parallel machines (values "Diff" in column "PT"), it seems preferable to take into account all available training examples, whereas the results observed for the tardiness instances (identical machines) highly suggest selecting only those examples that involve at least one critical job (see Sect. 3.6).

To assess the relative performance of the DT approach, we compared it with various different alternative classifiers, as provided by the popular, Python-based machine learning toolkit `scikit-learn` (Version 0.20.2). For this purpose, we exported all the pairwise comparison data, including all features and the targets as text files, then re-

Table 15 `scikit-learn` binary classification results (makespan, identical parallel machines, all examples)

	CB high		MB high		Mixed high	
Classifier	Accuracy (%)	Fit time	Accuracy (%)	Fit time	Accuracy (%)	Fit time
DT (8)	90.14	00:00.4	84.30	00:00.3	88.87	00:00.3
RF (8)	90.76	00:01.7	85.35	00:00.6	89.10	00:01.1
GNB	86.60	00:00.1	82.76	00:00.1	86.17	00:00.1
KNN	86.05	00:53.4	81.44	00:09.7	86.40	00:50.9
SGD	90.46	00:51.4	86.45	00:30.5	88.56	00:49.3
SVC	90.18	24:45.4	86.09	10:12.5	87.96	33:34.4

Table 16 `scikit-learn` binary classification results (tardiness, unrelated parallel machines, critical examples)

	CB high		MB high		Mixed high	
Classifier	Accuracy (%)	Fit time	Accuracy (%)	Fit time	Accuracy (%)	Fit time
DT (8)	90.05	00:00.4	84.19	00:00.4	89.34	0:00:01
RF (8)	90.71	00:02.6	85.05	00:00.9	89.44	0:00:02
GNB	86.58	00:00.1	82.82	00:00.3	86.76	0:00:00
KNN	86.68	01:50.7	81.33	00:34.4	85.15	0:00:56
SGD	90.46	01:05.5	86.43	00:50.4	89.25	0:00:57
SVC	90.19	39:24.5	86.06	17:20.6	89.20	0:22:43

imported them in `scikit-learn`. The same split between training and test examples served to train and rate a set of classifiers. Tables 15 and 16 show the achieved scores on the test set. The applied classification approaches include a DT and a RF, each with a depth limit of 8, a Gaussian naive Bayes (GNB) classifier, a k-nearest-neighbour (KNN) approach, a purely linear classifier trained using a stochastic gradient descent (SGD) method, and finally a support vector classifier (SVC). All methods run with their default settings. In addition to the accuracy results, the table lists the time (in minutes) required to fit the respective model to the training data. It must be emphasised that for both types of instances, the DT and/or the RF almost consistently achieve the highest accuracies. They are tied for the lead in the CB high setting, outperform the other methods in the mixed high setting, and are only slightly worse than SVC and SGD in the MB high case. Note that for this kind of comparison, we used the DT and RF implementations available in `scikit-learn`. Slight deviations from the values reported in Table 14 may be traced back to implementation differences between the two employed machine learning frameworks. The main goal of the comparison, however, was to provide additional evidence that tree-based classifiers are particularly well-suited for this kind of classification problem.

The second stage of our analysis centres on the accuracy of the DT after yes–no voting has occurred. The central question in this context is: Assuming that we are given decision points during scheduling, such as sets of movable jobs and their

Table 17 DT Accuracy after yes–no voting (makespan objective, identical parallel machines)

Instance set	Examples	Depth	PR1 (%)	PR2 (%)	DT (%)	DT job only (%)	Avg. movable
CB high	Critical	8	3	4	32	47	52.3
		50			39	60	
	All	8	2	4	41	62	52.8
		50			41	62	
MB high	Critical	8	8	11	43	59	37.5
		50			48	66	
	All	8	8	11	49	66	37.5
		50			50	68	
Mixed high	Critical	8	4	5	33	49	46.7
		50			40	61	
	All	8	4	5	33	49	46.7
		50			40	61	

Table 18 Accuracy after yes–no voting (total tardiness objective, identical parallel machines)

Instance set	Examples	Depth	PR1 (%)	PR2 (%)	DT (%)	DT job only (%)	Avg. movable
CB high	Critical	8	6	4	42	57	52.2
		50			43	57	
	All	8	6	3	38	48	52.2
		50			41	52	
MB high	Critical	8	13	14	57	65	29.7
		50			57	65	
	All	8	13	10	53	59	29.7
		50			53	59	
Mixed high	Critical	8	9	9	48	59	42.4
		50			48	59	
	All	8	9	6	47	55	42.4
		50			48	58	

potential destination machines, in how many cases can the DT choose exactly the same job *and* the same machine as the CP solver? The results in Tables 17 and 18 suggest disappointing percentages at first glance. However, when relating these values to the average number of movable job/machine pairs (column "Avg. movable") per decision point, the numbers support another, more positive view, which is reinforced by the poor performance of the priority rules that serve as the "baseline" results here. Accuracy also increases considerably if only the job part of the decision has to be matched exactly (column "DT job only"). This result offers a further indication of the relative importance of the machine assignment as an inherent component of every decision taken during scheduling processes.

6 Discussion and conclusion

We have introduced a machine learning approach for generating a tree-based dispatching rule making use of training examples taken from CP solutions. The underlying abstract model is based on a practical application setting. Our approach can deal with a greater degree of complexity, such as varying processing times per machine, varying number of parallel machines per stage, job types per order, number of processing stages, setup times, and number of jobs. We add further complexity by considering a transport resource with limited capacity. Existing contributions each cover only subsets of these aspects. We also analysed the DT and RF approaches in relation to two different objective functions: makespan and total tardiness.

Both the DT and RF, trained and tested on a specific instance scenario, undercut the makespan results of the priority rules in the medium and high job load scenarios, as confirmed by the cross-validation procedure. Due to the decreased number of training examples, this outcome does not appear possible for the low job load scenarios. However, the gaps relative to the priority rules are very small in these cases. When applying a generic DT and RF to the low, medium, and high job load scenarios, similar effects occur. The generic trees yield competitive results in the medium and high load scenarios. Furthermore, all approaches also perform well for the total tardiness objective with various instance scenarios and shop floor configurations.

We conclude that the high quality feasible solutions obtained with the help of CP provide a good basis for generating training examples. Furthermore, using the selected attributes in combination with the job-machine decision, and the yes–no voting, a well performing DT and RF can be generated. The computational results provide strong evidence that for each of the two objectives, the respective DT and RF approach can be used to generate a dispatching rule for various shop floor settings.

It has to be stated that our proposed approach might not always lead to one single, generic DT or RF that is able to outperform other approaches in all the instance scenarios and objectives. But the main advantage of our machine learning approach is the combination of an easy-to-implement, easy-to-learn, easy-to-adapt technique, with an interpretable model. This is especially interesting for decision makers, production schedulers, or other employees in the manufacturing industry, as the tree can be interpreted simply by reviewing its branches and leaves—in contrast to other classifiers introduced in Sect. 5.5.

Future research may address, for example, the possibility to include machine breakdowns as an extension of our approach to deal with a dynamic environment. Depending on the nature of such breakdowns, this could be accomplished either by scheduling preventive maintenance intervals or by a stochastic modelling approach. Another line of research might seek to develop a genetic programming or hyper-heuristic approach as even more complex competitors to our DT and RF implementation.

Acknowledgements Open access funding provided by University of Vienna. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged. We also thank Michael Stummer and Boris Serdar of Syngroup for demonstrating and explaining their real-world physical system, in combination with the Industry 4.0 application, in detail. We also appreciate their cooperation and willingness to answer all of our questions regarding this system.

References

Balas E (1969) Machine sequencing via disjunctive graphs: an implicit enumeration algorithm. Oper Res 17(6):941–957

Blockeel H, Struyf J (2003) Efficient algorithms for decision tree cross-validation. J Mach Learn Res 3:621–650

Braune R, Zäpfel G, Affenzeller M (2013) Enhancing local search algorithms for job shops with min-sum objectives by approximate move evaluation. J Sched 16(5):495–518

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Brucker P, Heitmann S, Hurink J (2003) Flow-shop problems with intermediate buffers. OR Spectr 25(4):549–574

Doh HH, Yu JM, Kwon YJ, Shin JH, Kim HW, Nam SH, Lee DH (2014) Decision tree based scheduling for flexible job shops with multiple process plans. Int J Mech Aerosp Ind Mechatron Manuf Eng 8(3):621–627

Hall NG, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. Oper Res 44(3):510–525

Ho TK (1995) Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition, vol 1, pp 278–282

Kurz ME, Askin RG (2004) Scheduling flexible flow lines with sequence-dependent setup times. Eur J Oper Res 159(1):66–82

Leisten R (1990) Flowshop sequencing problems with limited buffer storage. Int J Prod Res 28(11):2085–2100

Li X, Olafsson S (2005) Discovering dispatching rules using data mining. J Sched 8(6):515–527

Mascis A, Pacciarelli D (2002) Job-shop scheduling with blocking and no-wait constraints. Eur J Oper Res 143(3):498–517

Olafsson S, Li X (2010) Learning effective new single machine dispatching rules from optimal scheduling data. Int J Prod Econ 128(1):118–126

Panwalkar SS, Iskander W (1977) A survey of scheduling rules. Oper Res 25(1):45–61

Pinedo M (2002) Scheduling: theory, algorithms, and systems, 4th edn. Prentice Hall, Englewood Cliffs

Quinlan JR (1986) Induction of decision trees. Mach Learn 1(1):81–106

Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco

Ruiz R, Maroto C, Alcaraz J (2005) Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. Eur J Oper Res 165(1):34–54

Shahzad A, Mebarki N (2012) Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. Eng Appl Artif Intell 25(6):1173–1181

Vepsalainen APJ, Morton TE (1987) Priority rules for job shops with weighted tardiness costs. Manag Sci 33(8):1035–1047

Wuest T, Weimer D, Irgens C, Thoben KD (2016) Machine learning in manufacturing: advantages, challenges, and applications. Prod Manuf Res 4(1):23–45

Yang K, Trewn J (2004) Multivariate statistical methods in quality management. McGraw-Hill, New York

Affiliations

Frank Benda[1,2] · Roland Braune[3] · Karl F. Doerner[3,4] · Richard F. Hartl[3]

1 Christian Doppler Laboratory for Efficient Intermodal Transport Operations, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

2 Department of Business Administration, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

3 Department of Business Decisions and Analytics, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

4 Research Platform Data Science @ Uni Vienna, Vienna, Austria