universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation / Title of the Doctoral Thesis

"Machine Learning Approaches for Flexible Shop Scheduling Problems"

verfasst von / submitted by

Frank Benda

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doctor of Philosophy (PhD)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 794 370 403
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Logistics and Operations Management
Betreut von / Supervisor:	O. Univ.-Prof. Dipl.-Ing. Dr. Richard F. Hartl

Acknowledgements

First of all I want to thank my supervisor Prof. Richard F. Hartl for granting me the predoctoral position and for providing me with valuable scientific advises. I am really thankful for his patience that was needed to develop the research questions and solution approaches. Furthermore, I especially want to thank Prof. Karl F. Dörner and DI Dr. Roland Braune for fruitful discussions and their support regarding the research itself throughout the whole process of writing this thesis. I really appreciate the possibility to consult them for any organizational and scientific issue. Another important part within the last few years was the perfectly orchestrated management regarding all the organizational and administrative related issues of our secretaries Carina Artner-Konecny, Christine Mauric, and Alexandra Ederer. Without my colleagues the working time would not have been the same. So thank you for being part of a great time and those interesting discussions, not only about scientific topics. A special thanks also to Karl, both Daniels, and Markus, who joined me nearly every day for lunch at "Rebhuhn" with pleasant talks. All of you played an important role in the process of completing the thesis.
In addition, 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. I also want to thank Michael Stummer and Boris Serdar of Syngroup for demonstrating and explaining their Industry 4.0 application, in detail. I also appreciate their cooperation and willingness to answer all of my questions regarding this system. For the third part of the thesis, I also want to thank the EBG MedAustron GmbH for providing insight into their appointment scheduling problem.
Last but definitely not least, the biggest thank you goes to my precious friends, parents, Anneliese and Franz, and my sister Kathrin, who supported me through all the times of my life with their optimism, advice, by lending an ear whenever needed, and in so many different other ways.

Contents

List of Figures

List of Tables

List of abbreviations

ADD	Absolute due date
AI	Artificial intelligence
APT	Average processing time of jobs in queue of machine
ATC	Apparent tardiness cost
AWJ	Average waiting time of job
AWM	Average waiting time on machine
B	Blocking in front of processing slot
BS	Bottleneck stages
C	Crane position
CB	Crane blocking
CDDR	Critical due date ratio
CP	Constraint programming
CPS	Cyber-physical system
Cst	Constant
DT	Decision tree
EDD	Earliest due date
FE	Fully equipped shop floor
GA	Genetic algorithm
GNB	Gaussian Naive Bayes
GOR	Gesellschaft für Operations Research e.V.
GP	Genetic programming

ID3 Iterative Dichotomiser 3

IJCAI International Joint Conference on AI

IWOLIA International Workshop on Optimization in Logistics and Industrial Applications

KNN K-nearest-neighbour

KRR Kernel ridge regression

LB Lower bound

LINAC Linear particle accelerator

LR Lasso regression

LWKR Least work remaining rule

MB Machine blocking

MI Total machine idle time

MIP Mixed-integer programming

MLP Multilayer perceptron

MN Machine workload on next machine

MO Total machine occupation

MR Machine workload on current machine

MSE Mean squared error

MWKR Most work remaining rule

NP Next processing time

NT Job available on next machine

OL Operations left of job

OLA Workshop on Optimization and Learning

OLS Ordinary least squares

OS Occupied processing and buffer slots on next machine

PR1 Priority rule 1

PR2 Priority rule 2

RB Restricted buffer slots

RDD Time left until due date is reached

RF Random forest

RIT Recorded total machine idle time

RJV Recorded total number of jobs on machine

RMS Recorded makespan of job

RMW Recorded total waiting time of jobs on machine

RR Ridge regression

RW Remaining work of job

RWL Recorded total machine workload

RWT Recorded total waiting time of job on all machines

SGD Stochastic gradient descent

SL Job slack time

SPT Shortest processing time

ST Setup time on next machine

SVC Support vector classifier

SVM Support vector machine

SVR Support vector regression

T Transport time current machine to next machine

UB Upper bound

VSC3 Vienna Scientific Cluster 3

Chapter 1

Introduction

This thesis discusses research questions in the field of scheduling. For the first and second part, problem statements regarding flexible shop scheduling environments were covered. The considered problems were conducted within the framework of the research group "Industry 4.0" of the University of Vienna. The business consultancy "Syngroup", located in Vienna, was gained as a partner in the industrial sector "smart factories" and Industry 4.0. Within the cooperation, machine learning methods for their cyber-physical system (CPS) in a smart production environment were developed for the first and second publication. The CPS is used to demonstrate a decision support system of a typical shop floor configuration. Here, human planners can try to compete against basic priority rule approaches. The goal of the two papers in this framework consisted of developing more sophisticated priority dispatching rules based on the CPS. In order to achieve the goal, the level of complexity was extended. Therefore, the number of parallel machines, processing stages, jobs, and intermediate buffers in front of every machine were modified and transformed into more flexible shop floor configurations. With these settings, different instance scenarios were defined and for each of them random instances generated and split into a training and a test set. The machine learning approaches were trained on the training set and then, within experiments, applied to the test set. The solution methods contain a decision tree (DT), random forest (RF), and a genetic programming (GP) approach.

Originally, it was planned to conduct further research regarding machine breakdowns based on data provided by a second industrial partner for the third part of the thesis. Unfortunately, the company was not able to hand over data of a certain amount and quality. Hence, the topic was switched to appointment scheduling, as the data of the MedAustron ion beam center located in Wiener Neustadt was already existent. However, this part also contains an important field in relation to Industry 4.0 that gains more and more importance: Data science. Typical regression analysis methods used in data science are applied to replace complex sample average approximation methods for waiting time estimation in the field of radiotherapy appointment scheduling. For chronically ill patients with more than two consecutive treatments within the cancer treatment process, waiting time has to be estimated as precisely as possible to increase the patients' level of satisfaction. So far, two different approaches to estimate that waiting time in a stochastic environment were introduced by Vogl et al., 2020. Here, a genetic algorithm, an iterated local search method, and a combination of both of them are applied to

estimate the average waiting time. The goal of the third paper was to develop a replacement approach in form of a regression analysis with multiple regressors instead of a single one to estimate the average waiting time and, hence, significantly reduce the level of complexity for the estimation procedure. This waiting time estimation process might be integrated within the given GA framework by Vogl et al. (2020), but also in different stochastic scheduling problems.

The full list of publications and submissions for this thesis is listed below:

- Benda, F., Braune, R., Doerner, K. F., Hartl, R. F., 2019. A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. OR Spectrum 41 (4), pp. 871–893.

- Benda, F., Braune, R., Doerner, K. F., Hartl, R. F., 2020. Genetic Programming Dispatching Rules Learning Approaches for Flexible Shop Scheduling Problems. Submitted to the "Twenty-first International Working Seminar on Production Economics".

- Benda, F., Braune, R., 2020. Learning Approach to replace Sample Average Approximation for Appointment Scheduling. Technical report.

1.1 Research Topics and Methodology

The goal for the first paper was to develop priority dispatching rules in form of a DT based on the aforementioned CPS of the industrial partner. The main reason for considering a machine learning approach was the easy-to-apply character in terms of being able to provide competitive results for decision makers in the private sector within a split second. Once the DT and RF are trained, they can be applied to all production environments of the same configuration yielding competitive results, in contrast to other optimization methods that need to solve each problem individually. The typical decision in a shop floor setting is assigning the jobs that need to be processed to machines and sequence those jobs on each of the machines. Besides, the shop floor configuration in this thesis consists of a restricted transport resource in form of a crane moving a single job at a time from one machine to the next one, sequence-dependent setup times due to different job types, and blocking if the intermediate buffer in front of a machine is fully occupied and therefore the preceding stage is blocked. A unified representation of the DT containing the job assignment and machine sequencing in a single tree was developed. The RF approach was then implemented to counteract possible overfitting. For the experiments, instance scenarios with widely differing shop floor settings (e.g., differing number of parallel machines and stages) based on the real-world CPS were created to test the ability of the DT and RF approach to

generate decision trees that also perform well as a generic priority rule on different shop floor settings. For both approaches, high quality solutions of a constrained programming formulation for the different instance scenarios were used as training material. The goal was to minimize the overall makespan in the first and the overall tardiness in the second experiment run.

As the results of the first paper turned out to be promising, the second paper contained a slightly different learning approach in form of a GP. In the field of smart production, artificial intelligence (AI) is the trending topic in literature, but also regarding manufacturing processes in the private sector. Hence, well-established methods like machine learning as a subdomain of AI gain a revival and are widely used to develop complex decision support systems. GP, in turn, is a method of machine learning and was used similar to the DT and RF approach to generate a dispatching rule for flexible shop problems. However, the approach was extended by comparing the single-tree representation containing the job assignment and machine sequencing in one step with a multi-tree representation in which this decision was split into two separate decisions. The aspect of learning was also considered in terms of an iterative improvement of the resulting schedules. First, these approaches were tested on well-known benchmark instance sets and were then applied to real-world adapted instance sets, again based on the CPS of the industrial partner.

In contrast, for the third part of the thesis instead of production process data, patient data for appointment scheduling was used for methods in the field of data science. Here, the goal was to replace the complex procedure of waiting time estimation with simple linear regression with a single regressor and a sample average approximation with regression analysis methods with multiple regressors. Furthermore, an artificial neural network (as another popular method within the framework of data science for Industry 4.0 applications) was also implemented to predict the average patient waiting time. It was also intended to demonstrate, that making use of a feature set containing a variety of features and a feature selection procedure achieve better accuracy estimation results than a single feature with highest positive or negative correlation to the regressand. Therefore, 22 new features were introduced to cover important aspects regarding the scheduling process and waiting time occurrences. Three different feature sets were defined containing all features, the highest correlating feature, and a set of features determined with a feature selection procedure. The experiments were conducted with regression analysis methods with each of the three feature sets and applied to two instance sets containing randomized instances with different number of patients (50, 70, or 100). The patient mix distributions (e.g., treatment time for each patient) for these instances were either fixed or randomly distributed. The implementation of the regression analysis consists of support vector regression, kernel ridge regression, ordinary least squares, ridge regression, and lasso regression. In addition,

the aforementioned feedforward artificial neural network in form of a multilayer perceptron was implemented.

1.2 Structure

The cumulative thesis consists of three papers, as listed above. The first paper "A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking" was presented at the meeting of the working group "Complex Systems" of the "Gesellschaft für Operations Research e.V." (GOR) in Vienna 2017, at the "Workshop on Optimization and Learning" (OLA) in Alicante 2018, and at the "International Workshop on Optimization in Logistics and Industrial Applications" (IWOLIA) in Karlsruhe 2018. The GP approach was already part of given talks at the "International Joint Conference on AI" (IJCAI) in Stockholm 2019, at the OLA in Bangkok 2019, and at the "Twenty-first International Working Seminar on Production Economics" in Innsbruck 2020.

The underlying problems were motivated by the industrial partner, as mentioned above, for the first and second paper. The third paper was motivated by already existing data provided by MedAustron. Based on the basic problems at hand, the approaches were further developed through intense discussions with Prof. Richard F. Hartl, Prof. Karl F. Doerner, and Dr. Roland Braune. The co-authors' contribution took place in form of advises regarding designing the solution methods, analyzing results, and hints for the implementation procedure and the structure of the publications. My work contained the set up of the problem statements, development of the solution approaches, the implementation, interpretation of results, and writing the publications.

This thesis is structured as follows: Chapter 2 contains the first publication and describes the solution approach for the DT and RF method. Then, in Chapter 3 the GP is used as an alternative approach for the problem setting based on the industrial partner's CPS making use of the HeuristicLab[1]. The working paper on radiotherapy appointment scheduling with estimating the patients' waiting time through regression analysis and an artificial neural network is part of Chapter 4. Finally, the thesis is concluded in Chapter 5.

Note, that the form of the papers presented in this thesis might differ from the published ones as these only exist in PDF format. Hence, especially the format of the tables is different to the published papers. Furthermore, to harmonize the overall layout, abstracts and few other minor changes were applied. Further changes between the chapters are based on different styles regarding citation and journal requirements to which the papers were submitted to.

[1] `https://dev.heuristiclab.com/`

Chapter 2

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

Published in: Springer. *OR Spectrum* (2019). A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. F. Benda, R. Braune, K. F. Doerner, R. F. Hartl.
https://doi.org/10.1007/s00291-019-00567-8

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 random forest (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 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.

2.1 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, as stated by 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: Sect. 2.2 contains the problem statement in detail. In Sect. 2.3, 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. 2.4. The computational results for the DT and RF approach are summarised and discussed in Sect. 2.5. Finally, the discussion in Sect. 2.6 briefly reviews the contributions and offers suggestions for future research.

2.2 Problem Statement

We implemented an abstract model based on the basic shop floor configuration of the industrial partner, which is depicted exemplarily in Figure 2.1 for an exemplary configuration of the shop floor. Three different types 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.

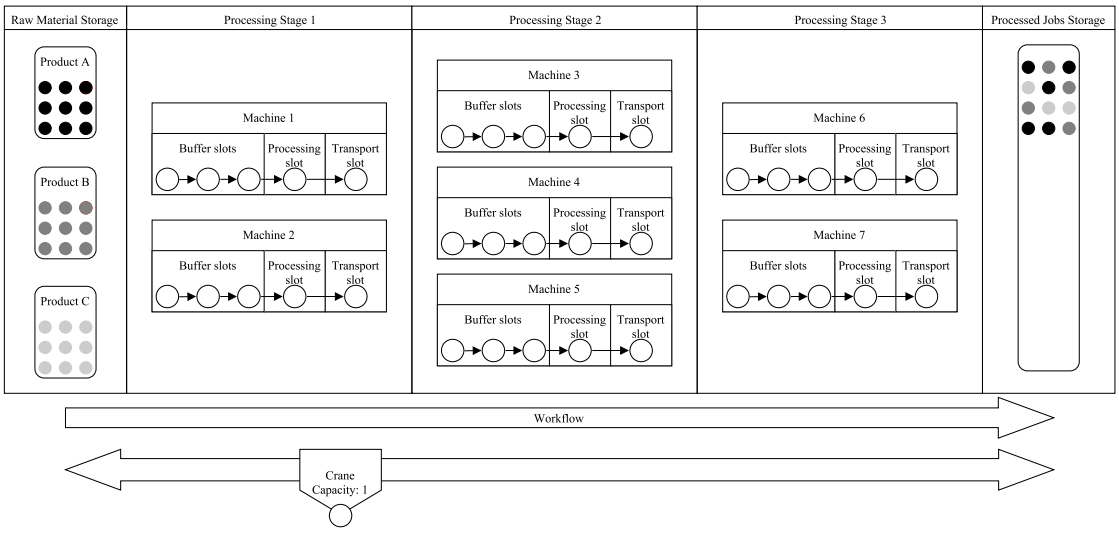

Figure 2.1: Exemplary configuration of the shop floor

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.

2.3 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. 2.2. 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

Table 2.1: List of attributes

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

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.

2.3.1 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 2.1. 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.

2.3.2 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. The schedules created by the CP solver are read in by a deterministic shop floor simulator, as described in Sect. 2.3.3. The training data is 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,

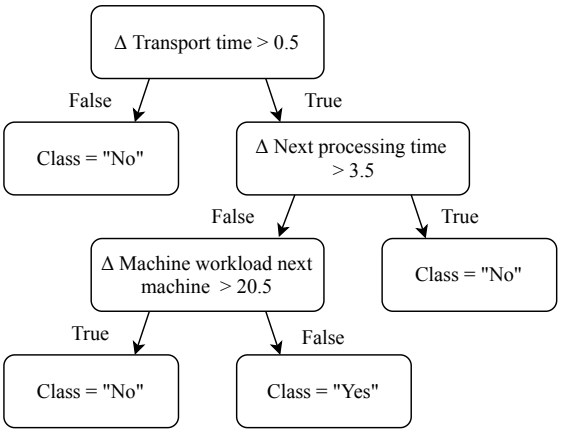

Figure 2.2: Decision tree example

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 (see Quinlan (1993)).

In Figure 2.2 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"), as proposed by 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.

2.3.3 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 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 short-hand 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 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 Figure 2.3. 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 2.2. 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

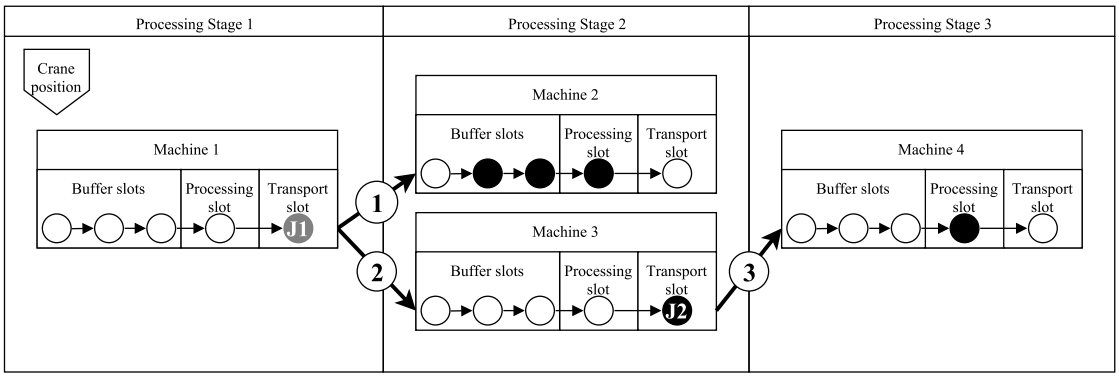

Figure 2.3: Example: Status quo on shop floor with possible job movements

Table 2.2: 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

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 2.2). 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. 2.3.2.

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.

2.3.4 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. 2.3.3 illustrates the transformation.

The status quo on the shop floor is the same as in Figure 2.3. 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

Table 2.3: 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

Table 2.4: 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

the DT is the classification of each pairwise comparison of all possible material movements, as listed in Table 2.3.

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, as demonstrated by Li and Olafsson (2005). It counts the amount of "Yes" and "No" responses for each possibility by transforming the classes in Table 2.3 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 2.4. 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.

2.3.5 Counteracting Overfitting using a RF Approach

To prevent the DT from overfitting the data, one effective technique is pruning, as introduced by 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 easy-to-use approach for this purpose. Ho (1995)

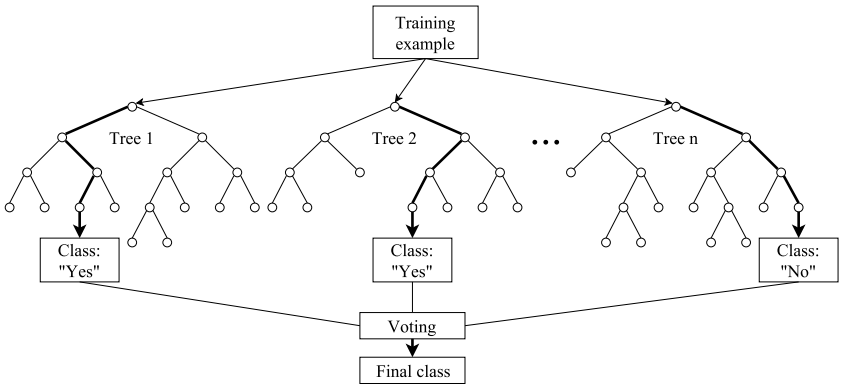

Figure 2.4: Simplified RF example

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, as listed by 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 2.4 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. 2.4.

2.3.6 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, 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, 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, as shown by 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 (see 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.

2.4 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.

2.4.1 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 2.5. The machine blocking (MB) 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

Table 2.5: 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

Table 2.6: 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

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 2.6. Note that the given number ranges in Tables 2.5 and 2.6 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 2.5 and 2.6. 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 as shown by 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 (see Panwalkar and Iskander (1977)). Priority rule 1 (PR1), extended to take setup time into consideration, selects the job with the shortest processing time

Table 2.7: 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

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 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.

2.4.2 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 2.7.

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 (e.g., 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, as introduced by Vepsalainen and Morton (1987). For both rules, the workload of possible next machines is also considered.

2.5 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 Optimiser 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. In this section, we present the best performing DT and RF configurations and discuss their results.

2.5.1 Results for the Makespan Objective with Unrelated Parallel Machines

The results for the makespan objective are listed in Table 2.8. 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 6,800 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 results in better results for the PR2 than the DT and the RF approach in all low job load sets.

Table 2.8: Cross-validation results for the instance scenarios with makespan objective and unrelated parallel machines

Approach	Configuration	Instance scenario: Machine bottleneck, Job load set: Low	Machine bottleneck, Med	Machine bottleneck, High	Crane bottleneck, Low	Crane bottleneck, Med	Crane bottleneck, High	Mixed, Low	Mixed, Med	Mixed, 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

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. 2.3.5, we started with randomly chosen parameters of 50 trees, 0.5 sample ratio, 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 2.8.

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.

Table 2.9: 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 Job load set Low	Machine bottleneck Med	Machine bottleneck High	Crane bottleneck Low	Crane bottleneck Med	Crane bottleneck High	Mixed Low	Mixed Med	Mixed 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

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 2.9 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.

2.5.2 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 2.10. 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 match the objective function value of the DT or priority rule schedules. Both example selection approaches are listed in Table 2.10 (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

Table 2.10: 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

slot scenario, in which fewer possible material movements occur at the same time, so that more training examples seem even more in favor 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 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.

2.5.3 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. 2.4.1 (see Tables 2.5 and 2.6). These modifications involved applying the same processing times on parallel machines and the critical path approach.

As shown in Table 2.11, 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.

Table 2.11: 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

Table 2.12: 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

In the next step, we calculated the due dates for the makespan instances with unrelated machines, as described in Sect. 2.4.2, 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.

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 2.12 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.

Table 2.13: Results for the generic trees

Objective	Machines	Training examples	EDD or PR1	ATC or 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

2.5.4 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 2.13. 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.

2.5.5 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. 2.3. More

Table 2.14: Binary classification results (before yes-no voting) for the DT with makespan objective and unrelated parallel machines ("Accuracy 1"), and the tardiness objective with identical parallel machines ("Accuracy 2")

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
Accuracy 1	84%	87%	88%	89%	82%	81%	85%	82%	85%	88%	85%	88%
Accuracy 2	88%	89%	81%	79%	83%	84%	74%	72%	87%	87%	79%	78%

Table 2.15: `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

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 2.5) 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 2.14 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. 2.3.6).

Table 2.16: `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

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-imported them in `scikit-learn`. The same split between training and test examples served to train and rate a set of classifiers. Tables 2.15 and 2.16 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 2.14 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 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 Table 2.17 and 2.18 suggest disappointing percentages at first glance. However, when relating

Table 2.17: 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 2.18: 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%	

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.

2.6 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. 2.5.5.

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.

Chapter 3

Genetic Programming Dispatching Rules Learning Approaches for Flexible Shop Scheduling Problems

Submitted to the "Twenty-first International Working Seminar on Production Economics", Jan. 17, 2020.
Genetic Programming Dispatching Rules Learning Approaches for Flexible Shop Scheduling Problems.
F. Benda, R. Braune, K. F. Doerner, R. F. Hartl.

Abstract This paper deals with a Genetic Programming (GP) approach for solving flexible shop scheduling problems. The aim of the adopted approach is to generate a priority rule in form of an expression tree for dispatching jobs. In a list-scheduling algorithm the available jobs can thus be ranked using a tree-based priority rule generated with the GP.

To achieve the goal of reducing the overall makespan, we consider job assignment and machine sequencing decisions simultaneously in a single tree representation and compare this single tree with a multi-tree approach. The single tree contains both terminal sets for the job assignment and the machine sequencing, whereas for the multi-tree approach the terminal set is split into job- and machine related terminals. This results in two parallel GP populations, used to firstly decide which job to choose and secondly, which machine it should be assigned to. We also apply a feature selection approach to determine the importance of terminals regarding their contribution to improve the objective function value. Furthermore, an iterative dispatching rule is generated with terminals that store recorded information of past schedules to improve the recent one.

The goal of the computational evaluation was a performance analysis of the single and multi-tree approach. To compare the performances of the proposed approaches, both were tested on benchmark instances,

such as those of Brandimarte and Lawrence. The GP approaches were then applied to a special case based on a real-world scenario. For all the experiments the GP approaches yielded competitive results by achieving good performance in terms of makespan minimization.

3.1 Introduction

Due to broad varieties of production processes with differing shop floor configurations in manufacturing companies, flexible flow and job shop problems play a major role in this context. Both are defined as multi stage shops with parallel unrelated or identical machines. Constructive heuristics, like list scheduling algorithms (e.g., Mainieri and Ronconi (2013)), are widely used methods for generating initial solutions for metaheuristics as advanced approaches. For problems in real-world settings, the constructive heuristics are able to deliver reasonable results within a relatively short time, as fast planning and decision processes are of vital importance in the manufacturing environment. For these decision making processes, priority rules are used to rank the possibilities regarding their importance. The aforementioned list scheduling algorithms are typically based on priority rules. Jobs that need to be processed on different machines have to be assigned to different parallel machines that conduct the same production process. Therefore, the first decision within these shop problems is to assign the jobs to their possible machines. The second decision contains the sequencing on the machine in terms of the order the assigned jobs shall be processed on the regarding machine.

There are plenty of ways to generate such priority rules, also called dispatching rules in the context of shop problems. These rules can be created manually by a human planner. The resulting production plans represent solutions that are feasible and quick to compute, but not competitive compared to approaches like metaheuristics. Therefore, a more systematic and automated synthesis of such rules would be desirable, for example machine learning as a method to create such rules. Due to the ability of collecting huge amounts of data during the production process in smart factory environments, machine learning approaches gain a revival for developing decision support systems. Machine learning as a subdomain of Artificial Intelligence (AI), is applied in a variety of production process optimization methods. Monostori (2003), for example, utilizes machine learning techniques in the setting of intelligent manufacturing systems to manage unforeseen problems during the manufacturing process. Wuest et al. (2016) state, that machine learning approaches are able to support the handling of big data in the production process with increasing complexity regarding manufacturing procedures. Hence, machine learning approaches can be implemented in manufacturing environments for different purposes, such as scheduling in our case. Genetic Programming (GP) as an

AI method can be used to solve machine learning problems. In our case, we seek to "learn" a dispatching rule by iteratively improving its corresponding expression tree by genetic evolution. The performance of a particular tree is not measured in terms of accuracy, but rather based on the achieved improvement in the objective function value. GP belongs to the group of evolutionary algorithms by following the approach "survival of the fittest". Here, a population of so called individuals undergoes a series of iterative transformations, using specialized crossover and mutation operators, originally described by Koza (1992). Ideally, the genetic material of high-quality individuals is effectively preserved and recombined to create even better "child" individuals in form of expression trees ("programs" in the case of GP). An expression tree can finally be translated into a dispatching rule that makes use of status quo information of the shop floor to create a priority ranking of possible decisions, such as the job assignment or the machine sequencing.

Successful application scenarios for GP in the field of scheduling and production planning include a variety of different machine environments (e.g., Jakobović et al. (2007), Tay and Ho (2008)), as well as lot sizing problems (e.g., Hein et al. (2018)). The main advantage of the resulting GP dispatching rule are the aspects of an easy-to-implement and easy-to-adapt dispatching rule. Furthermore, the GP approach generates easy-to-apply dispatching rules in the sense that once the GP dispatching rule for a certain problem setting has been trained, it can be used as dispatching rule for every other instance of a similar problem setting yielding results of a competitive quality in a split second. In contrast to that, "classical" optimization methods like metaheuristics need to solve every single instance from scratch, usually without exploiting "knowledge" from already seen instances.

As already mentioned above, flexible flow and job shop settings are quite common in industrial practice. In fact, the motivation of our approach is based on a cyber-physical system (CPS) of one of our industrial partners, a business consultancy in Vienna. It can be stated as a hybrid flow shop scheduling problem with alternative resources, sequence-dependent setup times, transport resource, limited intermediate buffers, and blocking. The idea of the CPS is simulating a shop floor on which human planners can compete against priority rule approaches. Their basic simulator was set up with the purpose of demonstrating the ability of simple priority rule approaches to perform better than the human planner. Therefore, we intend to evolve expression trees with a GP approach that yield better results in terms of makespan minimization than those priority rules.

Pickardt et al. (2013) made use of the GP approach for a work centers setting. Here, the GP approach generates dispatching rules with basic job attributes, whereas assigning these rules to the work centers is conducted with an evolutionary algorithm. We, in contrast, introduce a dispatching rule in form of a GP single-tree that combines the job assignment and machine sequencing in one step. This single-tree representation is compared to a multi-tree representation, as proposed by Zhang et al. (2018). Adding an additional aspect of learning, an iterative approach

was developed and compared to the basic results of the single- and multi-tree representation. The GP approach is suitable to build expression trees containing information about previous schedules in terms of an iterative dispatching rule, as introduced by Nguyen et al. (2013) and further analyzed by Froehlich et al. (2019) in the standard job shop context. Finally, a feature selection process was used to classify the terminals into important and less important terminals in terms of their ability to reduce the overall makespan.

In the experimental part of this paper, we first address the well-established benchmark instance sets of Brandimarte (1993), Lawrence (1984) for standard flexible flow and job shop scheduling, and, based on the latter, self-generated instances of larger size. Furthermore, these GP approaches were then applied to the special case in terms of the aforementioned CPS-related real-world problem. With the GP approach at hand, we intend to demonstrate that an expression tree in a single- or multi-tree representation yields competitive results regarding the instance sets based on the real-world CPS. The system represents a typical shop floor configuration that can be defined as hybrid flow shop scheduling problem. In Benda et al. (2019), a priority rule was evolved in the form of a decision tree (DT) and random forest (RF), trained on constraint programming (CP) formulation solutions, for dispatching jobs. The instances used for this purpose are based on a problem setting of a CPS simulator of our industrial partner.

Another contribution of this paper is the degree of complexity regarding the GP terminal selection for the real-world based instance sets. These terminal sets contain the aspects of a restricted transport resource, setup times, and blocking situations. To the best of our knowledge, this complexity level has not been dealt with in the literature until now.

The content of this paper is organized as follows: The problem statement is described in Section 3.2. Section 3.3 contains the GP single- and multi-tree algorithm description as well as the explanation of the iterative dispatching rule and the feature selection approach. Next, the generation of expression trees and application to the instance sets is described in Section 3.4 in detail. Section 3.5 contains the discussion and summary of the computational results. Finally, the contributions of this paper and suggestions for future research are briefly reviewed in Section 3.6.

3.2 Problem Statement

As outlined in the introduction, two different problem statements are presented. First, we made use of a flexible job shop setting in terms of the Brandimarte (1993) and Lawrence (1984) benchmark instance sets. And second, we considered the CPS simulator for a real-world-based instance set, as introduced in Benda et al. (2019),

that can be described as a hybrid flow shop scheduling problem. The objective for both sets is to minimize the total makespan.

For the benchmark instance sets the jobs have to be processed on several unrelated (Brandimarte instances) or identical (Lawrence instances) parallel machines. The number of operations for a single job are varying. Furthermore, the flexibility in terms of possible parallel machines conducting the same processing step for a jobs' operation differs for these instances. For each operation a certain amount of possible processing machines is determined out of all machines resulting in the possibility that jobs can be scheduled on the same machine several times for different operations.

In contrast to the benchmark instance sets, the aspects of transportation, setup times, and blocking, had to be considered for the real-world problem setting. The exemplary shop floor configuration of this set is depicted in Fig. 3.1. In this case three different job types exist with same processing times for each type on each machine. Each type has to be processed in a certain stage sequence, starting at the raw material storage in a forward processing procedure, not necessarily being processed on each stage. The machines consist of buffer, processing, and transport slots. Following that slot order, a job can be transported by a crane with limited capacity of one job at any time, to the next processing stage whenever it is situated in a transport slot. Overtaking on a machine is not permitted. Though the parallel machines conduct the same processing step, they might be unrelated in terms of differing processing times for the same job type. Furthermore, fully loaded machines in a subsequent stage might cause blocking also in the preceding stage. Finally, sequence-dependent setup times might occur for two consecutive jobs of different job types on the same machine. The setup activity can already be started before the next job has arrived on the respectively machine.

3.3 Solution Method

For pursuing the goal of reducing the overall makespan for the instance sets, the GP is used to generate a dispatching rule that takes into consideration every aspect of the shop floor configurations, e.g. machine workloads, transport and sequence-dependent setup times. The configuration of these GP terminals and functions, GP parameters, and the GP algorithm in general will be discussed in the following sections.

3.3.1 GP as a Method for Evolving Priority Dispatching Rules

As far as the evolutionary process is concerned, GP is thus very similar to classic Genetic Algorithms (GA). However, the structural representation of the individuals

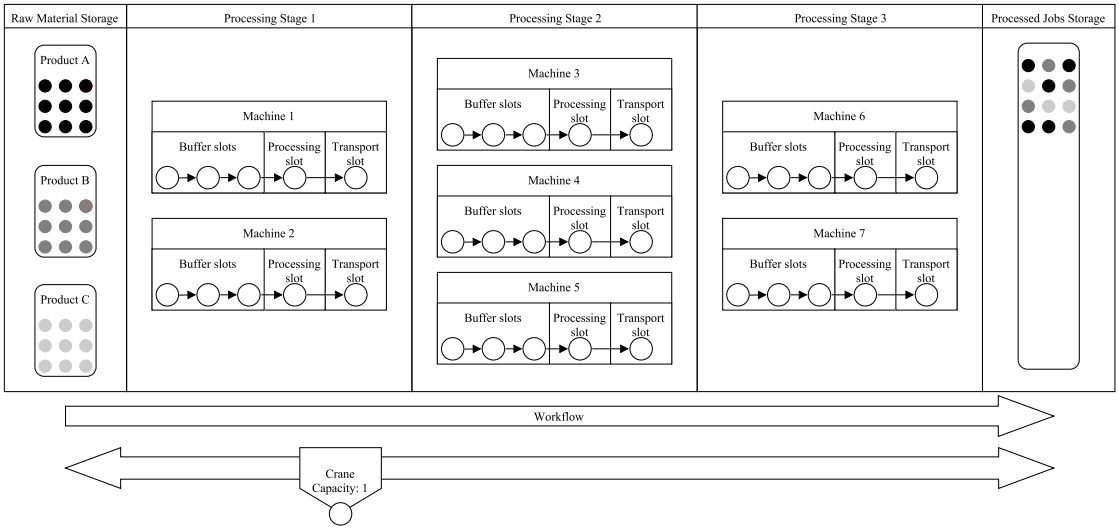

Figure 3.1: Exemplary shop floor configuration

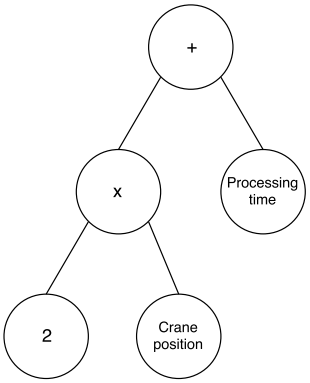

Figure 3.2: Example for a dispatching rule represented as expression tree

is fundamentally different. While GAs usually rely on string or permutation representations, GP is based on tree-like structures, as mentioned above. In this case, the tree nodes represent arithmetic operators or functions in general. The leaf nodes contain the so called terminal symbols, which are constants that are either fixed or "input" from the problem instance. These instances are split into training and test instance sets. The expression tree itself is built using the overall makespan for each training instance as the objective value (fitness value), which is explained in detail in Section 3.3.4. A hold-out set of up to 20 instances out of each instance set serves as test instances.

An example of a program in a tree-like structure is shown in Fig. 3.2. The functions used in this example are the mathematical operators + and *. The terminals consist of the constant value, "Crane position", and "Processing time". The resulting function is put together as follows: (2 * Crane position) + Processing time. It is a simplified example for a final priority rule for calculating the priority values of each possible job-movement according to aforementioned list scheduling algorithms. The priority calculation is applied to each of the possible movements. The job-machine option ranked with the lowest value is then executed by transporting the respective job to its next machine. We implemented and developed our GP approach with the HeuristicLab[1] framework due to predefined algorithms and a wide range of GP tools, as shown in Scheibenpflug et al. (2015). The shop floor is simulated with the same simulator introduced in Benda et al. (2019).

For the academic benchmark instance set, the chosen expression tree will be validated by computing the arithmetic mean makespan over all test instances and compare the result with simple priority rules. This will primarily be accomplished using a larger hold-out set of problem instances as the expected computational effort may render a k-fold cross-validation very difficult to perform within a reasonable time frame, at least on a single workstation PC. For the real-world based instance sets, we also calculate the arithmetic mean overall makespan for all test instances. However, the results are also compared with the CP, DT, and RF results, as introduced by Benda et al. (2019).

3.3.2 Terminal and Functions Set

Two different terminal sets for the academic benchmark instance set and the real-world based set were implemented. The function set for both instance sets contains basic mathematical operators, such as addition, subtraction, multiplication, and protected division. Additionally, conditionals in the form of "if-then-else statements" were implemented. Finally, a constant with values ranging from 0 to 1 chosen randomly is part of each set. Both terminal sets include terminals regarding jobs and machines widely used in the literature. Furthermore, the terminal sets are

[1] `https://dev.heuristiclab.com/`

Table 3.1: List of terminals and functions for the benchmark instance sets

Notation	Description	Category
AWJ	Average waiting time of job	Job
APT	Average processing time of jobs in queue of machine	Machine
MO	Total machine occupation	
MI	Total machine idle time	
AWM	Average waiting time on machine	
Cst	Constant (random number from 0 to 1)	

based on the list of attributes for the DT and RF approach of Benda et al. (2019). Both sets contain the terminals "Remaining work of job" (RW), "Operations left of job" (OL), "Job slack time" (SL), "Processing time on next machine" (NP), "Machine workload on next machine" (MN), and "Number of occupied slots on next machine" (OS). The additional terminals for the benchmark instance set are listed in Table 3.1.

In contrast to the benchmark instance set and other papers (e.g., Tay and Ho (2008) or Pickardt et al. (2013)), the terminal set for the real-world based set contains more complex elements, such as the crane position, sequence-dependent setup times, and buffer blocking. Hence, the terminal set consists of three different categories: job, machine, and transport related terminals. With these terminals, it is possible to capture every aspect of the status quo on the shop floor regarding workloads, crane and job positions, blocking situations, and setup times. The list of terminals is closer to the nine attributes of Benda et al. (2019) and therefore some terminals listed in Table 3.1 are left out. The list of additional terminals used for the real-world based instance sets is listed in Table 3.2.

Note that the categorization of the terminals in Table 3.1 and 3.2 also applies to the multi-tree terminals. Those marked as job-related are the terminals used for generating the job-decision trees, and vice versa for the machine-decision tree. The transport terminals on the other hand, are marked with the respective classification: "J" for job-related, "M" for machine-related.

3.3.3 Overall Structure of the Single- and Multi-Tree GP Approach

The GP approach is conducted as follows: In the first step, the initial population of a GP run is generated using the well known grow and full method so that the individuals in the initial population do not exceed the specified maximum tree depth. Applying the grow method, trees are created by adding a function and terminal randomly to the root of the tree. If a function was chosen, the tree is extended with

Table 3.2: List of terminals and functions for the real-world based adaption instance sets

Notation	Description	Category
ST	Setup time on next machine	Job
B^b	Does job block buffer in front of processing slot?	
MR	Machine workload on recent machine	Machine
C^J	Crane position in relation to job position	Transport
T^M	Transport time recent machine to next machine	
NT^{bM}	Is there a job on next machine on transport slot available?	
Cst	Constant (random number from 0 to 1)	

[b] Binary attribute values
[J] Job-related multi-tree terminal
[M] Machine-related multi-tree terminal

other functions and terminals. A branch of a tree ends, whenever a terminal was selected. This method does not guarantee a certain depth of the tree. In contrast, the full method creates trees randomly by assuring to achieve the set depth for the expression tree. Whether an expression tree is created with the grow or full method is determined with the help of a randomization parameter. The multi-tree approach consists of two parallel populations: the job-related and the machine-related expression tree population. Then, the fitness value of each expression tree within the population is evaluated by accumulating the makespan results achieved on the training instances for the respective expression tree. For manipulating the next generation candidates, we rely on standard operators, introduced by Koza (1992), to create a new expression tree offspring out of two "parent" trees:

- Single-point crossover: For the crossover operation expression tree parts of two "parent" expression trees are chosen, swapped, and transformed into a new "child" expression tree.

- Mutation: A certain part of the "parent" tree is randomly changed resulting in a new "child" expression tree.

The GP process is terminated as soon as the maximum number of generations is met. As the initial population for each GP run is generated randomly, the whole GP process is repeated four times to achieve a wider variety within the initial expression tree population resulting in four randomly generated initial GP populations with approximately 150 trees each. So for each run the best tree is selected and all of these four trees are applied to the test instances. The best performing tree out of these runs is the final, chosen tree.

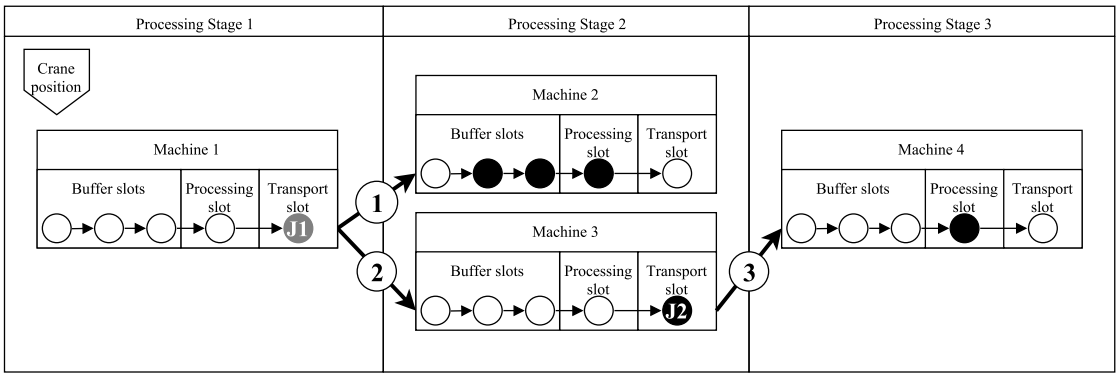

Figure 3.3: Example: Possible material movements

3.3.4 Fitness Evaluation

The fitness of an individual is typically linked to the objective function value of the solution represented by it. In our case, an individual is considered *fitter* than another one if it leads to a smaller overall makespan, which is calculated by adding up the makespans over all training instances. For solving each of those instances, the expression tree is used to determine the next decision of which job is assigned to which machine. Fig. 3.3 depicts an example for a possible decision process captured in a status quo of the shop floor during the production process of the real-world based instance set. The status quo of a (simple) flexible job shop instance would look quite similar but without the transport procedure and with infinite buffers in front of each processing slot.

With the help of the expression tree one out of three possible material movements shall be chosen. Here, three possible transport requests can be initiated: ① Put job 1 (J1) onto machine 2, ② put job 1 (J1) onto machine 3, or ③ put job 2 (J2) onto machine 4. For each of these possible movements a value is calculated considering the expression tree in observation. For example, the processing time of J1 on machine 2 may differ to the processing time on machine 3. Furthermore, the workload of machine 2 is higher as for machine 3. This might lead to a different value for the ranking—depending on which terminals are considered within the expression tree. The ranking is conducted by listing the option with the smallest value first. The decision influences the overall makespan for this instance and results in differing fitness values for the expression trees.

Within the single-tree approach the job-machine decision is conducted using one tree, while for the multi-tree approach the decision is split in choosing a job first with the expression tree containing job-related terminals and then determine the next machine for the job with the help of the second, machine-related tree.

3.3.5 Creating an Iterative Dispatching Rule

For generating an iterative dispatching rule, new terminals were introduced. The sets were expanded with the following job-related terminals: Recorded makespan of job (RMS), recorded total waiting time of job on all machines (RWT). Four terminals were added for the machine-relation: Recorded total machine workload (RWL), recorded total number of jobs on machine (RJV), recorded total machine idle time (RIT), and recorded total waiting time of jobs on machine (RMW).

The iterative approach is conducted within each generation before calculating a tree's fitness value. Once the new population was generated or modified, for each tree the iterative approach is conducted as follows: The tree is firstly applied to all training instances. The outcome is a initial overall makespan for this tree. In this first iteration, the iterative terminals RMS, RWT, RWL, RJV, RIT, and RMW are equal to 1 and the tree's fitness value is noted. Then, in the second iteration the terminal values of 1 are replaced with their corresponding values (e.g., 765 instead of 1 as the makespan of job number 2). With these values the fitness value of the tree is calculated again. If the overall makespan was improved in this iteration for the tree, the next iterations are conducted until no further improvement for the makespan minimization objective is achieved. This procedure is explained in the pseudo code shown in Algorithm 1.

Algorithm 1 Procedure to apply the iterative approach

```
for all trees in population do
    set all iterative terminal values to 1
    calculate basic objective value for tree
    update iterative terminals with new values
    calculate new objective value for tree
    while new objective value < basic objective value do
        set basic objective value as new objective value
        update iterative terminals with new values
        calculate new objective value for tree
    end while
end for
```

3.4 Experimental Set-up

Before the test runs were conducted, training instances for each instance set were generated randomly. Then, the experiments contained a series of tests regarding the GP parameters for training the expression trees. Finally, the resulting trees were tested on the corresponding test instances. The generation of the training instances and the testing procedure will be discussed in this section in detail.

Table 3.3: Configuration for Benchmark Instance Set IV and V

Configuration	1	2	3	4	5	6	7	8
Jobs	30	30	50	50	50	100	100	100
Machines	10	15	10	15	20	10	15	20
Operations	4-6	9-11	4-6	9-11	14-16	4-6	9-11	14-16
Processing times	5-99	5-99	5-99	5-99	5-99	5-99	5-99	5-99
Parallel machines for job	5.0	7.5	5.0	7.5	10.0	5.0	7.5	10.0

3.4.1 Configuration of the Benchmark Instance Sets

For the benchmark instance sets of Brandimarte and Lawrence, 60 to 64 instances were generated randomly within the given ranges. These instances served as training instances while the original benchmark instances were used as test instances. The training instances in this case were generated as follows:

- Benchmark Instance Set I: Consists of 60 randomly generated instances as training instances based on the Brandimarte set. The upper and lower bound for the number of jobs, machines, operations for jobs, unrelated parallel machines, and processing times, contain the minimum and maximum number for each of those aspects over all mk instances. The Brandimarte instance set consists of ten instances (mk1 to mk10), which served as test instances.

- Benchmark Instance Set II: Same as for Set I, but for each mk1 to mk10 instance 6 instances were generated randomly based on the properties of the respective instance.

- Benchmark Instance Set III: For each group of five Lawrence instances (la01 to la40) with similar ranges, 8 instances were generated randomly within the given range to a total of 64 training instances for this instance set. The la01 to la40 instances are used as test instances.

- Benchmark Instance Set IV: Based on the Brandimarte and Lawrence ranges a new instance set was generated with more jobs, machines, and operations with unrelated parallel machines. The configuration is shown in Table 3.3. For each configuration 8 training and 5 test instances were generated resulting in 64 training and 10 test instances. The instances are numerically ordered called bbdh-u1 to bbdh-u40. The goal was to generate instances with a greater flexibility in number of operations per job and job-machine-combinations.

- Benchmark Instance Set V: Same as set V but with identical parallel machines. These instances are named bbdh-i1 to bbdh-i40 as well.

For testing purposes the Benchmark Instance Set I and II will also be compared to the shortest processing time rule (SPT), the most work remaining rule (MWKR), and the least work remaining rule (LWKR), as listed in Brandimarte (1993).

3.4.2 Configuration of the Real-World Based Instance Sets

Three different instance scenarios were defined to examine the effect of machine blocking and a high degree of capacity utilization on the transport resource: The machine bottleneck (MB) scenarios with possibility of bottleneck stages containing only one machine, the crane bottleneck (CB) scenario with at least two parallel machines at any processing stage, and a mixture of both instances. In the CB case the crane is the scarcest resource and thus represents the bottleneck in the system. Furthermore, three different job load sets, low (6 to 15 jobs in total), medium (12 to 44) and high (21 to 72 jobs), were added for analyzing the impact of overall shop floor utilization resulting in 12 instance sets, as introduced by Benda et al. (2019). For each of those 12 sets, 80 instances were generated randomly with the help of an instance generator. For the three different job types the random parameter range from 3-5 processing stages, 1-3 buffer slots, 20-34 time units of processing time, and 8-13 time units of setup time. For the MB and mixed scenarios, the parallel machines range from 1-3, while for the CB scenario the range is 2-3.

3.4.3 Experiments for the Feature Selection Approach

The test series contained a feature selection procedure on the training instances of Benchmark Instance Set I, as listed above. 20 out of these training instances were selected, the initial population generated, and each trees' fitness value calculated. Then, within each tree, every single terminal was replaced with value 1, its fitness value calculated again, and compared to the original fitness value before the terminal's replacement. An increase for the makespan as fitness value is implied by a negative comparison result. Conversely, the makespan was reduced if the replacement yields a positive result. Thus, terminals that, if replaced, lead to a negative comparison result indicate a relative higher importance for the makespan minimization as those terminals leading to a lower comparison result. A full list of minimum and maximum differences to the basic fitness value, the overall sum and the average difference for each terminal is presented in Table 3.4.

Based on these test results, four different terminal sets were defined combining terminals that have a bigger effect on reducing the overall makespan than others:

- Terminal Set I: Contains only the 4 terminals with the biggest effect on the overall makespan, that is, RW, NP, MN, and APT
- Terminal Set II: Contains only the 4 terminals with the least effect, that is, OL, SL, MI, AWM

Table 3.4: Fitness value differences for feature selection approach

	Terminals										
	OL	RW	SL	NP	MN	OS	APT	MO	AWJ	MI	AWM
Min	-65	-299	-268	-617	-729	-74	-313	-258	-623	-103	-307
Max	50	182	362	323	687	436	572	657	70	318	289.00
Sum	79	-356524	583	-1590924	-1346862	-299	-80677	9683	-33317	22528	4323
AVG	0.001	-4.952	0.008	-22.096	-18.706	-0.004	-1.121	0.134	-0.463	0.313	0.060

Table 3.5: Results for terminal sets selected with feature selection on Brandimarte instances

	Single-tree				Multi-tree			
Terminal set	I	II	III	IV	I	II	III	IV
mk01	**45**	59	52	49	47	56	47	47
mk02	31	59	41	30	**29**	51	38	29
mk03	**204**	282	207	**204**	**204**	300	211	207
mk04	**69**	107	72	68	70	97	77	73
mk05	187	236	200	180	**178**	207	192	**178**
mk06	**73**	131	96	76	75	157	97	78
mk07	153	239	196	**148**	149	236	200	150
mk08	532	**531**	**531**	**531**	539	**531**	539	539
mk09	351	407	344	335	**333**	392	339	336
mk10	231	331	270	**229**	243	375	288	237
Average	187.6	238.2	200.9	**185.0**	186.7	240.2	202.8	187.4

Bold values indicate best values

- Terminal Set III: Contains all terminals except terminals of Set I resulting in excluding the ones with the biggest effect

- Terminal Set IV: Contains all terminals except terminals of Set II resulting in excluding the ones with the least effect

The results of applying these different sets on the Brandimarte instances for the single-tree and multi-tree approach are shown in Table 3.5 and indicate that terminal set II and III indeed lead to worse results than set I and IV, as expected. However, for the test series and experiments for the Brandimarte, Lawrence, and real-world based instance sets, the full terminal sets were used, as a significant improvement in comparison to the feature selection terminal sets was not achieved.

Table 3.6: Configuration for GP parameters

Parameter	Configurations
Population size	100, **150**, 200, 250
Maximum generations	80, 100, **150**, 200
Maximum depth of expression tree	4, **6**
Maximum length of expression tree	20, **40**
Crossover rate	**25%**

Basic configuration: Bold and underlined values

3.4.4 GP Parameter Tests

A variety of test runs were conducted on these instance scenarios changing some of the GP parameters. Tested parameter values and the basic configuration are listed in Table 3.6. For each test run one or more parameter was varied while the others were fixed to the basic configuration—bold and underlined. Secondly, the range of the GP parameters are configured for the test runs. In Table 3.6 the varying parameter configurations in combination with the fixed parameters are listed. For conducting the GP approach, the HeuristicLab 3.3 software was used, as already stated above. The test runs were conducted on a workstation with Windows 10, 64 bit, an i7-4790 CPU with 3.6 GHz, and 8 GB of RAM. Some of the experiments have also been carried out on the Vienna Scientific Cluster (VSC3) in single-threaded mode. The compute nodes of the VSC3 are equipped with two Intel Xeon E5-2650v2 (each with 8 cores, 2.6 GHz).

The real-world based instance sets were compared to a CP formulation. The CP formulation was solved using the IBM ILOG CP Optimizer 12.8. Furthermore, our results were compared to standard priority rules, as introduced by Benda et al. (2019). The first priority rule (PR1) is a standard shortest processing time rule also including setup time consideration. The second priority rule (PR2) contains PR1 and adds machine workload considerations.

3.5 Computational Results

In this section, the results of the aforementioned experiments on the different instance sets are presented and discussed in detail.

3.5.1 Results for the Brandimarte Benchmark Instances

The results for the benchmark instances of Brandimarte are listed in Table 3.7 with configurations of n jobs and m machines. The remarks in the table state on

Table 3.7: Results for the Brandimarte benchmark instances

Instance	n× m	SPT	LWKR	MWKR	UB	Single-tree				Multi-tree			
						Normal[1]	Gap	Iterative[2]	Gap	Normal[1]	Gap	Iterative[1]	Gap
mk1	10×6	65	76	54	39 [4]	45	15.38%	45	15.38%	47	20.51%	44	12.82%
mk2	10×6	44	48	41	26 [2]	30	15.38%	28	7.69%	29	11.54%	30	15.38%
mk3	15×8	397	506	296	204 [2]	204	0.00%	204	0.00%	204	0.00%	204	0.00%
mk4	15×8	109	121	89	60 [2]	73	21.67%	67	11.67%	70	16.67%	69	15.00%
mk5	15×4	231	291	188	172 [1]	180	4.65%	180	4.65%	178	3.49%	180	4.65%
mk6	10×15	128	143	139	58 [2]	76	31.03%	74	27.59%	79	36.21%	80	37.93%
mk7	20×5	188	238	262	139 [3]	156	12.23%	155	11.51%	150	7.91%	147	5.76%
mk8	20×10	670	1042	558	523 [2]	524	0.19%	532	1.72%	523	0.00%	523	0.00%
mk9	20×10	573	723	536	307 [2]	334	8.79%	333	8.47%	338	10.10%	336	9.45%
mk10	20×15	536	627	524	197 [1]	234	18.78%	234	18.78%	229	16.24%	233	18.27%

[1] Trained on Benchmark Instance Set I

[2] Trained on Benchmark Instance Set II

which benchmark instance set the best working expression tree was trained on. It can be stated that both expression tree approaches yield competitive results compared to the standard priority rules SPT, LWKR, and MWKR. The priority rules are undercut for all instances. The table also contains the gaps to the upper bounds of Gao et al. (2008) [1], Mastrolilli and Gambardella (2000) [2], Pezzella et al. (2008) [3], and Xing et al. (2010) [4]. It has to be stated that these approaches are upper bound calculations are made-to-measure approaches based on metaheuristics resulting in several hours of run time for result calculation of a single instance. Comparing the GP results to these bounds, it can also be stated that the single-trees and multi-trees are able to achieve results on upper bound level or close to it. For the mk3 and mk8 instances, a gap of 0.00% to 1.72% was achieved. Furthermore, the iterative approach leads to better results regarding the overall makespan for most mk instances compared to the basic single-tree approach. For the multi-tree approach, the iterative approach was able to reduce the gap from 20.51% to 12.82% for the mk1 instance. However, this does not hold for some other instances for which the iterative approach yields mixed results. Further tests on instance sets with a higher degree of flexibility in terms of machines, jobs, and parallel machines for each operation, as appearing in the Lawrence instances, might be more suitable for the iterative approach applied on the multi-tree representation.

3.5.2 Results for the Lawrence Benchmark Instances

As the Lawrence instances contain up to 30 jobs and a broader variety of configurations in terms of flexibility, number of machines, and processing times, the GP approaches yield significantly better results as for the Brandimarte instances with average gaps over all instances ranging from 4.44% up to 7.34%. The gap is calculated as average gap over all instances of a single approach compared to the lower bound result. As the lower bound equals the upper bound in nearly all

cases, the GP approach results are compared to the lower bound of Jurisch (1992). The results for the Lawrence instances are depicted in Table 3.8. Comparing the overall average gap, it can be stated that both the single-tree and multi-tree representations are able to deliver competitive results, whereas the single-tree performs slightly better than the multi-tree. Furthermore, for both expression trees the iterative approach further improves the results and indicates the ability to reduce the makespan for instances configurations as introduced by Lawrence.

Each five instances contain the same number of jobs n and machines m. When analyzing these packages of five instances, it emerges that the single-tree representation performs well especially in the 10×10 scenarios, while using the iterative approach for this representation leads to good results for many configurations (15×5, 20×5, 30×10, and 15×15). The same effect occurs for the multi-tree dispatching rule. Here, the simple multi-tree achieves competitive results for the 30×10 scenario and the approach with iterative dispatching rule can reduce the makespan in the 15×5, 20×5, 30×10, 15×15 sets.

3.5.3 Results for the New Instance Sets with Unrelated and Identical Machines based on Brandimarte and Lawrence

For the instances with unrelated and identical parallel machines, a higher flexibility in terms of number of jobs, machines, operations, and possible parallel machines for an operation, the results turn out to be even better as for the Lawrence instances. For all approaches it can be stated that they achieve better results in instance sets with more than 30 jobs and they undercut the CP upper bound (UB) for all instance sets except the 30×10, 30×15, and 50×10 sets. The results for the instances with unrelated machines are listed in Table 3.9 and these for the instances with identical parallel machines in Table 3.10. The average gaps over all instances for the different GP approaches compared to the CP UB indicate an improvement achieved with all GP approaches for the instance set with unrelated machines.

It becomes apparent that the average gap over all instances with unrelated machines was significantly improved in average for all approaches. Again, the iterative dispatching rules improve the overall result for the single- and multi-tree representation slightly. The multi-tree iterative dispatching rule reduced the overall makespan compared to other approaches by up to 1.21%. However, this does not hold for the instances with identical machines. Here, the four approaches do not undercut the average gap for all instances but still yield competitive results. In difference to the instance set with unrelated machines the iterative dispatching rules slightly improve the overall results.

Table 3.8: Results for the Lawrence benchmark instances

Instance	n× m	LB	Single-tree				Multi-tree			
			Normal	Gap	Iterative	Gap	Normal	Gap	Iterative	Gap
la01	10×5	570	649	13.86%	646	13.33%	611	7.19%	621	8.95%
la02	10×5	529	610	15.31%	547	3.40%	594	12.29%	610	15.31%
la03	10×5	477	519	8.81%	531	11.32%	606	27.04%	561	17.61%
la04	10×5	502	547	8.96%	545	8.57%	606	20.72%	531	5.78%
la05	10×5	457	507	10.94%	518	13.35%	547	19.69%	517	13.13%
la06	15×5	799	842	5.38%	823	3.00%	915	14.52%	825	3.25%
la07	15×5	749	781	4.27%	766	2.27%	810	8.14%	783	4.54%
la08	15×5	765	785	2.61%	785	2.61%	788	3.01%	774	1.18%
la09	15×5	853	876	2.70%	868	1.76%	925	8.44%	885	3.75%
la10	15×5	804	828	2.99%	828	2.99%	908	12.94%	825	2.61%
la11	20×5	1071	1098	2.52%	1092	1.96%	1085	1.31%	1097	2.43%
la12	20×5	936	952	1.71%	961	2.67%	975	4.17%	952	1.71%
la13	20×5	1038	1058	1.93%	1066	2.70%	1112	7.13%	1048	0.96%
la14	20×5	1070	1153	7.76%	1103	3.08%	1152	7.66%	1102	2.99%
la15	20×5	1089	1137	4.41%	1098	0.83%	1103	1.29%	1116	2.48%
la16	10×10	717	750	4.60%	750	4.60%	795	10.88%	757	5.58%
la17	10×10	646	650	0.62%	715	10.68%	669	3.56%	657	1.70%
la18	10×10	663	666	0.45%	675	1.81%	670	1.06%	681	2.71%
la19	10×10	617	670	8.59%	660	6.97%	646	4.70%	671	8.75%
la20	10×10	756	789	4.37%	776	2.65%	774	2.38%	760	0.53%
la21	15×10	800	903	12.88%	881	10.13%	863	7.88%	887	10.88%
la22	15×10	733	821	12.01%	788	7.50%	863	17.74%	791	7.91%
la23	15×10	809	880	8.78%	861	6.43%	941	16.32%	863	6.67%
la24	15×10	773	896	15.91%	834	7.89%	900	16.43%	877	13.45%
la25	15×10	751	908	20.91%	890	18.51%	914	21.70%	934	24.37%
la26	20×10	1052	1148	9.13%	1112	5.70%	1098	4.37%	1102	4.75%
la27	20×10	1084	1109	2.31%	1122	3.51%	1131	4.34%	1129	4.15%
la28	20×10	1069	1126	5.33%	1147	7.30%	1117	4.49%	1133	5.99%
la29	20×10	993	1064	7.15%	1060	6.75%	1144	15.21%	1037	4.43%
la30	20×10	1068	1145	7.21%	1123	5.15%	1163	8.90%	1114	4.31%
la31	30×10	1520	1587	4.41%	1565	2.96%	1597	5.07%	1561	2.70%
la32	30×10	1657	1731	4.47%	1695	2.29%	1700	2.60%	1687	1.81%
la33	30×10	1497	1549	3.47%	1532	2.34%	1553	3.74%	1513	1.07%
la34	30×10	1535	1588	3.45%	1567	2.08%	1569	2.21%	1571	2.35%
la35	30×10	1549	1600	3.29%	1586	2.39%	1605	3.62%	1603	3.49%
la36	15×15	948	1030	8.65%	956	0.84%	1021	7.70%	973	2.64%
la37	15×15	986	1087	10.24%	1024	3.85%	1058	7.30%	1043	5.78%
la38	15×15	943	1012	7.32%	943	0.00%	943	0.00%	944	0.11%
la39	15×15	922	974	5.64%	976	5.86%	996	8.03%	1007	9.22%
la40	15×15	955	910	-4.71%	969	1.47%	984	3.04%	959	0.42%
Gap			5.94%		4.44%		7.34%		4.76%	

Again, it can be observed that the iterative dispatching rule leads to slightly better results as the simple single- and multi-tree. Furthermore, the higher the degree of flexibility in terms of operations, number of jobs and machines, and parallel machines for an operation, the better the results of the GP approaches. Though, for scenarios with identical machines this flexibility is restricted and the approaches yield slightly worse results as for the scenarios with unrelated machines.

3.5.4 Results for the Real-World Based Instance Sets

The results of the GP approach are compared with the results of the decision tree, random forest, and CP runs of Benda et al. (2019). Furthermore, as a comparison baseline, two basic priority dispatching rules—based on Panwalkar and Iskander (1977)—were chosen. The basic principle of the two priority rules under consideration is simply choose the job with the shortest processing time. Priority rule 1 (PR1) includes the setup time consideration by selecting the job with the shortest processing time on the next machine also adding the inevitable setup time on that machine. Priority rule 2 (PR 2) is based on PR1 but also includes the factor of machine workload.

The results—as shown in Table 3.11—list the best performing expression tree (DT or RF) of each instance scenario set. The overall deviation of the expression tree for all 20 test instances within an instance set is determined by using the arithmetic mean of all those 20 overall makespans. For example, a value of 1.065 is interpreted as a deviation of 6.5% from the CP result, in terms of a 6.5% bigger average overall makespan. For the real-world based instance sets it can be stated that the GP approaches outperform the DT and RF for all sets, but there is no single approach that works best on all of them. For these sets the iterative approach is able to reduce the overall makespan for 4 out of 12 sets, but the single-tree still performs better in 4 out of 12 sets. Further test runs with a wider variation of GP configuration parameters might show that the iterative approach is able to undercut all the results of the regarding single- and multi-tree approaches. Nevertheless, it is shown that the GP approaches work well and reduce the overall makespan significantly.

3.6 Discussion and Conclusion

In this paper, a GP approach was introduced for generating expression trees that are used as priority dispatching rules for making job assignments and sequencing decisions, combined in a single tree or separated in a multi-tree approach. The trees were applied on instances of a flexible job shop and a hybrid flow shop scheduling problem by ranking the available material movements. With the help of the ranking, it is determined which job should be transported to which machine. First,

Table 3.9: Results for the new instance sets bbdh-u with unrelated machines

Instance	n× m	LB	CP UB	Single-tree				Multi-tree			
				Normal	Gap	Iterative	Gap	Normal	Gap	Iterative	Gap
bbdh-u1	30×10	283	304	374	18.72%	336	9.52%	350	13.14%	379	19.79%
bbdh-u2	30×10	274	291	372	21.77%	341	14.66%	364	20.05%	377	22.81%
bbdh-u3	30×10	258	295	380	22.37%	347	14.99%	358	17.60%	340	13.24%
bbdh-u4	30×10	385	409	443	7.67%	442	7.47%	446	8.30%	470	12.98%
bbdh-u5	30×10	412	454	512	11.33%	511	11.15%	498	8.84%	559	18.78%
bbdh-u6	30×15	352	414	455	9.01%	436	5.05%	451	8.20%	448	7.59%
bbdh-u7	30×15	868	923	931	0.86%	933	1.07%	939	1.70%	954	3.25%
bbdh-u8	30×15	864	969	942	-2.87%	937	-3.42%	935	-3.64%	980	1.12%
bbdh-u9	30×15	709	870	780	-11.54%	773	-12.55%	784	-10.97%	791	-9.99%
bbdh-u10	30×15	745	940	814	-15.48%	815	-15.34%	828	-13.53%	821	-14.49%
bbdh-u11	50×10	962	1097	1071	-2.43%	1034	-6.09%	1061	-3.39%	1079	-1.67%
bbdh-u12	50×10	928	1023	1016	-0.69%	1004	-1.89%	1005	-1.79%	1041	1.73%
bbdh-u13	50×10	870	997	959	-3.96%	939	-6.18%	949	-5.06%	974	-2.36%
bbdh-u14	50×10	895	977	977	0.00%	959	-1.88%	967	-1.03%	980	0.31%
bbdh-u15	50×10	913	1093	998	-9.52%	989	-10.52%	980	-11.53%	1002	-9.08%
bbdh-u16	50×15	642	822	735	-11.84%	755	-8.87%	739	-11.23%	758	-8.44%
bbdh-u17	50×15	638	779	729	-6.86%	717	-8.65%	736	-5.84%	746	-4.42%
bbdh-u18	50×15	635	762	738	-3.25%	729	-4.53%	735	-3.67%	760	-0.26%
bbdh-u19	50×15	604	777	701	-10.84%	669	-16.14%	672	-15.63%	693	-12.12%
bbdh-u20	50×15	635	816	711	-14.77%	772	-5.70%	736	-10.87%	734	-11.17%
bbdh-u21	50×20	1240	1533	1326	-15.61%	1312	-16.84%	1326	-15.61%	1373	-11.65%
bbdh-u22	50×20	1237	1342	1319	-1.74%	1314	-2.13%	1310	-2.44%	1333	-0.68%
bbdh-u23	50×20	1261	1383	1371	-0.88%	1362	-1.54%	1345	-2.83%	1374	-0.66%
bbdh-u24	50×20	1268	1443	1364	-5.79%	1353	-6.65%	1356	-6.42%	1356	-6.42%
bbdh-u25	50×20	1224	1389	1326	-4.75%	1322	-5.07%	1331	-4.36%	1378	-0.80%
bbdh-u26	100×10	864	1088	946	-15.01%	943	-15.38%	937	-16.12%	968	-12.40%
bbdh-u27	100×10	839	963	937	-2.77%	915	-5.25%	905	-6.41%	947	-1.69%
bbdh-u28	100×10	841	1081	950	-13.79%	932	-15.99%	946	-14.27%	942	-14.76%
bbdh-u29	100×10	814	1057	889	-18.90%	899	-17.58%	921	-14.77%	914	-15.65%
bbdh-u30	100×10	832	987	914	-7.99%	900	-9.67%	919	-7.40%	937	-5.34%
bbdh-u31	100×15	1577	1843	1662	-10.89%	1683	-9.51%	1670	-10.36%	1709	-7.84%
bbdh-u32	100×15	1565	1713	1652	-3.69%	1656	-3.44%	1659	-3.25%	1679	-2.03%
bbdh-u33	100×15	1550	1832	1630	-12.39%	1617	-13.30%	1628	-12.53%	1657	-10.56%
bbdh-u34	100×15	1539	1697	1647	-3.04%	1614	-5.14%	1655	-2.54%	1657	-2.41%
bbdh-u35	100×15	1571	1951	1656	-17.81%	1675	-16.48%	1684	-15.86%	1726	-13.04%
bbdh-u36	100×20	1026	1143	1123	-1.78%	1101	-3.81%	1114	-2.60%	1146	0.26%
bbdh-u37	100×20	1045	1206	1147	-5.14%	1140	-5.79%	1141	-5.70%	1161	-3.88%
bbdh-u38	100×20	1046	1167	1137	-2.64%	1121	-4.10%	1120	-4.20%	1162	-0.43%
bbdh-u39	100×20	1059	1258	1146	-9.77%	1159	-8.54%	1153	-9.11%	1200	-4.83%
bbdh-u40	100×20	1074	1238	1180	-4.92%	1170	-5.81%	1184	-4.56%	1218	-1.64%
Gap				-4.04%		-5.25%		-4.54%		-5.38%	

Table 3.10: Results for the new instance sets bbdh-i with identical machines

Instance	n× m	LB	CP UB	Single-tree				Multi-tree			
				Normal	Gap	Iterative	Gap	Normal	Gap	Iterative	Gap
bbdh-i1	30×10	839	840	885	5.08%	881	4.65%	876	4.11%	870	3.45%
bbdh-i2	30×10	839	840	885	5.08%	881	4.65%	876	4.11%	870	3.45%
bbdh-i3	30×10	996	998	1130	11.68%	1116	10.57%	1122	11.05%	1133	11.92%
bbdh-i4	30×10	1245	1245	1275	2.35%	1269	1.89%	1274	2.28%	1267	1.74%
bbdh-i5	30×10	1750	1752	1800	2.67%	1814	3.42%	1799	2.61%	1793	2.29%
bbdh-i6	30×15	1847	1851	1904	2.78%	1912	3.19%	1912	3.19%	1898	2.48%
bbdh-i7	30×15	2489	2490	2511	0.84%	2519	1.15%	2537	1.85%	2512	0.88%
bbdh-i8	30×15	3334	3335	3368	0.98%	3360	0.74%	3370	1.04%	3358	0.68%
bbdh-i9	30×15	3747	3748	3797	1.29%	3797	1.29%	3798	1.32%	3795	1.24%
bbdh-i10	30×15	3748	3749	3809	1.58%	3794	1.19%	3802	1.39%	3800	1.34%
bbdh-i11	50×10	3751	3753	3788	0.92%	3790	0.98%	3788	0.92%	3793	1.05%
bbdh-i12	50×10	3715	3717	3752	0.93%	3748	0.83%	3749	0.85%	3750	0.88%
bbdh-i13	50×10	3784	3785	3820	0.92%	3809	0.63%	3815	0.79%	3805	0.53%
bbdh-i14	50×10	3789	3791	3825	0.89%	3817	0.68%	3825	0.89%	3820	0.76%
bbdh-i15	50×10	3842	3844	3882	0.98%	3870	0.67%	3874	0.77%	3867	0.59%
bbdh-i16	50×15	3223	3227	3268	1.25%	3264	1.13%	3275	1.47%	3273	1.41%
bbdh-i17	50×15	3236	3240	3297	1.73%	3296	1.70%	3300	1.82%	3291	1.55%
bbdh-i18	50×15	3241	3245	3305	1.82%	3293	1.46%	3304	1.79%	3297	1.58%
bbdh-i19	50×15	3233	3237	3283	1.40%	3284	1.43%	3273	1.10%	3281	1.34%
bbdh-i20	50×15	3233	3237	3283	1.40%	3284	1.43%	3273	1.10%	3281	1.34%
bbdh-i21	50×20	5033	5034	5080	0.91%	5079	0.89%	5068	0.67%	5075	0.81%
bbdh-i22	50×20	4987	4987	5012	0.50%	5015	0.56%	5024	0.74%	5021	0.68%
bbdh-i23	50×20	5100	5102	5145	0.84%	5136	0.66%	5143	0.80%	5139	0.72%
bbdh-i24	50×20	4908	4910	4956	0.93%	4943	0.67%	4969	1.19%	4934	0.49%
bbdh-i25	50×20	4998	4999	5035	0.71%	5028	0.58%	5026	0.54%	5027	0.56%
bbdh-i26	100×10	4357	4359	4399	0.91%	4389	0.68%	4398	0.89%	4395	0.82%
bbdh-i27	100×10	4361	4363	4397	0.77%	4420	1.29%	4392	0.66%	4388	0.57%
bbdh-i28	100×10	4415	4418	4462	0.99%	4455	0.83%	4455	0.83%	4446	0.63%
bbdh-i29	100×10	4394	4397	4432	0.79%	4439	0.95%	4429	0.72%	4427	0.68%
bbdh-i30	100×10	4337	4340	4394	1.23%	4376	0.82%	4385	1.03%	4379	0.89%
bbdh-i31	100×15	6289	6291	6321	0.47%	6335	0.69%	6323	0.51%	6314	0.36%
bbdh-i32	100×15	6294	6295	6360	1.02%	6326	0.49%	6344	0.77%	6323	0.44%
bbdh-i33	100×15	6154	6155	6213	0.93%	6192	0.60%	6207	0.84%	6197	0.68%
bbdh-i34	100×15	6261	6261	6306	0.71%	6294	0.52%	6288	0.43%	6285	0.38%
bbdh-i35	100×15	6365	6366	6413	0.73%	6410	0.69%	6398	0.50%	6416	0.78%
bbdh-i36	100×20	5490	5492	5539	0.85%	5524	0.58%	5513	0.38%	5519	0.49%
bbdh-i37	100×20	5342	5343	5405	1.15%	5374	0.58%	5389	0.85%	5377	0.63%
bbdh-i38	100×20	5412	5413	5455	0.77%	5461	0.88%	5460	0.86%	5465	0.95%
bbdh-i39	100×20	5469	5472	5506	0.62%	5515	0.78%	5506	0.62%	5504	0.58%
bbdh-i40	100×20	5378	5379	5419	0.74%	5406	0.50%	5425	0.85%	5415	0.66%
Gap				1.58%		1.45%		1.48%		1.33%	

Table 3.11: Cross-validation results for the instance scenarios with makespan objective and unrelated parallel machines

Approach	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	1.201	1.216	1.178	1.322	1.379	1.318	1.223	1.318	1.244
PR2	1.155	1.195	1.168	1.254	1.367	1.303	1.184	1.308	1.226
DT or RF	1.179[1]	1.145[1]	1.094[1]	1.255[2]	1.162[2]	1.125[2]	1.234[2]	1.163[1]	1.111[2]
Single-tree	1.092	1.041	**1.029**	1.109	**1.032**	1.029	1.105	**1.055**	**1.005**
Multi-tree	1.088	1.045	1.036	1.104	1.057	**1.021**	1.110	1.256	1.009
Iterative single-tree	**1.063**	**1.033**	1.041	1.096	1.045	1.038	1.099	1.297	1.015
Iterative multi-tree	1.074	1.055	1.060	**1.082**	1.044	1.032	**1.088**	1.288	1.007

Bold values indicate best values

[1] Decision tree approach
[2] Random forest approach

academic benchmark instances of Brandimarte and Lawrence served as a basis for investigations regarding the performance of the single- and multi-tree approach, also combined with generating an iterative dispatching rule. Second, these test runs were also applied to special case instances based on a CPS configuration of an industrial partner. The goal was to deal with a variety of different settings regarding the shop floor, such as varying processing times of the same production stage for the same job, varying number of machines per stage, setup times, job types per order, and number of jobs per order.

It was shown that the GP approaches not only perform well on the benchmark instance sets, especially to the larger ones, compared to the lower bounds and CP upper bound obtained by a CP formulation. The GP results also undercut the results of simple priority rules and an existing tree learning-based approach (decision tree and random forest) on the real-world based scenario. The main advantage of our approach are easy-to-implement and easy-to-adapt criteria meaning that human planners might rely on such a decision support approach that yields competitive results on instances of the same problem setting within a split second. After the GP dispatching rule is trained, it can be applied to every other instance of the same configuration performing well regarding the overall makespan minimization objective, whereas other optimization methods, such as metaheuristics, need to solve each instance separately with crucial more amount of computational time.

Therefore we can conclude that the terminals and functions set in combination with the fitness value calculation is an adequate combination for obtaining results that range between significant improvements to close-to-lower bound results for each of the instance sets. And, furthermore, the expression tree representing a job-machine material movement combination performs well as a generalized dispatching rule. It became obvious, that the iterative dispatching rules improved to reduce the overall makespan in all academic benchmark instance sets. However, for some of these sets, the single-tree approach performed better as the multi-tree approach. Our proposed GP approaches perform especially well in the newly generated bigger instances sets based on Brandimarte and Lawrence compared to the CP formulation results. Furthermore, for the real-world instance sets, the overall makespan was reduced by all approaches significantly, whereas the iterative approaches yielded better results for most of the sets too.

Future research might address machine breakdowns in a dynamic environment. Occurrences of this kind might be considered with the help of predictive maintenance approaches. Therefore, we list three possible scenarios to be investigated. First, random machine breakdowns are assumed to follow known and also unknown distributions. The breakdowns may be correlated with regard to their frequency of occurrence. Second, schedule-dependent breakdowns based on sequencing information only. There might be a correlation with machine idle times, intensity of changeovers, and setup times, for example. And third, a sensor-based prediction based on time series using sensor data collected directly from the machines on the shop floor can help to prevent machine breakdowns. For our GP approach new "look-ahead" terminals would have to be implemented to process the predictive information concerning machine breakdowns and reassignments of jobs to different parallel machines.

Another line of research might consider unfixing the tree pairs in the multi-tree approach. In this approach, the job assignment and machine sequencing is split in two separate decision processes by managing fixed pairs of trees for each of those decisions within the GP population. In a new step, instead of fixed pairs, the two tree populations might be mixed completely, so that each job assignment tree is combined with each machine sequencing tree and their fitness value calculated.

Chapter 4

Learning Approach to replace the Sample Average Approximation for Appointment Scheduling

Technical report
Learning Approach to replace the Sample Average Approximation for Appointment Scheduling
F. Benda, R. Braune.

Abstract Cancer counts as one of the most difficult to treat illnesses. One of the promising alternatives to the typical chemotherapy is the ion beam therapy, which is conducted in specialized ion beam centers. During the whole treatment process, it is of importance to keep the time of a patient waiting for the treatment within tolerable limits. Hence, estimating the average waiting time correctly is the most important part to keep the patient's overall satisfaction on a adequate level. One possibility to estimate the average waiting time is the complex method of sample average approximation. The goal of this technical report is to demonstrate the ability of regression analysis with multiple regressors to replace the sample average approximation in order to reduce the level of complexity of the estimation procedure significantly. Therefore, new features used as regressors are implemented. Then, out of this list of features, different single feature and multi feature sets are defined. These feature sets, in turn, are then applied within a regression analysis and artificial neural network to estimate the average waiting time by defining the different features as regressors and the average waiting time as regressand. The regression analysis contains commonly used methods for regression analysis, such as a simple regression method (ordinary least squares), linear regression methods (lasso and ridge regression), nonlinear methods (support vector and kernel ridge regression), and an artificial neural network (multilayer perceptron). The experiments were carried out on different instance sets regarding the number of patients and different patient configuration mixes. Preliminary test results indicate that the multi feature sets perform significantly better than

the single feature set. The overall results also support the assumption that the learning approach might be used to replace the complex sample average approximation by achieving competitive prediction results for most of the instances.

4.1 Introduction

Cancer is one of the main threats regarding illnesses for human kind. And still, cancer is spreading steadily whilst research facilities worldwide are not able to develop and deliver a treatment that leads to significant success regarding destruction of cancer cells for most of the body parts. If a patient is diagnosed with cancer, typically a chemotherapy or surgery is conducted. The goal is to apply radiotherapy in a way, such that the healthy cells are not affected while those tumorous regions are killed. For this purpose, in common hospitals radiation with X-rays is produced using a linear particle accelerator (LINAC). However, it is possible to apply a different treatment procedure, the ion beam therapy. In contrast to typical radiotherapy, ion beam therapy makes use of protons and carbon ions (or both of them) that are able to prevent the creation of secondary tumors within the patients' bodies after their therapy. One of these ion beam centers is located in Wiener Neustadt. This technical report is based on the real-world appointment duration data of this facility.

The treatment phase contains two different beam types that might be applied: Carbon ions and protons. The ion beam center itself consists of three different treatment rooms and the ion beam technology, as shown in Fig. 4.1. The patients have to wait in these rooms until their treatment procedure is started. Here, unscheduled waiting times for the patients might occur due to unplanned delays. As chronically ill patients have to undergo several consecutive treatments, the unexpected waiting time is one of the major problems. Once a patient had to be delayed, this also affects all subsequent patients. Therefore, dealing with uncertainty regarding the waiting time in the treatment process in ion beam centers is one of the main challenges for radiotherapy appointment scheduling literature, as demonstrated by Vogl et al. (2020).

Within this field of radiotherapy scheduling, Vogl et al. (2019) propose a deterministic method for generating schedules including waiting time estimation using a genetic algorithm (GA), iterated local search method, and a method that combines both approaches. The initial solutions are created with a greedy randomized method. Further research was conducted by Vogl et al. (2020) with considering disruptions as stochastic aspect, especially in the context of patient waiting times that are extended due to unforeseen events. Again, the schedules are generated by creating several initial solutions with a greedy algorithm first. On this pool of

solutions, a GA metaheuristic is applied to determine those solutions of good quality. Out of the solution representations, a baseline schedule is generated, which is then evaluated after the optimization procedure, as described by Vogl et al. (2020). Here, the sample average approximation is used to evaluate the expectation of the average waiting time as a part of the objective function by sampling over 100 random scenarios. Another approach to determine the average waiting time, also proposed by Vogl et al. (2020), is called the "waiting time estimation strategy". Here, the correlation of the beam resource idle time and the patient's waiting time was used in the framework of a linear regression with a single regressor to estimate the average waiting time. The form of the linear regression line for this case is as follows:

$$\text{Waiting time} = A + B * \text{baseline schedule idle time}$$

The required data to perform the simple linear regression is collected during the optimization process in form of intermediate schedules to adjust the regression line accordingly during a GA run.

Both of these methods are of a high degree of complexity regarding the necessity to create the random scenarios, the evaluation process itself, and the extensive process to adjust the regression line throughout the optimization procedure. Therefore, the goal of this technical report is replacing the regression line adjustment process and the sample average approximation for estimating patients' waiting times within the framework of a stochastic optimization algorithm. This is achieved by applying methods of multiple instead of simple linear regression analysis and an artificial neural network. In an offline training implementation introduced in this report, the regressors are determined that can then be used in an online approach for average waiting time estimation in the framework of appointment scheduling. For the regression methods, new schedule-related features were introduced and a feature selection process conducted to define those features with a higher importance regarding their ability to predict the average waiting time accurately.

This technical report is organized as follows: The problem statement is described in Section 4.2. In Section 4.3, we discuss the feature selection process, the regression analysis methods, and the experiments in detail. Section 4.4 discusses the results of the multiple regression methods applied to the instance sets. Finally, Section 4.5 concludes and proposes possibilities for future research.

4.2 Problem Statement

We rely on the aforementioned specific ion beam center configuration of MedAustron in Wiener Neustadt, Austria. This center consists of the linear accelerator and three different treatment rooms, as depicted in Fig. 4.1. The availability of only one beam but three rooms leads to the definition of the beam as the bottleneck

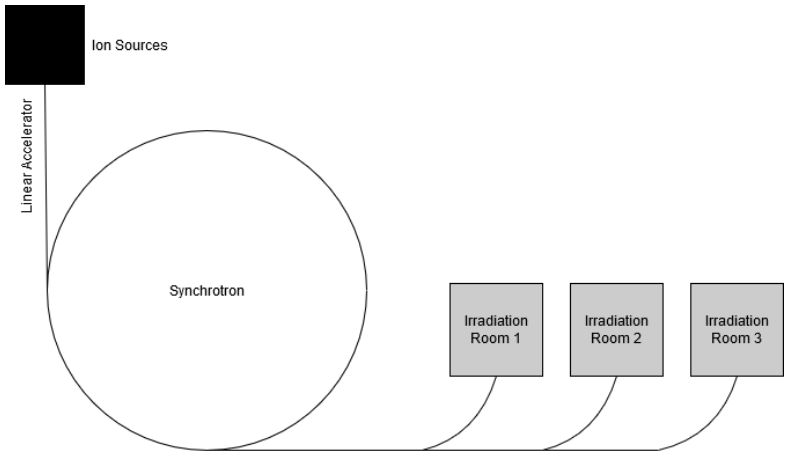

Figure 4.1: MedAustron ion beam center in Wiener Neustadt, Austria

resource. Each patient undergoes a certain number of treatments within the treatment rooms. For a more detailed overview of the treatment process, restrictions, setup times, and the scheduling process itself, we refer to Vogl et al. (2019).

The patients are clustered into four different groups which differ in treatment complexity and planned activity duration. Furthermore, nine different patient classes were implemented according to different treatment patterns, introduced by Vogl et al. (2020). The pattern here is described in terms of whether a treatment day takes place (indicated with 1) or not (indicated with 0) for patient classes 1 to 5, whilst each binary number stands for a weekday from Monday to Friday. Classes 6 to 9 each need four treatments with differing earliest brakes on weekdays, as follows: Class 1 pattern (1,1,1,1,1), class 2 pattern (0,0,1,1,1) , class 3 pattern (1,1,0,0,0), class 4 pattern (1,1,1,0,0), class 5 pattern (1,1,1,1,0), class 6 with earliest brake on Thursday, class 7 with earliest break on Wednesday, class 8 with earliest break on Tuesday, and class 9 with earliest break on Monday. The patient groups and classes are part of the feature set, as explained in Section 4.3.1.

Each appointment within a treatment room is divided into three consecutive phases, as stated by Vogl et al. (2019): First, the patient needs a in-room preparation time in one of the three treatment rooms before executing the treatment. Second, the irradiation process is conducted, resulting in occupying the beam resource as well as the treatment room. And third, after the session, the treatment room is still occupied due to a teardown time while the patient is exiting the room. Vogl et al. (2020) describe the transformation of the GA solution representation to the baseline schedule in which the starting times for the in-room preparations are determined. As idle time shall not occur between the phases, once the starting time for the in-room preparation is set, the starting times for the subsequent phases are dependent on the starting time of the in-room preparation. Within this stochastic approach, the waiting time extension can be caused in three different

ways: The patient needs longer in the pre-irradiation phase due to limited mobility and therefore delays the start of the irradiation and blocks the treatment room longer than scheduled. Then, in the irradiation phase itself, the patient might move during this phase resulting in the necessity of repeating or interrupting the process. Last, a machine breakdown might occur, such that maintenance needs to be performed. In each of these cases the appointment duration, and hence the patients' waiting times, are extended.

To estimate the expected average waiting time, the sample average approximation, introduced in Section 4.1, is replaced by applying ordinary least squares (OLS), different linear regression methods with multiple regressors, such as lasso (LR) and ridge regression (RR), the nonlinear regression methods support vector regression (SVR) and kernel ridge regression (KRR), and a multilayer perceptron (MLP) artificial neural network.

4.3 Learning-based Waiting Time Estimation

As already mentioned in Section 4.1, the two stochastic variants of a basic sample average approximation and a combination with simple linear regression to estimate the average patient waiting time by Vogl et al. (2020) are replaced with the proposed learning-based waiting time estimation approach. Instead of a single feature linear regression ("ion beam idle time" is utilized as regressor to predict the regressand), the waiting time estimation strategy is replaced with commonly used regression analysis methods and an artificial neural network. For this purpose, new features besides the "ion beam idle time" were applied and tested in a feature selection approach. These methods, as well as the features and regression analysis, will be discussed in this section in detail.

4.3.1 Introducing New Features and Defining the Feature Sets with Feature Selection

The first step within the process of waiting time estimation is setting up a list of features used as regressors in the regression models. Therefore, the resulting schedules generated with the deterministic approach of Vogl et al. (2019) and stochastic approach of Vogl et al. (2020) were examined in detail to determine factors that might influence the waiting time estimation, leading to a list of 22 features covering the aspect of idle times, patient requirements and treatment patterns, ion beam configurations, scheduled starting times, and room assignment cycle occurrences, as follows:

- Feature 1-4: Idle time of beam, room 1, 2, and 3
- Feature 5-8: Idle period count of beam, room 1, 2, and 3

- Feature 9-12: Mean starting time of patient group 1, 2, 3, and 4
- Feature 13-16: Number of switches between different patient groups on beam, room 1, 2, and 3
- Feature 17-20: Number of switches between different patient classes on beam, room 1, 2, and 3
- Feature 21: Total beam type switches
- Feature 22: Average room occurrence cycle lengths

The idle times describe the amount of time the respective beam and rooms are not occupied. The mean starting times determine the average starting time for each of the patient group, whilst the total beam switches states the number of how often a switch from beam type 1 to 2 was conducted. The cycle length value states the recurrence of a room number within a treatment sequence. The ion beam can be utilized only in one room at the same time. Therefore, a possible ion beam assignment might be first to room 1, then room 2, room 3, then the same sequence of 1, 2, 3 again. In this case, the cycle length would be 3, as each room is occupied every third time. Hence, the cycle length is a measure for the "distance" of a room's occupancy within the scheduling sequence.

The full list of the 22 features was used in the experiments as feature set 1. Out of this list, two more feature sets were defined. The second set only contains the highest positive or negative correlating feature regarding the respective instance. An exemplary heatmap of a correlation matrix for an instance including all of the 22 features and their correlation to each other is depicted in Fig. 4.2. Note, that only the correlation between a single regressor with the regressand "average waiting time" is taken into consideration. Here, "idle beam time" would be the chosen feature, as its correlation value of -0.7 is the highest (absolute value) among all the regressors. The scale on the right hand side indicates the level of correlation from deep read (highly positive correlated) to a light red (highly negative related). The third feature set was determined with the SelectKBest method. Here, the feature selection is processed by ordering each of these features according to the χ^2 Formula in a descending order and choose the eight most important ones.

4.3.2 Methods for and Implementation of the Regression Analysis

Multiple linear regression, as a widely used regression method, is conducted to find a relationship between the input and output variables that is assumed to be linear. For the experiments, we make use of the OLS, that can be declared as a rather simple method as only the difference between the predicted value and

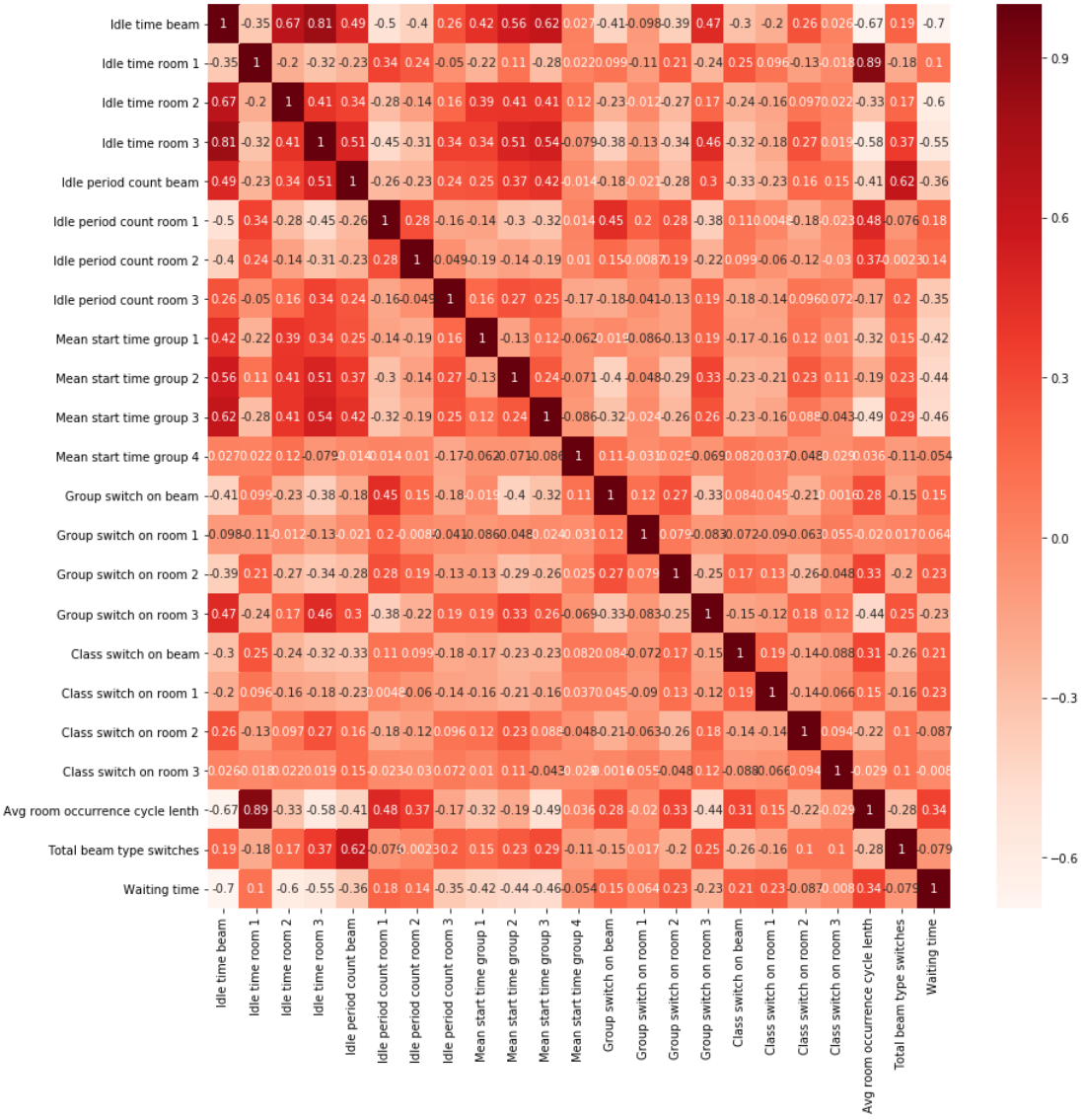

Figure 4.2: Exemplary feature correlation heatmap

the actual data is calculated. Compared to the OLS method, the LR and RR extend the OLS equation by adding a penalty through the L1 (for LR) and L2 (for RR) regularization. To estimate the waiting time, the desired combination of variance and bias is both to be as low as possible, as a high bias indicates underfitting. In contrast, high variance results in overfitting the model. The LR and RR method try to define a tradeoff between a slightly higher bias, but therefore a reduced value regarding the variance. This procedure is called regularization. As the OLS only tries to minimize the sum of the residual squares, the regularization procedure also adds a penalty term, which is applied to the magnitude of coefficients. For both the LR and RR, the strength of this penalty term is controlled by the parameter α. Whilst the LR adds this penalty in form of square of the magnitude (L2 regularization), the RR makes use of absolute values (L1 regularization). In order to compare the effect of the three methods regarding their ability to predict the average patient waiting time, the OLS, LR, and RR were included in the experiments. Further information about the regression techniques is provided by Hastie et al. (2009) and by Hoerl (1962) regarding the RR.

In contrast, the nonlinear regression approaches assume a nonlinear relationship between regressors and regressand. The SVR will be conducted in the experiments and is based on the support vector machine (SVM) method. Originally, the SVM method is mainly used for classification problems, but it can also be applied to regression problems, as stated by Vapnik (1998). Within the SVM classification procedure, the data points are classified by applying a hyperplane such that in an optimal case, the separating margin between the hyperplane and the data points is maximal in an N-dimensional space, whereat N equals the number of features. In difference to SVM, the SVR predicts continuous instead of discrete categorical variables. The main advantage of the SVR compared to a more simple regression method is its approach of not only minimizing the error rate but also making use of a threshold to fit the error between two boundary lines. The KRR, as described by Cristianini and Shawe-Taylor (2000), is used as second nonlinear regression method. The KRR differs to the ordinary RR in making use of kernel functions as the so called “kernel trick”. The advantage of the kernel trick is a reduced computational complexity, which results from skipping the computation of the coordinates when the data is transformed to a high-dimensional feature space. As all of these approaches offer advantages compared to the multiple linear regression methods introduced above, they were also included in the experiments.

Finally, a learning approach was implemented in terms of a MLP artificial neural network. Instead of only a single layer, the MLP makes use of multiple layers with an input and output layer, and several hidden layers in between. The input data is fed forward through these hidden layers passing the neurons within the layers, which, in turn, use a nonlinear activation function to predict the output. In this case, the input data equals the regressors of the regression methods and the output corresponds to the regressand, the waiting time that is ought to be

predicted. A more detailed description of the MLP was given by Hastie et al. (2009).

For the kernel-based approaches (SVR and KRR), as well as for the RR and MLP, the features were transformed by scaling their data. The range for the values in these features are set to 0 to 1 without changing the shape of the original distribution. Furthermore, the grid search method (hyper parameter tuning) was conducted to find the best performing aforementioned α for LR, RR, and KRR as regularization parameter within the given parameter ranges. For SVR, the parameter γ was tested in the grid search procedure, which is used to vary the support vectors' influence regarding a single training sample. In addition, the parameter "C" for the SVR is varied, that determines the weight of penalties for samples within the aforementioned margin. Finally, the parameter γ for KRR is used to define the fitting to the training data set, whereas a higher γ causes overfitting to the training set.

4.3.3 Experimental Set-up

The experiments were conducted on randomized instance scenarios which differ in number of patients (50, 70, or 100). For each number or patients 25 instances were generated containing different probability distributions for the patient groups irradiation activity duration, depicted in Fig. 4.3. These 25 instances, in turn, are split into two different patient group distribution mixes determining to which group a patient belongs. The distribution of the four different patient groups for instances 1 to 16 are chosen randomly with probabilities [%] {43, 29, 22, 6} for each of the four groups. For instances 17 to 25 the distributions are fixed, as listed in Table 4.2. For each of the instances, the patients are assigned randomly to a certain class, to one of the three treatment rooms, and to one of the two beam types. The original data set is split into a training set of 91% and a test set of 9% resulting in more than 800 training samples and more than 11.000 test samples. The implementation was conducted with `scikit-learn` version 0.21.3 in combination with Python.

4.4 Preliminary Results

The R^2 and mean squared error (MSE) results, as well as their respective variance for the instances with mixed distributions are presented in Table 4.1. The average values contain the results for all instances including all patient number sets (50, 70, and 100) for the different regression approaches and the whole feature set. It can be stated that using all features or the SelectKBest feature selection method (feature set 1 and 3) always leads to better results than using the highest correlated single feature approach (feature set 2). Especially the KRR method

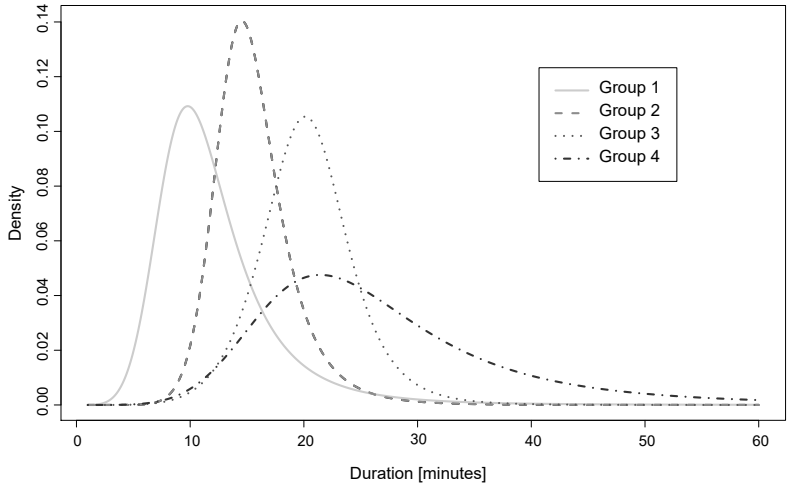

Figure 4.3: Irradiation activity duration distribution of the four patient groups

with feature set 1 achieves best results with a R^2 value of 0.74 and the lowest MSE of 587. However, the regression approaches do not differ significantly from each other. This also holds for the MSE comparison, where the feature set 1 and 3 always perform better than feature set 2 without differing significantly between the regression analysis methods. To conclude, each of the approaches in combination with feature set 1 leads to promising results regarding their ability to estimate the average waiting time.

In contrast to the results above, for the fixed distribution instances some of the regression approaches deliver excellent results regarding the reported R^2 values, as shown in Table 4.2. Note, that here the values are calculated for every instance (e.g., {100, 0, 0, 0}) separately containing the three results for each of the patient numbers (50, 70, and 100). The average results for feature set 1 range from 0.827 to 0.833, those for feature set 2 from 0.662 to 0.673, and those for feature set 3 from 0.772 to 0.781. With the results at hand, two major findings can be stated: First, it can be observed again that the feature set 1 and 2 perform better than the single feature approach of set 2 on all instances for all approaches. And second, instances {0, 0, 100, 0} and {0, 0, 0, 100} achieve remarkable results from 0.814 up to 0.976 for all approaches. A possible explanation for these near perfect 0.976 results of these two instances might be the probability distribution, as shown in Fig. 4.3. The fact that these instances contain only patients of group 3 or 4 leads to the assumption that the regression approaches are able to handle patient groups with a higher average value for the duration better than those with lower values (patient groups 1 and 2). Furthermore, nearly all of the best results are observed for the KRR method with feature set 1 and an average R^2 value of 0.833, although they do not differ significantly from other methods. This is also underlined by the values of the MSE results, listed in Table 4.2. Here, the MSE results differ

Table 4.1: Variance, MSE, and R^2 results of the regression analysis for instances with mixed distribution.

	Average	Feature set		
		1	2	3
SVR	Variance R^2	0.005	0.021	0.008
	R^2	0.702	0.434	0.575
	Variance MSE	76385	264676	162585
	MSE	594	1118	852
KRR	Variance R^2	**0.004**	0.023	0.008
	R^2	**0.704**	0.411	0.574
	Variance MSE	70938	291942	159149
	MSE	**587**	1163	853
OLS	Variance R^2	**0.004**	0.021	0.008
	R^2	0.703	0.433	0.569
	Variance MSE	**69412**	267471	154190
	MSE	587.3	1120.7	858.6
RR	Variance R^2	**0.004**	0.021	0.008
	R^2	0.703	0.433	0.568
	Variance MSE	69500	267945	154621
	MSE	587.5	1121.1	859.2
LR	Variance R^2	**0.004**	0.021	0.008
	R^2	0.703	0.433	0.568
	Variance MSE	69670	267645	155334
	MSE	588	1121	859
MLP	Variance R^2	0.005	0.021	0.008
	R^2	0.703	0.436	0.572
	Variance MSE	71446	262006	159991
	MSE	589	1115	856

Bold values indicate best values

significantly between the feature sets and instances, ranging between an average MSE of 458 for KRR with feature set 1 to 897 of OLS, RR, and LR with feature set 2. As for the instances with mixed distributions in Table 4.1, the KRR method here also yields most of the best results among all the other methods with feature set 1 and the lowest MSE average value of 458.

In summary, for instances with 100%,0% mixtures the approaches yield results up to 0.976, whilst for the other instances competitive values of up to 0.847 can be observed. Furthermore, feature set 1 in combination with the KRR technique is the most promising set as all values for this combination range from 0.714 for the {100, 0, 0, 0} instance to the 0.976 value of the {0, 0, 0, 100} instance. It can also be concluded that feature set 1 can be applied to replace the sample average approximation with partly excellent results regarding the waiting time estimation. Hence, a tendency within the different feature approaches or between the non-100%,0% mixtures for which approach might be applied to which instance cannot be identified.

4.5 Conclusion and Future Research

As outlined in the introduction, this technical report aimed at replacing a complex sample average approximation for estimating the patients waiting time with regression analysis methods in the field of appointment scheduling. Therefore, 22 new features were implemented to be used as regressors. Out of this list of features, three different feature sets were determined containing all features, the highest correlating single feature, and eight features selected with the SelectKBest method. These sets, in turn, were applied to different instances with differing patient mix distributions and tested using several regression analysis methods and an artificial neural network.

The major finding regarding the mixed distribution instances contains the fact, that both the multi feature approaches perform better than the single feature approach with only the highest correlating feature. Hence, all approaches yield mixed results. The more promising performance is achieved on the fixed distribution instances with results up to 0.976. The fact of better performing multi feature approaches also holds within this instance setting and can be defined as the second overall major finding. The overall results for all the instances yield a top performance level so that it can be stated, that the applied methods indeed are able to replace the sample average approximation, as proposed earlier. Especially the KRR method in combination with the feature set 1 turned out to achieve most of the best results regarding the R^2 and MSE values.

Future research might consider to implement the proposed replacement learning method with multiple regression in the GA optimization approach of Vogl et al. (2020) for radiotherapy appointment scheduling. The possible application area for

Table 4.2: R^2 results for the regression analysis for instances with fixed distributions

Feature set	Instance [%]	SVR	KRR	OLS	RR	LR	MLP
				[R^2]			
1	{100, 0, 0, 0}	0.713	**0.714**	0.710	0.710	0.710	0.713
	{0, 100, 0, 0}	0.724	**0.731**	0.718	0.719	0.718	0.729
	{0, 0, 100, 0}	0.912	**0.914**	0.903	0.903	0.903	0.913
	{0, 0, 0, 100}	**0.976**	**0.976**	**0.976**	**0.976**	**0.976**	0.975
	{25, 25, 25, 25}	**0.819**	0.818	0.813	0.813	0.813	0.813
	{40, 20, 20, 20}	0.847	**0.853**	0.844	0.844	0.842	0.853
	{20, 40, 20, 20}	**0.840**	**0.840**	0.839	0.839	0.839	0.839
	{20, 20, 40, 20}	0.830	**0.832**	0.828	0.828	0.827	0.828
	{20, 20, 20, 40}	0.820	0.815	0.814	0.814	0.815	0.819
	Average	0.831	**0.833**	0.827	0.827	0.827	0.831
2	{100, 0, 0, 0}	0.338	0.348	0.344	0.344	0.344	0.344
	{0, 100, 0, 0}	0.368	0.387	0.343	0.343	0.343	0.370
	{0, 0, 100, 0}	0.836	0.837	0.814	0.814	0.814	0.829
	{0, 0, 0, 100}	0.969	0.969	0.968	0.968	0.968	0.968
	{25, 25, 25, 25}	0.697	0.672	0.694	0.694	0.694	0.694
	{40, 20, 20, 20}	0.759	0.760	0.759	0.759	0.759	0.760
	{20, 40, 20, 20}	0.692	0.671	0.678	0.678	0.678	0.678
	{20, 20, 40, 20}	0.734	0.736	0.731	0.731	0.731	0.734
	{20, 20, 20, 40}	0.663	0.607	0.623	0.623	0.623	0.657
	Average	0.673	0.665	0.662	0.662	0.662	0.671
3	{100, 0, 0, 0}	0.587	0.605	0.588	0.588	0.588	0.600
	{0, 100, 0, 0}	0.683	0.692	0.676	0.676	0.675	0.682
	{0, 0, 100, 0}	0.898	0.900	0.886	0.886	0.886	0.898
	{0, 0, 0, 100}	0.973	0.973	0.973	0.973	0.973	0.973
	{25, 25, 25, 25}	0.756	0.756	0.751	0.750	0.750	0.748
	{40, 20, 20, 20}	0.794	0.806	0.797	0.798	0.798	0.805
	{20, 40, 20, 20}	0.755	0.761	0.751	0.751	0.750	0.748
	{20, 20, 40, 20}	0.773	0.775	0.772	0.772	0.772	0.771
	{20, 20, 20, 40}	0.768	0.764	0.757	0.758	0.757	0.767
	Average	0.776	0.781	0.772	0.772	0.772	0.777

Bold values indicate best values

Table 4.3: MSE results for the regression analysis for instances with fixed distributions

Feature set	Instance [%]	SVR	KRR	OLS	RR	LR	MLP
				[MSE]			
1	{100, 0, 0, 0}	813	**808**	817	817	817	812
	{0, 100, 0, 0}	776	753	780	780	780	**752**
	{0, 0, 100, 0}	464	**462**	542	541	542	470
	{0, 0, 0, 100}	99	99	99	99	**98**	103
	{25, 25, 25, 25}	409	**400**	409	411	410	410
	{40, 20, 20, 20}	386	**374**	394	392	398	375
	{20, 40, 20, 20}	450	**449**	453	453	454	453
	{20, 20, 40, 20}	474	**466**	476	476	478	475
	{20, 20, 20, 40}	304	306	304	305	**303**	299
	Average	464	**458**	475	475	476	461
2	{100, 0, 0, 0}	1760	1767	1742	1744	1743	1743
	{0, 100, 0, 0}	1647	1590	1654	1654	1654	1637
	{0, 0, 100, 0}	871	869	1007	1007	1007	877
	{0, 0, 0, 100}	130	129	132	132	132	132
	{25, 25, 25, 25}	684	729	681	681	681	680
	{40, 20, 20, 20}	637	635	636	636	636	634
	{20, 40, 20, 20}	838	866	892	893	893	892
	{20, 20, 40, 20}	754	748	758	758	758	751
	{20, 20, 20, 40}	542	678	573	573	573	550
	Average	874	890	897	897	897	877
3	{100, 0, 0, 0}	1214	1140	1186	1187	1186	1146
	{0, 100, 0, 0}	881	851	893	893	895	886
	{0, 0, 100, 0}	536	536	638	638	638	538
	{0, 0, 0, 100}	114	112	114	114	114	114
	{25, 25, 25, 25}	557	556	560	563	561	567
	{40, 20, 20, 20}	538	508	528	527	527	511
	{20, 40, 20, 20}	700	687	706	705	708	711
	{20, 20, 40, 20}	614	603	612	612	611	612
	{20, 20, 20, 40}	412	408	413	413	413	410
	Average	618	600	628	628	628	611

Bold values indicate best values

the regression analysis introduced in this technical report is not limited only in the field of appointment scheduling, but can also be applied to different stochastic scheduling problems.

Chapter 5

Conclusion

The thesis was parted into three major parts. The first and second phase contained topics motivated by the arising possibilities in the field of "smart production" and "Industry 4.0". Therefore, well-established learning approaches were applied to a industrial shop floor configuration of an industrial partner. In the first phase of the thesis, the goal was to implement a machine learning approach for generating decision trees used as priority rules for dispatching jobs based on the CPS of our industrial partner. To achieve solutions of high quality for training purposes, a CP formulation was set up. The solved instances, in turn, were used to build the DT and RF. But as the decision for material movements in the respective shop floor configuration contained two elements (choose job and the machine it should be assigned to), the most important part was developing a method to transform the binary decision tree output into a decision. This problem was solved with the yes-no voting. The experiments on different instance scenarios indicated, that the approaches perform well compared to simple priority rules on scenarios containing more jobs. It can be stated, that the DT and RF approach are ready-to-use for an implementation in the private sector with the objectives of makespan or tardiness minimization.

The second phase included a different learning approach in form of a GP implementation with a single- and multi-tree representation and an iterative approach to improve the baseline schedules. In experiments with benchmark instances, the method was tested regarding its ability to reduce the overall makespan significantly. As the approach succeeded in doing so, it was applied to instance scenarios based on the real-world adaption of the industrial partner's shop floor configuration. It can be stated that the GP yielded excellent results regarding the objective of reducing the makespan. It was proven that the learning approaches based on the CPS of the industrial partner were able to undercut other optimization processes in terms of reducing the computational time significantly and also achieving competitive results regarding the objective to reduce the overall makespan or tardiness in comparison to basic priority rules. These methods were tested on randomized problem data, but also on a real-world adapted problem setting. The observed results lead to the conclusion, that the DT, RF, and the GP approaches might be implemented in the private sector in decision support systems for production processes due to their easy-to-implement and easy-to-apply manner.

As already mentioned in Section 1, missing data of the second industrial partner lead to a switch from a stochastic scheduling approach with machine breakdowns

to a learning approach in the framework of appointment scheduling in the third phase. It was shown that a complex procedure of average waiting time estimation within a radiotherapy appointment scheduling framework can be replaced with the proposed learning approach making use of 22 features of regressors in a multiple regression analysis approach. The results indicate that the replacement learning approach offers a competitive alternative to the more complex sample average approximation used by Vogl et al. (2020) and therefore might be implemented in the radiotherapy appointment optimization scheduling procedure to reduce the level of complexity considerably.

Bibliography

Balas, E. (1969). Machine sequencing via disjunctive graphs: An implicit enumeration algorithm. *Operations Research*, 17(6):pages 941–957.

Benda, F., Braune, R., Doerner, K. F., and Hartl, R. F. (2019). A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. *OR Spectrum*, 41(4):pages 871–893.

Blockeel, H. and Struyf, J. (2003). Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research*, 3:pages 621–650.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):pages 157–183.

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

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):pages 5–32.

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

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Doh, H.-H., Yu, J.-M., Kwon, Y.-J., Shin, J.-H., Kim, H.-W., Nam, S.-H., and Lee, D.-H. (2014). Decision tree based scheduling for flexible job shops with multiple process plans. *International Journal of Mechanical, Aerospace, Industrial, Mechatronic and Manufacturing Engineering*, 8(3):pages 621–627.

Froehlich, G., Kiechle, G., and Doerner, K. (2019). Creating a multi-iterative-priority-rule for the job shop scheduling problem with focus on tardy jobs via genetic programming. volume 11353, pages 64–77. Springer Berlin Heidelberg New York.

Gao, J., Sun, L., and Gen, M. (2008). A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35(9):pages 2892–2907.

Hall, N. G. and Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):pages 510–525.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *Linear Methods for Regression*, pages 43–99. Springer New York, New York, NY.

Hein, F., Almeder, C., Figueira, G., and Almada-Lobo, B. (2018). Designing new heuristics for the capacitated lot sizing problem by genetic programming. *Computers & Operations Research*, 96:pages 1–14.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282.

Hoerl, A. E. (1962). Application of ridge analysis to regression problems. In *Chemical Engineering Progress*, volume 58, pages 54–59.

Jakobović, D., Jelenković, L., and Budin, L. (2007). Genetic programming heuristics for multiple machine scheduling. In Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., and Esparcia-Alcázar, A. I., editors, *Genetic Programming*, pages 321–330. Springer Berlin Heidelberg.

Jurisch, B. (1992). *Scheduling jobs in shops with multi-purpose machines*. Dissertation, Department of Mathematics/Computer Science of the University of Osnabrück.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Kurz, M. E. and Askin, R. G. (2004). Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159(1):66–82.

Lawrence, S. (1984). Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. *Graduate School of Industrial Administration, Carnegie-Mellon University*.

Leisten, R. (1990). Flowshop sequencing problems with limited buffer storage. *International Journal of Production Research*, 28(11):pages 2085–2100.

Li, X. and Olafsson, S. (2005). Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6):pages 515–527.

Mainieri, G. B. and Ronconi, D. P. (2013). New heuristics for total tardiness minimization in a flexible flowshop. *Optimization Letters*, 7(4):pages 665–684.

Mascis, A. and Pacciarelli, D. (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):pages 498–517.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):pages 3–20.

Monostori, L. (2003). AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. *Engineering Applications of Artificial Intelligence*, 16(4):pages 277–291. Intelligent Manufacturing.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2013). Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology*, 67(1):pages 85–100.

Olafsson, S. and Li, X. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, 128(1):pages 118–126. Integrating the Global Supply Chain.

Panwalkar, S. S. and Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, 25(1):pages 45–61.

Pezzella, F., Morganti, G., and Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):pages 3202–3212.

Pickardt, C. W., Hildebrandt, T., Branke, J., Heger, J., and Scholz-Reiter, B. (2013). Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics*, 145(1):pages 67–77.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, 4th edition.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):pages 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Ruiz, R., Maroto, C., and Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165(1):pages 34–54.

Scheibenpflug, A., Beham, A., Kommenda, M., Karder, J., Wagner, S., and Affenzeller, M. (2015). Simplifying problem definitions in the heuristiclab optimization environment. In *Companion Publication, Genetic and Evolutionary Computation Conference, GECCO'15*, pages 1101–1108.

Shahzad, A. and Mebarki, N. (2012). Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 25(6):pages 1173–1181.

Tay, J. C. and Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):pages 453–473.

Tran, T. T., Araujo, A., and Beck, J. C. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):pages 83–95.

Vapnik, V. (1998). *The Support Vector Method of Function Estimation*, pages 55–85. Springer US, Boston, MA.

Vepsalainen, A. P. J. and Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Manage Sci*, 33(8):pages 1035–1047.

Vogl, P., Braune, R., and Doerner, K. F. (2019). Scheduling recurring radiotherapy appointments in an ion beam facility. *Journal of Scheduling*, 22(2):pages 137–154.

Vogl, P., Braune, R., Gutjahr, W. J., and Doerner, K. F. (2020). Dynamic-stochastic radiotherapy appointment scheduling. Working paper, Department of Business Decisions and Analytics, University of Vienna.

Wuest, T., Weimer, D., Irgens, C., and Thoben, K.-D. (2016). Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):pages 23–45.

Xing, L.-N., Chen, Y.-W., Wang, P., Zhao, Q.-S., and Xiong, J. (2010). A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing Journal*, 10(3):pages 888–896.

Zhang, F., Mei, Y., and Zhang, M. (2018). Genetic programming with multi-tree representation for dynamic flexible job shop scheduling. In Mitrovic, T., Xue, B., and Li, X., editors, *AI 2018: Advances in Artificial Intelligence*, pages 472–484, Cham. Springer International Publishing.

Appendix for Chapter 2

Appendix A: A Mixed-integer Programming Formulation

The problem is specified by a set $\mathcal{O}$ of orders, each of which is partitioned in a set of jobs, where $\mathcal{I}^o$ denotes the set of jobs for a particular order $o \in \mathcal{O}$. The set union $\mathcal{I} = \bigcup_{o \in \mathcal{O}} \mathcal{I}^o$ of jobs has to be processed on a series of processing stages, denoted by the set $\mathcal{J}$, following a flow production scheme. Hence, the stages have to be always visited in the same order with the possibility of leaving out one or more stages, however. Each stage $j \in \mathcal{J}$ is made up of a set of parallel machines $\mathcal{M}^j$ that are all able to accomplish the same production step. Consequently, any of the available machines may be chosen for processing a job. The processing time p_{ijm} of a job $i \in \mathcal{I}$ on stage $j \in \mathcal{J}$ can be different for every machine $m \in \mathcal{M}^j$. All machines are unitary resources, in the sense that they can only process one job at a time. Furthermore, no preemption is allowed. Hence, once started, job processing cannot be interrupted until it is finished.

Table 1: Indices used in the MIP model formulation

o	Order
i	Job
j	Stage
m	Machine
b	Buffer/Processing slot
f	Setup family

Job processing on machines is subject to sequence-dependent setup times, based on *family setups*. A setup family is a set of jobs such that no changeover work is necessary when switching between two jobs of the same family. Each job $i \in \mathcal{I}$ belongs to exactly one setup family. Let $\mathcal{F}$ be the set of setup families. Then $s_{ff'}^{jm}$ denotes the changeover time for switching between setup families $f, f' \in \mathcal{F}$ on stage $j \in \mathcal{J}$ and machine $m \in \mathcal{M}^j$.

Material transport between stages requires a dedicated transport resource that can only handle one job at a time. The transport duration depends on the travel distance between the respective stages.

The proposed MIP model uses "building blocks" from various existing mathematical formulations in the context of flow shop scheduling with blocking or setup setup times. The notion of departure times to properly cover the blocking behavior can be found in Pinedo (2002). Binary variables $Y_{ii'jm}$, indicating whether job $i \in \mathcal{I}$ *directly* precedes another job $i' \in \mathcal{I}$ on stage $j \in \mathcal{J}$ and machine $m \in \mathcal{M}^j$, are used, for example, by Kurz and Askin (2004). Variables with the same semantics are also used on the transport resource. An overview of all decision variables is given in Table 4. The remaining symbols are listed in Tables 1, 2 and 3, representing indices, sets and parameters (constants), respectively. Depending on whether we use the maximum completion time (makespan) or, based on order due dates δ_o, the total tardiness as an objective, we either have

$$\text{Minimise} \quad C_{\max} \tag{1}$$

or

$$\text{Minimise} \quad \sum_{o \in \mathcal{O}} \max(C_o - \delta_o, 0) \tag{2}$$

subject to

$$\sum_{m \in \mathcal{M}^j} X_{ijm} = 1 \quad \forall i \in \mathcal{I}, \forall j \in \mathcal{J}^i, \tag{3}$$

$$t''_{ijm} + c''_{ijm} \leq M \cdot X_{ijm} \quad \begin{array}{l} \forall i \in \mathcal{I}, \\ \forall j \in \mathcal{J}^i, \\ m \in \mathcal{M}^j, \end{array} \tag{4}$$

$$\sum_{(i,i'') \in \mathcal{P}^j} Y_{ii''m} = X_{ijm} \quad \begin{array}{l} \forall j \in \mathcal{J}, \\ \forall m \in \mathcal{M}^j, \\ \forall i \in \tilde{\mathcal{I}}^j \cup \{0\}, \end{array} \tag{5}$$

$$\sum_{(i'',i) \in \mathcal{P}^j} Y_{i''im} = X_{ijm} \quad \begin{array}{l} \forall j \in \mathcal{J}, \\ \forall m \in \mathcal{M}^j, \\ \forall i \in \tilde{\mathcal{I}}^j \cup \{n+1\}, \end{array} \tag{6}$$

$$\begin{aligned} t'''_{i'jmb} \geq d'''_{ijmb} + s^{jm}_{f(i),f(i')} \cdot Y_{ii'jm} \\ -M \cdot (1 - Y_{ii'jm}) \end{aligned} \quad \begin{array}{l} \forall j \in \mathcal{J}, \\ \forall (i,i') \in \tilde{\mathcal{P}}^j, \\ \forall m \in \mathcal{M}^j, \\ \forall b \in \hat{\mathcal{B}}^{jm}, \end{array} \tag{7}$$

$$t'''_{i'jmb} \geq d'''_{ijmb} - M \cdot (1 - Y_{ii'jm}) \quad \begin{array}{l} \forall j \in \mathcal{J}, \\ \forall m \in \mathcal{M}^j, \\ \forall (i,i') \in \tilde{\mathcal{P}}^j, \\ \forall b \in \overline{\mathcal{B}}^{jm} \cup \underline{\mathcal{B}}^{jm}, \end{array} \tag{8}$$

$$t'''_{ijmb} = d'''_{ijm(b-1)} \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \\ \forall m \in \mathcal{M}^j, \\ \forall b \in \{2, \ldots, |\mathcal{B}^{jm}|\},\end{array} \tag{9}$$

$$c'''_{ijmb} \geq t'''_{ijmb} + p_{im} - M \cdot (1 - X_{ijm}) \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \\ \forall m \in \mathcal{M}^j, \forall b \in \hat{\mathcal{B}}^{jm},\end{array} \tag{10}$$

$$c'''_{ijmb} \geq t'''_{ijmb} - M \cdot (1 - X_{ijm}) \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \forall m \in \mathcal{M}^j, \\ \forall b \in \overline{\mathcal{B}}^{jm} \cup \underline{\mathcal{B}}^{jm},\end{array} \tag{11}$$

$$d'''_{ijmb} \geq c'''_{ijmb} \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \\ \forall m \in \mathcal{M}^j, \forall b \in \mathcal{B}^{jm},\end{array} \tag{12}$$

$$t''_{ijm} = t'''_{ijm1} \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \\ \forall m \in \mathcal{M}^j,\end{array} \tag{13}$$

$$t'_{ij} = \sum_{m \in \mathcal{M}^j} t''_{ijm} \quad \forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \tag{14}$$

$$c''_{ijm} = c'''_{ijm|\mathcal{B}^{jm}|} \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \\ \forall m \in \mathcal{M}^j,\end{array} \tag{15}$$

$$c'_{ij} = \sum_{m \in \mathcal{M}^j} c''_{ijm} \quad \forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \tag{16}$$

$$d''_{ijm} = d'''_{ijm|\mathcal{B}^{jm}|} \quad \begin{array}{l}\forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \\ \forall m \in \mathcal{M}^j,\end{array} \tag{17}$$

$$d'_{ij} = \sum_{m \in \mathcal{M}^j} d''_{ijm} \quad \forall j \in \mathcal{J}, \forall i \in \tilde{\mathcal{I}}^j, \tag{18}$$

$$t'_{ij'} = d'_{ij} \quad \forall i \in \mathcal{I}, \forall (j, j') \in \mathcal{Q}^i, \tag{19}$$

$$c''_{i\bar{j}(i)m} = d''_{i\bar{j}(i)m} \quad \forall i \in \mathcal{I}, \forall m \in \mathcal{M}^{\bar{j}(i)}, \tag{20}$$

$$C_o = c'_{i\bar{j}(i)} \quad \forall o \in \mathcal{O}, \forall i \in \mathcal{I}^o, \tag{21}$$

$$C_{\max} \geq C_o \quad \forall o \in \mathcal{O}, \tag{22}$$

$$t'_{ij} = \bar{c}_{ij} \quad \forall (i, j) \in \mathcal{R}, \tag{23}$$

$$\bar{t}_{ij'} = d'_{ij} \quad \forall i \in \mathcal{I}, \forall (j, j') \in \mathcal{Q}^i, \tag{24}$$

$$\sum_{(i',j') \in \mathcal{R}^*, (i',j') \neq (i,j)} Z_{iji'j'} = 1 \quad \forall (i, j) \in \mathcal{R} \cup \{(0, 0)\}, \tag{25}$$

$$\sum_{(i',j') \in \mathcal{R}^*, (i',j') \neq (i,j)} Z_{i'j'ij} = 1 \quad \forall (i, j) \in \mathcal{R} \cup \{(n+1, |\mathcal{J}|)\}, \tag{26}$$

$$\bar{t}_{i'j'} \geq \bar{c}_{ij} + \tau_{j\pi(i',j')} \cdot Z_{iji'j'} - M \cdot (1 - Z_{iji'j'}) \quad \forall (i, j), (i', j') \in \mathcal{R}, (i', j') \neq (i, j), \tag{27}$$

$$\bar{c}_{ij} \geq \bar{t}_{ij} + \tau_{\pi(i,j),j} \quad \forall (i, j) \in \mathcal{R}. \tag{28}$$

As far as the formulation is concerned, Constraint (3) ensures that exactly one machine is chosen for each job on a stage. Constraint (4) forces a job's starting/arrival and completion times to be zero whenever the job is *not* assigned to the respective machine. Constraints (5) and (6) require that each job has exactly one successor and one predecessor on the machine to which it is assigned. Temporal relations between pairs of jobs $(i, i') \in \tilde{\mathcal{P}}^j$ induced by setup times are enforced by Constraint (7). Note that setup can start once the predecessor job i has "departed," that is, when it has left the machine. Disjunctive constraints are also established for the buffer slots, forbidding that more than one job occupies such a slot at the same time (Constraint (8)). Buffer occupancy times are variable and only determined by transitions between the individual buffer slots, as described by Constraint (9). The slots of a machine are numbered consecutively, hence, $b = 1$ corresponds to the first buffer slot, while $b = |\mathcal{B}^{jm}|$ indicates the last transport slot of machine m on stage j.

Slot-based job completion times are linked to the corresponding starting/arrival time variables via Constraints (10) and (11). Constraints (12) ensures that a job cannot depart from a slot before its completion. Constraints (13) - (20) are linking constraints, putting start/arrival, completion, and departure time variables at different levels into relation with one another.

Order completion times and the maximum overall completion time are covered by Constraints (21) and (22). The link between the arrival time of a job on a stage and the end of the transport request moving it to that stage is established by Constraint (23). Complementary to that, the departure time of a job on stage j is synchronised with the start of the transport request moving it to the next stage j' (Constraint (24)). Note that the stage indices of all transport-related decision variables denote the *destination* node of the respective transport request. Since for each stage of a job $i \in \mathcal{I}$, there is at most one successor stage, described by index pairs in the set $\mathcal{Q}^i$, this notation scheme uniquely identifies all transport activities for that job. Analogously to the machines, each transport request has exactly one successor and one predecessor, indicated by Constraints (25) and (26), respectively. Constraint (27) implies that the transport resource may have to be re-positioned between two successive transport activities, in case the destination stage of the previous request and the source stage of the current one are different. Constraint (28) sets the completion time for a transport request that actually moves job.

When using the makespan objective (1), we also add two different variants of inequalities that both impose lower bounds on the maximum completion time $C_{\max}$ and thus help tighten the LP relaxation of the formulation. The idea of those constraints is to explicitly calculate resource completion times, as proposed by Tran et al. (2016), for example.

To introduce the first group of inequalities, one for each stage and machine, let us denote by r_{ij} the *head* of the processing activity associated with job $i \in \mathcal{I}$ on stage $j \in \mathcal{J}$. The head of an activity equals the *minimum* time required to

finish all predecessor stages for a particular job/stage combination. The quantity r_{ij} can be computed recursively as follows: for the first stage j^* of a job, we get $r_{ij^*} = 0$ ($j^* \in \mathcal{J}^i$ such that $\forall (j, j') \in \mathcal{Q}^i : j' \neq j^*$). For all subsequent stages, $r_{ij} = r_{i,\pi(i,j)} + \min_{m \in \mathcal{M}^{\pi(i,j)}} p_{i,\pi(i,j),m}$. Based on those activity heads, we can calculate the earliest machine completion times and set them as lower bounds on $C_{\max}$:

$$\sum_{(i,i') \in \mathcal{P}^j, i=0} r_{i'j} \cdot Y_{ii'jm} + \sum_{(i,i') \in \mathcal{P}^j} Y_{ii'jm} \cdot (s^{jm}_{f(i),f(i')} + p_{i'm}) \leq C_{\max} \quad \forall j \in \mathcal{J}, m \in \mathcal{M}^j. \tag{29}$$

As for the transport resource, we adopt a similar approach, leading to the following inequality:

$$\sum_{(i,j) \in \mathcal{R}} \sum_{(i',j') \in \mathcal{R}, (i',j') \neq (i,j)} Z_{iji'j'} \cdot (\tau_{j,\pi(i',j')} + \tau_{\pi(i',j'),j'}) \leq C_{\max}. \tag{30}$$

Note that the first transport time within the parentheses corresponds to the repositioning time, that is, the time needed to move from the destination stage of the first request to the source stage of the second request.

Appendix B: A Constraint Programming Formulation

As an alternative to the MIP model from Appendix A, we also develop a CP formulation for the problem at hand. The formulation relies on scheduling-specific model elements that can be found (or at least engineered) in various different CP solvers or frameworks. Because we rely on IBM ILOG CP Optimizer for the subsequent experimentation, we adopted the naming conventions of that environment.

All activities (intermediate storage, processing, and transport) are represented by interval variables. Table 5 provides a summary of all those interval variables, including all alternative ones that are necessary to ensure machine flexibility on the processing stages. The idea is to have master intervals for intermediate storage/buffering and processing on each stage. For each of those master intervals, multiple optional intervals can be created, one of which is set as "present" in case the respective machine is selected for processing.

Sequences of jobs on the machines and the transport resource are modelled by dedicated sequence variables (see Table 6). Sequence variables respect sequence-dependent setup times. For this purpose, a setup type has to be provided for

each interval variable. For the machine sequences, the setup type for an activity coincides with the job's setup family $f(i)$. For the transport resource sequence, we introduce an artificial mapping of transport requests to unique IDs. A further mapping is needed to obtain the repositioning times of the transport resource based on request IDs only. The corresponding parameters are stated in Table 7.

Intermediate buffer usage is modelled using cumulative resources. A concept called *cumul* functions is available for this purpose. One such function is used for each stage, machine and buffer type (see Table 8). Since buffer resources can be considered as storage resources or reservoirs, the *pulse* function is used to accumulate resource usage. The semantics of pulse is such that it increases the resource usage function by a predefined amount (1 in our case) when an activity starts, and maintains that level for the whole duration of the activity. The resource usage function is then decreased again. Again, we can use either the makespan or total tardiness as a minimisation objective, leading to

$$\text{Minimise} \quad \max_{o \in \mathcal{O}} \text{endOf}(U_o) \tag{31}$$

or

$$\text{Minimise} \quad \sum_{o \in \mathcal{O}} \max(0, \text{endOf}(U_o) - \delta_o) \tag{32}$$

subject to

$$\text{span}(U_o, \{V_i \mid i \in \mathcal{I}^o\}) \quad \forall o \in \mathcal{O}, \tag{33}$$

$$\text{span}(V_i, \{\hat{W}_{ij} \mid j \in \mathcal{J}^i\}) \quad \forall j \in \mathcal{J}, \tag{34}$$

$$\text{endBeforeStart}(\hat{W}_{ij}, \hat{W}_{ij'}) \quad \forall i \in \mathcal{I}, \forall (j, j') \in \mathcal{Q}^i, \tag{35}$$

$$\text{alternative}(\overline{W}_{ij}, \{\overline{W}^m_{ij} \mid m \in \mathcal{M}^j\}) \quad \forall (i,j) \in \mathcal{R}, \tag{36}$$

$$\text{alternative}(\hat{W}_{ij}, \{\hat{W}^m_{ij} \mid m \in \mathcal{M}^j\}) \quad \forall (i,j) \in \mathcal{R}, \tag{37}$$

$$\text{alternative}(\underline{W}_{ij}, \{\underline{W}^m_{ij} \mid m \in \mathcal{M}^j\}) \quad \forall (i,j) \in \mathcal{R}, \tag{38}$$

$$\text{noOverlap}(\Psi_{jm}, s^{jm}, 1) \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}^j, \tag{39}$$

$$g_{jm} \leq |\overline{\mathcal{B}}| \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}^j, \tag{40}$$

$$f_{jm} \leq |\underline{\mathcal{B}}| \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}^j, \tag{41}$$

$$\text{presenceOf}(\overline{W}^m_{ij}) \Rightarrow \text{presenceOf}(\hat{W}^m_{ij}) \quad \forall (i,j) \in \mathcal{R}, \forall m \in \mathcal{M}^j, \tag{42}$$

$$\text{presenceOf}(\hat{W}^m_{ij}) \Rightarrow \text{presenceOf}(\underline{W}^m_{ij}) \quad \forall (i,j) \in \mathcal{R}, \forall m \in \mathcal{M}^j, \tag{43}$$

$$\text{startAtEnd}(\hat{W}_{ij}, \overline{W}_{ij}) \quad \forall (i,j) \in \mathcal{R}, \tag{44}$$

$$\text{startAtEnd}(\underline{W}_{ij}, \hat{W}_{ij}) \quad \forall (i,j) \in \mathcal{R}, \tag{45}$$

$$\text{startAtEnd}(\overline{W}_{ij}, T_{ij}) \quad \forall (i,j) \in \mathcal{R}, \tag{46}$$

$$\text{startAtEnd}(T_{ij}, \underline{W}_{i\pi(i,j)}) \quad \forall (i,j) \in \mathcal{R}, \tag{47}$$

$$\text{noOverlap}(\Omega, \theta, 1), \tag{48}$$

$$\begin{aligned} &\text{sameCommonSubsequence}(\Psi_{jm}, \Omega, \\ &\qquad \{\hat{W}_{ij}^m \mid i \in \tilde{\mathcal{I}}^j\}, \\ &\{T_{ij'} \mid i \in \tilde{\mathcal{I}}^j, j' \in \mathcal{J}, \pi(i,j') = j\}) \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}^j, \end{aligned} \tag{49}$$

$$\begin{aligned} &\text{sameCommonSubsequence}(\Omega, \Psi_{jm}, \\ &\qquad \{T_{ij} \mid i \in \tilde{\mathcal{I}}^j\}, \\ &\{\hat{W}_{ij}^m \mid i \in \tilde{\mathcal{I}}^j\}) \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}^j. \end{aligned} \tag{50}$$

Besides the two alternative objectives (31) and (32), the formulation consists of a series of constraints, where Constraints (33) and (34) require order and job intervals to span all associated job and activity intervals, respectively. Constraint (35) is responsible for maintaining the correct precedences among the processing intervals of each job. Constraints (36), (37), and (38) establish a relationship between buffer/processing master intervals and their corresponding alternative intervals. The no-overlap constraint makes the machine resources "disjunctive," in the sense that only one interval of the respective associated sequence Ψ_{jm} can be processed at a time. The buffer capacity limits are set by Constraints (40) and (41), simply by imposing upper bounds on the cumul functions.

Once a machine assignment has been determined for one of the optional intervals that belongs to a job on a particular stage, the other optional intervals (buffer or processing) for that job/stage combination are also assigned to the same machine (Constraints (42) and (43)). The buffer and processing master intervals are linked together using rigid "startAtEnd" temporal Constraints (44) and (45). Further such links are established between transport request intervals and buffer master intervals (Constraints (46) and (47)). For the transport resource itself, and thus the associated sequence Ω, we impose an additional no-overlap requirement (Constraint (48)). Finally, Constraints (49) and (50) ensure for each machine that the relative order of interval variables on that machine and the corresponding transport requests (to and from the affected stage) stay the same.

Appendix C: CP Solution Quality Rating for the Instance Scenarios used in the Makespan Experiments with Unrelated Parallel Machines

The motivation for using the CP model described in Appendix B to generate the training examples results from its higher solution quality and the lower computation time compared with the MIP. Table 9 provides ratings of the CP results in terms

of average relative deviations from a reference lower bound, which we obtained by running the MIP formulation from Appendix A (including the bounding constraints) for eight hours and recording the best bound achieved within this time span. Note that the same time limit has been used for all CP and MIP runs to obtain feasible solutions (upper bounds). The values in Table 9 represent average factors, by which the reference lower bounds need to be multiplied to equal the respective CP or MIP objective function values. Furthermore, the last two rows of the table show factors obtained for the LP (root) relaxation of the MIP, with and without the additional bounding constraints, respectively. A factor of 0.953 means that the LP relaxation of the enhanced MIP reaches 95.3% of the reference bound quality. A much more drastic decline appears for the bound quality for the MIP without the additional bounding constraints, which strikingly yields LP objective values of zero for all three job load sets of the crane bottleneck scenario.

The CP apparently performs best under medium and high job loads, resulting in small "gaps" of up to 14.4%. Surprisingly, the MIP behaves in a directly opposite manner; smaller job loads seem to favor the formulation. However, the gaps with the CP results are consistently large, with (average) relative differences of up to 100%. Overall then, we can conclude that both, the enhanced LP/MIP proves successful in delivering notably better lower bounds than the respective basic version (even with the root node relaxation).

However, CPLEX runs on instances which can be solved to optimality reveal that the initial gaps are still relatively large, compared to the optimal solutions, also for the strengthened formulations. Therefore, our conjecture is that bound weakness is the most likely source for the higher CP gaps in particular cases (21.9 and 19.8%). From an overall point of view, the CP consistently outperforms the MIP and is able to provide solutions that are only up to 6.3% off the best lower bound.

Table 2: Sets used in the MIP model formulation

$\mathcal{O}$	Set of orders.
$\mathcal{I}$	Set of jobs.
$\mathcal{I}^o$	Subset of jobs i belonging to order o, $\mathcal{I}^o \subseteq \mathcal{I}$.
$\tilde{\mathcal{I}}^j$	Subset of jobs that are processed on stage j.
$\mathcal{F}$	Set of setup families $0, 1, \ldots, \bar{f}$, where 0 denotes the initial/terminal setup state (with no job on the machine) and $\bar{f}$ the total number of regular setup families.
$\mathcal{J}$	Set of stages, index j.
$\mathcal{J}^i$	Set of stages required for processing job i.
$\mathcal{M}^j$	Set of machines belonging to operation stage j.
$\mathcal{B}^{jm}$	Set of slots on machine m of stage j.
$\hat{\mathcal{B}}^{jm}$	Subset of processing slots on machine m of stage j, $\hat{\mathcal{B}}^{jm} \subset \mathcal{B}^{jm}$.
$\overline{\mathcal{B}}^{jm}$	Subset of buffer slots on machine m of stage j, $\overline{\mathcal{B}}^{jm} \subset \mathcal{B}^{jm}$.
$\underline{\mathcal{B}}^{jm}$	Subset of transport (buffer) slots on machine m of stage j, $\underline{\mathcal{B}}^{jm} \subset \mathcal{B}^{jm}$.
$\mathcal{P}^j$	Set of pairs of job indices that are to be processed on stage j. The job indices include two dummy jobs, one for the start and one for the end of each machine sequence. Hence, $\mathcal{P}^j = \{(i,i') \mid i,i' \in \tilde{\mathcal{I}}^j \cup \{0, n+1\} \wedge i \neq i'\} \setminus \{(n+1, 0)\}$.
$\mathcal{Q}^i$	Set of pairs (j, j') of stages that are immediately consecutive in the routing of job i.
$\mathcal{R}$	Set of all pairs (i, j), with $j \in \mathcal{J}$ and $i \in \tilde{\mathcal{I}}^j$, i.e., the set of all feasible job/stage combinations.
$\mathcal{R}^*$	The set of all feasible job/stage pairs including two "dummy"-pairs, i.e., $\mathcal{R}^* = \mathcal{R} \cup \{(0,0), (n+1, \lvert\mathcal{J}\rvert)\}$.

Table 3: Parameters used in the MIP model formulation

$s_{ff'}^{jm}$	Time required for switching between setup families $f, f' \in \mathcal{F}$ on machine m of stage j.
$f(i)$	Setup family of job $i \in \mathcal{I} \cup \{0, n+1\}$. According to the definition of $\mathcal{F}$, it is assumed that $f(0) = f(n+1) = 0$.
p_{ijm}	Processing time of job i on machine m of stage j.
$\tau_{jj'}$	Transport time required to move material from stage j to j'.
δ_o	Due date of order o.
$\pi(i,j)$	The predecessor stage j' of j in the routing of job i, that is, a stage j' such that $(j', j) \in \mathcal{Q}^i$.
$\bar{j}(i)$	The last processing stage of job i.
M	A very large number.

Table 4: Decision variables used in the MIP model formulation

t'_{ij}	Arrival time of job i in stage j.
t''_{ijm}	Arrival time of job i in stage j on machine m.
t'''_{ijmb}	Start/arrival time of job i in stage j on machine m and slot b.
c'_{ij}	Completion time of job i in stage j.
c''_{ijm}	Completion time of job i in stage j on machine m.
c'''_{ijmb}	Completion time of job i in stage j on machine m and slot b.
d'_{ij}	Departure time of job i in stage j.
d''_{ijm}	Departure time of job i in stage j on machine m.
d'''_{ijmb}	Departure time of job i in stage j on machine m and slot b.
$\bar{t}_{ij}$	Start time of transport request moving job i to stage j.
$\bar{c}_{ij}$	Completion time of transport request moving job i to stage j.
C_o	Completion time of order o.
$C_{\max}$	The maximum completion time of all orders.
X_{ijm}	Binary variable, indicating whether job i is performed on machine m on stage j.
$Y_{ii'jm}$	Binary variable, indicating whether job i directly precedes job i' on machine m on stage j.
$Z_{iji'j'}$	Binary variable, indicating whether job i with target stage j is moved immediately before job i' with target stage j'.

Table 5: Interval variables used in the CP model formulation

Symbol	Optional	Size	Description
U_o	no		Represents the total time span between start and finish of order o.
V_i	no		Represents the total time span between start and finish of job i.
$\overline{W}_{ij}$	no		Time spent by job i in a buffer slot of stage j.
$\overline{W}_{ij}^m$	yes		Alternative interval for $\overline{W}_{ij}$, seizing a buffer slot of machine m (on stage j).
$\hat{W}_{ij}$	no		Activity reflecting the processing of job i on stage j
$\hat{W}_{ij}^m$	yes	p_{ijm}	Alternative activity for $\hat{W}_{ij}$ taking place on machine m (on stage j).
$\underline{W}_{ij}$	no		Time spent by job i in a transport (buffer) slot of stage j.
$\underline{W}_{ij}^m$	yes		Alternative interval for $\overline{W}_{ij}$, seizing a transport slot of machine m (on stage j).
T_{ij}	no	$\tau_{\pi(i,j),j}$	Transport request for moving job i to stage j.

Table 6: Sequence variables used in the CP model formulation

Symbol	Interval Var. Set	Setup Type	Description
Ψ_{jm}	$\{\hat{W}_{ij}^m \mid i \in \tilde{\mathcal{I}}^j\}$	$f(i)$	Sequence of processing intervals on machine m of stage j
Ω	$\{T_{ij} \mid (i,j) \in \mathcal{R}\}$	$\lambda(i,j)$	Sequence of transport requests

Table 7: Parameters used in the CP model formulation

$\lambda(i,j)$	A unique ID for the pair (i,j), with $i \in \mathcal{I}$ and $j \in \mathcal{J}^i$.
$\theta(u,u')$	Gives the transport time required to move from the destination stage of the transport request with ID u to the source stage of the transport request with ID u'. Let j be the destination stage corresponding to u, and j' the source stage of u'. Then $\theta(u,u') = \tau(j,j')$.

Table 8: Cumul functions used in the CP model formulation

Symbol	**Definition**	**Description**
g_{jm}	$\sum_{i \in \tilde{\mathcal{I}}^j} \text{pulse}(\overline{W}_{ij}^m, 1)$	Cumulates buffer usage on the buffer slots of machine m on stage j.
h_{jm}	$\sum_{i \in \tilde{\mathcal{I}}^j} \text{pulse}(\underline{W}_{ij}^m, 1)$	Cumulates buffer usage on the transport slot(s) of machine m on stage j.

Table 9: CP and MIP solution quality rating for the instance scenarios used in the makespan experiments with unrelated parallel machines

Approach	Instance scenario								
	Machine bottleneck			Crane bottleneck			Mixed		
	Job load set								
	Low	Med	High	Low	Med	High	Low	Med	High
CP	1.219	1.144	1.103	1.179	1.063	1.068	1.198	1.085	1.083
MIP (enhanced)	1.646	1.683	2.046	1.474	1.760	1.960	1.213	1.673	1.975
LP (enhanced)	0.953	0.996	1.000	0.990	1.000	1.000	0.895	0.998	1.000
LP	0.093	0.043	0.027	0.000	0.000	0.000	0.036	0.019	0.010

Abstract

The proposed PhD thesis deals with different machine learning approaches for solving flexible shop scheduling problems in project 1 and 2, and a learning approach to replace the sample average approximation for radiotherapy appointment scheduling in project 3. The basis of the approach for project 1 and 2 is an Industry 4.0 application configuration of the industrial partner Syngroup, a business consultancy in Vienna with clients in manufacturing. The third project is based on data provided by MedAustron, located in Wiener Neustadt.

The goal of the first project was to generate a priority dispatching rule in form of a decision tree (DT). High quality solutions obtained using a constraint programming (CP) formulation were used to generate the DT. The unified representation of the job sequencing and machine assignment decision within a single tree was also part of the contribution, as well as the generation of random forests (RF) to counteract overfitting behavior. Both approaches were tested on a variety of differing instance scenario settings with the objective to either minimize the makespan or tardiness. Computational results indicate that the machine learning approach performs well on most instance sets and that a generic DT or RF achieve competitive results when applied to different instance scenarios.

The second topic deals with Genetic Programming (GP) with a single-tree and multi-tree representation for the same problem definition as for the first project. In this case the available jobs are ranked using a tree-based priority rule generated with the GP. The results of the GP approach are compared to those of the DT and RF approach. To conclude, the GP approach in combination with an iterative approach to improve schedules, yields better results than the DT and RF approach.

Finally, a regression analysis was applied to replace an existing sample average approximation approach to estimate the average patient waiting time in a radiotherapy appointment scheduling setting. Therefore, 22 features, used as regressors, were implemented. Furthermore, a feature selection procedure was conducted to define set of features regarding their importance to estimate the waiting time. Then, different regression analysis methods were tested (e.g., Kernel Ridge Regression, Lasso Regression) and an artificial neural network implemented. The results support the assumption, that the proposed replacement learning approach performs well and therefore might be implemented as a replacement for the complex sample average approximation.

Zusammenfassung

Die vorliegende PhD-Thesis bearbeitet im ersten und zweiten Projekt Themen aus dem Bereich Machine Learning für flexible Fließfertigungsprobleme. Die Basis für diese Projekte ist eine Industrie 4.0 Applikation des Industriepartners Syngroup. Für das dritte Projekt werden Daten des MedAustron Ionenstrahl-Centers in Wiener Neustadt herangezogen.

Das Ziel des ersten Projektes war die Entwicklung einer Prioritätsregel in Form eines Entscheidungsbaumes. Der Baum wird hierbei durch die Verwendung hochqualitativer Lösungen basierend auf einer Constraintprogrammierung-Formulierung erstellt. Die Neuheit liegt dabei in der vereinheitlichten Entscheidungsfindung bezüglich Job- und Maschinenauswahl. Ein Random Forest wurde ebenfalls implementiert, um Überanpassung entgegenzuwirken. Der Ansatz wurde auf verschiedensten Problemstellungen angewendet und lieferte überzeugende Ergebnisse in Hinblick auf Verkürzung der Produktionsdauer vor allem im Vergleich zu einfachen Prioritätsregeln. Das zweite Thema beinhaltete die Entwicklung eines evolutionären Algorithmus (genetische Programmierung) für dieselbe Problemstellung. Hier werden ebenfalls Entscheidungsbäume erstellt, die dafür verwendet werden, ein Ranking für anstehende Materialbewegungen zu erstellen und die höchst gerankte Option auszuwählen. Die Ergebnisse wurden mit denen aus Projekt 1 verglichen. Es konnte eine deutliche Verbesserung gegenüber der Ergebnisse aus Projekt 1 festgestellt werden. Ein iterativer Ansatz zur Verbesserung bestehender Ablaufpläne führte ebenfalls zum gewünschten Ziel, die Produktionsdauer zu verkürzen.

Im letzten Projekt wurde eine Regressionsanalyse durchgeführt. Hierbei wurden Daten von MedAustron bezüglich Ablaufplanung bei Behandlung von Krebs mittels Strahlentherapie dafür verwendet, die erwartete durchschnittliche Wartezeiten der Patienten in einem stochastichen Umfeld zu berechnen, um einen bestehenden, aufwändigen “sample average approximation”-Ansatz zu ersetzen. Dafür wurden zunächst Features definiert, die diese Wartezeit beeinflussen können und mittels Feature Selection die wichtigsten Features extrahiert. Mit diesen verschiedenen Feature Sets wurde dann eine Regressionsanalyse durchgeführt, sowie ein neuronales Netz für die Vorhersage implementiert. Die Ergebnisse zeigen, dass mit dem vorliegenden Ansatz eine vielversprechende Vorhersagegenauigkeit erreicht und der Ansatz durchaus als Ersatz für die “sample average approximation” verwendet werden kann.