applied sciences

Article

Choosing Solution Strategies for Scheduling Automated Guided Vehicles in Production Using Machine Learning

Felicia Schweitzer [1,2,*], Günter Bitsch [1] and Louis Louw [2]

1 ESB Business School, Reutlingen University, 72762 Reutlingen, Germany
2 Department of Industrial Engineering, Stellenbosch University, Stellenbosch 7600, South Africa
* Correspondence: felicia.schweitzer@student.reutlingen-university.de or 26426625@sun.ac.za

Featured Application: The artifact developed in this article is applicable to AGV and production planning optimization problems that align with the principles of the job-shop scheduling problem (JSSP) and the flexible job-shop scheduling problem (FJSSP).

Abstract: Artificial intelligence is considered to be a significant technology for driving the future evolution of smart manufacturing environments. At the same time, automated guided vehicles (AGVs) play an essential role in manufacturing systems due to their potential to improve internal logistics by increasing production flexibility. Thereby, the productivity of the entire system relies on the quality of the schedule, which can achieve production cost savings by minimizing delays and the total makespan. However, traditional scheduling algorithms often have difficulties in adapting to changing environment conditions, and the performance of a selected algorithm depends on the individual scheduling problem. Therefore, this paper aimed to analyze the scheduling problem classes of AGVs by applying design science research to develop an algorithm selection approach. The designed artifact addressed a catalogue of characteristics that used several machine learning algorithms to find the optimal solution strategy for the intended scheduling problem. The contribution of this paper is the creation of an algorithm selection method that automatically selects a scheduling algorithm, depending on the problem class and the algorithm space. In this way, production efficiency can be increased by dynamically adapting the AGV schedules. A computational study with benchmark literature instances unveiled the successful implementation of constraint programming solvers for solving JSSP and FJSSP scheduling problems and machine learning algorithms for predicting the most promising solver. The performance of the solvers strongly depended on the given problem class and the problem instance. Consequently, the overall production performance increased by selecting the algorithms per instance. A field experiment in the learning factory at Reutlingen University enabled the validation of the approach within a running production scenario.

Keywords: AGV scheduling; optimization; constraint programming; machine learning; algorithm selection

check for updates

Citation: Schweitzer, F.; Bitsch, G.; Louw, L. Choosing Solution Strategies for Scheduling Automated Guided Vehicles in Production Using Machine Learning. *Appl. Sci.* **2023**, *13*, 806. https://doi.org/10.3390/app13020806

Academic Editor: Dimitris Mourtzis

Received: 20 December 2022
Revised: 30 December 2022
Accepted: 4 January 2023
Published: 6 January 2023

1. Introduction

With the introduction of Industry 4.0, production systems have become increasingly complex within the last few years [1]. This is, on one hand, due to the growing size, but it is also due to the extended automation influences in manufacturing environments. As a result, the demands on intralogistics in terms of flexibility and internal control are increasing. One major development within the technology of material handling is the evolution of autonomous mobile robots (AMRs), including AGVs [2]. AGVs are wheel-based and unmanned vehicles that transport small or large unit loads along the floor of a facility. These load carriers provide the internal and external transport of materials [3]. Two of the main enabling software technologies behind AGVs are scheduling and routing.

Thereby, dynamic AGV scheduling models have the potential to minimize delays and reduce production costs by facilitating a minimal production makespan [4].

Scheduling, in general, is a decision-making process that deals with the allocation of resources to tasks over given time periods, while the goal is to optimize one or more objectives [5]. Thereby, resources and tasks can take many different forms, and the objectives themselves may range from a minimization of the completion time of the last task to the minimization of overall delay, depending on the use case. Scheduling can be either static or dynamic; in static scheduling, all task information is stable and obtained in advance, while dynamic scheduling considers uncertainties on the shop floor [6]. The majority of scheduling problems are NP-hard problems, which are more difficult to solve than others. This is because the computation time required for solving a problem increases faster as the size of the problem grows [7]. There are a wide variety of approaches for solving scheduling problems. However, they differ in their solution quality. Traditional scheduling approaches include three major areas: heuristic approaches, such as dispatching rules; metaheuristics, such as simulated annealing, tabu search, local search, and genetic algorithms; and exact methods, such as mathematical programming (MP), mixed-integer programming (MIP), linear programming (LP), integer linear programming (ILP), and mixed integer linear programming (MILP) [8,9]. Another emerging technology for optimizing scheduling problems is constraint programming (CP). The dynamic selection of the appropriate algorithm for a specific scheduling problem is a new challenge, especially in the case of changing framework conditions. Considering the new complexity arising within production systems, the dynamic selection of the most appropriate algorithm for a specific scheduling problem can be beneficial. Depending on the scheduling problem, some algorithms might be more efficient than others, especially when considering the specific instances within a scheduling problem.

Algorithm selection has been part of research for a long time. The algorithm selection problem was introduced in 1976 [10]. Three major characteristics help to categorize a model for algorithm selection: the problem space, algorithm space, and performance space. Algorithm selection has become especially relevant in the last decade, as researchers are increasingly investigating how to identify the most suitable existing algorithm for solving a problem instead of developing new algorithms [11]. This is because, in some scenarios, a new approach will improve the current state of the art but only for some problems. Selecting the most suitable algorithm for a particular problem aims to mitigating these problems and has the potential to significantly increase performance in practice [12]. Researchers use the benefits of algorithm selection to explore the details of data and to make use of data characteristics in order to choose the most promising algorithm to save computational time and increase performance.

For executing the algorithm selection process, researchers make use of different techniques to select algorithms from several algorithms for solving optimization problems, such as integration or hyper-heuristics. Hyper-heuristics select between low-level heuristics for solving optimization problems [13]. They operate at a higher level of abstraction than traditional heuristics, which makes them more flexible and adaptable to different problem domains. Additionally, it is possible to use integration as part of an algorithm selection process. Therefore, a set of algorithms and performance metrics is defined, and an integration method calculates the overall performance of each algorithm based on metrics [14]. The difference between these ideas, in essence, is that the algorithm selection method refers to the overall process of selecting an algorithm for a given problem, which can involve a variety of different techniques and approaches. These techniques can be integration, metaheuristics, or hyper-heuristics, but machine learning also finds frequent application.

Machine learning has received considerable attention in recent years due to its ability to learn patterns in data relating to a specified output. It extracts useful knowledge from the generated data throughout the search process and can support the detection of a problem class by evaluating incoming scheduling problems according to their individual characteristics. Therefore, machine learning algorithms were implemented to select algorithms for several problem areas [15].

The motivation lies in the successful application of algorithm selection approaches in other domains. One outstanding example within the field of satisfiability problems (SAT) is the portfolio-based algorithm selection *SATzilla* [16]. Instead of traditionally choosing the best solver for a given class of instances, this decision is online on a per-instance basis. The approach takes a distribution of problem instances and a set of component solvers as input. It constructs a portfolio that optimizes a given objective function. Furthermore, Refs. [17,18] introduced algorithm selection approaches for two well-known production planning problems: the job-shop scheduling problem (JSSP) and the flexible job-shop scheduling problem (FJSSP). In [17], the authors proposed a CP model with several solvers for the FJSSP and showed that there was no clear winner over all tested instances. The same results were obtained in [18] by testing several state-of-the-art algorithms for the JSSP. They both implemented machine learning algorithms to select the best solver for specific instances of the individual scheduling problem class.

This article aimed to develop an algorithm selector that is capable of finding the most suitable scheduling solution strategy with machine learning to determine the best schedule for a particular AGV problem instance. An automated algorithm selector was built from two state-of-the-art solvers for the JSSP and FJSSP, which were transferred to AGV scheduling. Several machine learning models were designed to decide which algorithm to run on an incoming instance. By selecting the algorithm for every given problem instance, a scheduling system can react dynamically to incoming scheduling problem types. There is no static algorithm presented. Instead, the instance is analyzed, and the suitable algorithm, based on the input features, is chosen. Furthermore, more than one overall problem type is taken into account. This enables a manufacturing system to be flexible in its approach to sending transport requests. These can already be assigned to an AGV, or the scheduler can allocate the orders to AGVs. The selection of an appropriate algorithm for individual scheduling scenarios enables a dynamic procedure of scheduling.

To this end, the applied methodology to develop the algorithm selector is first presented in Section 2. The analysis phase unveils the selection of the algorithm base and the theoretical concept of the selector. The design phase introduces the problem classification and the mathematical problem definition. The modeling of algorithms takes place, as well as the design of the dataset for training the machine learning algorithms. Section 3 presents the results of the implementation, which include the comparison of scheduling algorithms and the evaluation of machine learning models. A conclusion summarizing the results and the discussion follows in Section 4.

2. Materials and Methods

This study followed the principles of design science research [19]. Design science research aims to develop novel and innovative artifacts as an outcome of research. It reflects an engineering research tradition, and due to its ability to focus on the construction and evaluation of an artifact, it is beneficial for developing the intended algorithm selector. In particular, this article orients towards the cognition process introduced in [20]. The creation of the artifact takes place in four phases: analysis, design, evaluation, and diffusion.

2.1. Analysis Phase

During the analysis phase, the focus was on the assessment of existing scheduling algorithms for AGV scheduling to select the algorithm database for the selector. A semi-systematic literature review served to track the development of scheduling algorithms over time in order to identify appropriate algorithms for optimizing the AGV scheduling problems [21]. The literature analysis investigated journals, conference papers, and scientific books. The sources were in English, except when they contributed substantially to defining the research gap. Moreover, to ensure high-impact literature, the h-index for referenced conference proceedings needed to be higher than 30, and the Scimago Journal Rank needed to be higher than 0.5. The searched databases were Science Direct, SpringerLink, and Google Scholar. Furthermore, the articles had to contain research on AGV scheduling and consider disruptions like machine breakdowns, algorithm descriptions for AGV scheduling, or the classification of problem instances for scheduling. Articles that focused on multi-criteria optimization, such as multi-objectives, as well as articles focusing on AGV routing or path planning were not considered.

2.1.1. Scheduling Algorithms

As briefly described in the introduction, there are a variety of algorithms for solving scheduling problems (Figure 1). Heuristics and metaheuristics find frequent application for AGV scheduling because they provide exact solutions to scheduling problems in favor of fast computational times [22]. However, there are some drawbacks in practice, as the problem and relationship between elements must be well known and heuristics do not guarantee finding an optimal solution [23]. Besides heuristics, metaheuristics have been developed as another promising optimization-solving technique for AGV scheduling problems. Tabu search, simulated annealing, and genetic algorithms are high-level heuristics that guide local search heuristics to escape from local optima [23]. Several studies have shown the effectiveness of metaheuristics for scheduling in comparison to simple dispatching rules [24–26]. Nevertheless, despite the successful implementation of genetic algorithms for improving optimization problems, tabu search and simulated annealing tend to be more efficient in finding the optimal solution in a reasonable time due to their operation on a single configuration and not on an entire population [27]. Furthermore, evolutionary algorithms such as ant colony algorithms, particle swarm optimization, artificial bee colony, or hybrid water waver optimization are metaheuristics that developed rapidly in recent years. Several authors have conducted research on their application in optimization and scheduling scenarios [28–30]. Additionally, exact methods mostly provide optimal results. However, mathematical optimization can have difficulties in efficiently handling larger systems, as the computational time often exceeds the time limits for real-time scheduling [31].

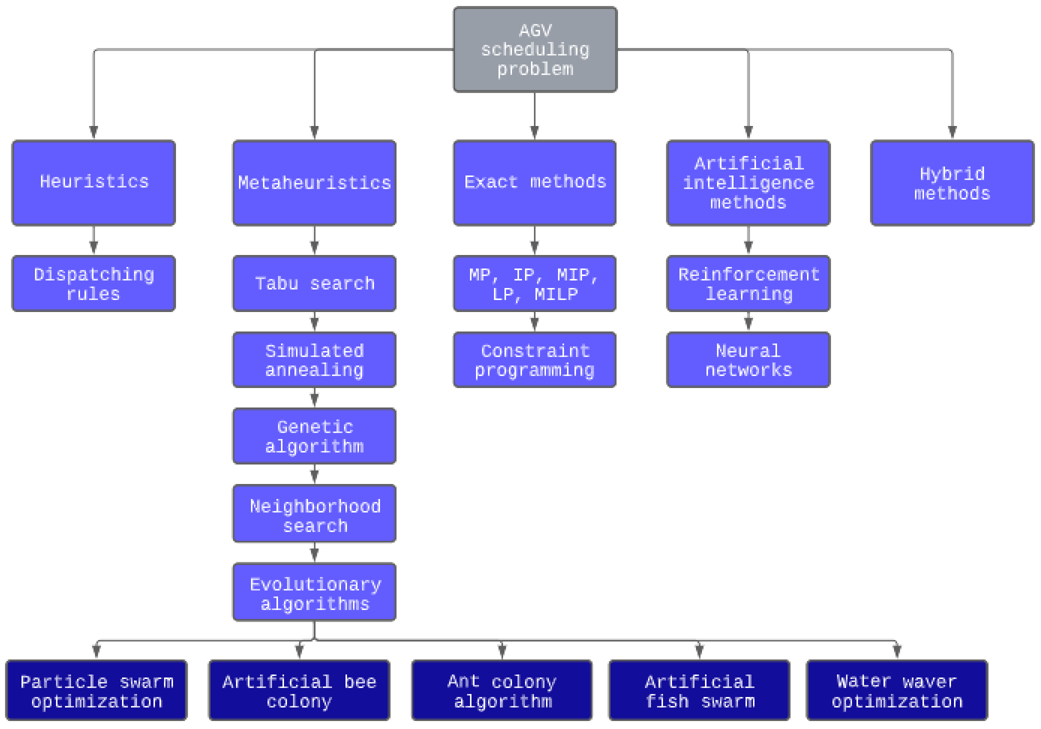


Figure 1. Algorithms for AGV scheduling problems identified in the literature.

One emerging optimization approach within the exact methods is CP. CP is a type of optimization algorithm that uses constraint satisfaction and search techniques to find solutions for given problems. Constraint-based scheduling is one of the most successful application areas of CP, a mathematical optimization technique that makes use of artificial intelligence logic. Researchers explain its success due to the combination of the best of two fields of research: operations research and artificial intelligence [32]. CP is more flexible in comparison to mathematical programming. This is because the constraints, as well as the objective functions, are more general due to the decision variables, besides representing integer values, real values, and set elements or their subsets of sets [5]. Many researchers make use of CP for machine scheduling problems such as the JSSP and FJSSP [17,18,33,34]. The major drawbacks of CP were long computational processing times for larger problem instances and that the building of models required empirical expertise [33,34]. Due to developments over the last 15 years, CP solvers are now able to solve even large-scale instances where CP solvers performed worse than heuristics or metaheuristics [9]. Currently, CP solvers perform exceptionally well for most scheduling problems due to the significant improvements in recent years for larger problem instances [9,17].

Furthermore, the application of artificial intelligence (AI) for solving scheduling problems is becoming more common [2]. One of the main application areas of AI to AGV scheduling that emerged during the review was reinforcement learning [6,22,35–37]. In addition, Ref. [38] proposed a neural-network-based multi-state scheduling algorithm for multi-AGVs in a flexible manufacturing system. Besides the individual application of artificial intelligence to AGV scheduling, more and more studies have focused on the integration of machine learning for traditional scheduling algorithms, and their findings illustrate performance improvements compared to individually applied methods [15,39,40]. Usually, state-of-the-art algorithms rely on handcrafted heuristics to make decisions that are otherwise too expensive to compute or mathematically not well defined [41]. This is where machine learning comes into play because it has the ability to make these decisions in a more principled and optimized way.

2.1.2. Theoretical Concept of Algorithm Selector

Due to the identified potential for combining machine learning with state-of-the-art algorithms, this article worked on the development of an algorithm selection approach that makes use of CP solvers as a solution strategy for the scheduling problem and machine learning algorithms that serve to select the optimal solver for given problem instances.

In the initialization phase, there is no evaluation of algorithms for specific problem instances available. Therefore, the selected algorithms are applied to a set of training scheduling problem instances. These instances are solved with several CP solvers. Their different mathematical solution methodologies and broad application to AGV scheduling present an interesting viewpoint on AGV scheduling approaches by comparing them. The objective is, at the same time, the performance metric for evaluating the scheduling algorithms. Therefore, the overall objective plays an important role in the scoring. After the application of all algorithms to the test instances, the labeling and feature selection for the dataset follow. Once the dataset has been labeled, machine learning is introduced. The input variables are the previously defined features, and the output is the winning solver measured with the scoring system. During the training phase, the machine learning algorithms learn the mapping of features to the output variable to select the most suitable solver for an incoming problem for future applications. The concrete algorithm selection model is presented in Figure 2. The details of the solvers, scoring system, features, and machine learning algorithms follow during the design phase.

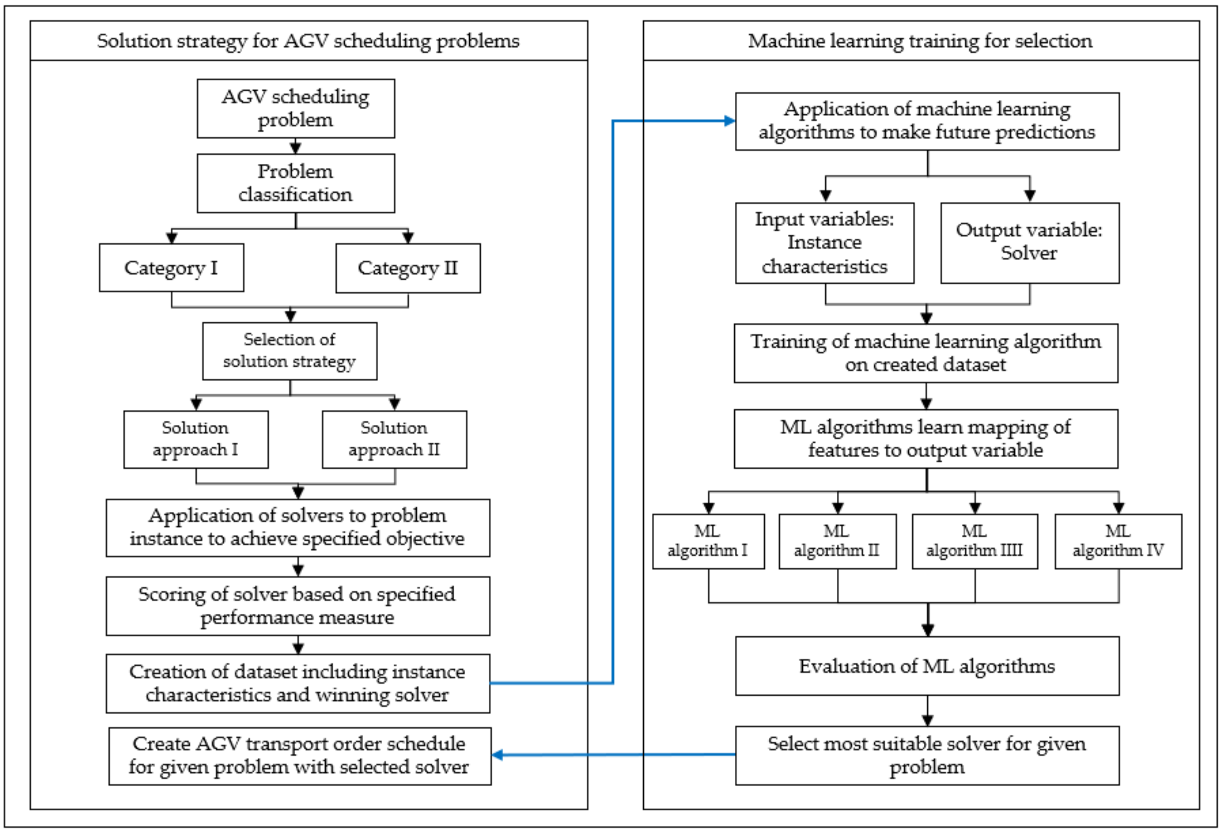

Solution strategy for AGV scheduling problems
AGV scheduling problem
Problem classification
Category I
Category II
Selection of solution strategy
Solution approach I
Solution approach II
Application of solvers to problem instance to achieve specified objective
Scoring of solver based on specified performance measure
Creation of dataset including instance characteristics and winning solver
Create AGV transport order schedule for given problem with selected solver
Machine learning training for selection
Application of machine learning algorithms to make future predictions
Input variables: Instance characteristics
Output variable: Solver
Training of machine learning algorithm on created dataset
ML algorithms learn mapping of features to output variable
ML algorithm I
ML algorithm II
ML algorithm IIII
ML algorithm IV
Evaluation of ML algorithms
Select most suitable solver for given problem

Figure 2. Theoretical concept of the algorithm selection approach.

2.2. Design Phase

The creation of the artifact appears during the design phase. The main parts include the exact problem classification and definition, the modeling of the problem, feature engineering, the scoring system, and machine learning model implementation.

2.2.1. Problem Classification

Before going into detail about the algorithm selection model, the problem must be classified. The determination of the exact problem type to solve is important because the problem type significantly influences the algorithm selection. One of the most discussed problems within the literature on scheduling is the JSSP. It is considered one of the hardest combinatorial optimization problems. The JSSP considers a job shop with a number of machines, where each job would have its own predetermined route to follow [5]. Each job has a series of tasks that requires the usage of a particular machine for a known duration. Additionally, the tasks need to be completed in a specified order. The objective is to schedule the jobs on the machines to minimize the time necessary for processing all jobs. Building on this, the FJSSP generalizes the job shop and parallel machine environment [5]. The difference to the job shop is that the flexible job shop considers work centers with a number of identical machines in parallel. Each job follows its own route and must be processed at each work center, but any machine can perform the job. Therefore, the scheduling problem becomes computationally more difficult to solve. This article focuses on the JSSP and FJSSP. This is because the JSSP is one of the most complex scheduling problems of all scheduling classes and is therefore a suitable initial scheduling scenario for testing the functionality of an algorithm selection approach. The reason for FJSSP is the flexibility it offers for scheduling. It is an extension to the JSSP, and especially as this article focuses on dynamic AGV scheduling, the characteristics of FJSSP align with the research objectives. Moreover, the FJSSP represents one of the most practically relevant problem settings and is of interest for implementation in real-world productions [17].

As the JSSP and FJSSP mainly focus on production scheduling, the problem classes must be transferred to AGV scheduling. AGV scheduling shows similarities to the production scheduling problem. Instead of machines as resources, there are AGVs within the manufacturing environment that need to process a certain number of transport orders with suboperations. Therefore, the AGV scheduling can be derived from the production scheduling problem in order to consider different possible AGV scenarios. Depending on the use case, several scheduling layouts can be displayed and tested. Within JSSP-oriented scheduling, each job has a series of tasks that require the usage of a particular AGV for a known duration. For the FJSSP, the processing of a transport order can be performed on any AGV of a certain AGV group, and the allocation of AGVs to jobs is executed during the scheduling process.

2.2.2. Problem Definition

The AGV scheduling problem studied in this article focuses on the transportation of n jobs, where each job has a source or a pickup node as well as a destination or delivery node. A set of AGVs provides the transportation of jobs. For the model, the fixed parameters of this optimization problem, the decision variables to be determined, the constraints to be satisfied, and the objectives to be optimized are described. The mathematical model is based on previous works [17,18,42,43]. For the JSSP, the following parameters and decision variables are set:

- Set $J = \{J_1, \ldots, J_n\}$ of *jobs*, whereas $n = |J|$ is the number of jobs;
- Each job $j_i \in J$ consists of a set of q_i *operations* $O_i = \{i_1, \ldots, i_{qi}\}$;
- They have to be transported on a set $A = \{A_1, \ldots, A_k\}$ of k *AGVs*, and $a = |A|$ is the number of AGVs;
- The transportation order of each job (j_i) is given by a permutation ($f_{j,1}, \ldots, f_{j,a}$) of A;
- Each job (j) and each AGV (k) is associated with a transport time ($t_{j,k}$).

The schedule is created by assigning each operation a start time ($s_{j,k}$) for all $j \in J$, $k \in A$. In order to be feasible, the schedule has to satisfy the following conditions:

Constraints

The constraints ensure the correctness of the model with regard to certain specifications of the problem domain. First, it needs to be ensured that all start times are positive.

$$s_{j,k} \geq 0 \text{ for all } j \in J, k \in A \tag{1}$$

Furthermore, the tasks within one job must be processed sequentially. That means that the transportation orders must align with the sequenced order of tasks.

$$s_{j,f_{j,i}} + t_{j,f_{j,i}} \leq s_{j,f_{j,i+1}} \text{ for all } j \in J,\ 1 \leq i \leq |A| \tag{2}$$

Additionally, there should be no overlap between transport requests processed on the same AGV.

$$s_{j,k} \geq s_{i,k} + t_{i,k} \vee s_{i,k} \geq s_{j,k} + t_{j,k} \text{ for all } i, j \in J, k \in A, i \neq j \tag{3}$$

Objective

For the decision variables, the makespan represents the time needed to complete all tasks. Furthermore, for each transportation task its start time ($s(t) \in \{0, \ldots, T\}$) must be determined, where T is the scheduling horizon. The scheduling horizon for this scenario is the sum of all transportation durations.

The makespan (C_{max}) is the time of completion of the last job, and the goal is to find a scheduling order that minimizes the overall makespan of the scheduling.

$$C_{max} = \max(\{s_{j,k} + t_{j,k} \mid j \in J, k \in A\}) \tag{4}$$

For the FJSSP, the following parameters and decision variables are set:

- Set $J = \{J_1, \ldots, J_n\}$ of n *jobs;*
- Each job $J_i \in J$ consists of a set of q_i *operations* $O_i = \{i_1, \ldots, i_{qi}\}$;
- They have to be transported on a set $A = \{A_1, \ldots, A_k\}$ of k *AGVs*.

Prerequisite Assumptions:

- The sets O_i are assumed to be linearly ordered for all $i \in \{1, \ldots, n\}$;
- All AGVs and jobs are available at the beginning of the scheduling horizon.

Constraints:

- Any pair of operations (i_j, $i_{j'} \in O_i$) with $j < j'$, $i_{j'}$ can only start to be transported after the transportation of i_j has completed;
- Each operation ($i_j \in O_i$, $i \in \{1, \ldots, n\}$) must be processed on exactly one AGV out of the set of possible AGVs ($A_{i_j} \subseteq A$) for that operation;
- Each AGV can transport only one operation at a time, and each operation can be transported by at most one machine at a time;
- The transportation time of an operation $i_j \in O_i$ of a job $J_i \in J$ on an eligible AGV ($A_k \in A_{i_j}$) is represented by $t^k_{i_j} \in N^+$;
- The set of eligible operations of an AGV ($A_k \in A$) is $E_k = \{i_j \mid J_i \in J, i_j \in O_i, A_k \in A_{i_j}\}$.

Objective:

- C_{i_j} is the completion time of an operation ($i_j \in O_i$) of job $J_i \in J$ in a schedule;
- C_i is the completion time of job $J_i \in J$.

A job is considered to be completed once all its operations are completed, for instance, $C_i = C_{i_{q_i}}$ for all $i \in \{1, \ldots, n\}$. The main objective is to minimize the makespan.

$$C_{max} = max_{i \in \{1, \ldots, n\}} C_i \tag{5}$$

2.2.3. Dataset

For implementing and training the selector, this study considered benchmark instances from the literature for the JSSP and FJSSP. This was for several reasons. Firstly, the instances were publicly available, which ensured transparency and reproducibility. Secondly, several papers made use of these instances, which facilitated an overall comparison more easily. Thirdly, the commonly used benchmark instances cover a wide range of problem sizes, from small instances of six jobs and six machines up to one hundred jobs with twenty machines. The data type of both instance sets was semi-structured. There was an internal structure but not one that was suitable for a machine learning process. The data did not provide any direct features; they had to be designed. The data was organized into a schema with patterns and logic, which made the data usable for further processing. The JSSP instances were taken from the instance collection of [44] (http://jobshop.jjvh.nl/, accessed on 10 December 2022) (Table 1).

Table 1. Overview of the JSSP literature benchmark instance collection.

Source	Abbreviation	Number of Instances	Number of Jobs	Number of AGVs
Fisher [45]	ft	3	6, 10, 20	5, 6, 10
Lawrence [46]	la	40	10, 15, 20, 30	5, 10, 15
Adams et al. [47]	abz	5	10, 20	10, 15
Applegate and Cook [48]	orb	10	10	10
Storer et al. [49]	swv	20	20, 50	10, 15
Yamada and Nakano [50]	yn	4	20	20
Taillard [51]	ta	80	15, 20, 30, 50, 100	15, 20
Demirkol et al. [52]	dmu	80	20, 30, 40, 50	15, 20

For the FJSSP, there were also various benchmark instances in the literature. The most common instances were from [53,54], who transformed the JSSP instances of [45,46]. Table 2 lists all available benchmark instances for the FJSSP and their characteristics.

Table 2. Overview of the FJSSP literature benchmark instance collection.

Source	Abbreviation	Number of Instances	Number of Jobs	Number of Operations	Number of AGVs	Processing Times
Brandimarte 1993 [53]	BR	10	10–20	55–240	4–15	1–19
Chambers and Barnes 1996 [54]	CH	21	10–15	100–225	10–18	2–99
Dauzère-Pérès and Paulli 1997 [55]	DA	18	10–20	196–387	5–10	10–100
Hurink et al., 1994 [56]	HU					
	edata	65	6–30	36–300	4–15	1–999
	rdata	65	6–30	36–300	4–15	1–999
	sdata	65	6–30	36–300	4–15	1–999
	vdata	65	6–30	36–300	4–15	1–999

To obtain a deeper understanding of the number of jobs and AGVs among the instances, Figure 3 displays the distribution of instance size regarding the number of jobs and AGVs over the entire dataset. The instances cover the problem space up to larger instances, but the majority of instances are in the area of 50 jobs x 20 AGVs for JSSP and up to 20 jobs x 10 AGVs for FJSSP.

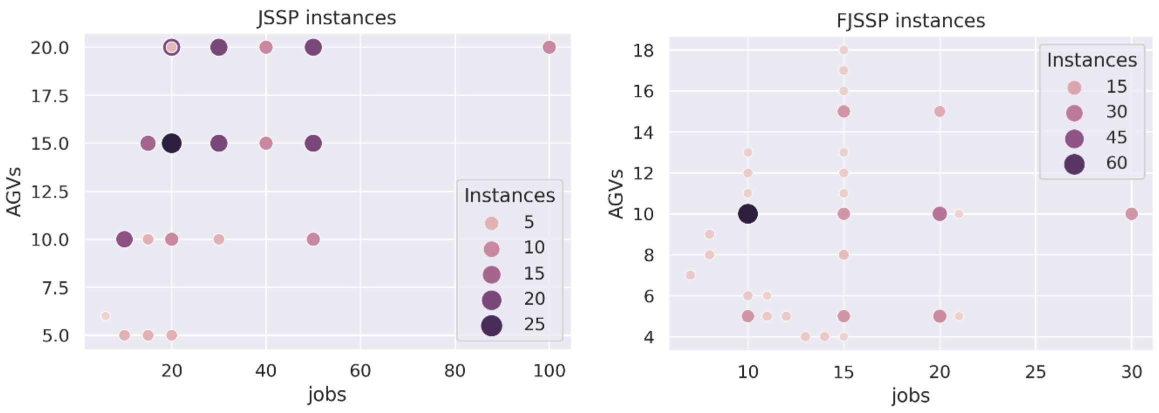


Figure 3. Distribution of jobs and AGVs in benchmark instances.

2.2.4. Dataset Labeling

To categorize the problem, the problem instances were labeled in order to obtain a defined feature set (Table 3). The feature set is crucial for any algorithm selection approach, as it characterizes the instances and links these characteristics to the performance of algorithms. First, instance-size-related characteristics seemed evident for one category of features. These included the number of jobs, the number of AGVs, the number of operations, the job–AGV ratio, and skewness. Furthermore, statistical features provided more insight into the structure of the input data, such as the arithmetic mean of job completion times or the corrected standard deviation of the job completion times divided by the corresponding arithmetic mean [17,57]. This labeling created the base for further feature engineering processes and feature designing to assess which features are the most important for the desired output.

Table 3. Feature description for scheduling problem instances.

Group	Feature	Explanation
Instance-size-related	Jobs	Number of jobs (n)
	AGVs	Number of AGVs (k)
	Operations	Number of operations (i)
	Job–AGV ratio	Job-to-AGV ratio (n/k)
	Skewness	Max. of job-to-AGV ratio and AGV-to job-ratio: max(n/k, k/n)
	Processing time	Travel time of AGV for each operation
Statistical features	Mean job	Arithmetic mean of job completion times
	Min job	Minimum job completion times
	Max job	Maximum job completion times
	Deviation job	Standard deviation of the job completion times divided by the corresponding arithmetic mean
	Min operation	Minimum operation duration time within the entire instance
	Max operation	Maximum operation duration time within the entire instance
	Horizon	The estimated total length of a schedule, calculated by summing the estimated travel times of AGVs

2.2.5. Scoring System

Before being able to appoint the best algorithm for a specific problem instance, the performances of the algorithms for that instance need to be compared with one another. Therefore, a scoring system was introduced. In order to measure the performance of the different algorithms for specific problem instances, each of the proposed solution algorithms was applied to each instance. The performance measure to compare the algorithm performance in this study was the calculated makespan of the instance. The scoring system assigns points to an algorithm depending on achieved results. If an algorithm achieves the optimal solution faster or finds a better solution than all other algorithms, it obtains a point. If the performance metric is equal, the algorithm requiring less computational time to solve the instance obtains the point. This scoring system was inspired by the scoring system used in the MiniZinc Challenge [58].

2.2.6. Modeling of CP Models

After selecting CP as a promising approach for solving scheduling problems during the analysis phase, the specific solvers still needed to be determined. There were a wide variety of solvers available for CP, and as the solution quality of traditional scheduling algorithms varied depending on the problem to solve, so did the performance of different CP solvers. The MiniZinc Challenge compared different solvers on a set of problem instances. MiniZinc is a solver-agnostic modeling language for defining and solving combinatorial satisfaction and optimization problems. The MiniZinc Challenge provided an overview of the current most successful solvers. OR-Tools is one of the top solvers, winning gold in the international constraint programming competition for several categories every year since 2013. OR-Tools is an open-source software suite for optimization developed by Google [59]. For solving CP problems, they provide two solvers: The CP-SAT solver, a CP solver that uses SAT methods, and the original CP solver. The CP-SAT solver is the primary OR-Tools solver for constraint programming and uses techniques for solving SAT problems along with CP methods. The job-shop scheduling problem is a common problem for the CP-SAT solver. Additionally, another promising solver is CP Optimizer, developed by IBM, which has the goal of focusing on industrial optimization problems. Due to their recent developments and the successful implementation of these solvers in other studies, this study selected OR-Tools and CP Optimizer for optimizing the scheduling problems.

The OR-Tools implementation for the JSSP and FJSSP was programmed with the Python API and used the CP-SAT solver provided by Google. The version used was 9.4.1874. CP Optimizer is a constraint optimization solver that is part of the IBM ILOG CPLEX Optimization Studio, with a focus on scheduling problems. This study used the

Python API to program the optimization in Python with the CP Optimizer interface, CPLEX version 22.1.0.0, and docplex version 2.23.221.

2.2.7. Algorithm Selection with Machine Learning

Due to the introduced scoring system, machine learning algorithms can learn which solver tends to provide satisfying results for a given instance. Machine learning can be divided into three major areas: supervised learning, unsupervised learning, and reinforcement learning [60]. Supervised learning is the form of machine learning that is most widely used in practice [61].

The abovementioned problem was modeled as a multi-class classification problem, which implies classes and a ground truth. The classes were the winning CP solvers, either CP Optimizer (CPO) or OR-Tools, and the previously determined features. The goal was to learn a mapping from inputs x that corresponded to the features of the dataset and to outputs y, where $y \in \{1, \ldots, K\}$, with K being the number of classes, in this case the set of solution algorithms.

There are several classification algorithms available for predicting the output of a set of labeled input features. Among the most popular classification algorithms are logistic regression, decision trees, random forest, K-nearest neighbors (KNN), support vector machine, and naïve Bayes. Furthermore, neural networks are also applicable to classification problems.

To make the final selection of one machine learning algorithm to determine the optimal solver, they have to be tested on the dataset. This is because their performance significantly depends on the given input data. Therefore, only a pre-selection based on preliminary criteria is reasonable, and the final decision takes place after applying them to the problems. Due to their characteristics and suitability for the algorithm selection approach, logistic regression, decision trees, random forest, KNN, and neural networks were pre-selected. Logistic regression uses the sigmoid function to return the probability of a label. It is often used for binary classification problems, for instance, true or false and positive or negative. The decision tree algorithm is one of the most popular algorithms in use today. Decision trees build tree branches using a hierarchy approach, where each branch can be considered to be an if–else statement [62,63]. The branches develop by partitioning the dataset into subsets based on the most important features. Their scale of explainability was ranked comparably very high in a study assessing several machine learning techniques in terms of their explainability. Especially when considering the fear of a majority regarding the black-box character of machine learning, this technique might provide broader consent [64]. Random forest represents a collection of decision trees. Because of the additional layers added to the model, random forest has better generalization but is less interpretable. KNN represents each data point in an n-dimensional space and calculates the distance between one point and another. Afterwards, it assigns labels to unobserved data based on the labels of the nearest observed data points [63]. Neural networks represent a set of algorithms that are modeled by the inspiration of the human brain. In this way, they are supposed to recognize patterns in data.

The algorithms were implemented with the Scikit-learn library, an open-source machine learning library in Python for predictive data analysis. It is publicly available and was built using NumPy, SciPy, and matplotlib. Furthermore, the neural network was implemented with two approaches, one approach with Scikit-learn and the other with Keras. It is the most widely used deep learning framework among the five top teams on Kaggle. The parameters were set to defaults if not otherwise mentioned.

3. Results and Discussion

The proposed algorithm selector and the scheduling algorithms were programmed in Python version 3.9 and run on an 11th generation Intel® Core i7-1165G7 @ 2.80 GHz processor with 16.0 GB of RAM. The time limits of the solvers were set to 30 s and 300 s to ensure realistic calculation times and to also test the effects of extending the time limit.

The experiments were conducted on all presented instances, and the solvers successfully solved the instances.

In order to ensure the correctness of the algorithms, the results were compared to the results of [65] for the FJSSP benchmark instances and the JSSP with the benchmark instance collection [66]. The comparison showed similar results, which ensured the overall correctness of the algorithms. Furthermore, for some instances, CP Optimizer or the OR-Tools solver found better values for the makespan. This illustrated the development of solvers over the years and their improvements in solving larger instances faster.

3.1. Computational Study on Benchmark Instances with CP Solvers

Before going into detail about the solvers' performances per instance, we present the evaluation of the overall performance. The solvers achieved optimal results for 27% of the instances when given a time limit of 30 s and 43% of the instances after the extension of the time limit to 300 s for the JSSP. Regarding the FJSSP, the solvers calculated the optimum for 71.3% of instances with a 30 s time limit and 76.9% of instances with a 300 s time limit, as shown in Figure 4.

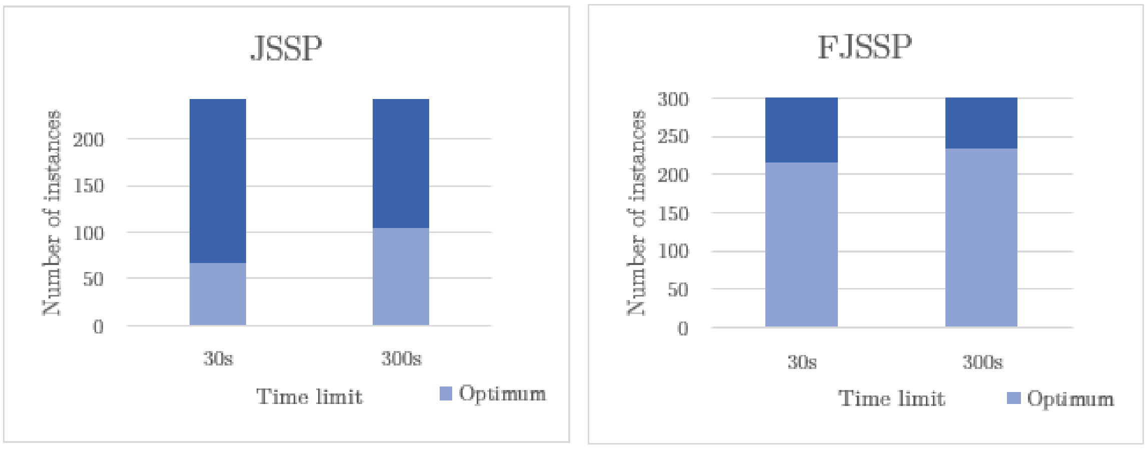


Figure 4. Optimal results for JSSP and FJSSP.

After calculating the makespan with the introduced solvers, the results were compared (Tables 4 and 5). During the computational study, it became clear that there was no overall winner for all problem instances. The distribution of wins for FJSSP instances was balanced between both solvers compared to the JSSP results. When extending the time limit to 300 s for calculating the makespan, the results for JSSP differed compared to the previous experiments with a 30 s time limit. For the FJSSP, the results were similar to the results with a 30 s time limit. However, the results showed that the winning solvers sometimes changed for individual instances when changing the time limit. This means that, for instance, OR-Tools provided better results with the 30 s time limits for one instance, while CP Optimizer provided a better makespan with a 300 s time limit. Consequently, the time limit influenced the performance of an algorithm for a specific instance. Nevertheless, this was mostly applicable for equal makespan values calculated by both solvers but with different solving times.

Table 4. Scoring points of CP Optimizer and OR-Tools for JSSP instances.

CP Solver	**30 s**	**300 s**
OR-Tools	50	71
CP Optimizer	192	171

Table 5. Scoring points of CP Optimizer and OR-Tools for FJSSP instances.

CP Solver	30 s	300 s
OR-Tools	148	146
CP Optimizer	155	157

To evaluate the effects of the algorithm selection approach for industrial applications, the assessment of the makespan was important. For 54 instances of the JSSP with a 30 s solving time, both solvers achieved the same makespan. Based on a time limit of 300 s, the solvers calculated the same makespan for 72 instances. In addition, it was interesting to evaluate the impact on the makespan by extending the time limit. For JSSP instances, the extension of the time limit to 300 s achieved time savings of 2.5% over all tested instances. Moreover, the makespan improvements by selecting between two solvers achieved a 5.9% decrease in the makespan for the 30 s time limit and a 2.9% decrease in the makespan for 300 s (Figure 5). Consequently, the selection has the potential to strongly influence the improvement in the overall productivity of production. The smaller decrease when extending the time limit is not surprising because the longer solving time led to a convergence of solutions at the same time. For instance, with a 300 s time limit, the solvers achieved more equal values for the makespan compared to the 30 s solving time. Consequently, especially for use cases where a smaller calculation time is necessary, for example, dynamically reacting production systems, an algorithm selection can be useful and can increase production performance by up to 5.9%.

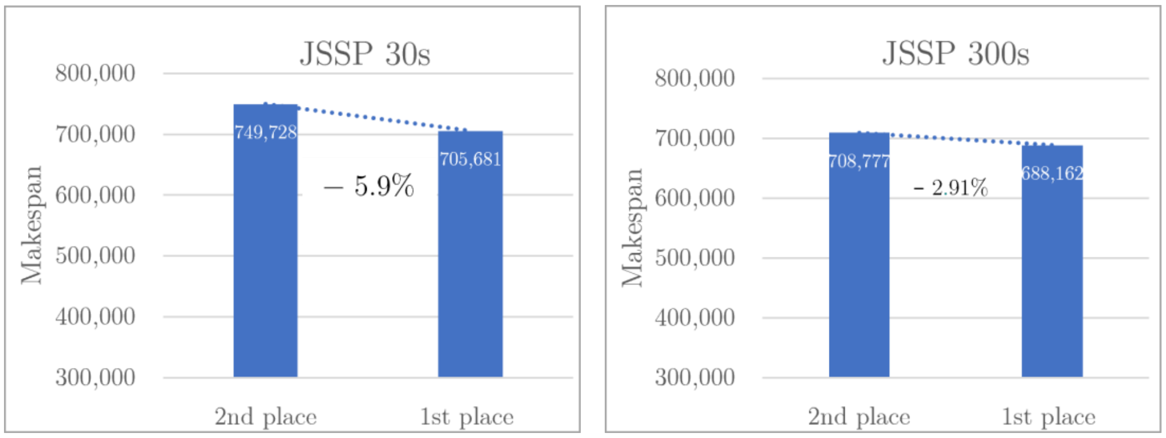


Figure 5. Makespan differences by algorithm selection per instance for JSSP.

Regarding the FJSSP, the extension of the time limit to 300 s achieved, over all instances, time savings of 0.18% compared to the winning makespan values for 30 s. Analyzing the minimization of the makespan by selecting the two solvers revealed a saving of 6329 time units for the 30 s time limit and 1245 time units for the 300 s time limit, calculated over all tested instances (Figure 6). This showed that the extension of the time limit did not substantially improve the overall performance regarding the makespan.

Within 30 s of solving time, both solvers achieved the same makespan for 187 instances. With a 300 s solving time, both solvers calculated the same makespan for 219 instances. Therefore, there were only small changes between first and second place regarding the makespan metrics. For the 30 s solving time, the difference was 1.31%, and for 300 s the makespan was reduced by 0.3%. Nevertheless, for those instances, the individual solving times deviated widely. Taking the solver with the smaller calculation time saved 77.5% in the calculation time for the 300 s time limit in comparison to the solving time of the second-choice solver and 43.1% for the 30 s maximum calculation time.

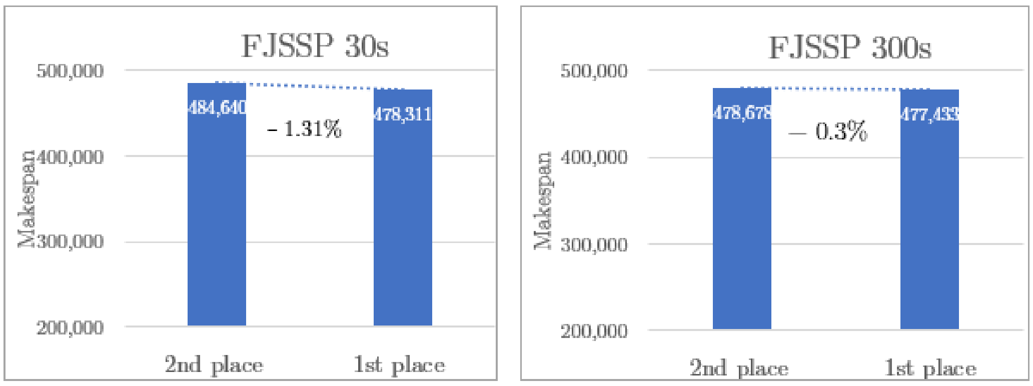


Figure 6. Makespan differences by algorithm selection per instance for FJSSP.

Considering the JSSP instances, there were fewer similar makespan values. Nevertheless, when comparing those instances with the similarly calculated makespan, the time savings by selecting between solvers was high as well. For the 30 s overall time limit, the selection saved 50.3% in the calculation time and 22.9% for a 300 s overall time limit.

Although the solvers achieved the same makespan, the difference in calculation time was high, and the algorithm selection could save up to 77.5% in the calculation time. Therefore, especially in industrial environments where time is an expensive factor, an algorithm selection can save costs by reducing the computational time and the production makespan.

3.2. Machine Learning Training and Evaluation

After implementing the selected machine learning models, they were trained on the designed datasets. Furthermore, feature engineering ensured the assessment of feature importance. Three different methods were applied for measuring feature importance: a heatmap with the Pearson correlations, the built-in class feature importance of tree-based classifiers, and random forest classifiers for evaluating feature importance. After selecting the features with the highest correlation to the output, the machine learning models were trained. The evaluation phase was ensured to test the robustness of the models and their validity. It implied the determination of the correct machine learning models to ensure the meeting of requirements and to solve the initial problem. For classification problems, the accuracy, recall, precision, and F1 score are key performance indicators that build on the principles of true positives, false negatives, true negatives, and false positives. The results of the JSSP dataset evaluation are presented in Tables 6 and 7.

Table 6. Performance metrics of machine learning models on JSSP datasets.

Machine Learning Algorithm	Solving Time	Parameter Tuning	Accuracy	Recall	Precision	F1 Score
Decision Tree	30 s	Depth = 1	0.8361	0.8361	0.6990	0.7614
	300 s	Depth = 17	0.8033	0.8033	0.8487	0.8254
Random Forest	30 s	Estimators = 100	0.8033	0.8033	0.8033	0.8033
	300 s	Estimators = 100	0.8525	0.8525	0.8580	0.8553
Logistic Regression	30 s	Max_iter = 1500	0.8197	0.8197	0.8580	0.8384
	300 s	Max_iter = 1500	0.8197	0.8197	0.8898	0.8533
KNN	30 s	Neighbors = 18	0.8525	0.8746	0.8525	0.8634
	300 s	Neighbors = 20	0.8525	0.8525	0.8750	0.8636

The tested models for classification problems provided successful implementations. Nevertheless, depending on the problem class, the performance altered. For the JSSP dataset with a solving time of 30 s, the neural network using the Keras implementation, decision trees, and KNN provided the best results, considering the tested performance metrics. For the 300 s solving time, random forest and KNN provided the best accuracy scores. However, the close values of the performance metrics made a concrete decision for the best machine learning technique difficult. Therefore, the selection of the best machine learning model must consider the dataset used.

Table 7. Performance metrics of the neural network for JSSP datasets.

Feed-Forward Neural Network *Sklearn*	**Solving Time**	**Accuracy**	**Recall**	**Precision**	**F1 Score**
Random_state = 43 activation = logistic	30 s	0.6885	0.6885	0.7416	0.7116
max_iter = 1000 hidden layer sizes (50,60)	300 s	0.7377	0.7377	0.8470	0.7661
Feed-Forward Neural Network *Keras*		**Accuracy**		**Loss**	
Loss = sparse categorical cross entropy activation = ReLU	30 s	0.8361		2.8486	
Optimizer = Adam hidden layer = 1 with density of 100	300 s	0.5902		2.8396	

The performance metrics for the FJSSP datasets are presented in Tables 8 and 9.

Table 8. Performance metrics of machine learning models for FJSSP datasets.

Machine Learning Algorithm	**Solving Time**	**Parameter Tuning**	**Accuracy**	**Recall**	**Precision**	**F1 Score**
Decision Tree	30 s	Depth = 2	0.7500	0.7500	0.7611	0.7555
	300 s	Depth = 2	0.6711	0.6711	0.6892	0.6800
Random Forest	30 s	Estimators = 300	0.6842	0.6842	0.6856	0.6849
	300 s	Estimators = 2	0.5789	0.5789	0.5789	0.5789
Logistic Regression	30 s	Max_iter = 3000	0.6974	0.6974	0.6979	0.6976
	300 s	Max_iter = 3000	0.6447	0.6447	0.6438	0.6442
KNN	30 s	Neighbors = 8	0.7105	0.7105	0.7113	0.7101
	300 s	Neighbors = 21	0.6842	0.6842	0.6865	0.6854

Table 9. Performance metrics of the neural network for FJSSP datasets.

Feed-Forward Neural Network *Sklearn*	**Solving Time**	**Accuracy**	**Recall**	**Precision**	**F1 Score**
Random_state = 43 activation = logistic	30 s	0.7500	0.7500	0.7611	0.7491
max_iter = 1000 hidden layer sizes (50,60)	300 s	0.6711	0.6711	0.6892	0.6705
Feed-Forward Neural Network *Keras*		**Accuracy**		**Loss**	
Loss = sparse categorical cross entropy activation = ReLU Optimizer = Adam	30 s	0.6316		0.5866	
hidden layer = 1 with density of 100	300 s	0.6579		0.6266	

Compared to the JSSP datasets, FJSSP achieved overall weaker results in terms of accuracy, precision, and recall. The highest scores for the 30 s solving time were obtained by the neural network with Sklearn and the decision tree, with an accuracy of 75%. For the 300 s solving time dataset, KNN and decision trees achieved the highest performance metrics. Consequently, there was not one machine learning algorithm providing the best results. The structure of the input data is an important criterion for future prediction accuracy. Therefore, the assessment of machine learning algorithm performance on the given data structure must be part of implementing the algorithm approach for an AGV scheduling use case.

3.3. Validation of Machine Learning Models

An additional measure to test the validity of the machine learning models is cross-validation. During the evaluation, the splitting of data into training and test datasets already validated the models in terms of how they react to unseen data. However, this validation method can lead to sampling bias if one subset includes only some of the existing instance groups. Therefore, cross-validation served as another method to ensure the validity of the created models. K-fold cross-validation ensures more reliability in terms of accuracy and parameter determination. It is a technique that is commonly used to measure the performance and generalizability of a model and helps to reduce the chances of overfitting [67]. For the presented models, a five-fold cross-validation was applied to measure the overall performance (Table 10).

Table 10. Five-fold cross-validation of mean validation accuracy results.

Machine Learning Algorithm with Five-Fold Cross-Validation	JSSP 30 s	JSSP 300 s	FJSSP 30 s	FJSSP 300 s
Decision Tree	0.7510	0.6934	0.5970	0.5640
Random Forest	0.7263	0.7181	0.5937	0.5378
Logistic Regression	0.8094	0.7266	0.6600	0.6102
KNN	0.7309	0.6315	0.6404	0.5840
Neural network (Sklearn)	0.7843	0.6527	0.6338	0.6001

Furthermore, in order to validate the model, two new datasets were created with literature data and randomly created test instances. In this way, the machine learning models were validated with previously unseen data. To properly validate the results of the design phase, CP Optimizer and OR-Tools both solved the given instances to evaluate the prediction by the machine learning models. The comparison of the selector predictions with the actual results of the two solvers showed satisfying results. For the JSSP, the decision tree achieved an accuracy of 78.5%, the random forest model achieved an accuracy of 70.1%, and the neural network achieved an accuracy of 67.1%. For the FJSSP validation dataset, the accuracy scores of the correct prediction were even higher (Table 11).

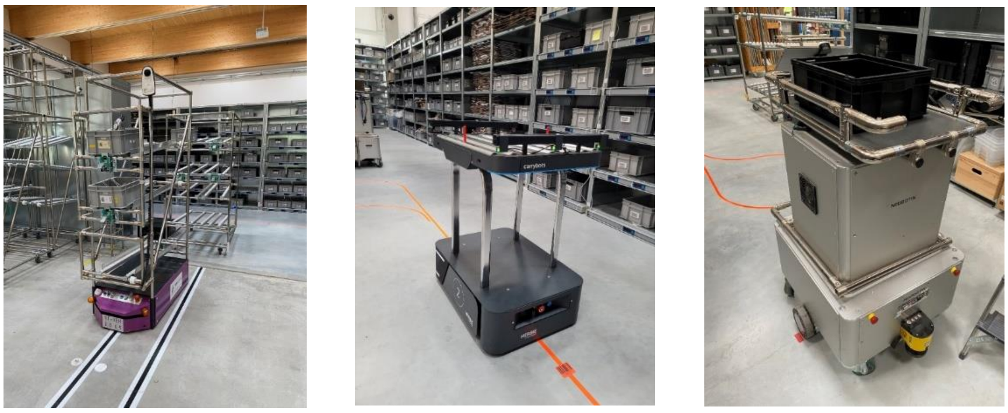
Table 11. Accuracy of classification models for validation datasets.

Algorithm Accuracy Scores	JSSP	FJSSP
Decision Tree	0.7848	0.6406
Random Forest	0.7089	0.9531
Logistic Regression	0.3924	0.8750
KNN	0.3797	0.8281
Neural network (Sklearn)	0.6709	0.8750

The testing of the individual machine learning models on the new datasets emphasized the interdependency between datasets and algorithm performance. Overall, the results showed high performance metrics and consequently validated the concept of the algorithm selection approach for AGV scheduling problems.

3.4. Validation of Approach with Disruptions and Rescheduling

To test the approach in a running factory environment, the algorithm selector made predictions on a dataset created from real-world data from the learning factory “Werk 150” at Reutlingen University. The learning factory represents a scooter production that uses AGVs to transport goods between different stations. “Werk 150” is a production company that replicates processes in the areas of product and work system engineering, incoming goods, storage, order picking, production, assembly, and distribution, which are fundamental for the verification of a scheduling system for AGVs [68]. All necessary processes that impact AGV scheduling are present. For delivering parts, there are three different transport vehicles available (Figure 7). Two AGVs and one AMR, which travel with an average velocity of 0.9 m/s. The transport orders of the vehicles vary among the production, depending on the outstanding work step. The transport orders are in JSON format and include information about the operation, the machine ID, the area where the transport takes place, the line, and the planned quantity. This file format complies with VDA5050, a standardization approach in the automotive industry in Germany, which shall ensure conflict-free and transparent interfaces between AGV fleets and the central executing software system, independent of the manufacturer.



(**a**) (**b**) (**c**)

Figure 7. AGVs in "Werk 150": (**a**) AGV from the company Beewatec that navigates on a designed track, (**b**) AGV from the company Carrybots that travels along a marked track, and (**c**) AMR from the company Neobotix.

To test the functionality of the algorithm selector, the experiment included different scheduling scenarios. One scenario considered AGVs that were already assigned to tasks, whereas the second scenario offered alternatives for assigning the AGVs to orders. In this case, the scheduler allocated the eligible AGVs to individual tasks to minimize the makespan of the entire scenario. Furthermore, disruptions within the system were included to test the reaction of the approach to disturbances. The entering disruption included a timestamp that specified the already executed orders. These were deleted in the original scheduling problem in order to reschedule only the pending orders. In the tested scenario, an AGV failure appeared, and the system replaced the defective AGV with the corresponding substitute. After the adaptation, a new schedule was generated, and the execution of the plan continued, considering the downtime of the respective AGV.

4. Conclusions

This research focused on the analysis of AGV scheduling and on potential fields of artificial intelligence to improve the performance of current solutions. The purpose was to discover the possibility of increasing efficiency with the help of artificial intelligence for traditional scheduling approaches. By investigating current scheduling algorithms, constraint programming emerged as a promising solution technique, and the suitability of the developed artifact for dynamic scheduling approaches was tested with real-world data from a learning factory environment.

The algorithm selection model resulting from this research helped to save computational time and improve production efficiency by minimizing the makespan. The experimental study showed that the automated selection process can increase production performance by up to 5.9% and can decrease computational power by up to 77.5%. Furthermore, the research supported the statement of other researchers that CP improved and can solve large instances in a reasonable time. Additionally, CP is easy to adapt to different constraints and goals, which makes them dynamically adjustable for industrial requirements. The detailed performance evaluation of the two tested CP solvers showed great potential for selecting between both solutions, depending on the instance. Machine learning algorithms evaluated the performances of existing methods based on the characteristics of a given scheduling instance and achieved up to 88.9% precision in predicting the optimal solver for a given scheduling problem.

In the future, further parameters such as job priorities, real-time AGV location data, and battery constraints could be included. Additionally, the application of more objectives could be of great interest, especially when it comes to how this influences the performances of algorithms among various problem instances. This work concentrated on the implementation of constraint programming to solve the scheduling problem. Future research could test more algorithms, such as tabu search or genetic algorithms, to compare the performances among the algorithm categories.

Author Contributions: Conceptualization, F.S., G.B. and L.L.; methodology, F.S.; software, F.S.; validation, F.S.; writing—original draft preparation, F.S.; writing—review and editing, G.B. and L.L.; visualization, F.S.; supervision, G.B. and L.L. All authors have read and agreed to the published version of the manuscript.

Funding: The article processing charge was funded by the Baden-Württemberg Ministry of Science, Research, and Arts and by Reutlingen University through the funding program for open-access publishing.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The created codes for the constraint programming solvers are stored in the following repository on GitHub: https://github.com/feli-s/algorithm-selector-for-AGV-scheduling.git, accessed on 19 December 2022. Additionally, the created datasets for the machine learning training are also available on this repository.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Kagermann, H. Change Through Digitization—Value Creation in the Age of Industry 4.0. In *Management of Permanent Change*; Albach, H., Meffert, H., Pinkwart, A., Reichwald, R., Eds.; Springer Fachmedien Wiesbaden: Wiesbaden, Germany, 2015; pp. 23–45, ISBN 978-3-658-05014-6.
2. Fragapane, G.; de Koster, R.; Sgarbossa, F.; Strandhagen, J.O. Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. *Eur. J. Oper. Res.* **2021**, *294*, 405–426. [CrossRef]
3. Vis, I.F. Survey of research in the design and control of automated guided vehicle systems. *Eur. J. Oper. Res.* **2006**, *170*, 677–709. [CrossRef]
4. Zhang, L.; Yan, Y.; Hu, Y.; Ren, W. A dynamic scheduling method for self-organized AGVs in production logistics systems. *Procedia CIRP* **2021**, *104*, 381–386. [CrossRef]
5. Pinedo, M.L. *Scheduling: Theory, Algorithms, and Systems*, 5th ed.; Springer International Publishing: Cham, Switzerland, 2016; ISBN 978-3-319-26580-3.
6. Hu, H.; Jia, X.; He, Q.; Fu, S.; Liu, K. Deep reinforcement learning based AGVs real-time scheduling with mixed rule for flexible shop floor in industry 4.0. *Comput. Ind. Eng.* **2020**, *149*, 106749. [CrossRef]
7. Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.H.; Shmoys, D.B. Chapter 9 Sequencing and scheduling: Algorithms and complexity. In *Handbooks in Operations Research and Management Science: Logistics of Production and Inventory*; Elsevier: Amsterdam, The Netherlands, 1993; pp. 445–522, ISBN 0927-0507.
8. Fazlollahtabar, H.; Saidi-Mehrabad, M. Methodologies to Optimize Automated Guided Vehicle Scheduling and Routing Problems: A Review Study. *J. Intell. Robot. Syst.* **2015**, *77*, 525–545. [CrossRef]
9. Da Col, G.; Teppan, E.C. Industrial-size job shop scheduling with constraint programming. *Oper. Res. Perspect.* **2022**, *9*, 100249. [CrossRef]
10. Rice, J.R. TThe Algorithm Selection Problem**This work was partially supported by the National Science Foundation through Grant GP-32940X. This chapter was presented as the George E. Forsythe Memorial Lecture at the Computer Science Conference, February 19, 1975, Washington, D. C. In *Advances in Computers*; Rubinoff, M., Yovits, M.C., Eds.; Elsevier: Amsterdam, The Netherlands, 1976; pp. 65–118, ISBN 0065-2458.

11. Kotthoff, L. Algorithm Selection for Combinatorial Search Problems: A Survey. In *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*; Bessiere, C., de Raedt, L., Kotthoff, L., Nijssen, S., O'Sullivan, B., Pedreschi, D., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 149–190, ISBN 978-3-319-50137-6.
12. Messelis, T.; de Causmaecker, P. An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **2014**, *233*, 511–528. [CrossRef]
13. Drake, J.H.; Kheiri, A.; Özcan, E.; Burke, E.K. Recent advances in selection hyper-heuristics. *Eur. J. Oper. Res.* **2020**, *285*, 405–428. [CrossRef]
14. Yoon, B. A machine learning approach for efficient multi-dimensional integration. *Sci. Rep.* **2021**, *11*, 18965. [CrossRef]
15. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.-G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *Eur. J. Oper. Res.* **2022**, *296*, 393–422. [CrossRef]
16. Xu, L.; Hutter, F.; Hoos, H.H.; Leyton-Brown, K. SATzilla: Portfolio-based Algorithm Selection for SAT. *Artif. Intell. Res.* **2008**, *32*, 565–606. [CrossRef]
17. Müller, D.; Müller, M.G.; Kress, D.; Pesch, E. An algorithm selection approach for the flexible job shop scheduling problem: Choosing constraint programming solvers through machine learning. *Eur. J. Oper. Res.* **2022**, *302*, 874–891. [CrossRef]
18. Strassl, S.; Musliu, N. Instance space analysis and algorithm selection for the job shop scheduling problem. *Comp. Oper. Res.* **2022**, *141*, 105661. [CrossRef]
19. Hevner, A.R.; March, S.T.; Park, J.; Ram, S. Design Science in Information Systems Research. *MIS Q.* **2004**, *28*, 75. [CrossRef]
20. Österle, H. *Gestaltungsorientierte Wirtschaftsinformatik: Ein Plädoyer für Rigor und Relevanz*; Infowerk: Nürnberg, Germany, 2010; ISBN 978-3-00-030310-4.
21. Snyder, H. Literature review as a research methodology: An overview and guidelines. *J. Bus. Res.* **2019**, *104*, 333–339. [CrossRef]
22. Popper, J.; Yfantis, V.; Ruskowski, M. Simultaneous Production and AGV Scheduling using Multi-Agent Deep Reinforcement Learning. *Procedia CIRP* **2021**, *104*, 1523–1528. [CrossRef]
23. Ouelhadj, D.; Petrovic, S. A survey of dynamic scheduling in manufacturing systems. *J. Sched.* **2009**, *12*, 417–431. [CrossRef]
24. Chryssolouris, G.; Subramaniam, V. Dynamic scheduling of manufacturing job shops using genetic algorithms. *J. Intell. Manuf.* **2001**, *12*, 281–293. [CrossRef]
25. Mousavi, M.; Yap, H.J.; Musa, S.N.; Tahriri, F.; Md Dawal, S.Z. Multi-objective AGV scheduling in an FMS using a hybrid of genetic algorithm and particle swarm optimization. *PLoS ONE* **2017**, *12*, e0169817. [CrossRef]
26. Zheng, Y.; Xiao, Y.; Seo, Y. A tabu search algorithm for simultaneous machine/AGV scheduling problem. *Int. J. Prod. Res.* **2014**, *52*, 5748–5763. [CrossRef]
27. Glover, F.; Kelly, J.P.; Laguna, M. Genetic algorithms and tabu search: Hybrids for optimization. *Comp. Oper. Res.* **1995**, *22*, 111–134. [CrossRef]
28. Tang, J.; Liu, G.; Pan, Q. A Review on Representative Swarm Intelligence Algorithms for Solving Optimization Problems: Applications and Trends. *IEEE/CAA J. Autom. Sin.* **2021**, *8*, 1627–1643. [CrossRef]
29. Wu, X.; Zhang, M.-X.; Zheng, Y.-J. An Intelligent Algorithm for AGV Scheduling in Intelligent Warehouses. In *Advances in Swarm Intelligence*; Tan, Y., Shi, Y., Eds.; Springer International Publishing: Cham, Switzerland, 2021; pp. 163–173, ISBN 978-3-030-78743-1.
30. Zou, W.-Q.; Pan, Q.-K.; Wang, L. An effective multi-objective evolutionary algorithm for solving the AGV scheduling problem with pickup and delivery. *Knowl.-Based Syst.* **2021**, *218*, 106881. [CrossRef]
31. Erol, R.; Sahin, C.; Baykasoglu, A.; Kaplanoglu, V. A multi-agent based approach to dynamic scheduling of machines and automated guided vehicles in manufacturing systems. *Appl. Soft Comput.* **2012**, *12*, 1720–1732. [CrossRef]
32. Baptiste, P.; Laborie, P.; Le Pape, C.; Nuijten, W. Chapter 2—Constraint-Based Scheduling and Planning. In *Foundations of Artificial Intelligence: Handbook of Constraint Programming*; Rossi, F., van Beek, P., Walsh, T., Eds.; Elsevier: Amsterdam, The Netherlands, 2006; pp. 761–799, ISBN 1574-6526.
33. Corréa, A.I.; Langevin, A.; Rousseau, L.-M. Scheduling and routing of automated guided vehicles: A hybrid approach. *Comp. Oper. Res.* **2007**, *34*, 1688–1707. [CrossRef]
34. Khayat, G.E.; Langevin, A.; Riopel, D. Integrated production and material handling scheduling using mathematical programming and constraint programming. *Eur. J. Oper. Res.* **2006**, *175*, 1818–1832. [CrossRef]
35. V Sagar, K.; Jerald, J. Real-time Automated Guided vehicles scheduling with Markov Decision Process and Double Q-Learning algorithm. *Mater. Today Proc.* **2022**, *64*, 279–284. [CrossRef]
36. Chen, C.; Hu, Z.-H.; Wang, L. Scheduling of AGVs in Automated Container Terminal Based on the Deep Deterministic Policy Gradient (DDPG) Using the Convolutional Neural Network (CNN). *JMSE* **2021**, *9*, 1439. [CrossRef]
37. Zhou, B.; Zhao, Z. A hybrid fuzzy-neural-based dynamic scheduling method for part feeding of mixed-model assembly lines. *Comput. Ind. Eng.* **2022**, *163*, 107794. [CrossRef]
38. Wang, X.; Wu, W.; Xing, Z.; Chen, X.; Zhang, T.; Niu, H. A neural network based multi-state scheduling algorithm for multi-AGV system in FMS. *J. Manuf. Syst.* **2022**, *64*, 344–355. [CrossRef]
39. Song, H.; Triguero, I.; Özcan, E. A review on the self and dual interactions between machine learning and optimisation. *Prog. Artif. Intell.* **2019**, *8*, 143–165. [CrossRef]
40. Talbi, E.-G. Combining metaheuristics with mathematical programming, constraint programming and machine learning. *Ann. Oper. Res.* **2016**, *240*, 171–215. [CrossRef]

41. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.* **2021**, *290*, 405–421. [CrossRef]
42. Geitz, M.; Grozea, C.; Steigerwald, W.; Stöhr, R.; Wolf, A. Solving the Extended Job Shop Scheduling Problem with AGVs—Classical and Quantum Approaches. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*; Schaus, P., Ed.; Springer International Publishing: Cham, Switzerland, 2022; pp. 120–137, ISBN 978-3-031-08011-1.
43. Cordeau, J.-F. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Oper. Res.* **2006**, *54*, 573–586. [CrossRef]
44. van Hoorn, J.J. The Current state of bounds on benchmark instances of the job-shop scheduling problem. *J. Sched.* **2018**, *21*, 127–128. [CrossRef]
45. Fisher, H.; Thompson, G.L. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial Scheduling*; Muth, J.F., Thompson, G.L., Eds.; Prentice-Hall: Englewood Cliffs, NJ, USA, 1963; pp. 225–251.
46. Lawrence, S. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*; GSIA, Carnegie Mellon University: Pittsburgh, PA, USA, 1984.
47. Adams, J.; Balas, E.; Zawack, D. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Manag. Sci.* **1988**, *34*, 391–401. [CrossRef]
48. Applegate, D.; Cook, W. A Computational Study of the Job-Shop Scheduling Problem. *ORSA J. Comput.* **1991**, *3*, 149–156. [CrossRef]
49. Storer, R.H.; Wu, S.D.; Vaccari, R. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Manag. Sci.* **1992**, *38*, 1495–1509. [CrossRef]
50. Yamada, T.; Nakano, R. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. In *Parallel Problem Solving from Nature, 2*; Manner, R., Manderick, B., Eds.; Elsevier Science Publishers, BV: Amsterdam, The Netherlands, 1992.
51. Taillard, E. Benchmarks for basic scheduling problems. *Eur. J. Oper. Res.* **1993**, *64*, 278–285. [CrossRef]
52. Demirkol, E.; Mehta, S.; Uzsoy, R. A Computational Study of Shifting Bottleneck Procedures for Shop Scheduling Problems. *J. Heuristics* **1997**, *3*, 111–137. [CrossRef]
53. Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.* **1993**, *41*, 157–183. [CrossRef]
54. Chambers, J.B.; Barnes, J.W. (Eds.) *Tabu Search for the Flexible-Routing Job Shop Problem*; Technical Report Series ORP96-10, Graduate Program in Operations Research and Industrial Engineering; The University of Texas: Austin, TX, USA, 1996.
55. Dauzère-Pérès, S.; Paulli, J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Ann. Oper. Res.* **1997**, *70*, 281–306. [CrossRef]
56. Hurink, J.; Jurisch, B.; Thole, M. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Oper.-Res.-Spektrum* **1994**, *15*, 205–215. [CrossRef]
57. Strassl, S. Instance Space Analysis for the Job Shop Scheduling Problem. Diploma Thesis, Technische Universität Wien, Wien, Austria, 2020.
58. Stuckey, P.J.; Feydy, T.; Schutt, A.; Tack, G.; Fischer, J. The MiniZinc Challenge 2008–2013. *AIMag* **2014**, *35*, 55–60. [CrossRef]
59. Google Developers. Google OR-Tools. Available online: https://developers.google.com/optimization (accessed on 3 October 2022).
60. Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. *Science* **2015**, *349*, 255–260. [CrossRef]
61. Murphy, K.P. *Machine Learning: A Probabilistic Perspective*; MIT Press: Cambridge, MA, USA, 2012; ISBN 9780262304320.
62. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
63. Gong, D. Top 6 Machine Learning Algorithms for Classification: How to Build a Machine Learning Model Pipeline in Python. Available online: https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501 (accessed on 5 October 2022).
64. Oxborough, C.; Cameron, E.; Rao, A.; Birchall, A.; Townsend, A.; Westermann, C. Explainable AI Driving Business Value through Greater Understanding. Available online: https://www.pwc.co.uk/audit-assurance/assets/explainable-ai.pdf (accessed on 19 December 2022).
65. Behnke, D.; Geiger, M.J. *Test Insances for the Flexible Job Shop Scheduling Problem with Work Centers*; Helmut-Schmidt-University: Hamburg, Germany, 2012. [CrossRef]
66. van Hoorn, J.J. Job Shop Instances and Solutions. Available online: http://jobshop.jjvh.nl (accessed on 18 December 2022).
67. Berrar, D. Cross-Validation. In *Encyclopedia of Bioinformatics and Computational Biology*; Ranganathan, S., Gribskov, M., Nakai, K., Schonbach, C., Eds.; Academic Press: Oxford, UK, 2019; pp. 542–545, ISBN 978-0-12-811432-2.
68. Hummel, V. ESB Logistics Learning Factory: The Authentic Learning, Research and Development Environment at ESB Business School. Available online: https://www.esb-business-school.de/en/research/training-and-research-centre-added-value-and-logistics-systems/research-infrastructure/werk150/ (accessed on 8 August 2022).