applied sciences

Article

Deep Reinforcement Learning for Selection of Dispatch Rules for Scheduling of Production Systems

Kosmas Alexopoulos *, Panagiotis Mavrothalassitis, Emmanouil Bakopoulos, Nikolaos Nikolakis and Dimitris Mourtzis

Laboratory for Manufacturing Systems & Automation (LMS), Department of Mechanical Engineering & Aeronautics, University of Patras, Rio, 26504 Patras, Greece; mavrothalassitis@lms.mech.upatras.gr (P.M.); bakopoulos@lms.mech.upatras.gr (E.B.); nikolakis@lms.mech.upatras.gr (N.N.); mourtzis@lms.mech.upatras.gr (D.M.)

* Correspondence: alexokos@lms.mech.upatras.gr; Tel.: +30-2610-910160

Abstract: Production scheduling is a critical task in the management of manufacturing systems. It is difficult to derive an optimal schedule due to the problem complexity. Computationally expensive and time-consuming solutions have created major issues for companies trying to respect their customers' demands. Simple dispatching rules have typically been applied in manufacturing practice and serve as a good scheduling option, especially for small and midsize enterprises (SMEs). However, in recent years, the progress in smart systems enabled by artificial intelligence (AI) and machine learning (ML) solutions has revolutionized the scheduling approach. Under different production circumstances, one dispatch rule may perform better than others, and expert knowledge is required to determine which rule to choose. The objective of this work is to design and implement a framework for the modeling and deployment of a deep reinforcement learning (DRL) agent to support short-term production scheduling. The DRL agent selects a dispatching rule to assign jobs to manufacturing resources. The model is trained, tested and evaluated using a discrete event simulation (DES) model that simulates a pilot case from the bicycle production industry. The DRL agent can learn the best dispatching policy, resulting in schedules with the best possible production makespan.

Keywords: artificial intelligence; deep reinforcement learning; production scheduling; deep Q-learning; discrete event simulation

Academic Editors: Jose Machado, Ran Ji and Zhengyang Fan

Received: 4 November 2024
Revised: 21 December 2024
Accepted: 26 December 2024
Published: 30 December 2024

Citation: Alexopoulos, K.; Mavrothalassitis, P.; Bakopoulos, E.; Nikolakis, N.; Mourtzis, D. Deep Reinforcement Learning for Selection of Dispatch Rules for Scheduling of Production Systems. *Appl. Sci.* **2025**, *15*, 232. https://doi.org/10.3390/app15010232

1. Introduction

In the era of Industry 4.0, manufacturing technologies are undergoing a revolutionary transformation, embracing digitalization and intelligent automation for a new era of interconnected and smart production [1]. In the new manufacturing revolution, called Industry 4.0, the digitization of all systems that comprise a manufacturing system is a key objective towards smart manufacturing [2,3]. Information technology is used in various applications, including data collection, data management, knowledge development and sophisticated tasks such as the scheduling of complex production systems [4]. Scheduling is crucial in connecting customer orders to the production capacity, aiming to realize the efficient production of high-quality products. This involves determining the optimal sequence and timing for the processing of jobs with the available resources, considering complexities such as resource constraints, job characteristics and the dynamic nature of manufacturing environments, utilizing optimization methods and advanced technologies like artificial intelligence (AI) [5]. Due to its importance, production scheduling has been studied for several decades by many researchers. In particular, a genetic algorithm-based scheduling approach has been shown to enhance the adaptability in manufacturing job shops through the iterative optimization of task sequencing [6]. In parallel, a cloud-based cyber-physical system for adaptive shop-floor scheduling and condition-based maintenance integrates real-time data, resulting in more responsive and resilient operations [7]. The importance of synchronizing both domains for holistic optimization was addressed by a systematic review of integrated production scheduling and process control [8]. Another work reports a real-time scheduling method tailored to Industry 4.0 conditions, which addresses uncertainties like fluctuating job arrivals and machine breakdowns, ensuring robust performance in dynamic manufacturing environments [9]. Drawing on data from cyber-physical production systems (CPPS) and digital twin (DT) frameworks [10], innovative, data-driven approaches have emerged to effectively tackle scheduling problems. This study focuses on optimizing production scheduling using a deep reinforcement learning (DRL) agent for job shop problems. Trained and tested within a discrete event simulation (DES) model, and specifically designed to simulate a pilot case from the bicycle production industry, the DRL agent dynamically selects dispatching rules to achieve the best possible production makespan. More specifically, the contributions of this work are as follows:

1. We propose Markov decision process (MDP) modeling for a DRL approach to a scheduling problem;
2. We demonstrate that deep neural networks (DNN) and reinforcement learning (RL) can be effectively used for the selection of the best dispatching policy (one rule per resource) given a scheduling problem instance;
3. We present the design, modeling and implementation of a framework for the training and testing of a DRL scheduling agent;
4. We present the validation of the DRL framework and a performance analysis of the DRL scheduling agent based on an industrial case from the bicycle industry.

The remainder of this work is organized as follows: Section 2 reviews the development and latest research in scheduling with a focus on the applications enabled by RL, Section 3 presents the proposed approach to designing, modeling and implementing a scheduling framework, Section 4 presents the implementation of the proposed scheduling solution, and Section 5 presents the results of the experiments and the comparison of the proposed DRL scheduling agent with typical dispatch rules using data from the bicycle production industry. Finally, in Section 6, the conclusions and future directions for research are discussed.

2. Literature Review

In manufacturing, scheduling refers to the process of arranging, controlling and optimizing the work and workloads in a production system [11]. The fundamental objective is to arrange the manufacturing activities in such a way that the production costs and lead times are minimized, and quality standards are achieved. In industrial practice, dispatch rules represent one of the approaches used for scheduling, as they are simple to implement and they can deliver acceptable solutions quickly In the literature, it has been demonstrated how dynamic modifications to dispatching rules and the incorporation of real-time data can increase the overall efficiency, decrease lead times and improve responsiveness [12–14]. Although the first two cited studies focus on customizing rule selection for particular manufacturing contexts, such as general systems or unique production, the latter offers a comprehensive overview that situates these strategies within the changing Industry 4.0 environment, emphasizing the value of informed, data-driven decision-making. In the past few years, more sophisticated scheduling methods have been investigated, such as smart search [15–17] or genetic algorithms [6,18]. A resource planning methodology specifically tailored to the installation phase of industrial product–service systems has been proposed, integrating advanced planning and forecasting techniques to ensure the efficient allocation of personnel and equipment, ultimately reducing delays and improving coordination among stakeholders [15]. In [16], an integrated, simulation-based approach is introduced for refinery short-term scheduling that concurrently manages tank farms, inventories and distillation processes. Through the holistic modeling of interconnected operations, it achieves optimized production sequencing, minimized downtimes and improved overall refinery performance. Cai proposed a real-time scheduling and simulation optimization framework for job shops operating in a production–logistics collaborative environment. By continuously adjusting the schedules based on live data and accommodating both production and material flow constraints, it enhances the responsiveness, shortens lead times and aligns operational execution with the evolving shop floor conditions [17]. A dynamic scheduling approach was proposed for manufacturing job shops by employing genetic algorithms. Improved adaptability and efficiency were demonstrated as task sequencing and resource allocation were optimized in rapidly changing conditions [6]. A unified multi-objective genetic algorithm was introduced to achieve energy-efficient job shop scheduling. Reduced energy consumption and shortened production times were obtained by simultaneously optimizing multiple criteria, thereby enhancing the overall operational sustainability [18].

In practice, most of the time, there are deviations from the initial schedule due to unexpected events such as new jobs or machine breakdowns. In such cases, a mechanism to reschedule is needed. Much research has been carried out around the general concept of dynamic scheduling. A heuristic was developed for adaptive production scheduling and control within flow shop environments. Improved responsiveness and reduced lead times were achieved by continuously adjusting schedules based on the real-time conditions [19]. A two-layer dynamic scheduling method was proposed to minimize earliness and tardiness in re-entrant production lines. Enhanced schedule quality was obtained by layering decision-making processes to better handle complex production flows [20]. A dynamic scheduling method was introduced for integrated process planning and scheduling tasks affected by machine faults. More robust operational performance was ensured by rapidly adapting schedules to disruptions and maintaining the continuity of production [21]. An intelligent scheduling approach based on deep reinforcement learning was presented for discrete automated production lines. Improved scheduling quality and efficiency were demonstrated through the autonomous learning of optimal actions in complex operational settings [22]. An integrated scheduling method was developed, incorporating dispatching strategies and the conflict-free routing of autonomous mobile robots (AMRs) in flexible job shops. Enhanced coordination and reduced delays were realized by simultaneously optimizing job assignments and AMR paths [23]. A digital-twin-driven service model was proposed to optimize the allocation of manufacturing resources within a shared production environment. Significant improvements in resource utilization and operational flexibility were achieved, highlighting the value of virtualization and real-time data in modern manufacturing systems [24]. Heuristics [25], meta-heuristics [26], swarm optimization [27], immune algorithms [28], knowledge-based systems [29], fuzzy logic, neural networks [30–32] and hybrid techniques [33] are the predominant strategies that have been developed over the years to tackle the dynamic scheduling problem [34]. Real-time information during production on a shop floor is essential to identify the current situation, the level of progress until a certain checkpoint and the issues that have been raised, so as to react and execute a rescheduling task if needed [35].

Apart from the above-mentioned approaches, data-driven methods such as machine learning (ML) are receiving attention. A toolbox of agents was developed to schedule tasks within a bicycle industry paint shop. Improved coordination and reduced processing delays were achieved by dynamically assigning operations to the available resources [36]. A deep reinforcement learning framework was implemented to enhance tool life prior to failure. Increased efficiency and extended tooling usage were obtained by autonomously adjusting the cutting parameters based on real-time performance feedback [37]. A deep reinforcement learning approach was applied in assembly sequence planning while accounting for user preferences. Enhanced flexibility and user satisfaction were realized as the algorithm adapted assembly orders to specific constraints and requirements [38]. A comprehensive review of deep reinforcement learning methods was presented for smart manufacturing in the Industry 4.0 and 5.0 contexts. Emerging trends, potential benefits and practical implications were identified, offering insights for future research and industrial implementation [39]. The ascension of Industry 4.0 has positioned ML approaches as an appealing solution to tackle manufacturing challenges due to the availability of data, high computing power and large storage capacity. In this context, researchers have tried to develop and apply new technologies and algorithms [40]. An intelligent scheduling approach using reinforcement learning was developed and evaluated. Enhanced decision-making capabilities and improved scheduling performance were demonstrated by adapting to changing conditions in real time [41]. In another approach, to select the best dispatch rule for a machine at each decision point, a neural network was trained, using shop floor characteristics inherited from a simulation [42]. When there are two or more jobs in the queue, the model decides which rule a machine should follow in order to choose the next job for processing. However, despite the large amount of ML applications for scheduling, in recent years, another popular ML approach has garnered attention: reinforcement learning (RL).

Sutton and Barto [43] define RL as being simultaneously a problem, a class of solution methods that work well on a class of problems and the field in which these problems and their solution methods are studied. The key issue that this technique raises is that no prior knowledge of previously applied schedules is needed. The purpose of RL is for the agent to learn an optimal, or nearly optimal, policy that maximizes the reward. Hubbs et al. [44] presented a DRL agent for automatic decision-making for a chemical reactor operation. The model was successfully implemented in a simulation designed from historical production data, and, as expected, the testing results outperformed those of human schedulers. Another solution was proposed by Zhou et al. [45] that considered an AI scheduler with self-organizing and self-learning capabilities, developed through multiple training sessions and the formulation of a composite reward function, to enable scheduling for multi-objective learning. The trained scheduling agent decides the optimal job allocation directly via a single machine. A proximal policy optimization (PPO) combined with dispatch rules and a simulation was proposed by Wang and Liao [46] for the training of an agent to decide the optimal distribution of a job's tasks when a random job arrives. Tang and Salonitis [47] proposed a methodology to minimize reconfiguring actions in a production system. In their work, DRL was implemented to make autonomous decisions with a built-in DES model. As a result of self-learning, the agent outperformed the conventional First-In-First-Out (FIFO) dispatching rule by completing assigned order lists while minimizing reconfiguration actions. Kardos et al. [48] proposed a DRL-based approach to select the best resource for job processing in a given state. A deep reinforcement learning framework was developed to manage dynamic scheduling in smart manufacturing environments, leveraging real-time data for continuous decision-making. Notable improvements in responsiveness and resource utilization were reported, demonstrating the potential of AI-driven approaches to adapt schedules under shifting operational conditions [49]. Shi et al. [22] implemented DRL for the determination of process allocation to resources. The agent was able to handle a single process at a time. Thus, when iterations of this process occurred, the agent achieved the allocation of all processes on the machines.

Combining reinforcement learning and dispatch rules, a single machine agent that employs Q-learning was proposed by Wang and Usher [50]. Through RL, a decision-making policy is created for the selection of the appropriate dispatching rule for a resource each time this is required. The scheduling agent proposes the job allocation policy for a single machine. Improved scheduling performance and reduced processing times were achieved by dynamically adapting the dispatching decisions based on the current shop conditions. A variation considering the continuous state features as input in a deep Q-network (DQN) for the approximation of a state–action value function was proposed. The agent proposes a dispatching rule each time a new job arrives and needs to be assigned to a machine [51]. Zhang et al. [52] proposed a graph neural network (GNN) to solve job shop scheduling as the agent can learn dispatch rules from scratch with elementary raw features. The agent selects the optimal dispatch rule for job allocation at each decision point for the production resources. Luo [53] proposed a DRL approach implementing a DQN and dispatching rules for dynamic job scheduling to determine the most suitable dispatching rule at each rescheduling point. The agent selects one custom composite dispatching rule that will be applied to all machines. Qin et al. [54] proposed a multi-agent deep reinforcement learning approach for dynamic job shop scheduling environments. An adaptive multi-objective multi-task scheduling method was introduced, using hierarchical deep reinforcement learning to manage complex operational requirements under multiple objectives [55]. Meanwhile, a hybrid intelligence approach combining deep reinforcement learning with an attention mechanism was proposed to enhance dynamic job shop scheduling, offering improved responsiveness and decision quality [56]. Both studies underscore the growing influence of AI-driven techniques in optimizing modern manufacturing processes [56].

Nevertheless, the aforementioned approaches do not implement a large-scale solution. Most of the approaches consider either a single machine for job allocation or multiple machines' job allocation with the use of a single dispatch rule, which has limited performance. For scheduling in a more complex and real-world production system, Lin et al. [57] propose a multiclass DQN (MDQN) to solve the job shop problem with the use of dispatch rules. Each job has to be dispatched to a single machine when an order arrives. The input state consists of both static and dynamic parameters, while the action consists of groups of dispatching rules. More specifically, the output layer of the DQN consists of groups of nodes, with each group dedicated to a single machine. Thus, contrary to the classical DQN, the MDQN proposed by the authors selects multiple output nodes, one for each group. After training, the agent learns to select the best dispatching rule for each machine. However, this implementation refers to the scheduling of only one order at a time and its performance for larger-scale problems needs investigation. To address a larger solution, a multi-agent training approach with the use of deep reinforcement learning was proposed [58] in order to realize a cooperative multi-agent scheduling system. According to the authors, each agent is dedicated to one work center and all agents cooperate to provide the optimal policy.

To find the best dispatch policy, this work uses DRL. The action space consists of all possible combinations of dispatch rules and the available production resources at a given decision point. The DRL agent's mission is to find the best dispatching policy by selecting the best action, i.e., a set of dispatch rules.

3. Method

As already mentioned, the purpose of this work is to design, develop, implement and evaluate a scheduling solution based on RL, in order to support scheduling problems.

Before presenting the solution's architecture design, the main concepts of the RL approach are highlighted, as well as the problem's definition and objective. The main component in the proposed scheduling approach is a scheduling agent capable of learning optimal scheduling policies to dispatch jobs to machines for various workloads. To develop such a scheduling algorithm, the modeling of the scheduling problem and the RL method are addressed below.

3.1. Scheduling Problem Formulation

The production scheduling problem under study refers to job allocation to machines (Figure 1), in such a sequence that results in an optimal plan regarding a predefined objective. In this work, the objective of the scheduling algorithm will be the production makespan's minimization. For the scheduling problem model, the parameters of a production system should be defined. Moreover, the scheduling parameters and variables are defined below.

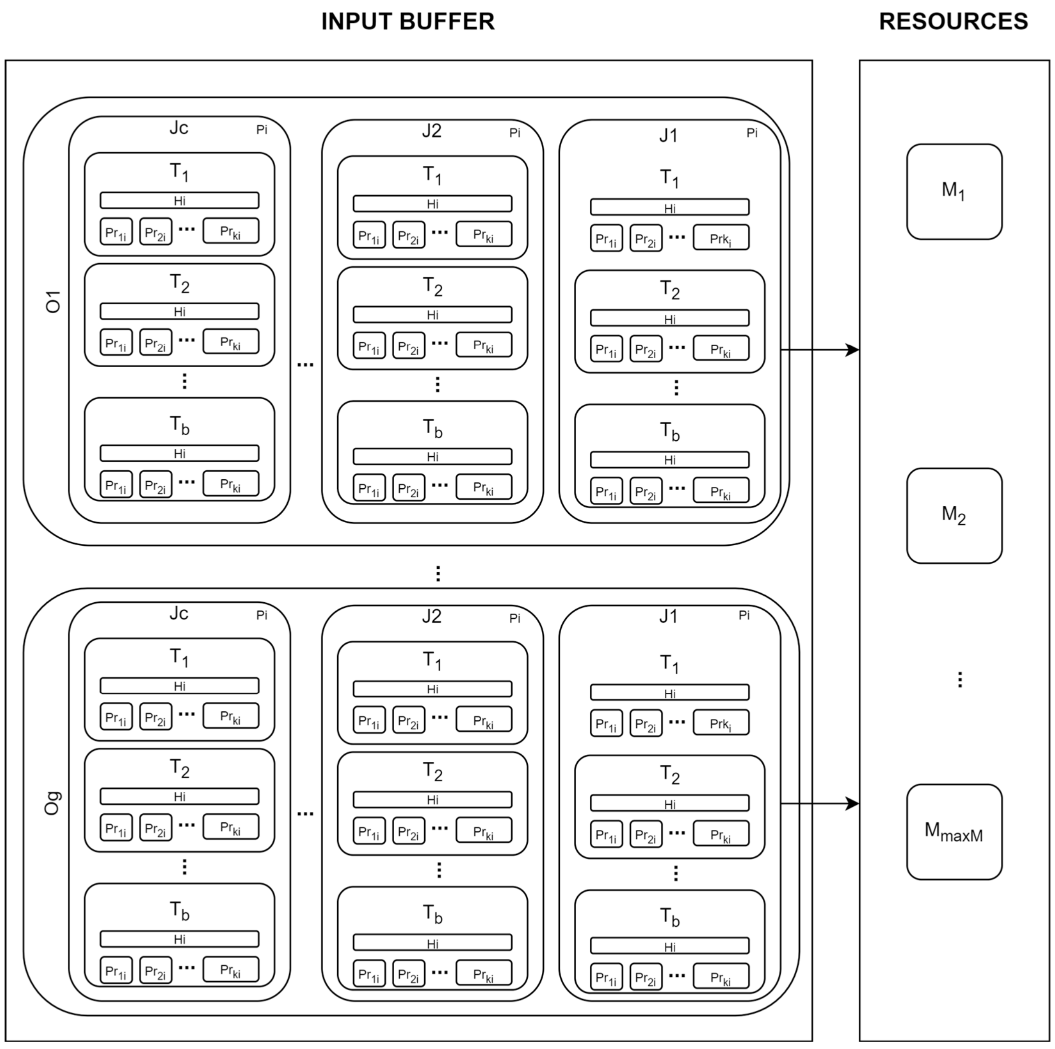

Figure 1. Overview of scheduling problem under study.

- Product types: A set of product type IDs is denoted as $P = \{P_1, P_2, \ldots, P_i\}$, with $i = 1, 2, \ldots, maxP$, where $maxP$ is the maximum number of product types that the production system can produce.
- Process plan: Each product type has a sequence of processes that define its process plan. Let us denote this set of processes as $H_i = \{Pr_{1i}, Pr_{2i}, \ldots, Pr_{ki}\}$, with

$k = 1, 2, \ldots, maxH_{Pi}$, where $maxH_{Pi}$ is the maximum number of processes in a product type P_i process plan. For the various product types, let us denote $H = \{H_1, H_2, \ldots, H_h\}$, with $h = 1, 2, \ldots maxP$.

- Resources: Let us denote the list of resources in a production system as $M = \{M_1, M_2, \ldots, M_r\}$ with $r = 1, 2, \ldots, maxM$, where $maxM$ is the maximum number of resources in the production system.
- Suitability and processing times: Let us denote the processing time of a process for a resource as $I(M_r, Pr_{ki}) \geq 0$. If a process Pr_{ki} cannot be executed by a resource M_r,j, then $I(M_r, Pr_{ki}) = 0$. In this way, the suitability of tasks for resources is also defined.
- Tasks: Let us denote as a task T_b, with $b = 1, 2, \ldots, maxT$, where $maxT$ is the maximum number of tasks T_b in a scheduling problem. For this scheduling problem, a product from a certain product type to be produced is represented as a task T_b.
- Jobs: Let us denote as a job J_c, with $c = 1, 2, \ldots, maxJ$, where $maxJ$ is the maximum number of jobs J_c in a scheduling problem. For this scheduling problem, a certain quantity of identical products to be produced is represented as a job J_c.
- Order: Let us denote as an order O_g, with $g = 1, 2, \ldots, maxO$, where $maxO$ is the maximum number of orders O_g in a scheduling problem. For this scheduling problem, a certain quantity of jobs J_c, regarding a specific product, is represented as an order O_g.
- Scheduling workload: The scheduling workload consists of all the orders O_g, tasks T_b and jobs J_c that need to be scheduled. The workload follows a hierarchical structure. The scheduling solution refers to the jobs' allocation to resources.

The above-mentioned parameters formulate the production system and the scheduling variables that need to be considered when formulating the scheduling input, as well as the production environment model. Apart from the above parameters, any other production constraints should be defined. For this work, the following constraints are defined.

- Constraint 1 (C1): Each machine can perform at most one process at a time.
- Constraint 2 (C2): A job's tasks should be processed by the same set of machines. This means that the tasks of a particular job cannot be processed by different workstations.
- Constraint 3 (C3): The transportation times and setup times are negligible.

3.2. Reinforcement Learning Approach

In order to apply the RL approach for the scheduling problem, the main concepts should be defined from the scheduling perspective. In the following subsections, the MDP, learning algorithm, environment, state, action and reward are defined.

The RL algorithm proposed in this work is rooted in Q-learning, which is implemented using deep learning techniques—specifically, a neural network. This approach is commonly referred to as deep reinforcement learning (DRL). The following sections will describe the Q-learning method and its variation when incorporating a deep neural network.

3.2.1. Q-Learning and Deep Q-Network

Q-learning is a fundamental reinforcement learning algorithm in the field of AI and ML, used to solve problems where an agent interacts with an environment to learn an optimal policy for sequential decision-making tasks. In reinforcement learning, an agent learns to make a sequence of decisions (actions) in an environment to maximize a cumulative reward. Q-learning is typically applied to problems modeled as MPDs, where the agent interacts with an environment composed of states, actions, transition probabilities and rewards. MDPs have the following components.

- State space (S): A set of all possible states that the agent can be in, where s represents the current situation in the environment.

- Action space (A): A set of all possible actions that the agent can take, where a denotes the decision that the agent makes in a given state s.
- Transition function (P): This describes the probability of transitioning from one state to another after taking a specific action.
- Reward function (R): This defines the immediate reward that the agent receives for taking an action in a particular state.
- Q-value (action value) function: This represents the expected cumulative reward that an agent can obtain by taking action a in state s and following an optimal policy thereafter.

The key insight behind Q-learning is the Bellman equation, which expresses the relationship between the Q-values of the current and future states:

$$Q(s,a) = R(s,a) + \gamma * \max\left[Q\left(s', a'\right)\right] \tag{1}$$

where $Q(s, a)$ is the Q-value for state–action pair (s, a), $R(s, a)$ is the immediate reward when taking action a in state s, γ (gamma) is the discount factor that represents the agent's preference for immediate rewards over future rewards and $max[Q(s', a')]$ represents the maximum Q-value over all possible actions in the next state, denoted as s'.

The idea of the Q-learning is to use the Bellman equation as an iterative update. The agent, at each decision point t in a state $s_t \in S$, selects an action $a_t \in A$ according to a policy π. After the agent takes the action a_t, it enters a new state s_{t+1} with the transition probability $p(s_{t+1}|s_t, a_t) \in P(S \times A \to S)$ and reward $r_t \in R$. The agent's objective is to find the optimal policy $\pi*$ that maximizes the expected sum of rewards. A policy is a rule that an agent follows when selecting an action by considering the given state that the agent is in. The model consists of an agent, a state space S and a set of actions for each state A (action space). All the information is stored in a Q-table, which consists of the Q-values between different states. The goal of the agent is to maximize the sum of reward r_t discounted by γ at each time step t, based on $\pi = P(a|s)$, after making an observation s and taking action α.

$$Q^{\pi}(s_t, a_t) = \max_{\pi}\mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \middle| s_t = s, a_t = a, \pi\right] \tag{2}$$

where α is the learning $0 < \alpha \leq 1$ and $\gamma \in (0,1]$. In a further analysis, when $\gamma \to 0$, this means that there are near-term goals, but when $\gamma \to 1$, this means that there are long-term rewards. Based on the Bellman optimality equation, the standard Q-learning algorithm can be derived. However, Q-learning has some limitations when the environment becomes more complicated. There are problems with the storage capacity when the state space is large. To solve this issue, neural networks are used and we implement the Q-learning method.

In order to overcome the limitations mentioned in the previous section regarding Q-learning's inability to deal with unknown states, the concept of the DQN is used. The DQN is an RL algorithm that combines Q-learning and DNNs to approximate the Q-values for high-dimensional state spaces. The DQN was proposed by Watkins and Dayan [59]. Mnih et al. [60] combined RL with deep learning techniques, which can be regarded as a neural network Q-function approximator with weights. The DQN that integrates deep learning and reinforcement learning is regarded as a powerful method in RL.

The DQN can be regarded as a neural network Q-function approximation with weights θ. It takes the tuple of features that characterizes a state as an input. The DQN can handle decision processes with a large state space. In this work, the DQN is used because of the huge state space that a scheduling problem might have. Let us denote as $Q(s, a; \theta_i)$ the approximate value using a deep convolutional neural network, where θ_i are the weights of the Q-network at iteration i. The experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time t are stored in a data set $D_t = \{e_1, \ldots, e_t\}$. Choosing uniformly at random an instance from the pool of stored instances, a Q-learning update is applied for each experience $(s, a, r, s') \sim U(D)$.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right] \tag{3}$$

where θ_i are the weights of the Q-network at iteration i, and θ_i^- are the network weights used to compute the target in iteration i. The θ_i^- weights are only updated for the Q-network weights θ_i at every c steps, where c is a constant number.

3.2.2. Environment

In the context of scheduling, the model is trained and evaluated in a simulated production system representing the environment. Since it is not feasible to use a real production system for training purposes, the simulation approach is a realistic and viable solution. The simulation offers a realistic and efficient alternative, enabling faster training under different scenarios [61]. Thus, apart from its feasibility, the training will be more efficient, since different situations can be modeled in a cost-effective manner. For this work, a DES is chosen. The operations in a manufacturing system can be modeled as discrete processes and thus it fits well with scheduling. A DES models the operation of a system as a sequence of events in time, where each event occurs at a particular instant in time and marks a change in status in the system.

A DES is used to model the behavior of a production system and simulate different production scenarios, which are used for training and testing purposes [62]. In the simulation model, the production elements' specific characteristics must be defined, such as input and output rules for the resources and buffers and the quantity of product units that each resource can handle at a time. Additionally, uncertainties in production, such as machine breakdowns, can also be modeled in a DES environment. Information regarding the sequence of each product type, the processing times and the suitability is also defined. Lastly, a DES gives the ability to easily calculate business key performance indicators (KPIs), such as the production makespan. The production makespan is the amount of time between the start and the completion of a production run.

3.2.3. State

The environment where the agent is trained should be adequately explored. Thus, the modeling of the environment from the agent's perspective is one of the most important aspects in such an ML approach. However, the environmental parameters that can be recognized by the RL agent should not consist of every asset in detail in a production system. The agent, in order to perform well and with stability and generalize, needs to identify the production status and the variables that matter. Hence, the overall status of the production at time t should be mapped in some predefined parameters that formulate the state s at time t. This state will be given as input to the DRL agent to calculate the output, which is the next action a in the environment.

State s_t is a vector of features representing the production status at a time t. The state consists of information regarding the following features:

- Jobs status—job IDs, types, the remaining quantity to be produced, the remaining cumulative processing time, the processing times and the suitability;
- Resource status—the available and unavailable resources (e.g., due to breakdown or performing another process).

In a production environment, the state space is huge, especially when uncertainties are included. Thus, it is important to ensure sufficient exploration and training to reduce the state space without missing any important information. The state vector consists of $maxJ$ sub-vectors, where $maxJ$ is the maximum number of jobs defined in the scheduling problem requirements. Each sub-vector has the following parameters.

- Job ID: In the workload, there are orders O_g with jobs J_c and tasks T_b, following a hierarchical structure. A job ID is an integer value representing a job J_c.
- Job type ID: A job type ID is an integer value representing the product type P_i of the job J_c.
- Job remaining workload: An integer value representing the number of remaining tasks of a job to be executed. Let us denote it as Jrw_c.
- Job minimum cumulative processing time: Each resource that can perform the tasks of a job has its own processing time. The minimum cumulative processing time is a float value representing the cumulative processing time of the tasks of a job, if this job is processed by the resource with the minimum processing time. Let us denote this as $Jtmin_c$.
- Job maximum cumulative processing time: Each resource that can perform the tasks of a job has its own processing time. The maximum cumulative processing time is a float value representing the cumulative processing time of the tasks of a job, if this job is processed by the resource with the maximum processing time. Let us denote this as $Jtmax_c$.
- Job processing time and suitability: A product type has one processing time per resource. If the tasks T_b of a job J_c are not suitable for the resource M_r or the resource M_r is unavailable, the value is 0. Thus, it is a list of values with a length equal to $maxM$. Let us denote it as $Jps_c = \{Jps_{c1}, Jps_{c2}, \ldots, Jps_{cr}\}$, with $r = 1, 2, \ldots, maxM$.

Thus, a state sub-vector is the following:

$$Ssub = \{J_c, P_i, Jrw_c, Jtmin_c, Jtmax_c, Jps_c\} \tag{4}$$

The state is the input of the DRL agent's neural network. The DRL agent is a DQN. Thus, the state vector's length should be defined. There are two limitations in this state approach:

- Maximum number of jobs $maxJ$;
- Maximum number of resources $maxM$.

The maximum number of jobs $maxJ$ is the maximum number of sub-vectors that can be formulated in the state vector. Let us denote the state vector length as

$$Ls = maxJ * Lsub \tag{5}$$

where the sub-vector length in this work is $Lsub = 5 + len(Jps)$, with $len(Jps) = maxM$. When defining the Ls and $Lsub$ values, the scheduling problem's specific characteristics should be taken into account. All the states that an environment can provide formulate the state space of the reinforcement learning approach. The state vector is the following:

$$s = \left\{Ssub_1, Ssub_2, \ldots Ssub_{maxJ}\right\} \tag{6}$$

3.2.4. Action

The DRL agent explores the environment by visiting the various states. The transition from a state s to a state s' is defined as action a. The agent shall perform actions to visit the states of the environment under exploration, and the set of all possible actions formulates the action space. Intuitively, the action of the DRL agent in a scheduling problem should be the next jobs' or tasks' allocation at a certain point in time. Indeed, this is the goal when defining the action vector and the action space. However, in this work, the DRL agent is a DQN, and the output layer of the neural network has to be strictly defined. Moreover, it is important to mention that, in each iteration, the DRL agent chooses an output node as the proposed action. In a production environment with varying workloads, the one-to-one mapping of job allocations to resources may differ. It is crucial to address this problem, since the solution that needs to be developed refers to the scheduling of different workloads.

One way to obtain a static output layer in a neural network and, at the same time, achieve job dispatching to resources is through the use of dispatch rules. Dispatch rules are independent of the scheduling problem's volume and work well as a traditional method when a fast scheduling decision is required. However, a question arises: what is the best dispatch rule in each situation? This is what the DRL agent learns to determine. In each production situation, the DRL agent should be able to propose the best dispatch rule for each resource. Thus, the DRL agent learns the best dispatch policy.

Let us denote the set of dispatch rules as $r = \{r_1, r_2, \ldots, r_q\}$, with $q = 1, 2, \ldots maxR$, where $maxR$ is the maximum number of dispatch rules that will be considered as available options in selecting the dispatch policy. An action a is the following:

$$a = \{r_q, r_q, \ldots, r_q\}, \text{ with } len(a) = maxM \tag{7}$$

The action space A is defined as follows:

$$A = \{a_1, a_2, \ldots, a_y\} \text{ with } len(A) = maxM^{maxR} \text{ and y} \leq \text{len(A)} \tag{8}$$

3.2.5. Reward

At each decision point where the agent takes an action to transition from state s to state s', the reward function evaluates the quality of the action based on environmental observation. For this work, the production makespan is chosen as the reward function variable. The reward is the following:

$$R(s_t, a_t) = \frac{const}{makespan_t} \tag{9}$$

where $makespan_t$ is the production time achieved in a simulation production run and $const$ is a constant value to normalize the reward. Reward normalization should be applied, since the makespan of a production run depends heavily on the workload. The reward is higher when the makespan is lower, resulting in objective maximization and makespan minimization. A training episode is considered a simulation production run. The goal is to maximize the reward taken by the agent in a training episode.

3.3. Framework for DRL Scheduling Agent Training

Now that the scheduling problem and the reinforcement learning approach have been described, the last concept to be described is the DRL scheduling agent framework and workflow. To connect the DRL agent, the scheduling input and the environmental supporting functionality have to be defined. A custom-made component, namely an execution controller, is designed with the following functionalities.

- Pre-processing: The DQN input vector has to be created from the output of the DES model. The output of the DES model consists of the resources and job status. The execution controller executes the formulation of the DQN input vector. Moreover, it provides the DRL agent with the episode reward using the makespan obtained from the simulation.

- Post-processing: The DES model input has to be created from the DQN output. The output is a vector with the dispatch rules encoded. The execution controller passes this information to the DES model.
- Simulation control: The execution controller is responsible for performing and coordinating the DQN and DES model's operation, ensuring the stability of the framework.
- Scheduling input: The scheduling workload is either outsourced or created from the execution controller's inner functionality. The execution controller is capable of performing both processes (in testing mode, it is outsourced, while, in training mode, the workload is created with the inner functionality).

The DRL scheduling framework's components and functionalities are illustrated in Figure 2.

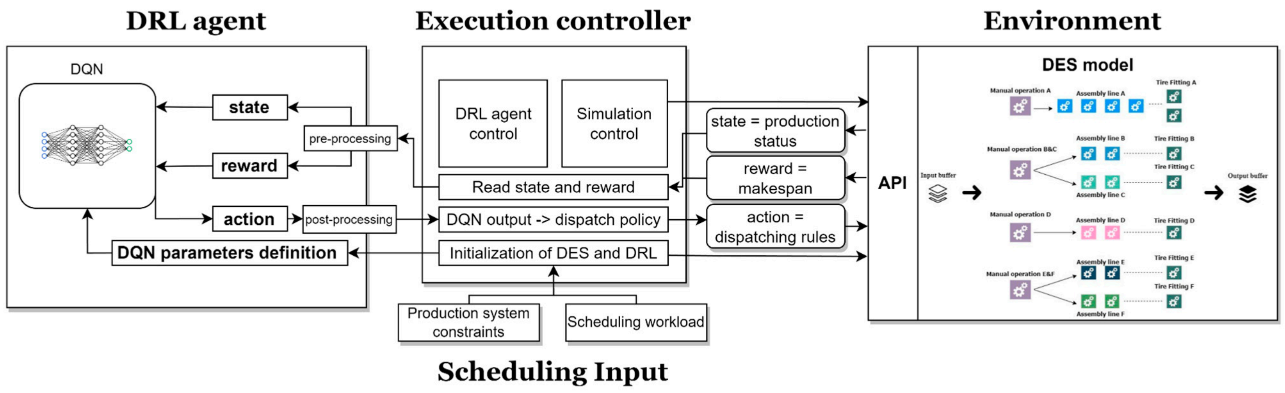

Figure 2. Scheduling framework architecture.

4. Industrial Pilot Case

4.1. Pilot Case Description

A real production system from the bicycle industry is employed in order to validate and evaluate the performance of the DRL framework. The production environment is a wheel assembly shop floor, with a set of operations, product types and resources. The shop floor consists of four assembly lines, responsible for wheel assembly. Before the assembly operation, a manual process must be performed. Depending on the product type, some types of wheels go through a final tire-fitting process, and then they are sent to the warehouse. The department is capable of producing 12 types of wheels. Each type of wheel has a specific processing time in the assembly line. The total processing time for a complete wheel assembly, as well as the suitability, is presented in Table 1. The shop floor of the production system is presented in Figure 3.

For the representation of the production system, a simulation environment is developed. The production system's variables and the wheel attributes must be defined. Additionally, the input rules of the assembly lines have to be defined. These input rules are the dispatch rules that are used to formulate the action space. Three particular rules are chosen for this study in order to offer a thorough evaluation. These three rules are the Most Tasks Remaining per Job (MTRJ), Longest Processing Time (LPT) and Shortest Processing Time (SPT). These guidelines each provide unique benefits and insights into the effectiveness and efficiency of resource management systems.

- SPT: The resource selects the job that consists of the tasks that have the shortest processing times among all suitable tasks. The SPT rule is selected because it focuses on minimizing the processing time for individual tasks. This method helps to maintain a constant flow of work and reduces waiting times for subsequent tasks, making it especially useful in situations where quick task turnaround is essential.

- LPT: The resource selects the job that consists of the tasks that have the longest processing times among all suitable tasks. The LPT rule, as opposed to the SPT, is used to investigate the effects of prioritizing jobs with the longest processing times. This method is particularly effective in settings where finishing long-duration tasks is crucial for sustaining overall system efficiency and avoiding extended delays.
- MTRJ: The resource selects the job that consists of the most remaining suitable tasks. The MTRJ rule is chosen to assess how well jobs are prioritized according to the quantity of tasks that remain. The MTRJ rule is especially helpful in situations where completing multiple tasks is key to the system's overall progress.

Table 1. Product types, suitability and processing times.

Product Type ID	Product Type	Assembly Line 1 (min)	Assembly Line 2 (min)	Assembly Line 3 (min)	Assembly Line 4 (min)
1	A	10	15	0	10
2	B	5	0	0	5
3	C	8	0	6	8
4	D	12	18	8	0
5	E	0	6	4	2
6	F	0	5	0	0
7	G	0	7	0	10.5
8	H	6	9	0	0
9	I	0	0	13	0
10	J	0	4	0	0
11	K	0	5	2	3
12	L	14	7	10	0

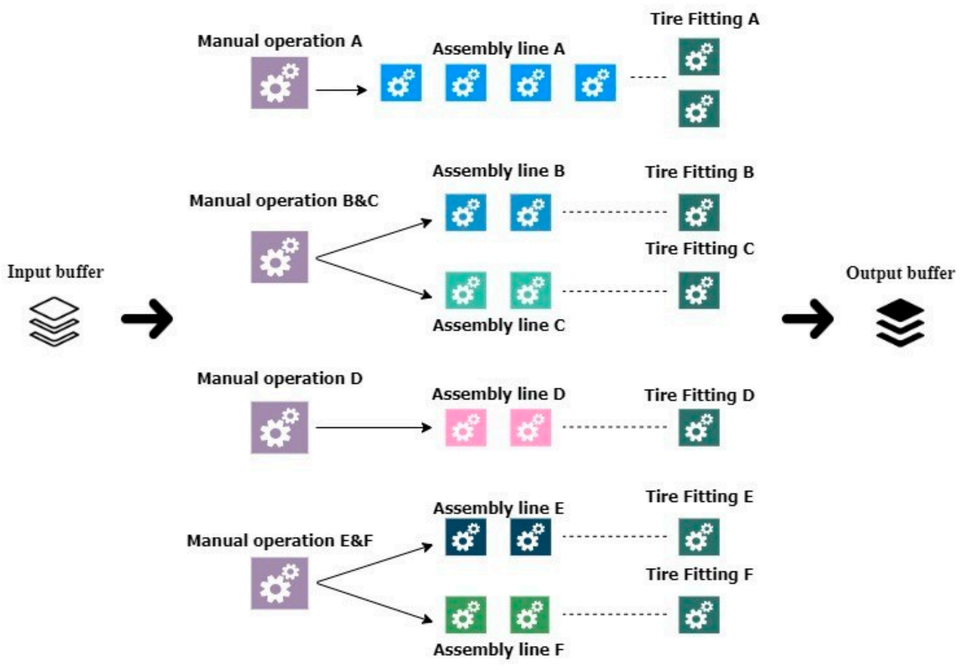

Figure 3. Wheel assembly department shop floor design.

Each production line consists of parallel machines that can perform the assembly process, machines for manual operation and, lastly, machines for the tire-fitting process, if this is needed based on the process plan.

4.2. Scheduling Problem Description

The use case specifications and description should be mapped into production variables, formulating the scheduling problem's parameters. The scheduling problem is defined below.

- Product types: The set of product types is $P = \{P_1, P_2, \ldots, P_i\}$, with $i = 1, 2, \ldots, maxP$, where $maxP = 12$.
- Process plan: There are three operation categories in the system, which create two different process plans based on the product type. The three operations are $Pr_1 = manual\ process$, $Pr_2 = assembly$ and $Pr_3 = tire\ fitting$. Thus, there are two types of process plans: $H_1 = \{Pr1, Pr2, Pr3\}$ and $H_2 = \{Pr1, Pr2\}$.
- Resources: The assembly lines can be modeled as resources, since each production job can be allocated to one and only one assembly line during production. Thus, $M = \{M_1, M_2, \ldots, M_r\}$ with $r = 1, 2, \ldots, maxM$, where $maxM = 4$.
- Tasks: As a task is considered, a wheel process plan must be executed. This means that a product is represented as task T_b, with $b = 1, 2, \ldots maxT$, where $maxT = 1500$. The workload consists of a maximum of 1500 wheels to be produced.
- Jobs: As a job is considered, a certain number of units must be produced for a certain wheel type. A job J_c, with $c = 1, 2, \ldots, maxJ$, where $maxJ = 60$, consists of 25 tasks or less.
- Order: An order is considered as a certain amount of jobs to be executed. An order O_g with $g = 1, 2, \ldots, maxO$, where $maxO = 30$, consists of two jobs or less.
- Suitability and processing times: A processing time for each task T_b is defined for each resource (assembly line) M_r. If the value is 0, it means that the task is not suitable for the resource: $I(M_r, Pr_{ki}) \geq 0$. This is presented in Table 1.
- Scheduling workload: For this scheduling problem, the maximum number of orders is $maxO = 30$, the maximum number of jobs is $maxJ = 60$ (each order consists of two jobs) and the maximum number of tasks is $maxT = 1500$ (each job has a maximum number of 25 tasks). The scheduling workload consists of all orders O_g, jobs J_c and tasks T_b that need to be executed. The scheduling solution refers to the job's allocation to resources (J_c allocation to M_r).

5. Method Implementation and Evaluation

5.1. DRL Agent Implementation

The neural network used for this work is sequential. A sequential model is a linear stack of layers, where one layer is added at a time from input to output. There are four layers.

- INPUT LAYER (1): This is the first layer of the model. It is a flatten layer, which is used to normalize the input. The shape of the input vector is (1, Ls). This means that the input is expected to be a one-dimensional array with state elements.
- MIDDLE LAYERS (2): There are two fully connected dense layers with 1024 units and a ReLU activation function in each. The ReLU activation function helps to introduce non-linearity into the model.
- OUTPUT LAYER (1): This is the final dense layer. This layer has action units, which correspond to the number of possible actions that the agent can take. The 'linear' activation function is used, which means that the output of this layer will be a linear

combination of the inputs. This is used to represent action values (Q-values) directly. The shape of the output vector is (1, len(A)).

In Table 2, the DRL agent parameters used in implementing the aforementioned DQN network are presented. The stability and convergence of the network are achieved after multiple training parameter combinations, with the best ones resulting in the following table.

Table 2. DRL agent model parameters.

POLICY	Epsilon-greedy (EpsGreedyQPolicy), eps = 0.1
MEMORY	Sequential
DQN	Model = Q-network, gamma = 0.99
LEARNING RATE	Adam optimizer, a = 0.001
TRAINING	5000 episodes

Regarding the DRL agent's DQN, it has already been mentioned that the input and output vector depend on the implementation approach. For this work, there are four assembly lines and a maximum of 60 jobs; thus,

$$Ssub = \{J_c, P_i, Jrw_c, Jtmin_c, Jtmax_c, Jps_{c1}, Jps_{c2}, Jps_{c3}, Jps_{c4}\} \tag{10}$$

For the output vector, which is the action space, the three dispatch rules over the four resources create an action space of 81 actions. Let us encode the dispatch rules as follows:

- $Shortest\ Processing\ Time\ (SPT) = 1$;
- $Longest\ Processing\ Time\ (LPT) = 2$;
- $Most\ Tasks\ Remaining\ per\ Job\ (MTRJ) = 3$.

The action space consists of all combinations of the three dispatch rules over the four resources, which are the assembly lines.

5.2. DRL Framework Implementation

The environment for implementation is a DES model developed with the use of the LANNER WITNESS HORIZON software v23.1. Moreover, additional external files, such as excel and text files, are used. The product type IDs, process plan, suitability and processing times are loaded to the DES model with the use of external files. These files can be either Excel files or text files. For this work, Excel files are used. Moreover, the scheduling problem input and the workload are also loaded through external files, i.e., Excel and text ones. In addition, to control the training execution, a custom-made Python v3 API, as already mentioned, is deployed. This API gives the execution controller the ability to control the workflow. Simulation commands like run, stop and reset are integrated into this API and provide the ability for the external deployment of a DES model.

5.3. DRL Model Training

For training purposes, random workloads with the above-mentioned requirements have been developed. The execution controller loads the workload of each episode, which is a simulated production run. The DRL agent, having the initial state fed from the DES model through the execution controller, proposes an action. The set of dispatch rules is then passed through the execution controller, and the simulation run starts. When the production has finished, the final status of the production system, as well as the reward, is given as feedback to the DRL agent, through the execution controller. The controller then resets the DES model environment and creates a new workload, and a new production run is simulated. After 5000 episodes and one day of training, the agent is adequately trained and able to perform well in the given production environment. This DRL approach addresses the aspect of machine breakdowns as unexpected events; thus, retraining is only mandatory if there is another production setup or additional products. The neural network parameters are chosen after trials, with the best results being achieved with the current setup of parameters (Table 2). In Figures 4–7, the training results are presented. In these figures, the moving averages of the makespans and rewards during the training session are presented. The training scenarios are randomly generated. From one episode to another, the workload could differ significantly. Thus, to overcome this issue, the moving averages of the makespans and the rewards are selected as the measurement method. In Figures 4 and 5, the moving average of the makespan achieved from the DRL agent's actions is presented. The first figure shows the progress of the makespan's moving average for 30 measurements and the second one for 100 measurements. Each value is the average value of 30 or 100 production scenarios. As the figures show, the average makespan tends to be shorter in the last few steps of the training, starting at 700–750 min and finishing at 550–600. This is a 20–21% makespan reduction observed in the training procedure. The PC specifications for the training of the DRL model are the following: CPU Intel i7 @2.2 GHz, RAM 16 GB (Santa Clara, CA, USA).

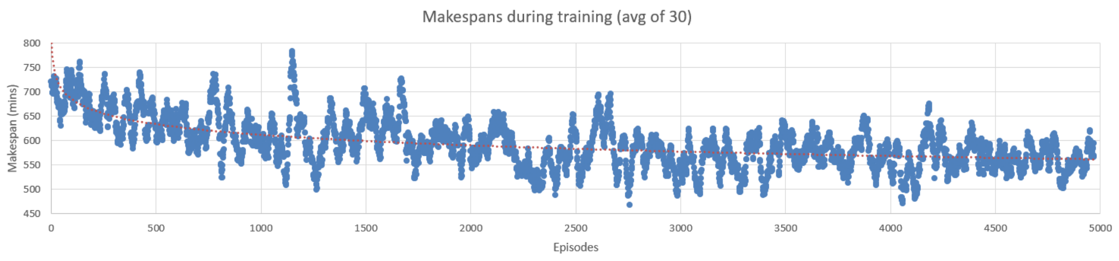

Figure 4. Makespan during training (moving average of 30 values).

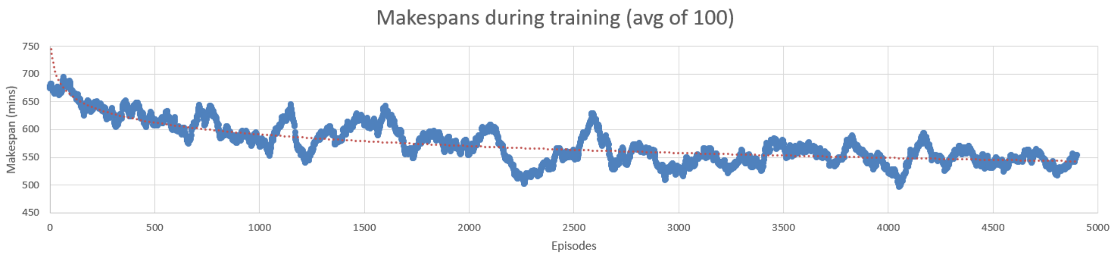

Figure 5. Makespan during training (moving average of 100 values).

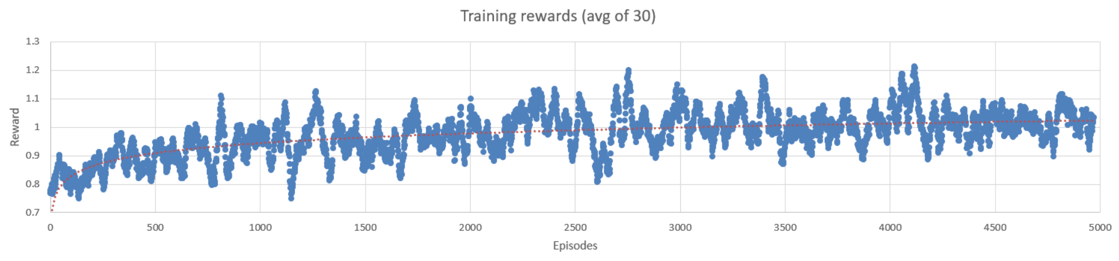

Figure 6. Rewards during training (moving average of 30 values).

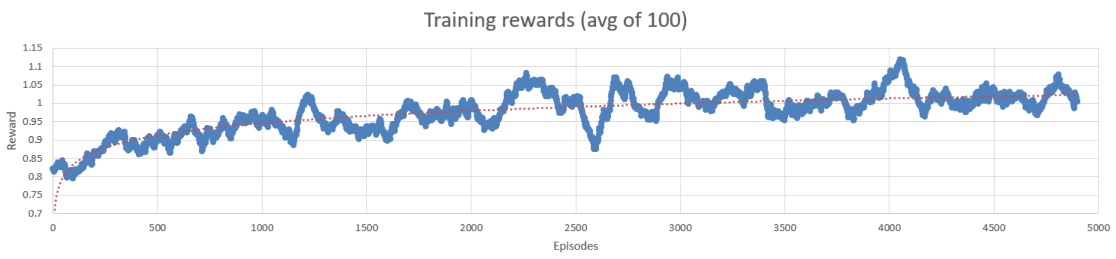

Figure 7. Rewards during training (moving average of 100 values).

In the next section, the results achieved and a comparative analysis are presented.

5.4. Evaluation and Discussion

The DRL agent was trained by exploring the use case's production environment. The trained model was then able to perform under different circumstances and obtain scheduling solutions. To evaluate the proposed method, the DRL scheduling algorithm was compared with random decision-making and dispatch rules.

For testing, production scenarios were randomly generated with the following characteristics.

- Workload: Random workloads were generated with the limitations described in Section 4.2.
- Resource availability: In each episode, one line may be unavailable. This is a random event happening in the simulation.

The implementation of the testing phase was realized with the use of Python as well. For random workloads, each algorithm should propose an action and be evaluated. An iteration included five steps (one for each algorithm). In each iteration, a production scenario was generated. The initial status of the production system, as well as the workload for each run, was randomly generated based on the production system requirements as addressed in Section 4.2. In the beginning, one or no machines was unavailable, while all tasks were pending and ready to be dispatched to resources. After this, each algorithm proposed its action, and the simulation run was initialized. The result of each run, which was the makespan returned by the DES model, was stored. Thus, for each scenario, one could identify the performance of each algorithm.

In Figures 6 and 7, the moving average of the rewards from the DRL agent's actions is presented. The first figure shows the progress of the rewards' moving average for 30 measurements and the second one for 100 measurements. Each value is the average value of 30 or 100 production scenarios. As the figures show, the average reward tends to be larger in the last few steps of the training, starting at 0.8–0.85 and finishing at 1–1.05. This is a 19–20% reward improvement observed in the training procedure.

From the training procedure, there are two main conclusions. The first one is that the DRL agent was able to train well and learn better actions over time. Starting from large makespans, on average, it was able to reduce the average scheduling makespan by 20%. In the same manner, a reward improvement is observed, which is inversely related to the makespan. The second observation is that the DRL agent does not follow the typical reward progress seen in other reinforcement learning problems. In other words, the reward does not start from zero and reach higher values at the end of the training. Initially, the training rewards are high, and this is because there is not a terminal state other than the production's completion. This means that the agent's reward cannot be negative or zero. It can only be proportional to the makespan achieved, as defined in the methodology. In this implementation, the agent does not learn how to navigate between states following actions in order to reach a terminal state and achieve its goal. From the beginning, the agent can perform any action and reach a terminal state. However, the agent learns how to navigate through these states in order to maximize the reward. Thus, despite the fact that the rewards during the first phase of the training are already large, the agent is able to find even better solutions to the problem. Lastly, the agent explores a vast environment, affecting its performance by necessitating the learning of problem-solving in a production system and the ability to generalize to achieve optimal outcomes across a variety of circumstances.

One hundred testing scenarios were randomly generated in order to identify average behavior, since the randomness introduced by the production scenarios, as well as the scale of the problem, had to be adequately addressed in the results. In Table 3, the makespans achieved for each algorithm for the testing scenarios are presented. From the last column of the table, one can see that the DRL agent finds a better solution compared to all other algorithms by 85%. In addition, the makespan achieved with DRL is, on average, shorter than the best ones achieved by the other scheduling algorithms. In Figure 8, the graph shows better the average makespan achieved by each algorithm.

Table 3. Makespans achieved with the different scheduling algorithms in the testing phase. With bold the best performance in each production run is highlighted.

PROD RUN	DRL	RANDOM	SPT	LPT	MTRJ
	Makespan (min)				
1	**569**	878	835	748	631
2	**561**	976	944	724	696
3	**484**	731	731	509	603
4	**393**	614	582	492	496
5	**691**	1117	1165	901	752
6	**438**	606	740	525	543
7	**900**	1201	1249	900	999
8	**315**	324	324	440	295
9	**470**	802	809	632	535
10	**548**	780	690	701	565
11	355	524	**334**	505	372
12	373	380	412	483	**362**
13	**467**	608	796	598	529
14	729	886	958	**720**	729
15	**1108**	1108	1551	1108	1205
16	**367**	479	627	460	455
17	**458**	464	647	622	502
18	**652**	652	971	652	652
19	**601**	699	846	601	649
20	**495**	729	751	656	513
21	**443**	715	720	588	627
22	**413**	608	509	570	416
23	**564**	886	852	711	607
24	**632**	725	866	816	586
25	**1016**	1448	1458	1016	1053
26	**353**	369	584	443	409
27	**547**	891	918	800	668
28	**445**	532	717	489	476
29	**458**	664	686	458	458
30	**540**	748	717	708	587
31	**640.5**	970	1046	744	826
32	**432**	432	679	447	499
33	**529**	608	739	572	577

Table 3. *Cont.*

PROD RUN	DRL	RANDOM	SPT	LPT	MTRJ
	Makespan (min)				
34	**496**	496	821	607	682
35	**365**	524	514	458	366
36	**335**	381	394	473	363
37	**1069**	1069	1389	1069	1069
38	**575**	692	780	575	575
39	**574**	738	932	591	624
40	**425**	603	614	556	432
41	**795**	1239	1251	795	909
42	**691**	828	851	691	740
43	**484**	875	806	593	523
44	**652**	911	929	652	652
45	**483.5**	512	734	626	625
46	562.5	692	681	658	**516**
47	**506**	521	764	637	575
48	**1186**	1466	1570	1186	1337
49	**360**	412	463	455	416
50	**509**	509	697	509	509
51	**691**	1003	998	691	799
52	**1029**	1029	1511	1029	1029
53	**505**	623	636	677	523
54	**575**	575	834	575	575
55	**717**	1081	1097	756	746
56	**574**	805	886	574	596
57	**455**	510	520	560	469
58	**601**	799	1002	750	678
59	**561**	561	810	561	579
60	**653**	871	893	653	653
61	**717**	793	1115	770	793
62	**416.5**	421	550	554	435
63	**717**	1134	1095	772	918
64	316	357	**285**	387	315
65	**496**	496	716	496	596
66	**432**	774	721	578	461
67	**470**	710	645	660	515
68	**783**	783	1033	783	783
69	**587**	587	952	727	658
70	**492**	805	823	621	610
71	**406**	437	646	490	437
72	321.5	326.5	393	370	**307.5**
73	**990**	990	1287	990	990
74	**562**	857	872	600	591
75	**418**	418	634	418	427
76	**388**	411	412	522	398
77	599	**552**	864	732	608
78	**365**	531	577	433	366
79	**410**	537	464	540	410
80	**407**	612	584	523	407
81	649	667	829	**639**	639
82	**413**	418	562	459	429
83	**474**	521	614	691	502
84	**310**	396	382	396	335
85	466	592	636	696	**460**

Table 3. *Cont.*

PROD RUN	DRL	RANDOM	SPT	LPT	MTRJ
	Makespan (min)				
86	**666**	992	1142	833	903
87	**484**	496	557	610	507
88	**562**	754	750	562	573
89	**319**	441	548	445	425
90	**574**	602	578	684	596
91	**991**	1042	1410	991	1014
92	587	628	778	**546**	587
93	**616**	799	900	799	672
94	574	591	903	686	**553**
95	**367**	367	554	416	397
96	**493**	797	795	660	584
97	692	**678**	1147	678	759
98	**808**	897	1164	808	851
99	**618**	657	812	832	626
100	**372**	515	460	496	380
AVG	554.63	694.61	795.31	638.73	599.14

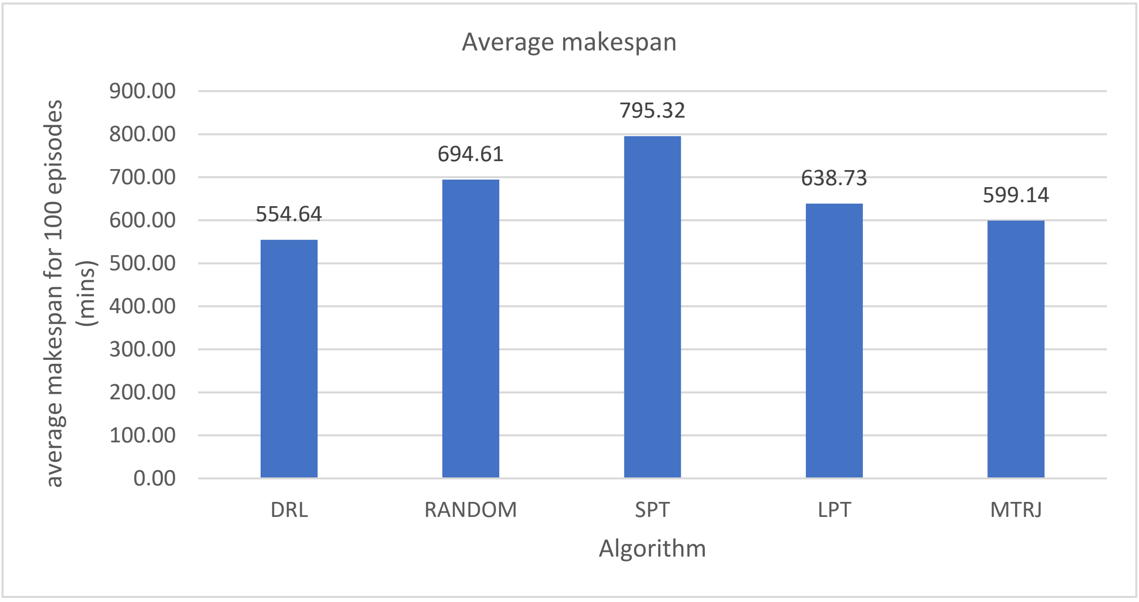

Figure 8. Scheduling algorithms' average makespan achieved.

In Figure 9, the results of all testing algorithms are shown in a graph. Firstly, one can see that the DRL scheduling algorithm performs, on average, better than all other algorithms. At this point, it should be mentioned that the performance of an algorithm also depends on the use case. Thus, the DRL agent should be evaluated for different scheduling methods. In Figure 10, it is obvious that the DRL agent outperforms the random job selection. This is expected, since the DRL agent has been trained to choose the best actions in each situation. This also validates the training of the DRL agent and its ability to adapt to a particular production scenario and perform well in different situations. Moreover, in Figures 11–13, a comparison of the DRL with each dispatch rule is presented. In the next section, a comparison of the different algorithms' results is reported.

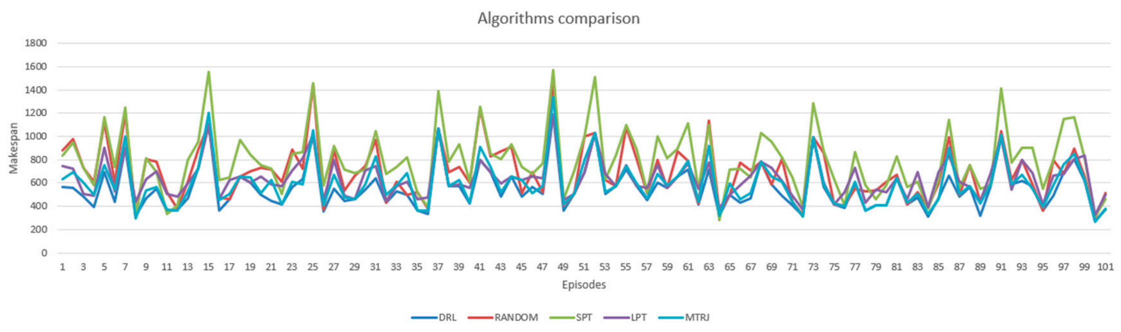

Figure 9. Comparison of all algorithms' performance regarding makespan achieved through 100 testing scenarios.

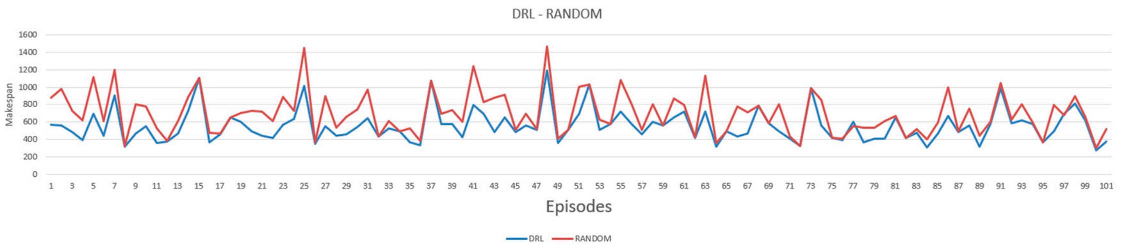

Figure 10. Comparison of DRL and RANDOM algorithms' performance regarding makespan achieved through 100 testing scenarios.

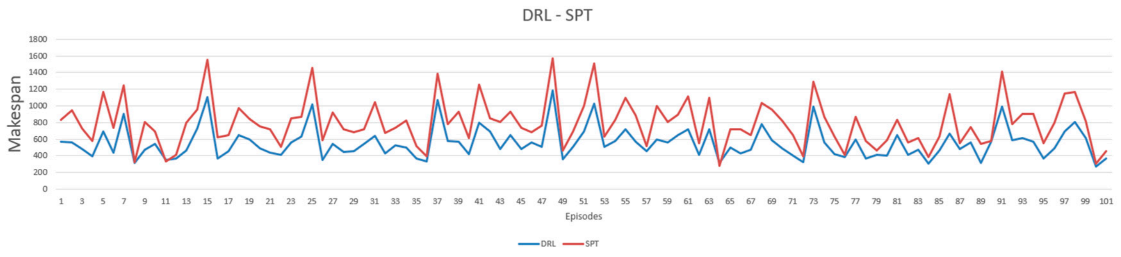

Figure 11. Comparison of DRL and SPT algorithms' performance regarding makespan achieved through 100 testing scenarios.

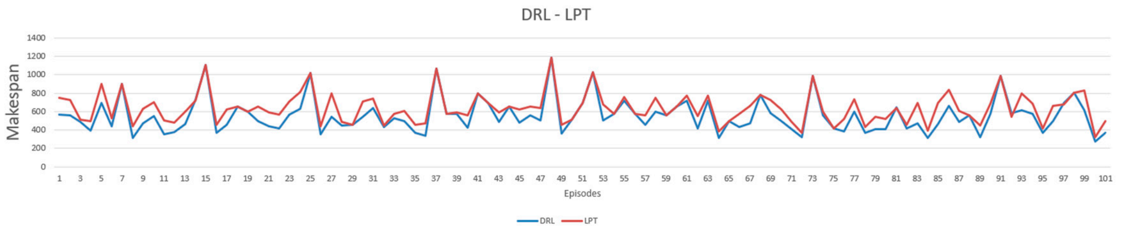

Figure 12. Comparison of DRL and LPT algorithms' performance regarding makespan achieved through 100 testing scenarios.

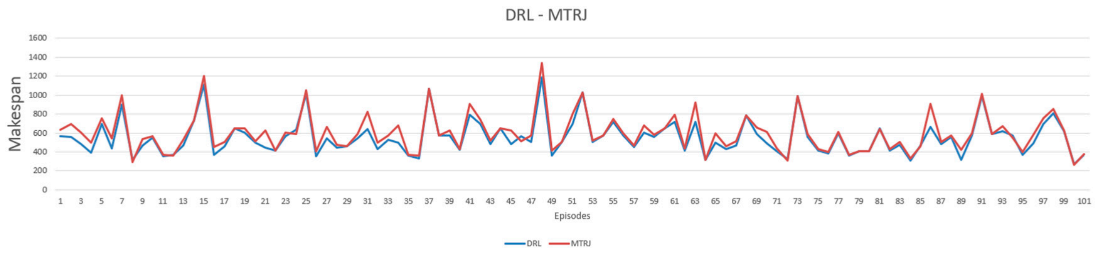

Figure 13. Comparison of DRL and MTRJ algorithms' performance regarding makespan achieved through 100 testing scenarios.

6. Discussion

The results show that the DRL approach for scheduling is a good solution with promising results and outperforms traditional methods. More specifically, the DRL agent is able to propose a schedule that results in a shorter makespan than the other scheduling approaches by 85%. This means that, in 85/100 testing scenarios, the DRL proposes a more efficient schedule, resulting in a shorter production makespan. Comparing the DRL with each other algorithm separately shows that the DRL performs better in 98% of the cases compared to RANDOM, 98% compared to SPT, 96% compared to LPT and 90% compared to MTRPJ. Lastly, in terms of the average makespan achieved by the algorithms, the DRL results in a makespan that is shorter than the ones achieved by RANDOM, SPT, LPT and MTRJ by 20%, 30%, 13% and 7.5%, respectively.

To further validate the findings, 95% confidence intervals were computed for the mean makespan achieved by the DRL approach and the other methods. The intervals demonstrate a statistically significant improvement in the makespan performance for DRL across all test scenarios. Additionally, a paired t-test was conducted, with the p-values confirming that the differences in the makespan between DRL and the baseline methods were statistically significant ($p < 0.05$). Moreover, variability in the DRL performance across different scenarios was observed, which was due to the dynamic nature of the production workloads and the DRL agent's policy adjustments based on state exploration. Despite this, DRL consistently outperformed the other methods, reflecting its ability to adapt effectively to uncertain and diverse conditions.

Traditional optimization methods, such as genetic algorithms (GAs), are well established for their ability to handle complex scheduling problems through global search heuristics [6]. However, these methods often require significant computational time and manual parameter tuning, particularly for dynamic and uncertain production environments. In contrast, DRL offers several distinct advantages. DRL adapts dynamically to changing workloads and system states without requiring complete rescheduling from scratch, leading to significantly reduced computational times. While these methods often require significant computational resources during training due to their iterative and data-intensive nature, their ability to generalize across varying scenarios significantly outweighs this initial cost. Furthermore, DRL integrates decision-making flexibility by learning from the environment, rather than relying on predefined heuristic rules [46,54]. It effectively navigates vast state and action spaces, making it particularly suited for dynamic and uncertain environments. While traditional methods excel in deterministic scenarios, they may struggle with real-time adaptability in stochastic environments, a challenge addressed effectively by DRL. Additionally, DRL's capacity to optimize the scheduling decisions while generalizing across different problem scales further positions it as a competitive alternative for modern manufacturing systems [51]. Despite its computational intensity during training, DRL achieves superior long-term adaptability and scalability, as highlighted by the results of this study, with an observed makespan reduction of up to 20%. The proposed implementation demonstrates notable advantages, such as scalability to larger production setups and adaptability to dynamic workloads. This aligns with other findings in the literature, such as those of Mnih [60], who highlighted the efficiency of DQNs in approximating optimal policies in high-dimensional state spaces. Furthermore, by leveraging state space reduction and task prioritization, the computational burden is minimized without sacrificing the solution quality, underscoring its practical applicability in smart manufacturing environments [51].

7. Conclusions

In conclusion, this study has successfully introduced and implemented a novel approach to production scheduling, employing a DRL agent guided by a DNN. The DRL scheduling agent exhibited remarkable performance, achieving an 85% improvement in selecting optimal dispatch rules (SPT, LPT and MTRJ) for each resource in a scheduling problem instance. The comparison against traditional rules, such as SPT, LPT, MTRJ and random job selection, highlighted the superiority of the DRL agent, particularly when applied to real-world data from the bicycle industry. The development of a DES model, replicating the dynamics of the bicycle industry, served as a crucial training basis for the DRL agent. The results not only validate the effectiveness of the DRL agent but also emphasize its potential as a valuable tool to enhance production scheduling, especially in conjunction with established dispatch rules.

While DRL demonstrates significant potential in dynamic scheduling, it has limitations that warrant consideration. Training DRL models, such as the implemented deep Q-network, is computationally intensive, requiring substantial time and resources, especially for large-scale or complex production environments [60]. Additionally, the proposed implementation does not fully capture real-world uncertainties, such as supply chain disruptions and the addition of new products or product variants. Lastly, DRL's sensitivity to hyperparameter tuning and the potential for suboptimal policies in highly stochastic settings pose challenges, requiring careful design and the evaluation of the reward structure and state action representation.

As future work, we envision expanding the scope of this study by incorporating additional dispatch rules like FIFO and the Earliest Due Date (EDD) into the DRL scheduling agent framework. Furthermore, future endeavors may explore the integration of standardized models, such as the asset administration shell concept (AAS), to parameterize the DRL agent. These anticipated extensions aim to further refine and generalize the applicability of the DRL-based approach, contributing to the ongoing evolution of intelligent production scheduling methodologies. Another topic for improvement is the optimization goal, seeking to include more parameters, such as energy efficiency, the production rate, etc., resulting in a multi-objective DRL agent in a manufacturing environment. The comparison of such multi-objective agents' implementations, following a central architecture and methodology, is a topic of great interest, especially when considering other production environments with different characteristics and conditions, where more limitations or applicability issues may occur. Scalability advancements and framework validation in more complex environments will be addressed in future research as well. This will be also supported by a variability analysis of the deployed scenarios to identify the implications for the DRL performance related to the workload or production setup scale. In addition, the effects of the environmental parameters, requirements and limitations may be another topic of great intertest, especially in larger production systems. Dynamic environments with more unexpected events could also be addressed with the implementation of the DRL framework, such as changes in the capacity and the processing power of resources or urgent production orders and delivery. Lastly, when the scale of the problem is much greater, the state representation and state space reduction could be a topic for further research, regarding the impact of the state space formulation on the quality of the decisions.

Author Contributions: Conceptualization, K.A., N.N. and D.M.; Methodology, K.A., P.M. and E.B.; Software, P.M. and E.B.; Validation, P.M. and E.B.; Formal analysis, K.A.; Data curation, P.M.; Writing—original draft, P.M. and E.B.; Writing—review & editing, K.A., P.M., N.N. and D.M.; Supervision, K.A., N.N. and D.M.; Project administration, K.A. and N.N.; Funding acquisition, K.A. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 957204 MAS4AI. The dissemination of the results herein reflects only the authors' views, and the Commission is not responsible for any use that may be made of the information that it contains.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to privacy.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Chryssolouris, G.; Alexopoulos, K.; Arkouli, Z. Artificial Intelligence in Manufacturing Systems. *Stud. Syst. Decis. Control* **2023**, *436*, 79–135. [CrossRef]
2. Mourtzis, D. Advances in Adaptive Scheduling in Industry 4.0. *Front. Manuf. Technol.* **2022**, *2*, 937889. [CrossRef]
3. Cioffi, R.; Travaglioni, M.; Piscitelli, G.; Petrillo, A.; Parmentola, A. Smart Manufacturing Systems and Applied Industrial Technologies for a Sustainable Industry: A Systematic Literature Review. *Appl. Sci.* **2020**, *10*, 2897. [CrossRef]
4. Bai, C.; Dallasega, P.; Orzes, G.; Sarkis, J. Industry 4.0 technologies assessment: A sustainability perspective. *Int. J. Prod. Econ.* **2020**, *229*, 107776. [CrossRef]
5. Chryssolouris, G. *Manufacturing Systems: Theory and Practice*; Springer: New York, NY, USA, 2013.
6. Chryssolouris, G.; Subramaniam, V. Dynamic scheduling of manufacturing job shops using genetic algorithms. *J. Intell. Manuf.* **2001**, *12*, 281–293. [CrossRef]
7. Mourtzis, D.; Vlachou, E. A cloud-based cyber-physical system for adaptive shop-floor scheduling and condition-based maintenance. *J. Manuf. Syst.* **2018**, *47*, 179–198. [CrossRef]
8. Baldea, M.; Harjunkoski, I. Integrated production scheduling and process control: A systematic review. *Comput. Chem. Eng.* **2014**, *71*, 377–390. [CrossRef]
9. Ghaleb, M.; Zolfagharinia, H.; Taghipour, S. Real-time production scheduling in the Industry-4.0 context: Addressing uncertainties in job arrivals and machine breakdowns. *Comput. Oper. Res.* **2020**, *123*, 105031. [CrossRef]
10. Alexopoulos, K.; Nikolakis, N.; Chryssolouris, G. Digital twin-driven supervised machine learning for the development of artificial intelligence applications in manufacturing. *Int. J. Comput. Integr. Manuf.* **2020**, *33*, 429–439. [CrossRef]
11. Framinan, J.M.; Leisten, R.; García, R.R. *Manufacturing Scheduling Systems: An Integrated View on Models, Methods and Tools*; Springer: London, UK, 2014; pp. 1–400. ISBN 978-1-4471-6272-8. [CrossRef]
12. Pierreval, H.; Mebarki, N. Dynamic scheduling selection of dispatching rules for manufacturing system. *Int. J. Prod. Res.* **1997**, *35*, 1575–1591. [CrossRef]
13. Choi, B.K.; You, N.K. Dispatching rules for dynamic scheduling of one-of-a-kind production. *Int. J. Comput. Integr. Manuf.* **2006**, *19*, 383–392. [CrossRef]
14. Zhang, J.; Ding, G.; Zou, Y.; Qin, S.; Fu, J. Review of job shop scheduling research and its new perspectives under Industry 4.0. *J. Intell. Manuf.* **2017**, *30*, 1809–1830. [CrossRef]
15. Alexopoulos, K.; Koukas, S.; Boli, N.; Mourtzis, D. Resource planning for the installation of industrial product service systems. *IFIP Adv. Inf. Commun. Technol.* **2017**, *514*, 205–213. [CrossRef]
16. Chryssolouris, G.; Papakostas, N.; Mourtzis, D. Refinery short-term scheduling with tank farm, inventory and distillation management: An integrated simulation-based approach. *Eur. J. Oper. Res.* **2005**, *166*, 812–827. [CrossRef]
17. Cai, L.; Li, W.; Luo, Y.; He, L. Real-time scheduling simulation optimisation of job shop in a production-logistics collaborative environment. *Int. J. Prod. Res.* **2023**, *61*, 1373–1393. [CrossRef]
18. Wei, H.; Li, S.; Quan, H.; Liu, D.; Rao, S.; Li, C.; Hu, J. Unified Multi-Objective Genetic Algorithm for Energy Efficient Job Shop Scheduling. *IEEE Access* **2021**, *9*, 54542–54557. [CrossRef]
19. Li, W.; Luo, X.; Xue, D.; Tu, Y. A heuristic for adaptive production scheduling and control in flow shop production. *Int. J. Prod. Res.* **2011**, *49*, 3151–3170. [CrossRef]
20. Yan, Y.; Wang, Z. A two-layer dynamic scheduling method for minimising the earliness and tardiness of a re-entrant production line. *Int. J. Prod. Res.* **2012**, *50*, 499–515. [CrossRef]
21. Wen, X.; Lian, X.; Qian, Y.; Zhang, Y.; Wang, H.; Li, H. Dynamic scheduling method for integrated process planning and scheduling problem with machine fault. *Robot. Comput. Integr. Manuf.* **2022**, *77*, 102334. [CrossRef]
22. Shi, D.; Fan, W.; Xiao, Y.; Lin, T.; Xing, C. Intelligent scheduling of discrete automated production line via deep reinforcement learning. *Int. J. Prod. Res.* **2020**, *58*, 3362–3380. [CrossRef]

23. Liu, J.; Sun, B.; Li, G.; Chen, Y. An integrated scheduling approach considering dispatching strategy and conflict-free route of AMRs in flexible job shop. *Int. J. Adv. Manuf. Technol.* **2023**, *127*, 1979–2002. [CrossRef]
24. Wang, G.; Zhang, G.; Guo, X.; Zhang, Y. Digital twin-driven service model and optimal allocation of manufacturing resources in shared manufacturing. *J. Manuf. Syst.* **2021**, *59*, 165–179. [CrossRef]
25. Zadeh, M.S.; Katebi, Y.; Doniavi, A. A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times. *Int. J. Prod. Res.* **2019**, *57*, 3020–3035. [CrossRef]
26. Liu, W.; Jin, Y.; Price, M. New meta-heuristic for dynamic scheduling in permutation flowshop with new order arrival. *Int. J. Adv. Manuf. Technol.* **2018**, *98*, 1817–1830. [CrossRef]
27. Mansouri, N.; Zade, B.M.H.; Javidi, M.M. Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. *Comput. Ind. Eng.* **2019**, *130*, 597–633. [CrossRef]
28. Wang, P.; Lei, Y.; Agbedanu, P.R.; Zhang, Z. Makespan-Driven Workflow Scheduling in Clouds Using Immune-Based PSO Algorithm. *IEEE Access* **2020**, *8*, 29281–29290. [CrossRef]
29. Wang, J.-J.; Wang, L. A Knowledge-Based Cooperative Algorithm for Energy-Efficient Scheduling of Distributed Flow-Shop. *IEEE Trans. Syst. Man Cybern. Syst.* **2020**, *50*, 1805–1819. [CrossRef]
30. Park, J.; Chun, J.; Kim, S.H.; Kim, Y.; Park, J. Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. *Int. J. Prod. Res.* **2021**, *59*, 3360–3377. [CrossRef]
31. He, P. Optimization and Simulation of Remanufacturing Production Scheduling under Uncertainties. *Int. J. Simul. Model.* **2018**, *17*, 734–743. [CrossRef]
32. Shahzad, A.; Mebarki, N. Learning Dispatching Rules for Scheduling: A Synergistic View Comprising Decision Trees, Tabu Search and Simulation. *Computers* **2016**, *5*, 3. [CrossRef]
33. Sun, J.; Zhang, G.; Lu, J.; Zhang, W. A hybrid many-objective evolutionary algorithm for flexible job-shop scheduling problem with transportation and setup times. *Comput. Oper. Res.* **2021**, *132*, 105263. [CrossRef]
34. Mohan, J.; Lanka, K.; Rao, A.N. A Review of Dynamic Job Shop Scheduling Techniques. *Procedia Manuf.* **2019**, *30*, 34–39. [CrossRef]
35. Priore, P.; Gómez, A.; Pino, R.; Rosillo, R. Dynamic scheduling of manufacturing systems using machine learning: An updated review. *Artif. Intell. Eng. Des. Anal. Manuf.* **2014**, *28*, 83–97. [CrossRef]
36. Vasilis, S.; Nikos, N.; Kosmas, A.; Dimitris, M. A toolbox of agents for scheduling the paint shop in bicycle industry. *Procedia CIRP* **2022**, *107*, 1156–1161. [CrossRef]
37. Taha, H.A.; Yacout, S.; Shaban, Y. Deep Reinforcement Learning for autonomous pre-failure tool life improvement. *Int. J. Adv. Manuf. Technol.* **2022**, *121*, 6169–6192. [CrossRef]
38. Neves, M.; Neto, P. Deep reinforcement learning applied to an assembly sequence planning problem with user preferences. *Int. J. Adv. Manuf. Technol.* **2022**, *122*, 4235–4245. [CrossRef]
39. Torres, A.d.R.; Andreiana, D.S.; Roldán, O.; Bustos, A.H.; Galicia, L.E.A. A Review of Deep Reinforcement Learning Approaches for Smart Manufacturing in Industry 4.0 and 5.0 Framework. *Appl. Sci.* **2022**, *12*, 12377. [CrossRef]
40. Panzer, M.; Bender, B.; Gronau, N. Deep Reinforcement Learning In Production Planning And Control: A Systematic Literature Review. In Proceedings of the Conference on Production Systems and Logistics, Online, 10–11 August 2021; pp. 535–545. [CrossRef]
41. Cunha, B.; Madureira, A.; Fonseca, B.; Matos, J. Intelligent Scheduling with Reinforcement Learning. *Appl. Sci.* **2021**, *11*, 3710. [CrossRef]
42. Mouelhi-Chibani, W.; Pierreval, H. Training a neural network to select dispatching rules in real time. *Comput. Ind. Eng.* **2010**, *58*, 249–256. [CrossRef]
43. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
44. Hubbs, C.; Wassick, J.M.; Hubbs, C.D.; Kelloway, A.; Sahinidis, N.V.; Grossmann, I.E. An Industrial Application of Deep Reinforcement Learning for Chemical Production Scheduling. In *Machine Learning for Engineering Modeling, Simulation, and Design*; Researchgate.net: Berlin, Germany, 2020.
45. Zhou, T.; Tang, D.; Zhu, H.; Wang, L. Reinforcement Learning with Composite Rewards for Production Scheduling in a Smart Factory. *IEEE Access* **2020**, *9*, 752–766. [CrossRef]
46. Wang, Z.; Liao, W. Smart scheduling of dynamic job shop based on discrete event simulation and deep reinforcement learning. *J. Intell. Manuf.* **2023**, *35*, 2593–2610. [CrossRef]
47. Tang, J.; Salonitis, K. A Deep Reinforcement Learning Based Scheduling Policy for Reconfigurable Manufacturing Systems. *Procedia CIRP* **2021**, *103*, 1–7. [CrossRef]
48. Kardos, C.; Laflamme, C.; Gallina, V.; Sihn, W. Dynamic scheduling in a job-shop production system with reinforcement learning. *Procedia CIRP* **2021**, *97*, 104–109. [CrossRef]
49. Zhou, L.; Zhang, L.; Horn, B.K. Deep reinforcement learning-based dynamic scheduling in smart manufacturing. *J. Manuf. Syst.* **2020**, *93*, 383–388. [CrossRef]

50. Wang, Y.-C.; Usher, J.M. Learning policies for single machine job dispatching. *Robot. Comput. Integr. Manuf.* **2004**, *20*, 553–562. [CrossRef]
51. Hu, L.; Liu, Z.; Hu, W.; Wang, Y.; Tan, J.; Wu, F. Petri-net-based dynamic scheduling of flexible manufacturing system via deep reinforcement learning with graph convolutional network. *J. Manuf. Syst.* **2020**, *55*, 1–14. [CrossRef]
52. Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P.S.; Xu, C. Learning to dispatch for job shop scheduling via deep reinforcement learning. In Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, BC, Canada, 6–12 December 2020.
53. Luo, S. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Appl. Soft Comput.* **2020**, *91*, 106208. [CrossRef]
54. Qin, Z.; Johnson, D.; Lu, Y. Dynamic production scheduling towards self-organizing mass personalization: A multi-agent dueling deep reinforcement learning approach. *J. Manuf. Syst.* **2023**, *68*, 242–257. [CrossRef]
55. Zhang, J.; Guo, B.; Ding, X.; Hu, D.; Tang, J.; Du, K.; Tang, C.; Jiang, Y. An adaptive multi-objective multi-task scheduling method by hierarchical deep reinforcement learning. *Appl. Soft Comput.* **2024**, *154*, 111342. [CrossRef]
56. Zeng, Y.; Liao, Z.; Dai, Y.; Wang, R.; Li, X.; Yuan, B. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism. *arXiv* **2022**, arXiv:2201.00548.
57. Lin, C.-C.; Deng, D.-J.; Chih, Y.-L.; Chiu, H.-T. Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4276–4284. [CrossRef]
58. Waschneck, B.; Reichstaller, A.; Belzner, L.; Altenmüller, T.; Bauernhansl, T.; Knapp, A.; Kyek, A. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP* **2018**, *72*, 1264–1269. [CrossRef]
59. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
60. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
61. Mourtzis, D.; Papakostas, N.; Mavrikios, D.; Makris, S.; Alexopoulos, K. The role of simulation in digital manufacturing: Applications and outlook. *Int. J. Comput. Integr. Manuf.* **2015**, *28*, 3–24. [CrossRef]
62. Mourtzis, D. Simulation in the design and operation of manufacturing systems: State of the art and new trends. *Int. J. Prod. Res.* **2020**, *58*, 1927–1949. [CrossRef]