RAIRO-Oper. Res. 60 (2026) 2065–2094
https://doi.org/10.1051/ro/2026055

RAIRO Operations Research
www.rairo-ro.org

TARDY JOBS MINIMIZATION IN A DYNAMIC JOB-SHOP ENVIRONMENT USING DOUBLE Q-LEARNING ALGORITHM

Manal Abir Belmamoune[1,*], Zakaria Yahouni[2] and Latéfa Ghomri[1]

Abstract. Job-shop scheduling problems are complex optimization challenges frequently encountered in manufacturing and production systems, especially in environments with high market demand and customized production modes. The goal of this study is to minimize delays in a dynamic job-shop environment using reinforcement learning. Specifically, we introduce a Double Q-Learning (Double-QL) approach to dynamically manage operations, accounting for varying processing times, unpredictable job arrivals, and machine constraints. A new state representation has been developed to accommodate workshops of all sizes, along with a novel reward function tailored to the agents' specific contexts. Transfer learning is utilized by leveraging the Q-tables learned from previous environmental scenarios. The simulation was carried out with four machines and varying job limits. To assess the robustness of our approach, we compared our Double-QL algorithm with dispatching rules related to tardiness and evaluated its performance in multiple scenarios.

Mathematics Subject Classification. 90B36, 90C40, 68M20.

Received March 28, 2025. Accepted May 8, 2026.

Introduction

The manufacturing environment is characterized by high complexity, dynamic production conditions, and volatile markets. Moreover, to remain competitive in a globalized world, companies must offer customized products while minimizing costs and delays [1]. To achieve these goals, companies need to improve production management by selecting the right strategies and procedures that can lead to overall improvements in manufacturing systems, and by translating these strategies and forecasts into actionable timelines through task scheduling. Scheduling, a critical decision-making process, is used regularly in various manufacturing and service industries. It involves allocating resources to tasks over defined time periods, with the objective of optimizing one or more goals [2]. Most real-world scheduling problems are classified as NP-hard. Among them is the Job-Shop Scheduling Problem (JSSP), in which a set of jobs and machines or resources are involved, and each job follows its own predetermined route [2].

Keywords. Dynamic job-shop, scheduling, reinforcement learning, double Q-Learning, transfer learning, uncertainties, machine failure, job arrivals, robust.

[1] Manufacturing an Engineering Laboratory of Tlemcen (MELT), University of Tlemcen, Tlemcen, Algeria.

[2] Sciences of Conception, Optimization and Production (G-SCOP) University of Grenoble Alpe, CNRS, Saint-Martin-d'Héres, France.

*Corresponding author: manalabir.belmamoune@univ-tlemcen.dz

In real-world environments, scheduling cannot be considered a problem where all information is known in advance [3]. The occurrence of dynamic events, such as machine failures or random job arrivals, is frequent, meaning that the initial schedule may not always be applicable. Therefore, the development of a robust scheduling system capable of handling these uncertain events is a central focus of the research.

To overcome these uncertainties, various methods, collectively known as robust approaches, have been proposed in the literature [4]. These approaches include exact methods such as branch and bound [5], and linear and dynamic programming. When the size of a given scheduling problem is relatively small, most of these techniques can generate schedules within a reasonable time frame and guaranty optimal solutions [6]. However, since exact methods are often impractical for large-scale problems, researchers turn to metaheuristics and artificial intelligence-based techniques. Examples include metaheuristics such as genetic algorithms [7] and tabu search [8], which can effectively provide acceptable (though not always optimal) solutions [6]. In addition, learning-based methods, such as machine learning techniques, have demonstrated their effectiveness in various fields, including image classification, autonomous driving, and more. As a result, scheduling researchers have increasingly turned to these strategies to address the challenges associated with real-time scheduling.

Machine Learning (ML) is an artificial intelligence technique that enables machines or robots to perform tasks in a manner similar to that of humans, although with some differences. There are three main types of ML techniques: Supervised Learning (SL), which is based on labelled data. As a result, the efficiency of these models is directly influenced by the precision of the input data [9]. Unsupervised learning (UL) works with unlabelled data, and examples of such techniques include clustering and feature detection [10]. The third type of ML is Reinforcement Learning (RL), where autonomous agents learn to make decisions in a given state (such as a workshop configuration at a specific time) to achieve their goals by interacting with the environment and maximizing reward signals. An example of RL in a scheduling context is the work by Belmamoune *et al.* [11], which applied RL to optimize the maximum completion time in a job-shop environment. Similarly, [12] used an RL algorithm to minimize job tardiness. Numerous studies have used RL to address JSSP, consistently showcasing its effectiveness due to its ability to support real-time decision making.

In this paper, we aim to leverage RL to minimize the number of delays (tardy jobs) in a job-shop environment under uncertainty. The number of tardy jobs is a critical factor directly related to customer satisfaction. Our work attempts to simulate a realistic workshop environment by introducing key uncertainties, including; stochastic variations in job arrivals, processing times, production routing, and machine reliability, across different workshop sizes. The originality of our work lies in:

(1) The employment of a new Double Q-Learning algorithm that combines the simplicity of traditional Q-Learning with double estimation to improve accuracy and reduce overestimation of action values, particularly in dynamic environments.
(2) A new scalable state representation that can be adapted to workshops of various sizes and a novel reward function designed to enhance scheduling by using predictive component to anticipate machine failures and prevent delays. Unlike traditional reactive models, it proactively plans tasks before the next expected failure, resulting in more robust and reliable policies.
(3) We introduce a transfer of learned Q-values between different setups, a topic rarely explored in the existing literature, especially with tabular-based learning algorithms.

The paper is structured as follows: Section 1 reviews contributions from various authors in the context of both static and dynamic job-shop scheduling problems. In Section 2, we describe the problem under study and the approach employed. Section 3 provides a detailed explanation of the Double Q-Learning algorithm utilized in our work. The results of our study and their discussion are presented in Section 4. Finally, we conclude with a summary and future perspectives in Section 5.

1. Related works

In this section, we review some works in the literature on the approaches and methods used to schedule jobs. We first address the case of a static environment, where all the information required for scheduling is available from the outset. We then examine the case of a dynamic environment, characterized by uncertain events.

1.1. Application of machine learning in static scheduling

A JSSP is said to be static if a finite number of jobs must be completed by a finite number of machines. Each job is made up of a predetermined sequence of task operations that must be completed uninterrupted for a set amount of time on a specific machine. Tasks from the same job cannot be processed simultaneously and each job must visit each machine exactly once. This is how [13] defined the JSSP in its static form. In the field of static JSSP, researchers have access to various standardized benchmarks [11, 14–20], allowing them to compare their results with those of other studies and objectively evaluate the effectiveness of the proposed approaches. The works cited above have all used existing job-shop benchmarks in the literature such as Lawrence instances [21] to study the job-shop.

In general, when using these instances, the main objective is to optimize the makespan. However, this is not the only criterion that can be optimized. Other objectives, such as minimizing waiting times, balancing workloads, or reducing energy consumption, can also be considered. This diversity of objectives and the complexity of the environment highlights the relevance of approaches such as machine learning.

Belmamoune *et al.* [11] used a Q-Learning algorithm to solve a job-shop scheduling problem one of the RL model-free algorithms with a new representation of the state of the environment based on machine loads. The agent was evaluated twice using two different methods. The reward depends on the objective function, which is the makespan. The actions selected by the agent are the dispatching rules. Tassel *et al.* [17] and Wang *et al.* [18] implemented a Proximal Policy Optimization (PPO). It is a policy-based method that limits policy updates. A meaningful state representation of the job's state in terms of allocation and remaining operations was presented by the two last cited works. The reward function that defines the difference between the duration of allocated operations and the idle time of a machine was closely related to the minimisation of the makespan, which is the objective function. In addition to the difference in learning algorithms, the presentation of the state of the environment, as well as the action space and reward function, differs from one contribution to another.

Another work related to the minimization of tardiness in a job-shop environment is the one of [12]. They used a Deep Q-Network (DQN) where ten well-known heuristic dispatching rules are chosen as the DQN's action set, and five state features of continuous value ranges related to machine utilization and job's remaining processing time are created for input to a Deep Neural Network (DNN).

Other works were interested in the flexible job-shop, which is closer to the real workshops. The authors in [19] used the actor critic architectures which consists of the decision-making actor and the evaluating critic. In general, researchers have relied on the connection between the reward function and the objective function, despite the variations observed in the learning algorithms and state representations.

1.2. Application of machine learning in dynamic scheduling

Dynamic scheduling, on the other hand, addresses scenarios where problem parameters are subject to changes, such as stochastic job arrivals, machine breakdowns, or other disruptions. This approach is typically used when disturbances occur frequently or when problem variables cannot be predetermined, making it impossible to construct a reference schedule during the offline phase [4]. Dynamic scheduling is more suitable for real-world applications where flexibility and adaptability are essential to respond to unexpected events.

It is difficult to design benchmarks for dynamic scheduling, which is why in the current literature we find examples of applications that vary from one scientific contribution to another. It is also interesting that because a dynamic scheduling problem has to be studied over a long enough period, a parameter like makespan cannot be taken into consideration in this case because of uncertain events that can prevent the completion time of

all tasks. For this reason, most of the works that are interested in the dynamic environment use an objective function related to the tardiness.

Shahrabi *et al.* [22] used a Q-Leaning algorithm to improve the performance of the scheduling method suggested for the dynamic job-shop problem, which takes into account machine failures and random job arrivals. In actuality, an optimization process's parameters at any given rescheduling point were chosen by a continuously evolving policy derived from RL. A variable neighborhood search was introduced as the foundation for the scheduling method. In order to determine reward values in learning processes based on the quality of chosen parameters, a novel method is also presented.

Workneh and Gmira [23] used DQN algorithm on dynamic job-shop problem to minimize the makespan. A DQN algorithm was designed with state features which represent important information about the environment for the DQN agent. They introduced the information in the workshop related to the objective function, which is the makespan. Also, for the reward, it represents the difference between the previous makespan and the current one.

Multi-agent systems (MAS) consist of multiple agents that interact with the same environment to learn how to solve a task simultaneously. Can simulate cooperative and competitive situations; they have been applied in many different domains [24]. It also represents a subfield of RL where agents can learn in the same environment using one of the RL algorithms. Liu *et al.* [25] suggested a multi-agent deep RL method to address the dynamic scheduling issue in job-shops. The decentralized scheduling agents were equipped with the DDQN algorithm, which was used to learn the connections between scheduling goals and production data and to make scheduling decisions almost instantly. The state was represented in two important characteristics: information of queuing jobs that are always visible to the scheduling agents, and the next machine of the operating job.

Zhang *et al.* [26] implemented a DRL based on MAS that combines the self-learning approach and the self-organization mechanism. The equipment agent's AI scheduler decision-making module is established using a multi-layer perceptron. The AI scheduler intelligently creates the best production plan to carry out task allocation based on the perceived state data of the workshop. The AI scheduler is updated using the PPO algorithm to enhance its decision-making performance based on the sample scheduling process. They took into account dynamic events such as random machine failures and stochastic job insertions.

In addition to DRL, PPO has often been used in dynamic environments [27,29]. Wang and Liao [27] proposed a PPO scheduling agent is trained based on a framework that integrates discrete event simulation. Copies of the trained agent can be linked to each machine for distributed control.

In [28], the authors present an optimization algorithm based on the integration of Long Short-Term Memory (LSTM) neural network and PPO in a DJSP. This algorithm dynamically adjusts scheduling strategies based on changes in the production environment. The LSTM can handle the memory in the neural network and integrates the historical scheduling information of the job-shop with the current information and improves the scheduling efficiency of the job-shop. Wu *et al.* [29] used PPO with hybrid prioritized experience replay. A novel state representation is designed based on the feasible solution matrix, which depicts the scheduling order of a scheduling task, a set of paired priority dispatching rules was used as the actions, and a reward function based on the machine idle time.

Most of the uncertain events introduced in the literature were the random arrival of jobs.

In most of the works cited, the authors use dispatching rules as actions to guide the decision-making process in their model. Samsonov *et al.* [30] modeled a dynamic environment using random processing time and operation assignment. An RL agent can choose actions using a new reward shaping and an action space related to the dispatching rules.

Yang *et al.* [31] demonstrate how combining RL with graph neural networks (GNNs) yields improved performance for dynamic JSSP by encoding the workshop state in a relational latent space that supports more informed scheduling decisions. Zhang *et al.* [32] employ DRL with Proximal Policy Optimization to learn dynamic scheduling policies for JSSP manufacturing systems. The agent uses historical data which represents the outcomes of interacting with the dynamic simulation. The approach was tested in real settings and outperformed current top methods, aiming for more sustainable production.

Pu *et al.* [33] introduce a novel Deep Reinforcement Learning (DRL) approach for Job Shop Scheduling using Graph Neural Networks (GNNs). The problem is modeled as a sequential decision-making process represented by a graph. A Graph Embedding–Heterogeneous Graph Neural Network (GE-HetGNN) encodes the state of the shop floor into node representations. These representations are then used by a multi-agent system, trained with a PPO algorithm, to jointly learn optimal machine assignment and job sequencing strategies that maximize a global reward. Zeng *et al.* [34] propose a flexible, hybrid framework that uses disjunctive graphs to represent state and a set of dispatching rules as actions, requiring minimal prior knowledge. It employs an attention mechanism for graph representation learning and a sophisticated double dueling deep Q-network with prioritized replay and noisy networks (D3QPN) algorithm to select the optimal rule for any given state. Song *et al.* [35] was interested in a human-machine collaborative decision-making by synergistically integrating human expertise with computational power. Its innovative components are structured as follows: First, the scheduling problem is represented using disjunctive graphs to model its complex relational structure. Second, a refined set of dispatching rules, selected via Data Envelopment Analysis, constitutes a flexible and efficient action space. Third, a Transformer model serves as the core feature extraction module, adeptly capturing intricate state relationships for powerful representations. Furthermore, the framework utilizes a Duelling Double Deep Q-Network with Prioritized Experience Replay to learn the optimal state-to-rule mapping.

1.3. Resume

Table 1 resumes the contributions of the cited works. Each of the authors in the aforementioned works made significant contributions to the algorithm's adaptation and implementation. We can observe a wide variety of algorithms employed to address problems in job-shop environments. When comparing their simplicity, Q-Learning stands out as one of the most straightforward RL algorithms. Its ease of adaptation to different environments makes it a practical choice for solving scheduling and optimization challenges in such contexts. The most variable factors from one author to the other when discussing RL algorithms are the way of representing the state of the environment and the reward function. The action space was most of the time the dispatching rules.

The state presentation provides the agent with the current situation of the environment. Some studies have used all the available information in the workshop as the state, but this approach can confuse the learning process by including unnecessary information. On the other hand, when too little information is provided about the environment's state, the learning process becomes weak and ineffective. The optimal approach is to analyze the influence of the current environmental characteristics on the overall learning objective and select the most relevant ones to represent the state. In most studies, the state information was tied to job attributes, which vary from one instance to another. The idea of generalizing the state presentation to make it adaptable to all workshops has not received much attention.

Unlike the approaches where rewards are given reactively after a decision, or after the end of the scheduling by comparing previous and current agent performances [23], in the studied approach we have introduces a predictive component. It continuously monitors machine queues and their operational status (working or failed) to allow the agent to anticipate and prevent potential delays or subsequent breakdowns during restarts. A distinctive feature of our approach is the integration of MTBF into the decision process, which enables the planning of pending tasks that should be executed before the next expected failure a strategy seldom explored in the literature, where failures are typically modeled as simple binary events. Ultimately, our reward mechanism encourages the agent to forecast and evaluate the impact of critical future states, thereby enhancing the robustness and reliability of the final scheduling policy.

Concerning dynamic environments, authors have defined uncertainties in various ways to better reflect real-world conditions. In all cases, the goal is to demonstrate the effectiveness of the proposed approach.

Due to the challenges of learning in dynamic environments, some authors have proposed an offline learning approach, with rescheduling implemented whenever uncertainties arise. However, when considering robustness, the ideal scenario is a learning method that can operate effectively in uncertain environments or through online learning.

TABLE 1. State of art table.

Author	Environment	Objective	Dynamic event	Algorithm	State	Action type	Reward
Yang *et al.* [31]	Dynamic	Minimize the earlier and later completion time	Job arrivals	GNN	A graph neural network (GNN)	Dispatching rules	The difference between the best and the current completion time.
Zhang *et al.* [32]	Dynamic	Maximize machine utilization and minimize order waiting time	Machine breakdowns and processing times	PPO	Machine status, remaining operations, action validation, remaining free buffer spaces of each machine and waiting times	Jobs in the queue of machines	Average utilization, waiting time
Shahrabi *et al.* [22]	Dynamic	Mean Flow Time	Job arrivals and Machine breakdown	Q-Learning and VNS	Number of jobs in the shop floor, Mean Operations Processing Time	Dispatching rules	VNS parameters
Pu *et al.* [33]	Dynamic	Minimize total energy consumption and Makespan	Machine failure, order insertion, and job cancellation	GNN	Inter-process transfer time, machine processing time, and delivery time	Machine selection and job operation selection	The difference between the best and the current completion time
Samsonov *et al.* [30]	Dynamic	Makespan	Processing time/ Operation assignment	DQN	Remaining times for the operations currently being processed, The sum of all operations' processing times currently in the queue of each individual machine, The sum of all operations' processing times for each individual job, The duration of the next operation for each individual job, The index of the next required machine for each individual job	Dispatching rules	Difference between expected Makespan and the found one
Zeng *et al.* [34]	Dynamic	Makespan	Machine breakdowns and different order requirement	D3QPN	Disjunctive graph of all possible operations	Dispatching rules	Related to the machine utilization
Yejian *et al.* [12]	Static	Tardiness	/	DQN	The utilization of all machines , The completion rate of all jobs, The remaining process waiting rate of all jobs, and the load rate of all machines	Dispatching rules	Slack time and remaining processing time
Wang *et al.* [18]	Static	Makespan	/	PPO	Processing status matrix, The machine and processing time matrix for each operation in jobs	Jobs in the queue of machines	Machine utilization
Tassel *et al.* [17]	Static	Makespan	/	PPO	All characteristics of jobs and machines	Dispatching rules	The difference between the duration of the allocated operations and the idle time of a machine
Zhang *et al.* [26]	Dynamic	Tardiness	Job arrivals	Multi agent and PPO	Operating time and order urgency, the difference between due time and current time	Jobs in the queue of machines	Cumulative ahead-of-schedule time, Workload balance standard deviation and product profit
Song *et al.* [35]	Dynamic	Makespan	Job arrivals	D3QPN	Disjunctive graph of all possible operations	Dispatching rules	Related to the machine utilization
Burggräf *et al.* [19]	Static	Makespan	/	Actor Critic	Number of operations, remaining time and processing time	Dispatching rules	Total lead time across all jobs
Liu *et al.* [25]	Dynamic	Cumulative tardiness	Job arrivals	Multi-agent Deep Q Learning	Information of queuing jobs, the next machine of the operating job	Dispatching rules	Job's tardiness
Belmamoune *et al.* [11]	Static	Makespan	/	Q-Learning	Jobs in the queue of the machine	Dispatching rules	Machine loads
Workneh and Gmira [23]	Dynamic	Makespan	Machine failure, Reprocessing of the job	DQN	Job index, The machine's end time	Jobs in the queue of machines	The difference between the previous makespan and the current one
Wu *et al.* [29]	Dynamic	Makespan	Processing time	PPO	A matrix of job number and operation number of each job	Dispatching rules	Completion time of jobs
Wang and Liao [27]	Dynamic	Tardiness	Job arrival	PPO	Number of jobs in the system, Number of jobs in the queue, Remaining time and remaining operations, Processing time of current and next operation, Slack time	Dispatching rules	Difference between the total slack time of time t and t + 1
Chen *et al.* [28]	Dynamic	Makespan and Tardiness	Machine failure	LSTM-PPO	Number of remaining operations, Machine utilization and Completion rates	Dispatching rules	Difference between the completion time (difference between the maximum tardiness time of the process) of the current state and the previous state operation
Our paper	Dynamic	Tardy jobs	machine failure, Job arrivals, Processing times	Double Q-Learning	Remaining time to due dates, Machine status	dispatching rules	Related to job's due date and machine status

In studies that have utilized tabular algorithms such as Q-learning and double Q-learning, authors typically apply learning to each workshop separately, without leveraging previous experiences from other instances. This approach is defined in the literature as Transfer Learning, a concept often associated with deep learning algorithms. However, adapting this idea to tabular algorithms has not been extensively explored in the literature. This gap has motivated us to combine the benefits of this approach with the simplicity of tabular algorithms.

After giving a general idea about the environment we are focusing on, and the various contributions, we will now explore the description of the dynamic job-shop environment as well as the approach used in the next section.

2. Problem description

In this section, we will present three key concepts relevant to the contribution of this paper: JSSP, RL, and transfer learning. The JSSP is a well-known combinatorial problem, and as such, we will briefly introduce it without going into extensive detail. The second part of this paper will focus on RL, with particular emphasis on the Q-learning algorithm. This section will be given greater attention, as adapting Q-learning to solve the JSSP is a key contribution of this work. Finally, we will introduce the concept of transfer learning, which we believe has significant potential for advancing learning techniques in this field.

2.1. Job shop environment

The Job Shop Scheduling Problem (JSSP) is a classic production scheduling challenge concerned with determining the optimal sequence for processing n jobs on m machines to minimize a specific objective function, such as makespan or total tardiness. Given that its solution space consists of all possible combinations of job sequences, the JSSP is categorized as a Combinatorial Optimization Problem (COP) [36, 37]. Due to its combinatorial nature and computational complexity, the JSSP is widely recognized in the literature as NP-hard [38–40]. In addition, scheduling a set of independent jobs with release times on an arbitrary number of unrelated machines so as to minimize the number of late jobs has been proved as Strongly NP-Hard by [41].

The JSSP is one of the most studied scheduling problems due to its industrial applications (production system management, logistics planning, manufacturing process optimization).

In a more formal manner, a JSSP can be defined as follows: Given a set J of n jobs $J = \{j_1, j_2, j_3, \ldots, j_i, \ldots, j_n\}$, a set of m machines $M = \{m_1, m_2, m_3, \ldots, m_j, \ldots, m_m\}$. Each job consists of an ordered sequence of operations to be processed on the machines: $J_i = \{O_{i1}, O_{i2}, O_{i3}, \ldots, O_{ij}, \ldots, O_{im}\}$, where each operation must be executed on a specific machine. Each machine can process at most one operation at a time. The order of operations within a job is constrained: an operation can only start after the completion of the previous operation; Each operation is associated with a machine on which it must be executed and a fixed and known processing time.

The JSSP environment is subject to the following constraints:

- Precedence constraints: Operations within the same job must be executed in a strictly defined order.
- Machine exclusivity constraints: A machine can only process one operation at a time.
- Non-pre-emption constraints: An operation cannot be interrupted once it has started.
- Machine availability constraints: Machines are available from the beginning of the scheduling horizon and cannot be shared between multiple operations simultaneously.

A Gantt diagram is usually used to present a feasible solution that satisfies the precedence constraints among jobs on the same machine and operations of the same job. In Figure 1, we represent a job-shop environment consisting of three jobs and four machines. Each job is represented by a distinct color, along with its corresponding processing path. Figure 2 presents a Gantt diagram of an arbitrary solution to the JSSP corresponding to Figure 1. The scientific community working on the JSSP agrees on the notation presented in Table 2.

A scheduling problem is considered dynamic if it exhibits one or more of the following characteristics:

1. Random job arrivals: New jobs can arrive at random times during the execution of the schedule, requiring

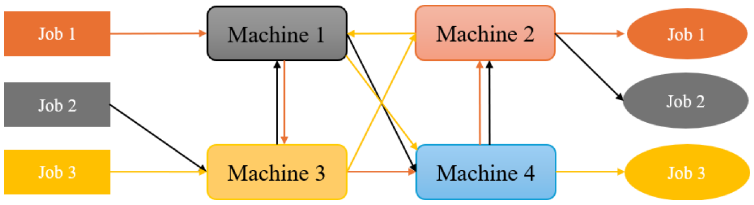

FIGURE 1. Example of a job-shop with 3 jobs and 4 machines.

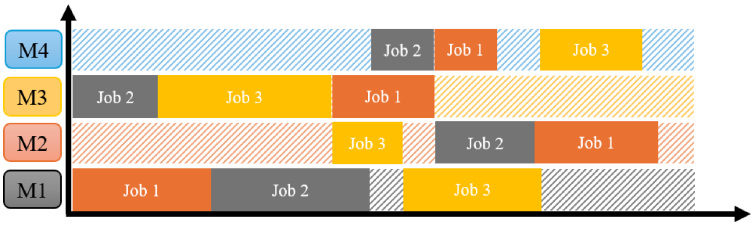

FIGURE 2. Gantt diagram that represents the solution of the problem.

TABLE 2. Notation of the scheduling variables.

Variable	Meaning
t_{ij}	Starting time of execution of job j_i in machine m_j / operation O_{ij}
r_i	Release date of job j_i
p_{ij}	Processing time of job j_i in machine m_j / operation O_{ij}
c_{ij}	Completion time of job j_i in machine m_j / operation O_{ij}
d_i	Due date of job j_i
C_i	Completion time of all operations of job j_i
C_j	Completion time of all operations in machine m_j

continuous reassessment of the plan to account for these stochastic arrivals.
2. Changes in job parameters: jobs parameters as priorities may change based on evolving system conditions, new information, or operator decisions, affecting the processing order.
3. Machine failures: Unexpected machine breakdowns can disrupt the schedule, requiring quick adjustments.
4. Variability in processing times: Task durations can vary due to factors such as material quality or machine conditions.
5. Changes in machine availability: Machine availability may fluctuate due to planned or unplanned maintenance or shifts in product demand.
6. Urgent requests: Urgent orders may arise, altering production schedules or accelerating deadlines for certain tasks.

These characteristics make the dynamic JSSP require a flexible and adaptive approach, often utilizing real-time optimization techniques, artificial intelligence, or machine learning for schedule adjustments.
In the specific case of the dynamic JSSP we are studying here, we consider:

- Random arrival times of jobs following a Poisson distribution.
- Random processing times follow an exponential distribution.
- Random number of tasks (operations) for each job. The number of operations follows a uniform distribution.
- Machines are subject to failures, where only one machine can be in a failure state at any given time. Each machine can enter a failure state after processing at least one task.

For the environment we have just described, namely the dynamic JSSP, we will apply RL techniques to address the problem, the principles of which will be presented hereafter.

2.2. Reinforcement learning

RL is one of the most widely used ML algorithms for real-time decision-making. It is closely related to a Markov decision process (MDP), where the agent takes actions in a given state s_t of the environment to transition to a subsequent state s_{t+1}. This transition is typically represented as a 3-tuple (s_t, a_t, s_{t+1}) where s_t is the current state, a_t is the action taken, and $s_t + 1$ is the resulting next state.

In RL, the agent observes the environment and takes actions based on these observations. In response, the environment provides a reward signal and transitions to a new state. This process is typically represented by a 4-tuple (s_t, a_t, r_t, s_{t+1}) by adding to the previous 3-tuple a fourth element which represents the reward r_t. The agent uses this feedback (reward) to update its policy, which is a mapping from observations to actions. The agent's goal is to learn a policy that maximizes long-term returns, which is the sum of the rewards received over time [42].

QL is a model-free form of RL, often considered a method of asynchronous dynamic programming. It allows agents to learn optimal actions in Markovian environments by experiencing the consequences of their actions [43]. The QL considers a Q-function that updates a Q value associated with the action applied in a given observed state $Q(s_t, a_t)$. The character Q refers to the word quality of the action a_t in a state s_t at the moment t. The Q-values will be updated after each experience using Bellman equation (Eq. 1).

$$Q^{\text{new}}(s_t, a_t) = (1 - \alpha)Q^{\text{old}}(s_t, a_t) + \alpha(r_t + \gamma \max(Q(s_{t+1}, a))). \tag{1}$$

Where $Q^{\text{new}}(s_t, a_t)$ is the new Q value of the action a_t in the state s_t. α is the learning rate, *i.e.* the probability of accepting the target value $(r_t + \gamma \max(Q(s_{t+1}, a)))$. γ is a discounting factor, used to balance the immediate reward r_t and the future reward $\max(Q(s_{t+1}, a))$. This later represents the maximum Q value of the next state s_{t+1}. The idea behind the Q-Learning algorithm is to find the best action in a given state, taking into account both the immediate and the future reward.

The updated Q values will be saved in a table named Q table. Each row in the Q table represents the state of the environment and the columns are the actions. At the end of the learning process, this table is used to choose the best action (with a maximum value of $Q^{\text{a}}(s_t, a_t)$. Algorithm 1 explains the basic Q-Learning steps. To make the agent learn how to choose the optimal actions in the learning phase, it must typically navigates the same environment numerous times. Exploration and exploitation are two fundamental strategies used by the agent in RL to balance the trade-off between discovering new actions and leveraging existing knowledge [44].

Algorithm 1: Q-Learning.

Initialize $Q(s, a)$ arbitrarily
  Repeat for each episode:
  Initialize s_t
  Repeat each step of the episode:
  Choose a_t given s_t using the policy derived from $Q - table$ (*e.g.*, $\epsilon - greedy$)
  Take action a_t, observe r_t, s_{t+1}
    $Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max(Q(s_{t+1}, a)))$
  $s_t \leftarrow s_{t+1}$
  Until s_t is terminal

Exploration involves taking random actions to explore different possibilities within the environment. This prevents the agent from getting stuck in a suboptimal region of the solution space and encourages it to discover new strategies. Exploitation, on the other hand, refers to choosing the action that has the highest Q-value, based on the agent's current knowledge. This approach aims to maximize the agent's reward by selecting the most rewarding action in the current state [45].

At the beginning of the learning process, the agent prioritizes exploration to gather more information about the environment. Over time, as the agent learns more about the potential outcomes of different actions, it gradually shifts towards exploitation. This balance between exploration and exploitation allows the agent to improve its decision-making and refine its policy [46]. The $\epsilon - greedy$ policy is commonly used to manage this balance. In this policy, a random number is generated and compared to a threshold value $\epsilon \in [0, 1]$. If the random value is greater than ϵ, the agent explores the environment by selecting a random action. However, if the random value is less than ϵ, the agent exploits its knowledge and selects the action with the highest Q-value, thereby maximizing the expected reward. By using $\epsilon - greedy$ exploration, we balance the trade-off between discovering new actions (exploration) and selecting actions that are known to yield high rewards (exploitation). The $\epsilon - greedy$ policy is described in Algorithm 2.

Algorithm 2: $\epsilon - greedy$.

Choose a value of $\epsilon \in]0, 1[$
Choose a random value $\eta \in]0, 1[$
If $\eta \leq \epsilon$:
    The agent chooses a random action
Else:
    The agent chooses an action with maximum Q-value

A new algorithm called Double Q-learning have been developed from the classical QL algorithm which has not been touched upon much in literature [47–49]. It uses a double estimator approach (two Q-values) to determine the value of the next state. In an uncertain environment, the overestimation that results from the single estimator approach can have a large negative impact on algorithms that use this method. Double Q-learning stores two Q functions: Q^{A} and Q^{B}. Each Q function is updated with a value from the other Q function for the next state [50].

Since Q^{B} was updated on the same problem, but with a different set of experience samples, this can be considered an unbiased estimate of the value of this action. A similar update is used for Q^{B}, using the maximum Q^{A} value of the next state. It is important that both Q functions learn from separate sets of experiences, but to select an action to perform, both value functions can be used. In our experiments, we calculate the average of the two Q values for each action and then perform $\epsilon - greedy$ exploration with the resulting average Q values.

This dual-Q function setup is particularly beneficial in environments with complex, noisy, or uncertain dynamics, where overestimation can significantly impair learning efficiency. By utilizing two independent value estimates, Double Q-learning leads to more stable and reliable learning, resulting in improved performance in RL tasks. The idea of the Double Q-Learning is presented in Algorithm 3.

Algorithm 3: Double Q-Learning.

Initialize Q^{A} ,Q^{B} arbitrarily
    Repeat for each episode:
    Initialize s_t
    Repeat each step of the episode:
    Choose a_t given s_t using, $\epsilon - greedy$
    Take action a_t, observe r_t, s_{t+1}
    Choose (*e.g.* random) UPDATE Q^{A} orQ^{B}
    If UPDATE Q^{A} :
        $Q^{\mathrm{A}}(s_t, a_t) = (1 - \alpha)Q^{\mathrm{A}}(s_t, a_t) + \alpha\big(r_t + \gamma \max\big(Q^{\mathrm{B}}(s_{t+1}, a)\big)\big)$
    If UPDATE Q^{B} :
        $Q^{\mathrm{B}}(s_t, a_t) = (1 - \alpha)Q^{\mathrm{B}}(s_t, a_t) + \alpha\big(r_t + \gamma \max\big(Q^{\mathrm{A}}(s_{t+1}, a)\big)\big)$
    $s_t \leftarrow s_{t+1}$
    Until s_t is terminal

2.3. Transferred Q-Learning

Transfer Learning is a machine learning paradigm in which knowledge acquired from a previously solved task is leveraged to improve learning efficiency and performance on a new, yet related, task [51]. In the literature, we found the concept of TL in the context of classification, multitask learning and inductive learning [52]. In Reinforcement Learning (RL), an agent learns to maximize a cumulative reward signal which may be time-delayed by taking sequential actions. A more recent development in the field is the application of TL to RL tasks [53].

In RL, learning a task from scratch can be highly time-consuming and computationally expensive due to the exploration of large state spaces and the need to gather extensive experience through trial and error. However, Transfer Learning offers a powerful mechanism to overcome these challenges by enabling an agent to leverage knowledge acquired from previously learned tasks or environments. By transferring this prior knowledge, the learning process in new, yet related tasks can be significantly accelerated. This is especially beneficial when the new task shares similar characteristics or features with the previous ones, allowing the agent to build upon existing knowledge rather than starting from zero. Transfer Learning not only reduces training time but also opens the door to solving more complex tasks that might be too difficult or time-prohibitive to learn from scratch, especially in dynamic environments where new configurations or challenges frequently arise [54].

The approach we propose here to address the dynamic JSSP adopts the principle of Transfer Learning to optimize the efficiency of the RL algorithm in JSSP. Rather than initiating the learning process from scratch each time we encounter a new workshop configuration with varying parameters, we apply the knowledge gained from previous learning episodes. Specifically, we transfer the Q-tables, which encapsulate the learned policies from prior configurations, and use them as starting points in the new configuration. This allows the agent to immediately benefit from prior experiences, reducing the time spent exploring the state space and enabling faster adaptation to the new environment.

The Q-values stored in the transferred Q-tables represent the agent's previous encounters with the environment, effectively encoding the knowledge of state-action pairs that have led to successful outcomes. By reusing this prior knowledge, the agent does not need to relearn from scratch, but instead can fine-tune its policy to accommodate the new configuration's parameters. The transferred experiences help the agent to quickly identify potentially successful actions in the new configuration, thereby reducing the need for extensive exploration and accelerating the learning process. This not only reduces the overall time required for the agent to become proficient in the new configuration but also facilitates more efficient use of resources, making it possible to tackle increasingly complex tasks within reasonable time frames.

Ultimately, Transfer Learning helps us overcome the limitations of traditional RL, allowing for faster, more efficient learning, and enabling the agent to solve increasingly complex problems that would otherwise be too resource-intensive to approach from scratch.

3. Proposed double Q-Learning framework

3.1. General framework

In this section, we outline the key contributions of this work, specifically the used approach is based on a multi-agent system applied to the dynamic JSSP. The proposed framework utilizes a multi-agent system operating within the scheduling environment, where each agent is assigned to a particular machine and collaborates with others to optimize the objective function. We introduce two distinct types of agents: job agents and machine agents.

The primary function of the job agents is to monitor the environment by encoding pertinent information regarding the jobs on the shop floor, such as job priorities, processing times, and deadlines. These agents continuously gather data about the jobs in progress and communicate this information to the machine agents, ensuring that the machines have up-to-date insights into the job requirements and operational conditions.

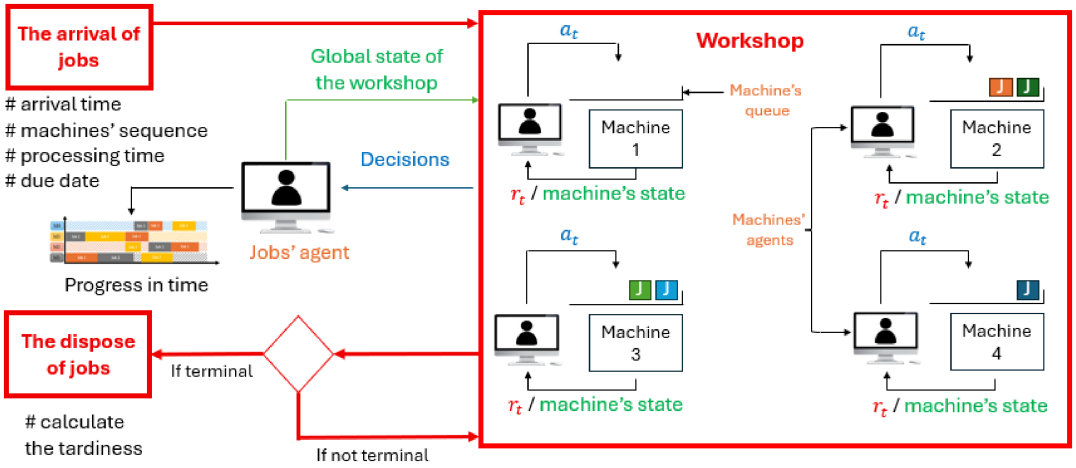

FIGURE 3. The dynamic job shop environment.

Conversely, the machine agents are tasked with managing the machine queue. They observe the status of their assigned machine, taking into account factors such as availability, workload, and operational state. Based on this information, each machine agent selects a job from the queue of competing tasks for execution. The decision-making process of the machine agents is critical for optimizing the overall system performance, as they must efficiently balance competing job requirements while minimizing delays and maximizing machine utilization. The agents receive feedback from the environment in the form of rewards or penalties based on their actions. Positive actions, such as efficient job scheduling or reduced idle time, yield rewards, while suboptimal decisions, such as causing bottlenecks or missing deadlines, result in penalties. This feedback loop enables the agents to refine their decision-making processes over time, thereby improving their performance.

Figure 3 provides a visual representation of the interactions between the two types of agents and their roles within the dynamic JSSP framework. It illustrates how the job agents and machine agents collaborate, exchange information, and make decisions in real time to achieve an optimized scheduling solution.

3.1.1. State presentation

An accurate representation of the system state is essential for effective decision-making and process optimization. To achieve this, we define the system state as a vector composed of three key attributes, which collectively capture the status of job flow and machine availability. These three key attributes are:

(1) **Job Queue Ratio (C1):** The percentage of jobs currently in the queue relative to the total number of jobs awaiting processing within the system.
(2) **Remaining Time Until Due Dates (C2):** The time remaining before the jobs in the queue reach their respective due dates. It is important to note that if a job is overdue, this value becomes negative, indicating a delay.
(3) **Machine Status (C3):** The operational state of the machine, which can be either in operational mode or failure mode (denoted as op/fail).

The primary objective of this state representation is to provide a comprehensive overview of the job queue's status, including both its size and the time constraints associated with the pending jobs.

Additionally, this formulation is adaptable and can be applied to workshops of any scale, ensuring its relevance across different production environments.

3.1.2. Action space

In scheduling and production control, dispatching rules play a crucial role in determining job sequencing and overall system performance. Several studies, as referenced in Section 2, have employed dispatching rules to define the action space in decision-making models. Similarly, in the studied approach, we have selected four well-established dispatching rules as the actions taken by each agent to optimize job prioritization.

The decision to define actions as the application of dispatching rules stems from the need for a simple yet effective decision-making framework that can be easily implemented in dynamic shop floor environments. Dispatching rules provide a straightforward mechanism for job selection without requiring extensive computational resources, making them well suited for real-time scheduling. Moreover, they allow flexibility and adaptability to different production scenarios while aligning with our main objective: minimizing tardy jobs. The five dispatching rules we consider are as follows:

– **Shortest Processing Time (SPT):** gives priority to jobs with minimal processing time.

$$\min(p_{ij}). \tag{2}$$

– **Earliest Due Date (EDD):** gives the priority to jobs with the minimal due date.

$$\min(d_i). \tag{3}$$

– **Least Slack Time (LST):** gives the priority to jobs with the minimal slack time s_i or the $Margin$, which represents the difference between the due date d_i and the current time t_{now} minus the sum of the remining processing times.

$$s_i = d_i - t_{\text{now}} - \sum p_i j \tag{4}$$

$$\min(s_i). \tag{5}$$

Where t_{now} represents the time now or the time at this moment.

– **Margin and operating time (TZAR):** gives priority to jobs with minimal *margin* except for the next operation.

$$\min(s_i + p_{ij}). \tag{6}$$

The selection of these dispatching rules aligns with our primary objective of minimizing tardy jobs. By incorporating due dates and slack times into the decision-making process, these rules contribute significantly to optimizing scheduling efficiency and improving overall system performance.

3.1.3. Reward function

To effectively optimize job scheduling in a dynamic manufacturing environment, it is crucial to define a robust reward function that guides the agent decision-making process. In this context, we propose a reward mechanism tailored to two distinct machine states: operational and failure. This approach ensures that the scheduling strategy adapted to machine availability while minimizing job tardiness and maximizing throughput.

In this paper, the reward function is defined for two different scenarios. The first scenario occurs when the machine is in the "op" state (operational mode), where the reward is determined based on the margin of the selected job for execution. The second scenario arises when the machine is in the "fail" state (failure mode), in which case the reward is computed based on the margin of the job that will be executed after the machine is repaired. To achieve this, the agent will calculate the Mean Time Between Failures (MTBF) and estimate the remaining time before the next machine failure. The primary objective is to maximize the number of jobs completed before the next failure occurs.

Algorithm 4: Reward function.

$Margin$ = Calculate the margin of the chosen job
if $machinestate = op$:
    if $Margin$ = minimal one in the queue of the machine
        $r_t = +2$
    else:
        $r_t = -2$
else:
    if $Margin \leq \text{MTBF} - t_{\text{now}}$:
        $r_t = -1$
    else:
        $r_t = -2$

Algorithm 5: Adapted double Q-learning algorithm.

Define the number of agents depending on machines' number
Initialize the Q^A,Q^B tables for each agent
Initialize Q^A,Q^B arbitrarily
Initializing the hyper parameters of the Bellman equation $(\alpha, \gamma, \epsilon)$
Define the ϵ_step
For (number of episodes) or While $\epsilon < 1$:
    Terminal = False
    For each machine:
        If there is more than one job in the queue on a machine:
            Initialize s_t
            Choose a_t given s_t using, $\epsilon - greedy$
            Take action a_t, observe r_t, s_{t+1}
            Choose (*e.g.* random) UPDATE (Q^A) or UPDATE (Q^B)
            If UPDATE Q^{A} :
                $Q^{\text{A}}(s_t, a_t) = (1-\alpha)Q^{\text{A}}(s_t, a_t) + \alpha\big(r_t + \gamma \max\big(Q^{\text{B}}(s_{t+1}, a)\big)\big)$
            If UPDATE Q^{B} :
                $Q^{\text{B}}(s_t, a_t) = (1-\alpha)Q^{\text{B}}(s_t, a_t) + \alpha\big(r_t + \gamma \max\big(Q^{\text{A}}(s_{t+1}, a)\big)\big)$
        $s_t \leftarrow s_{t+1}$
        $\epsilon += \epsilon\_step$
        If s_t is terminal
            Terminal = True
    If reaching final episode or $\epsilon >= 1$:
        break

The details of the reward function are outlined in Algorithm 4. The decision to assign rewards of -2 and -1 in the failure scenario stems from the fact that the agent's action has not yet been executed; rather, it represents a prediction of what the agent would select once the machine returns to operational mode. Managing the job queue during machine breakdowns is critical, as the queue may continue to grow if new jobs arrive from other machines that remain in operation. Figure 4 provides an overview of the proposed approach implemented by the agents controlling the machines, while Algorithm 5 details the process of learning updates. Notably, each agent maintains its own Q^A and Q^B tables to facilitate learning through Double Q-learning, ensuring a more stable and efficient decision-making process. To validate the effectiveness of the proposed approach, we conduct a series of numerical experiments, comparing its performance with various well-known dispatching rules. These experiments evaluate the efficiency of our method in different operational scenarios by analyzing two key performance metrics: the number of tardy jobs and the total tardiness. The following section presents the numerical results, highlighting the advantages and limitations of the used approach compared to traditional scheduling strategies.

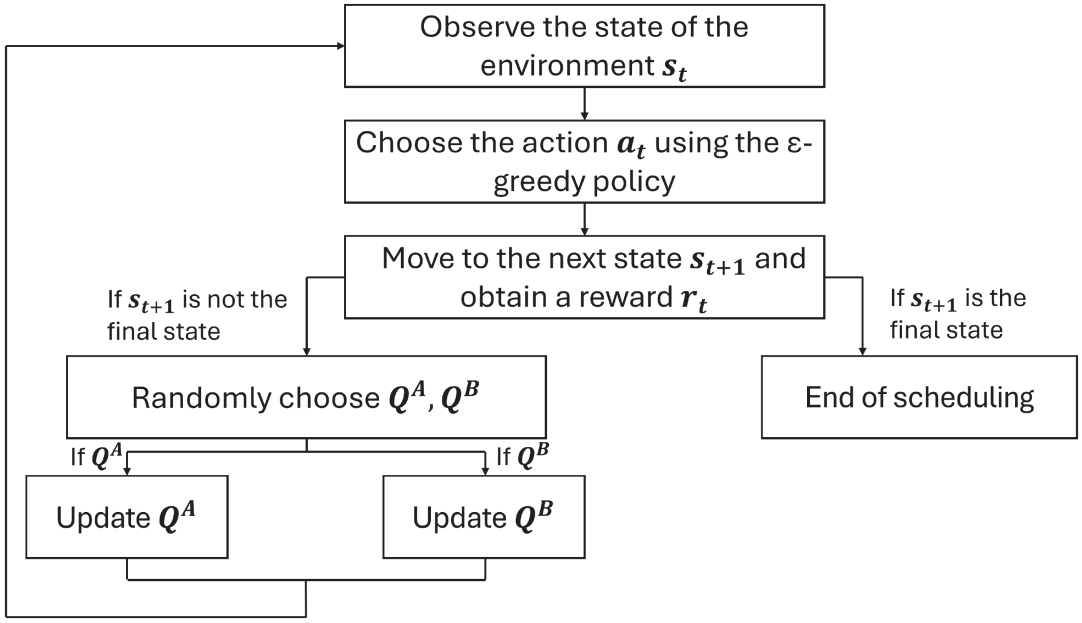

FIGURE 4. Approach organigram.

4. EXPERIMENTATION, RESULTS AND ANALYSES

4.1. Dynamic environment modeling

To model a dynamic environment that closely resembles real-world industrial conditions, we explore various approaches and adopt the workshop configuration proposed by [55], This setup provides a realistic representation of industrial production systems, making it well-suited for studies on dynamic scheduling [56].

The workshop consists of four machines, a number deemed appropriate for analyzing scheduling dynamics. The arrival of jobs in the system is limited to a predefined maximum, ensuring a controlled yet dynamic workflow. Each newly inserted job follows a sequence of operations, which is uniformly distributed between two and six operations [2,6], meaning that a job may visit the same machine multiple times. However, a job never executes two consecutive operations on the same machine, ensuring that the workflow remains distributed across multiple machines.

The processing time for each operation is generated using an exponential distribution with a mean of 1, simulating realistic variability in execution times. The due date for each job is determined using equation (7), where w represents a due date factor that influences scheduling constraints. In this work, we adhere to this environment generation method to ensure consistency with prior studies and enable meaningful performance comparisons.

$$d_i = w \cdot \sum_{j=0}^{m} p_{ij}. \tag{7}$$

In the following, we will explain each step of the environment modelling process.

4.1.1. Job arrivals

Job arrivals in the system are modeled as a Poisson process, *i.e.*, the inter arrival times follow an exponential distribution with a parameter defining to the machine's utilization rate.

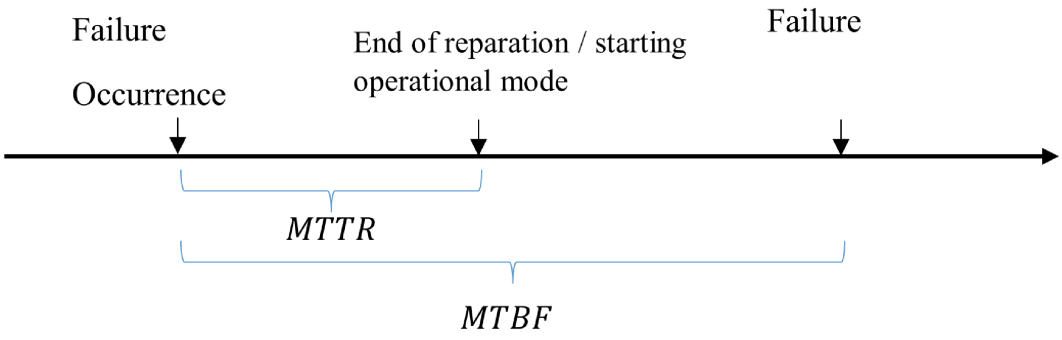

FIGURE 5. Breakdown presentation.

TABLE 3. Numerical data.

Number of jobs	$[50, 100, 500, 1000]$
Utilization rate of the machine	$0,8$
Job arrival rate	$1/0,8$
Due date factor	$[uniform(1.5, 3), uniform(4.5, 6)]$
Failure frequency μ	$[0.5, 0.75]$
MTTR	$[50, 100]$
Number of learning episodes	5000
Epsilon	from 0 to 1
Epsilon_step	from 0.001
Learning rate	0,5
Discount factor	0.5

4.1.2. Machine breakdown

when addressing machine breakdowns, two key metrics must be considered: the Mean Time to Repair (MTTR) and the Mean Time Between Failures (MTBF). In our case, both follow an exponential distribution. Additionally, we introduce σ, the failure frequency in equation (8).

$$\sigma = \text{MTTR}/\text{MTBF}. \tag{8}$$

Figure 5 illustrates the relationship between MTTR, MTBF, and the estimated failure time. The mean values of MTTR and σ are provided as inputs, while MTBF is derived from these parameters.

4.2. Results and discussion

The results of used approach are summarized in Table 4, which presents the average number of late jobs for 1000 executions for each instance, along with the standard deviation of the delay calculated from these 1000 results. Detailed numerical data are provided in Table 3. We compared the Double Q-learning-based transfer algorithm with the same parameters used with several heuristic methods, including **SPT (Shortest Processing Time)**, **FIFO** (First In First Out), **LST** (Least Slack Time) , **EDD** (Earliest Due Date) **MDD** (Modified Due Date) and **RAND** (Random). The RAND heuristic employs a random assignment rule for each decision; to ensure a fair comparison, we executed this heuristic 100 times for each generated instance and selected the best outcome.

For simulation modeling, Python programming language is used on an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz and a RAM of 8GB. The learning parameters such as the epsilon value and the learning rate, discount factor, and number of episodes are summarized in Table 3.

TABLE 4. Mean tardy jobs and standard deviation of a set of 1000 instances for each number of jobs with the parameters: (Machine utilization rate = $0.8/w \rightarrow$ due date factor/$\mu \rightarrow$ failure factor/MTTR $\rightarrow$ Mean Time To Repair).

Jobs	SPT		FIFO		LIFO		LST		EDD		MDD		RAND		Double-QL	
	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV
Case 01 → w = uniform (1,5; 3)/μ = 0,5/MTTR = 50																
50	24,34	6,31	32,55	6,75	25,93	5,04	31,56	7,78	29,83	7,52	27,07	6,46	27,28	6,82	**23,20**	1,49
100	53,60	10,07	73,36	10,87	56,59	7,60	73,25	11,89	69,81	11,99	62,64	10,47	63,02	11,22	**52,53**	6,23
500	297,36	26,58	411,00	29,50	310,00	21,90	421,00	31,10	410,00	33,20	364,00	28,80	363,00	32,20	**285,12**	9,85
1000	910,65	14,87	964,31	14,68	872,57	14,84	963,06	14,87	961,15	15,93	902,48	16,26	932,25	15,05	**570,95**	38,60
Case 02 → w = uniform (4,5; 6)/μ = 0,5/MTTR = 50																
50	31,92	5,62	34,19	5,66	32,42	5,29	33,10	6,47	32,60	6,18	29,79	6,09	31,23	6,76	**22,48**	5,43
100	69,64	7,73	76,94	8,36	70,53	6,83	75,12	8,50	74,12	8,84	67,83	8,58	70,91	7,67	**54,75**	1,96
500	434,09	14,06	457,09	16,49	421,53	12,84	454,02	17,81	452,46	18,13	427,50	15,72	440,12	16,64	**361,66**	20,77
1000	807,18	22,84	946,57	19,97	769,75	20,92	939,69	21,38	940,04	21,09	874,08	22,14	874,33	21,77	**720,68**	33,17
Case 03 → w = uniform (1,5; 3)/μ = 0,75/MTTR = 50																
50	41,50	2,64	42,92	2,44	41,02	2,52	42,33	2,47	42,33	2,56	41,47	2,57	41,28	2,59	**39,43**	3,15
100	86,79	3,54	91,61	2,85	84,83	3,60	91,11	3,05	90,62	2,93	87,46	3,48	88,01	3,38	**84,69**	3,75
500	461,51	7,10	488,69	3,64	442,06	7,41	487,62	3,86	487,10	3,91	467,57	6,29	472,77	5,86	**420,26**	23,37
1000	922,20	11,99	968,27	13,48	882,24	12,60	966,38	14,24	964,89	14,97	928,54	14,27	940,76	14,58	**840,04**	28,01
Case 04 → w = uniform (4,5; 6)/μ = 0,75/MTTR = 50																
50	32,83	4,44	35,47	3,94	33,96	3,54	34,38	4,67	33,94	4,66	**31,47**	4,22	33,14	4,18	32,97	4,43
100	71,75	5,50	78,27	5,83	72,05	4,64	76,03	6,76	75,72	6,60	**69,79**	5,85	72,61	5,38	72,41	6,20
500	393,15	10,97	453,76	15,62	380,61	10,50	448,08	17,37	447,31	17,69	403,58	14,20	414,18	14,58	**340,45**	26,59
1000	814,21	14,96	947,16	18,66	778,14	14,38	939,12	21,32	940,10	20,82	876,79	19,68	875,85	20,28	**774,40**	33,14
Case 05 → w = uniform (1,5; 3)/μ = 0,5/MTTR = 100																
50	39,73	4,38	40,66	3,96	39,36	4,71	40,40	3,96	40,15	4,31	39,38	4,68	39,18	4,61	**37,91**	4,28
100	81,45	6,99	84,95	5,55	81,69	5,85	84,19	5,52	83,57	6,18	81,77	6,31	82,19	5,46	**79,02**	6,37
500	438,87	13,93	460,61	15,51	426,40	12,17	459,17	17,13	457,05	17,60	442,03	15,54	445,74	15,88	**378,66**	25,52
1000	901,43	19,82	949,17	21,68	868,26	17,01	946,68	22,41	943,59	23,64	908,24	19,18	920,47	20,89	**813,61**	38,08
Case 06 → w = uniform (4,5; 6)/μ = 0,5/MTTR = 100																
50	20,90	12,37	24,77	11,25	23,37	10,25	21,26	14,21	21,30	13,81	**20,01**	12,30	20,68	13,37	20,35	13,16
100	**51,40**	22,48	63,49	18,62	56,15	16,40	56,66	24,40	58,95	22,41	52,03	21,63	55,96	21,27	52,00	11,15
500	**331,82**	57,24	429,67	38,14	333,94	41,61	418,06	46,54	417,26	46,88	366,83	44,43	379,71	44,36	346,14	31,97
1000	801,70	30,82	919,77	30,73	769,15	24,83	911,42	34,72	910,69	34,01	852,21	33,23	850,10	30,96	**745,20**	48,62
Case 07 → w = uniform (1,5; 3)/μ = 0,75/MTTR = 100																
50	40,64	2,91	41,78	2,92	40,73	2,84	41,30	2,97	41,09	3,12	40,80	2,91	40,64	3,10	**38,74**	3,69
100	82,78	4,35	85,73	5,07	82,78	3,92	84,90	5,30	84,20	5,24	83,17	4,64	83,45	4,65	**81,59**	5,20
500	440,85	12,15	460,35	15,99	428,66	9,97	458,97	16,89	457,36	17,11	442,81	13,34	447,30	14,70	**379,79**	26,14
1000	897,23	17,66	943,77	22,81	865,58	15,40	940,29	23,10	939,45	24,31	887,43	21,56	911,67	22,85	**800,35**	34,74
Case 08 → w = uniform (4,5; 6)/μ = 0,75/MTTR = 100																
50	36,53	3,85	37,93	3,03	36,95	3,53	37,28	3,46	37,24	3,79	36,68	3,48	36,43	3,63	**33,43**	4,74
100	74,13	4,80	80,19	5,74	73,87	4,66	79,31	6,22	78,95	6,45	75,69	5,23	75,38	5,09	**72,97**	6,33
500	394,63	11,50	439,30	21,38	381,66	10,07	435,01	22,45	433,66	23,46	410,12	16,83	407,84	17,10	**344,94**	24,72
1000	804,71	17,18	916,62	29,94	774,52	14,90	909,61	32,77	908,52	31,76	830,14	23,97	845,80	25,60	**729,4**	34,53

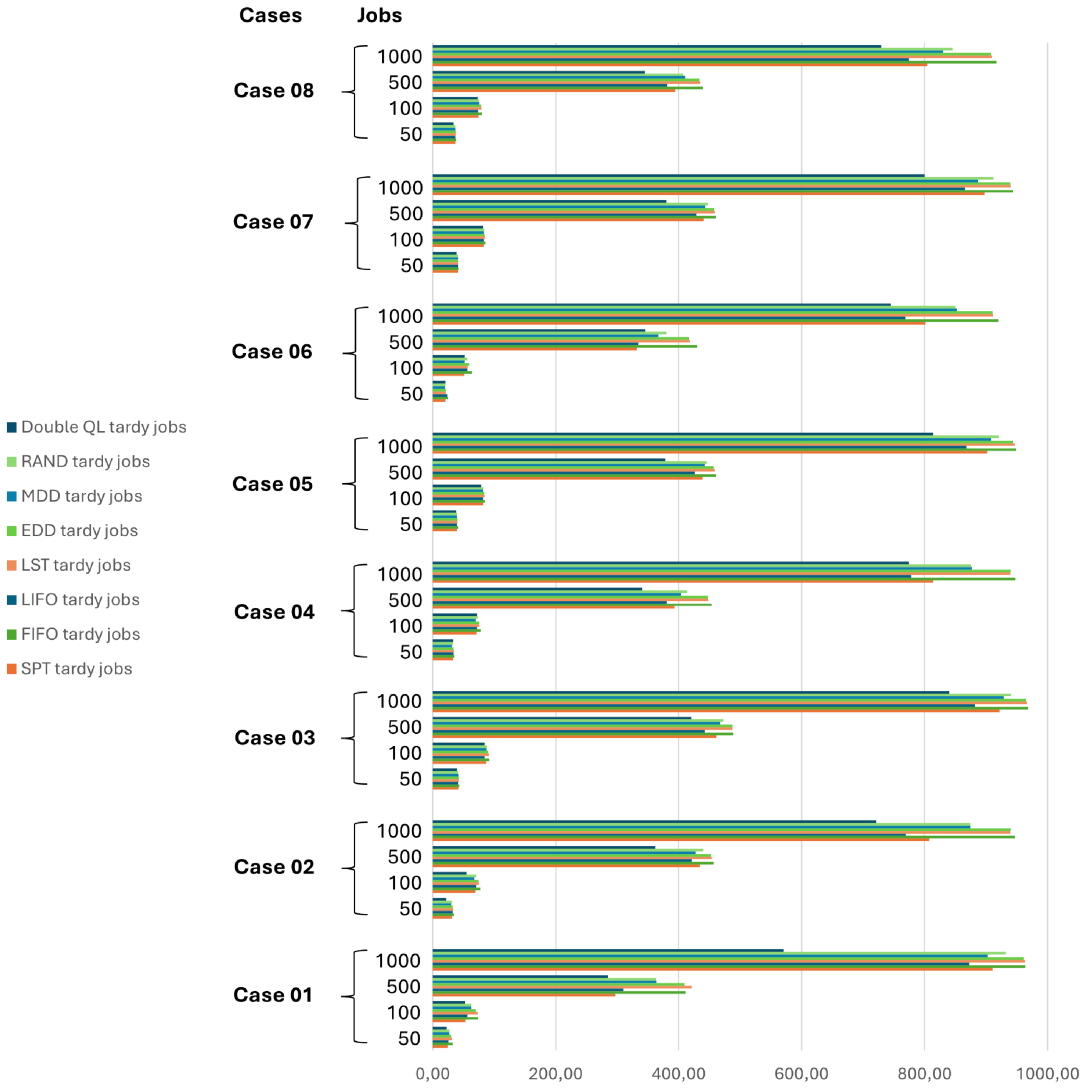

FIGURE 6. A comparative chart of the mean number of tardy jobs between the Double-QL algorithm and the heuristics (SPT, FIFO, LST, EDD, MDD, and RAND) across all cases listed in Table 4 for the four chosen workshop sizes.

In total, 32 experiments are presented in Table 4 were carried out with the parameters mentioned in Table 3 which are classified into 8 cases. The objective is to evaluate our method in various configurations and under different conditions.

To have a better vision of our results, Figure 6 represents a graph that illustrates the results in terms of tardy jobs. The dispatching rules SPT and LIFO perform satisfactorily against the other ones, especially in large workshops. On the other hand, the Double-QL outperforms the other rules in most cases, with a percentage of 81%.

As expected, a dispatching rule that performs well in one instance may not necessarily yield good results in another. Dispatching rules do not guarantee the robustness of their policies across different workshops. Using a single policy is not always reliable. Therefore, combining dispatching rules could potentially yield better

FIGURE 7. Box plot of tardy jobs of two cases (case 02 and case 08) of two different workshop sizes (50 and 100).

results. However, the combination must be carefully selected. As demonstrated using the Rand heuristic, despite randomly selecting policies for decision-making and choosing the best solution out of 100 executions, Rand did not perform as well as the other dispatching rules. The key difference between the used approach and Rand lies in the incorporation of a learning process that considers the state of the environment and a reward mechanism.

In our experiments, we identified two scenarios related to standard deviation and average delay.

(1) A maximum average delay (case 01, case 02, case 03 and case 05) was observed on the results yielded by FIFO.
(2) In most instances but specifically large ones (number of jobs 500 and 1000) our algorithm presented minimum average delay but larger standard deviations compared to other heuristics.

As mentioned above, we examined four different workshop sizes (50, 100, 500, 1000). The probability of delays (ranging from 0 to the maximum number of jobs) is greater with fewer jobs (50, 100), but it becomes much smaller with many jobs (500, 1000). This is why large workshops show a higher standard deviation in the number of tardy jobs. Figure 7 illustrates this phenomenon by presenting the box plot for 50 and 100 jobs. The boxes are closely grouped, and the standard deviation is minimal in the case of Double-QL. For the 50-job scenario, 75% of the Double-QL results fall below the median of the other heuristics. This indicates that, despite the variability in the results, 75% of Double-QL's results are equivalent to or better than 50% of the results produced by the other heuristics.

Figure 8 illustrates the box plot for case 7 from the set of cases in the solution table, focusing on instances with 500 and 1000 jobs. It can be observed that the population of delay results for the double-QL algorithm lies below those of the other dispatching rules, despite the variance in the solutions generated by double-QL. This serves as further evidence that our algorithm performs well in dynamic workshop environments, particularly in terms of minimizing the number of late jobs and for workshops with a large number of jobs. The illustrated figures represent the majority of cases in the results table and explain the standard deviation values obtained.

4.3. Improving the learning phase

In traditional tabular algorithms, learning is generally limited to a specific environment. This approach can lead to an increase in learning time, especially when the training is repeated for each individual instance. To address this, we optimized the learning process by limiting it to 100 jobs for each scenario described in the table. This strategy significantly reduces learning time while maintaining the effectiveness of the algorithm.

In the learning phase, we make our agents learn for a number of 5000 episodes, knowing that the agent does not always arrive at the last episode when the value of epsilon reaches its maximum ($epsilon = 1$) before that. The learning is done only once for 100 jobs for each case, and the double Q table obtained at the end of the learning will be used for the other sizes of workshops.

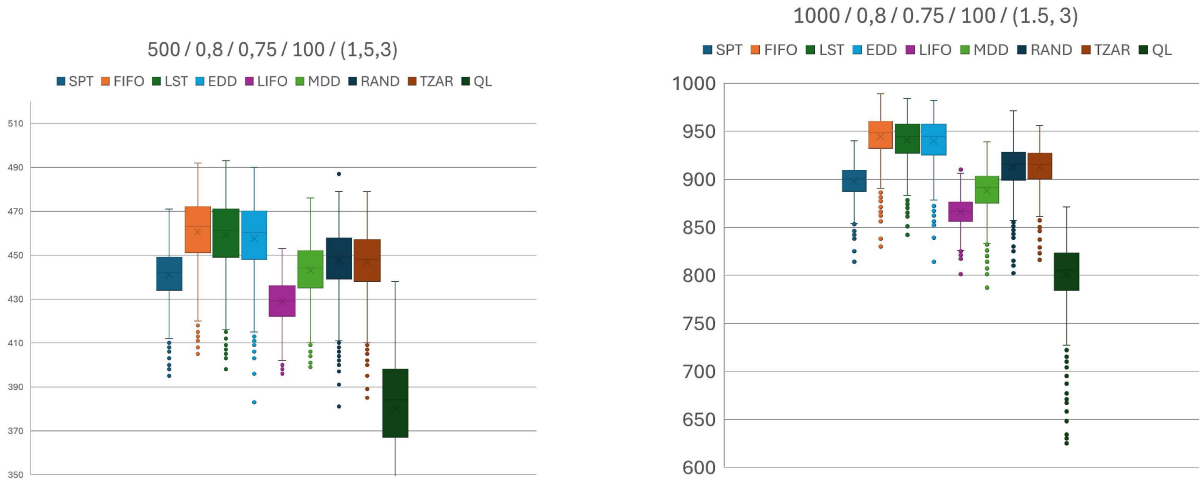

FIGURE 8. Box plot of tardy jobs of case 07 of two different workshop sizes (500 and 1000).

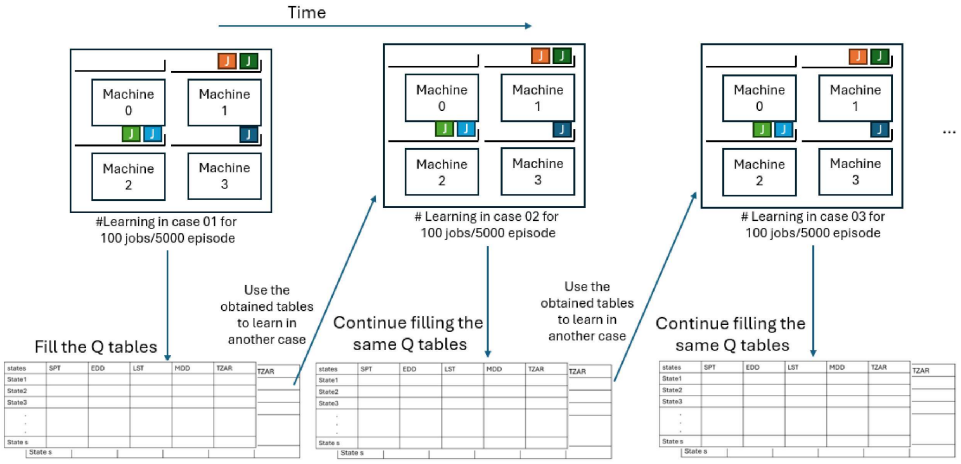

FIGURE 9. Learning process.

A Transfer learning (detailed in Sect. 3) was employed to share learned experiences across different cases. This method optimized learning efficiency by leveraging knowledge from previous cases to the current one. Our standardized state representation ensured that the states observed in one case could reappear in others, facilitating the reuse of accumulated experience. The idea of transfer learning is illustrated in Figure 9.

Since our dynamic workshop environment involves uncertain events, individual states may occur too infrequently and agents could not learn optimal policies for these states. Transfer learning addressed this by effectively increasing the frequency of state visits, improving the agents' ability to learn robust policies and make informed decisions. In addiction, agents may encounter entirely new states that have not been observed during training. Rather than resorting to random actions, they can extrapolate from similar states in past cases to select near-optimal actions.

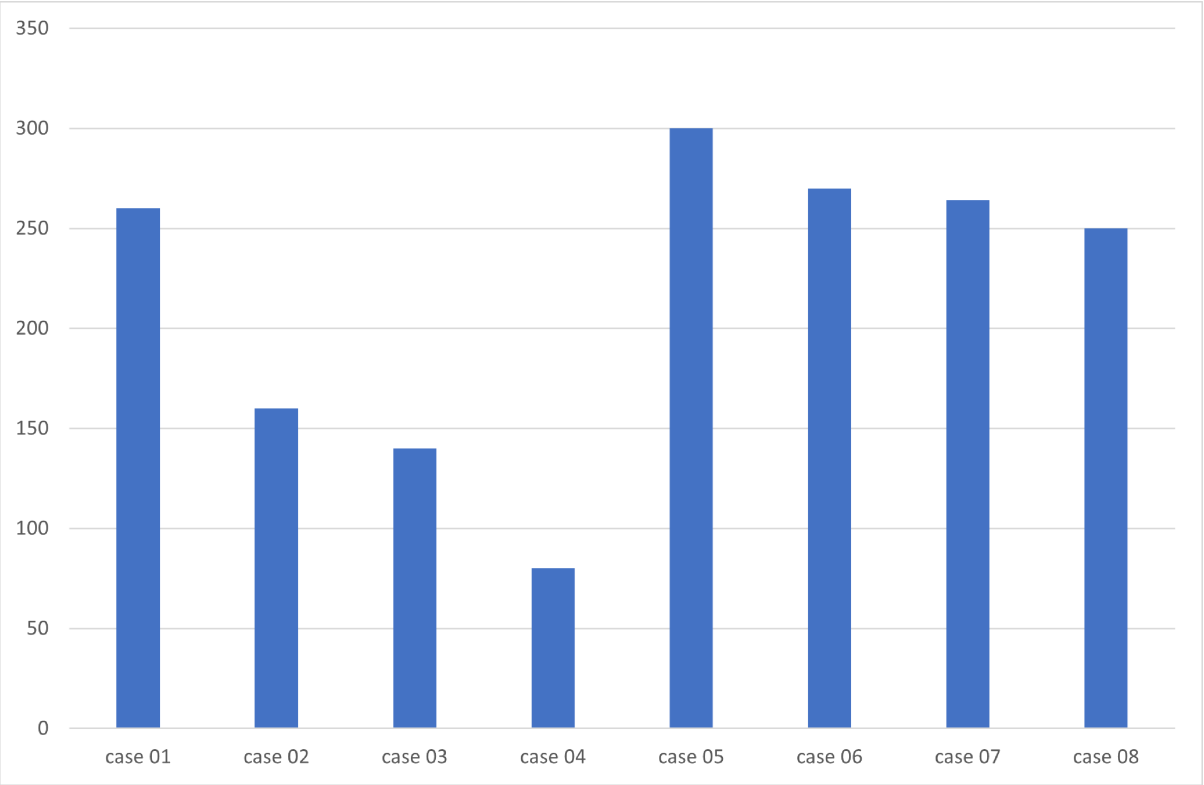

FIGURE 10. Learning time (minutes) from case 01 to case 08.

Although the idea of transferring Q tables in a tabular algorithm like Double-QL has not been widely explored in the literature, this contribution played a crucial role in the used approach by significantly reducing the learning time (Fig. 10).

To validate the robustness of the used Double-QL algorithm and the transfer learning approach used, we maintained identical parameters (Tab. 3) and test cases (Tab. 4) while increasing the machine utilization rate to 0.9 representing a tighter job arrival scenario. The resulting performance metrics are detailed in Table 9.

4.4. Computational time

Computation time serves as a critical metric for comparing methodologies that address the same problem. In this study, we evaluated a learning-based algorithm against traditional scheduling heuristics. The learning algorithm operates in two distinct phases: a learning phase, where the agent explores and exploits the environment (*e.g.*, via an epsilon-greedy policy), and a final phase, where the agent executes decisions based on its acquired knowledge without further exploration.

4.4.1. Learning time

This study utilizes a standardized state representation that is agnostic to workshop size, facilitating knowledge transfer across different operational scenarios. A key component of our methodology is the application of transfer learning, where the Q-table from a preceding case is used to initialize learning in the next. This approach prevents the computationally expensive process of learning from scratch for each new environment.

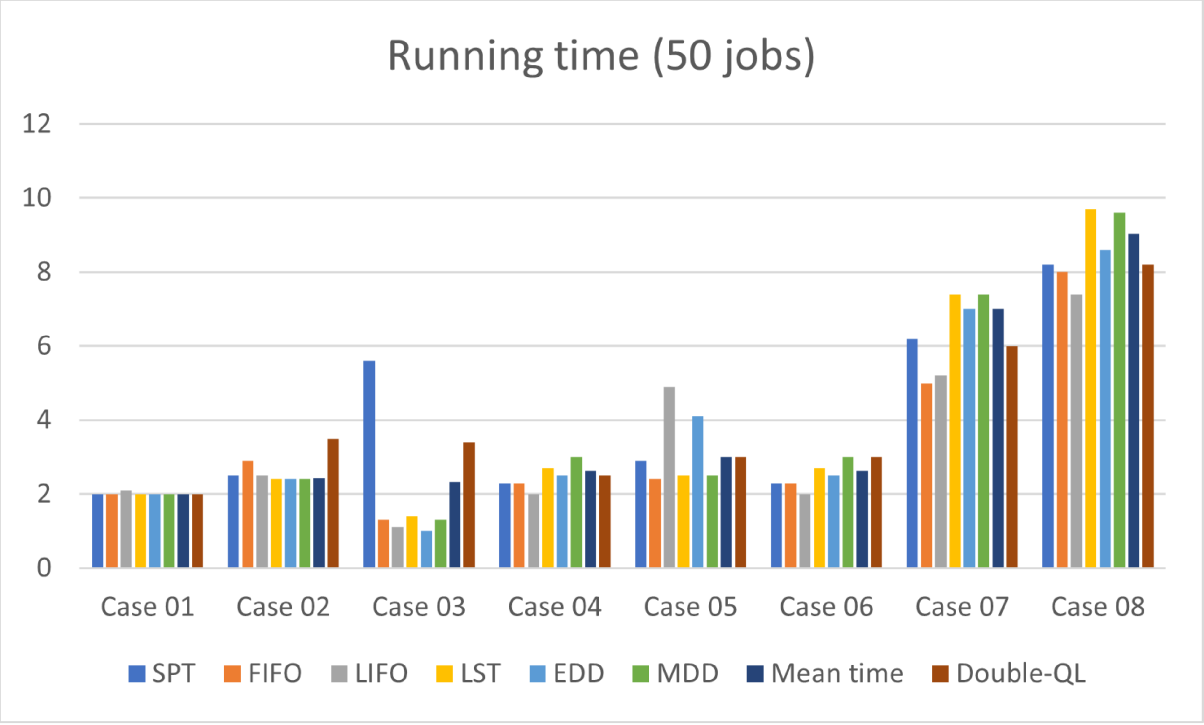

FIGURE 11. Running time (minutes) comparison from case 01 to case 08 for 50 jobs.

Figure 10, which shows the learning time per case for 500 jobs, demonstrates the effectiveness of this approach. The observed decrease in learning time between Cases 01 and 04 suggests that the agent successfully transmitted and improved its policy, reducing the need for additional research.

A notable deviation from this trend occurs in Case 05, where a significant increase in learning time coincides with the introduction of a longer machine repair time (MTTR = expo(100)). This new condition altered the system's dynamics, rendering a portion of the agent's prior experience suboptimal. Consequently, the agent required additional exploration to adapt its policy to these new constraints.

For the remaining cases (06 onward), learning times decreased but plateaued at a level higher than the initial baseline. This sustained elevation is a direct consequence of the increased MTTR, which prolongs job processing cycles and, by extension, the time required for the agent to accumulate a robust set of experiences within the environment.

4.4.2. Final phase computational time

We compared the execution time of our trained agent against the dispatching rules, which are known for their rapid computation. The agent's initial learning time was justifiably excluded from this comparison, as it is a one-time cost for building a policy designed to achieve higher-quality schedules during this decision-making phase.

The Figures 11–14 below represent the computation time for each job under the eight cases. We also calculated the average computation time for the heuristics, which were used as actions during the learning phase and subsequently in the final scheduling phase.

The results show that computation time varies significantly from one case to another, depending on dynamic workshop conditions such as: The frequency of machine breakdowns, the time required for repairs, the due date

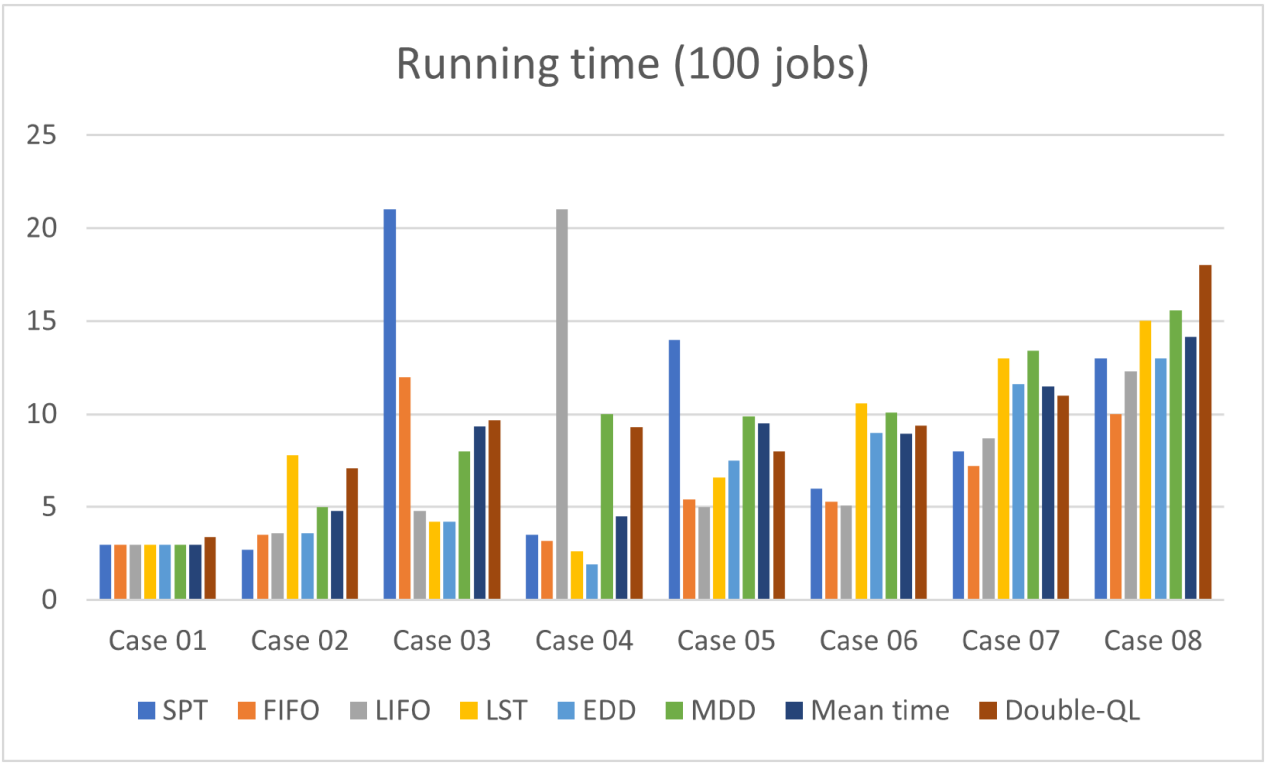

FIGURE 12. Running time (minutes) comparison from case 01 to case 08 for 100 jobs.

tightness factor. In most cases, the heuristics perform well in terms of computation time during this final phase. This efficiency is particularly evident for rules that require minimal calculation, such as FIFO and LIFO. The SPT heuristic also falls into this category, as it only requires examining the processing times of jobs in the queue. In contrast, more complex heuristics like MDD and LST involve active calculations based on the current time and remaining slack. For these rules, we observe a noticeable increase in computation time.

A direct comparison between a single, fixed heuristic and an algorithm like Double Q-Learning which dynamically selects from a portfolio of heuristics (multiple policies), would be methodologically unsound. The results show that the average processing time of the individual heuristics and the performance of the Double Q-Learning algorithm are comparable. This indicates that the Double Q-Learning agent effectively learned to emulate the collective performance of the heuristic set by dynamically selecting the most appropriate rule from its diverse action space for any given situation.

The increased computation time of the Double Q-Learning algorithm is a direct result of its decision-making mechanism. For every scheduling event, the algorithm must perform a "lookup" in its Q-table to identify the action with the highest Q-value for the current state. Although the computational cost of a single "lookup" for one state is minimal, this operation is executed for every state encountered during the job assignment process. The cumulative effect of these sequential operations results in a total computation time that is substantially greater than that of static dispatching rules.

4.5. Statistical tests (Analyse of variance)

Since the performance of scheduling heuristics can fluctuate due to stochastic simulation effects, it is important to determine whether the observed differences in average tardiness are significant. The one-way ANOVA test

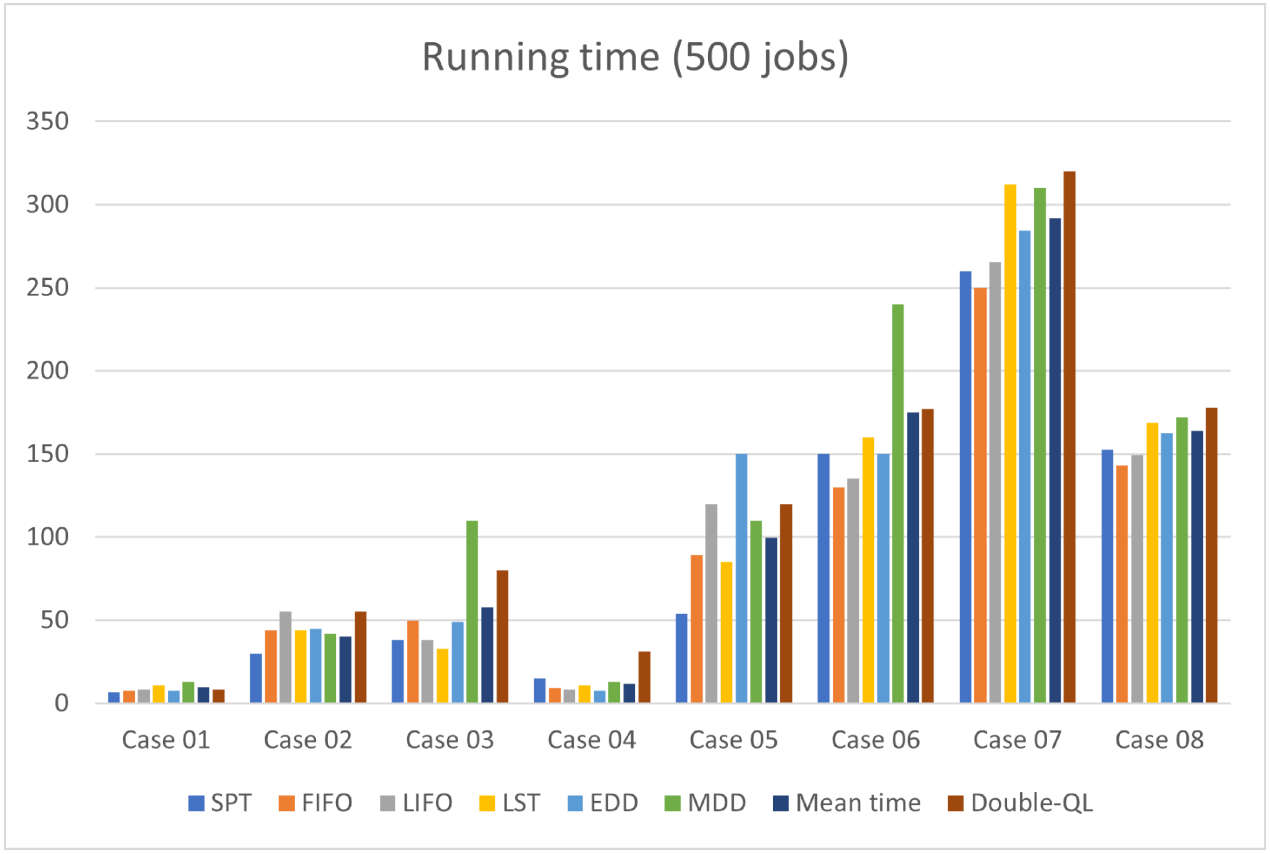

FIGURE 13. Running time (minutes) comparison from case 01 to case 08 for 500 jobs.

was conducted to determine whether there are statistically significant differences in the mean number of tardy jobs among the eight dispatching rules: SPT, FIFO, LST, EDD, LIFO, MDD, RAND, and QL. After calculating the source of variation, the sum of squares (SS), the degrees of freedom (df), the mean squared (MS), and the F-statistic, we arrived at the results shown in the following tables (Tabs. 5–8).

For all the presented tables, the results indicate a highly significant difference among the mean tardiness values of the eight dispatching rules. The computed **F-value (376.91/1280.99/564,5040385/3930,732172)** is far greater than the critical F-value (2.01), while the corresponding P-value is very close to zero. Hence, the null hypothesis of equal means is strongly rejected, confirming that the performance of the used methods is not equivalent.

From the ANOVA results, we can conclude that the Double-QL method differs significantly from the other dispatching rules. This difference can be explained by the Double-QL approach's intrinsic architecture, which incorporates several decision-making techniques into a RL framework rather than depending on a single static dispatching policy.

Every action in the Double-QL method relates to a different dispatching rule that is used for choices about job assignments in the queue, combines the strengths of several traditional heuristics, such as prioritizing short processing times, due dates, or slack times.

The Double-QL agent successfully adjusts its scheduling policy over time by learning to choose the best rule for every system state through constant contact with the simulated environment.

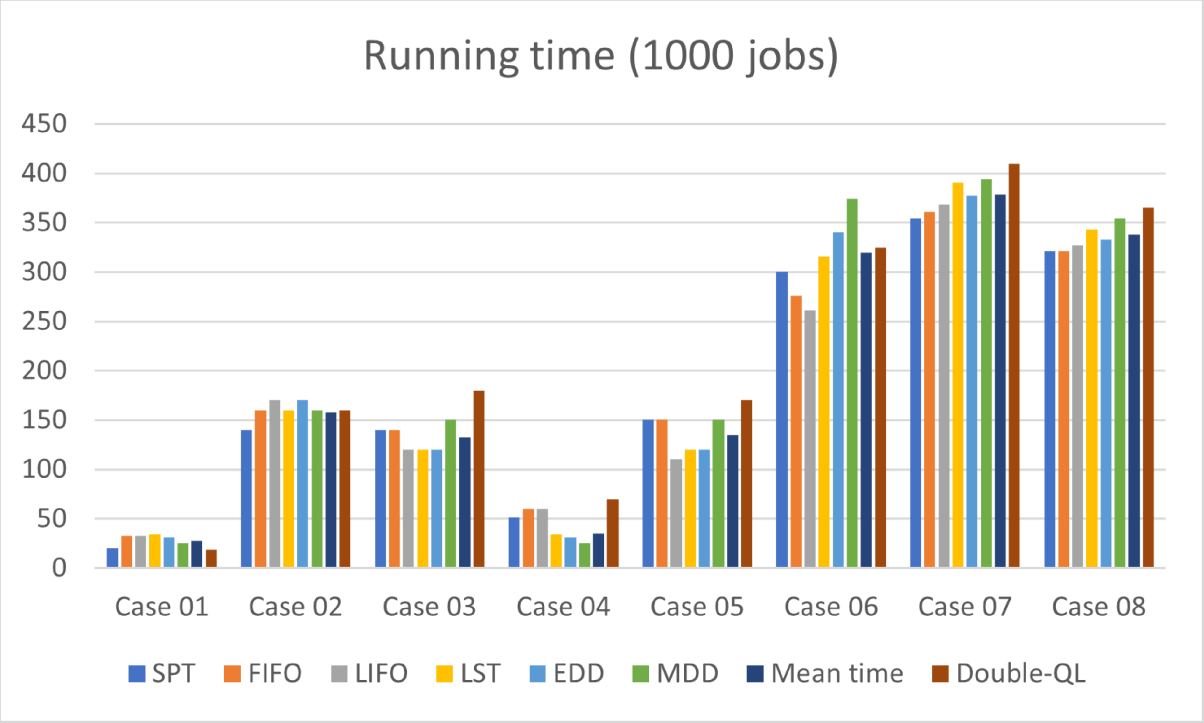

FIGURE 14. Running time (minutes) comparison from case 01 to case 08 for 1000 jobs.

TABLE 5. ANOVA results for 50 jobs case 02.

Source of variation	SS	df	MS	F	P-value	F crit
Between Groups	94054,9575	7	13436,4225	376,9054142	<0.0001	2,01073174
Within Groups	284909,382	7992	35,64932207			
Total	378964,3395	7999				

TABLE 6. ANOVA results for 100 jobs case 02.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	618393,1175	7	88341,87393	1280,98586	<0.0001	2,01073174
Within Groups	551160,07	7992	68,96397272			
Total	1169553,187	7999				

This dynamic and context-sensitive learning mechanism allows Double-QL to make more balanced and globally optimal decisions, which explains its superior performance and the statistically significant differences observed in the ANOVA results.

TABLE 7. ANOVA results for 500 jobs case 07.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1011474,22	7	144496,3171	564,5040385	<0.0001	2,01073174
Within Groups	2045715,332	7992	255,9703869			
Total	3057189,552	7999				

TABLE 8. ANOVA results for 1000 jobs case 02.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	31981534,28	7	4568790,611	3930,732172	<0.0001	2,01073174
Within Groups	9289306,156	7992	1162,325595			
Total	41270840,44	7999				

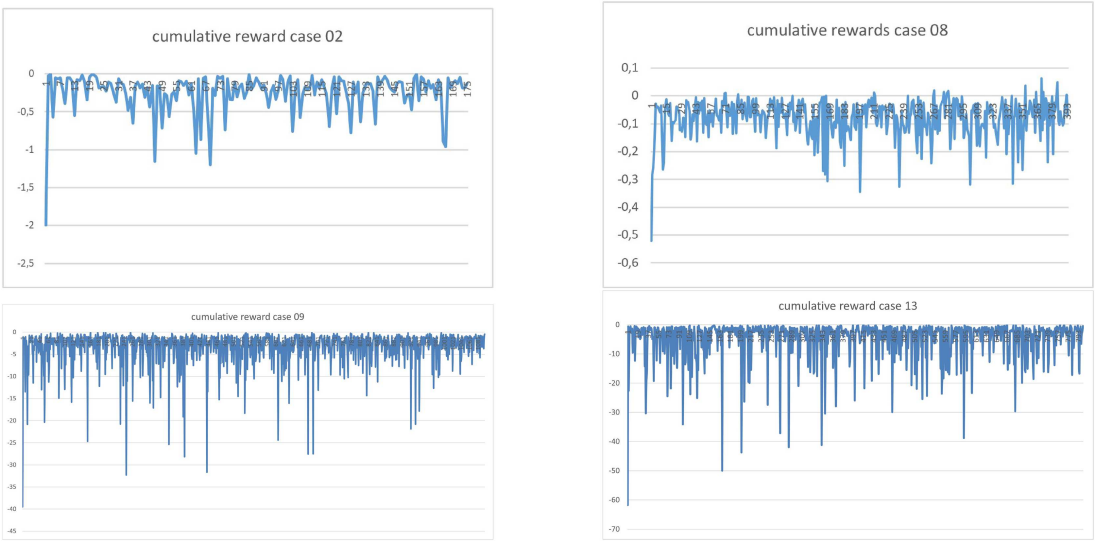

FIGURE 15. Reward convergence.

4.6. Reward convergence

As we have already mentioned in the previous sections, the reward is related to the objective function that represents the number of delays. In order to see an efficiency of learning. We have used the principle of exploitation and exploration. The exploitation phase that occurs at the beginning of learning to discover the environment and the exploitation phase that serves to choose the actions with a high reward because they represent the best actions in the current state. In this case, the cumulative rewards at the end of the learning will be the highest. In a dynamic workshop like the one we have modeled, having a high cumulative reward at the end of learning is challenging. Because, even if we are in the exploitation phase, there is a chance that we will encounter a new state. In order to increase the number of state visits and reduce the encounter of states that have never been seen, we have in this instance relied much more on the previously discussed concept of state transfers. The convergence of cumulative reward during the learning phase in two different scenarios is depicted in Figure 15. Given that the epsilon plays an important role in the exploration and exploitation policy (the $\epsilon - greedy$ algorithm), we calculated the cumulative reward values for various epsilon values (Tab. 9).

TABLE 9. Mean tardy jobs and standard deviation of a set of 1000 instances for each number of jobs with the parameters: (Machine utilization rate = 0.9/w → due date factor/μ → failure factor/MTTR → Mean Time To Repair).

Jobs	SPT		FIFO		LIFO		LST		EDD		MDD		RAND		Double-QL	
	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV	TARDY	SDEV
Case 09 → w = uniform (1,5; 3)/μ = 0,5/MTTR = 50																
50	41,14	2,94	42,27	2,86	41,18	2,69	41,79	3,14	41,48	3,08	40,97	2,89	41,06	2,96	**39,70**	3,54
100	84,84	4,82	88,32	4,88	83,84	4,35	87,63	5,27	87,04	5,62	85,12	4,89	85,47	4,92	**83,70**	5,10
500	454,74	10,16	477,09	10,83	**438,46**	10,32	476,17	11,68	474,98	11,71	457,57	10,66	463,24	11,66	462,86	12,92
1000	921,27	13,29	947,11	19,07	**771,03**	19,60	939,24	22,51	940,18	21,32	874,62	20,99	873,01	21,14	841,60	26,96
Case 10 → w = uniform (4,5; 6)/μ = 0,5/MTTR = 50																
50	34,90	5,43	37,25	4,70	35,57	5,06	36,21	5,61	35,82	,30	35,56	4,87	34,83	6,14	**31,36**	6,27
100	72,98	6,98	79,77	7,04	73,58	6,43	78,33	7,85	77,59	9,19	74,60	7,69	74,24	7,70	**70,28**	8,07
397,80	17,56	459,37	16,41	**385,25**	13,38	454,19	19,58	453,43	19,13	424,79	16,20	424,49	18,65	407,40	25,41	
810,16	22,34	949,18	21,67	**778,68**	18,89	943,67	21,97	942,96	22,69	853,13	22,47	881,32	22,64	784,41	34,78	
Case 11 → w = uniform (1,5; 3)/μ = 0,75/MTTR = 50																
50	41,50	2,70	43,01	2,89	41,61	2,64	42,54	3,13	42,17	3,03	41,54	2,94	41,63	2,99	**40,50**	3.05
100	85,37	4,43	88,43	4,97	84,61	3,71	87,78	5,22	87,53	5,23	85,65	4,54	85,71	4,54	**84,24**	4,94
500	456,00	9,62	477,23	11,13	440,23	8,76	476,20	11,49	476,07	11,20	457,63	11,28	463,83	11,19	**412,44**	23,79
1000	930,43	11,92	974,11	12,14	892,19	11,84	973,02	12,76	970,51	14,28	936,90	13,61	949,32	13,87	**872,66**	32,01
Case 12 → w = uniform (4,5; 6)/μ = 0,75/MTTR = 50																
50	36,07	3,48	38,40	3,08	36,74	3,35	37,33	3,78	37,15	3,63	36,59	3,69	36,41	3,47	**33,61**	4,25
100	**73,85**	5,08	80,52	5,59	74,01	4,40	78,81	5,95	78,71	6,36	75,88	5,60	75,30	5,13	74,06	6,19
500	402,39	11,07	460,64	16,47	388,29	9,89	454,32	18,93	454,04	18,58	426,48	15,01	428,15	15,46	**370,13**	23,07
1000	825,68	14,83	953,40	19,53	788,27	14,24	946,30	22,91	946,68	22,17	884,85	19,95	892,02	20,53	**784,41**	34,80
Case 13 → w = uniform (1,5; 3)/μ = 0,5/MTTR = 100																
50	38,68	4,54	40,13	3,95	39,20	3,85	39,66	3,93	37,18	4,22	37,26	4,22	38,58	4,26	**27,22**	5,36
100	80,46	6,46	83,98	5,85	80,60	5,93	83,28	5,77	81,86	5,90	79,01	5,44	81,41	5,41	**62,64**	9,65
500	439,21	13,20	460,88	16,76	427,47	13,00	457,77	18,96	456,77	18,86	434,12	14,26	444,86	16,42	**388,99**	30,15
1000	904,78	18,93	953,61	21,63	873,02	17,13	949,80	22,15	946,68	24,20	897,04	21,49	922,21	22,00	**807,87**	39,60
Case 14 → w = uniform (4,5; 6)/μ = 0,5/MTTR = 100																
50	31,49	7,95	33,71	6,66	32,58	6,20	31,87	8,88	31,79	8,62	30,05	7,19	30,05	10,17	**22,20**	5,27
100	68,84	12,61	75,41	8,46	69,58	10,08	72,23	11,61	71,77	11,32	67,50	10,73	70,01	10,85	**54,86**	8,64
500	392,02	20,61	442,03	23,38	383,61	16,59	434,48	26,51	433,65	28,88	409,28	24,39	410,98	24,28	**365,53**	27,10
1000	801,49	33,78	923,37	32,46	**777,33**	23,15	914,58	34,57	911,27	37,32	836,84	28,67	856,10	30,76	799,60	39,12
Case 15 → w = uniform (1,5; 3)/μ = 0,75/MTTR = 100																
50	39,79	3,00	40,61	3,34	40,09	2,79	39,96	3,43	39,60	3,49	38,48	3,03	39,39	3,23	**28,53**	5,02
100	81,95	4,48	84,96	5,09	82,42	4,09	83,78	5,31	82,81	5,38	79,75	4,58	82,22	5,10	**63,03**	9,71
500	440,73	13,13	462,09	16,27	429,99	10,19	459,18	17,80	455,96	19,14	434,56	14,19	445,53	16,12	**394,43**	28,65
1000	913,90	17,91	951,77	22,31	876,57	14,39	948,86	22,50	945,01	24,52	898,34	20,99	929,56	21,61	**841,33**	39,39
Case 16 → w = uniform (4,5; 6)/μ = 0,75/MTTR = 100																
50	33,59	4,14	35,58	4,10	34,44	3,27	34,32	4,90	34,48	4,83	31,87	4,08	33,53	4,63	**23,16**	4,41
100	72,66	5,50	77,11	5,80	73,01	5,08	74,92	6,41	74,45	6,35	70,63	5,58	72,82	5,62	**53,13**	8,55
500	398,10	12,42	443,78	21,70	387,40	10,36	436,95	24,25	435,66	25,10	412,96	18,63	413,64	19,15	**347,42**	29,23
1000	812,77	16,78	923,00	31,18	782,96	15,22	914,90	33,58	913,06	35,05	838,98	25,84	857,85	29,33	**750,82**	46,37

We can notice fluctuations in the cumulative reward that start to decrease as the epsilon approaches 1 (at the end of learning). Noting that the cumulative reward is calculated during a learning episode, and in this episode the agent can be blocked in a state because of a machine failure. and we have specified in the function that the agent will have a penalty in the case of the failure. This fluctuation is found a lot in the learning phase, where we have cases of high frequency of failure and duration of repairs.

5. Conclusion

In this work, We have introduced a Double Q-Learning algorithm to solve a Dynamic Job Shop Scheduling Problem. By deploying a multi-agent system, each machine agent is capable of making localized decisions while incorporating global information shared by the job agent. This framework enhances adaptability to dynamic environments, particularly in scenarios involving unpredictable events such as machine failures. Our primary contribution lies in the new state representation, which focuses on critical information related to tardiness. This presentation is distinguished by its adaptability to workshops of any size. In addition, the reward mechanism was carefully designed to guide the agent in making effective decisions in both operational and failure mode. This strategy not only maximizes rewards but also optimizes the number of tardy jobs, ensuring efficient task execution even when parameters are subject to change.

Another approach that helped us optimize learning time using standard state presentation is to use the Q tables from one case to another using a transfer of learning, in order to benefit from the experiences of a previous case in a current case, avoiding learning again. This method is not often used in the literature for tabular algorithms.

This approach has obtained satisfactory results for a dynamic workshop in which there are uncertainties, with the emergence of new states during the final phase. It is not excluded at all, but these new states could have coincided in a previous case, so we will make the decision immediately. We have presented 64 possible situations which are close to the real case in order to verify the robustness of the used approach. The findings underscore the robustness and adaptability of our model.

However, there are still areas for enhance. One major area is improving the model's alignment with different workshop layout designs by increasing the number of machines in the system to higher quantities, which will more accurately represent real industrial settings. Another important perspective is benchmarking The Double Q-Learning Approach against exact methods on static instances. It represents a critical validation step. Through systematic comparison of the agent's solutions with established optimal benchmarks, we can rigorously quantify the inherent quality and efficiency of the learned policy, thereby establishing a robust foundation for performance evaluation. Furthermore, we intend to test more advanced RL methods PPO and DQN to see how well they operate under these differing conditions in relation to our Double Q-learning algorithm.

Data availability statement

The examples of Data and the code generating the Data used in this paper is available online in a Github repository https://github.com/abirmanal27/Dynamic-job-shop-instances [57].

References

[1] J.P. Usuga Cadavid, S. Lamouri, B. Grabot, R. Pellerin and A. Fortin, Machine learning applied in production planning and control: a state-of-the-art in the era of industry 4.0. *J. Intell. Manuf.* **31** (2020) 1531–1558.

[2] M.L. Pinedo, Scheduling, 6th edition. Springer, Cham (2022).

[3] Y.M. Jiménez, *A generic multi-agent reinforcement learning approach for scheduling problems*. Ph.D. thesis, Vrije Universiteit Brussel 128:11 (2012).

[4] Z. Yahouni, Le Meilleur Des Cas Pour l'ordonnancement de Groupes: Un Nouvel Indicateur Proactif-Réactif Pour l'ordonnancement Sous Incertitudes. Thesis, École centrale de Nantes; Université Abou Bekr Belkaid (Tlemcen, Algérie) (2017).

[5] Z. Yahouni, N. Mebarki and F. Belkadi, Human-machine cooperation in planning and scheduling: a case study on an unstable environment. *Eur. J. Ind. Eng.* **12** (2018).
[6] W. Zhang, Reinforcement Learning for Job Shop Scheduling. Oregon State University (1996).
[7] C. George and S. Velusamy, Dynamic scheduling of manufacturing job shops using genetic algorithms. *J. Intell. Manuf.* **12** (2001) 281–293.
[8] J.P. Watson, L.D. Whitley and A.E. Howe, A dynamic model of Tabu search for the job-shop scheduling problem (2005). www.ms.ic.ac.uk/info.html.
[9] C. Schmidl, T.D. Simão and N. Jansen, A supervised learning approach to robust reinforcement learning for job shop scheduling, in International Conference on Agents and Artificial Intelligence. Science and Technology Publications, Lda (2024) 1324–1335.
[10] V. Thiagarajan, R. Rajkanth, C. Rajendran and T.N. Srikantha Dath, Machine learning algorithm to solve dynamic jobshop problem to deal with digitalization of manufacturing process – a real-world case study, in Recent Advances in Industrial and Systems Engineering, edited by S.G. Ponnambalam, P. Damodaran, N. Subramanian and J. Paulo Davim. Springer Nature Singapore, Singapore (2024) 225–235.
[11] M. Belmamoune, L. Ghomri and Z. Yahouni, Solving a job shop scheduling problem using QLearning algorithm, in International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing (2023) 196–209.
[12] Z. Yejian, W. Yanhong, T. Yuanyuan, Z. Jun and Y. Hongxia, Dynamic jobshop scheduling algorithm based on deep Q network. *IEEE Access* **9** (2021) 122995–123011.
[13] B.M. Ombuki and M. Ventresca, Local search genetic algorithms for the job shop scheduling problem. *Appl. Intell.* **21** (2004) 99–109.
[14] B.J. Park, H.R. Choi and H.S. Kim, A hybrid genetic algorithm for the job shop scheduling problems. *Comput. Ind. Eng.* **45** (2003) 597–613.
[15] L. Asadzadeh, A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Comput. Ind. Eng.* **85** (2015) 376–383.
[16] B.A. Han and J.-J. Yang, Research on adaptive job shop scheduling problems based on dueling double DQN. *IEEE Access* **8** (2020) 186474–186495.
[17] P. Tassel, M. Gebser and K. Schekotihin, A reinforcement learning environment for job-shop scheduling. Preprint arXiv:2104.03760 (2021).
[18] L. Wang, X. Hu, Y. Wang, S. Xu, S. Ma, K. Yang, Z. Liu and W. Wang, Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Comput. Netw.* **190** (2021).
[19] P. Burggräf, J. Wagner, T. Saßmannshausen, D. Ohrndorf and K. Subramani, Multi-agent-based deep reinforcement learning for dynamic flexible job shop scheduling. *Procedia CIRP* **112** (2022) 57–62.
[20] M. Boukedroun, D. Duvivier, A. Ait-el-Cadi, V. Poirriez and M. Abbas, A hybrid genetic algorithm for stochastic job-shop scheduling problems. *RAIRO-Oper. Res.* **57** (2023) 1617–1645.
[21] S. Lawrence, Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement) (1984).
[22] J. Shahrabi, M.A. Adibi and M. Mahootchi, A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Comput. Ind. Eng.* **110** (2017) 75–82.
[23] A.D. Workneh and M. Gmira, Deep Q network method for dynamic job shop scheduling problem, in International Conference on Artificial Intelligence & Industrial Applications (2023) 137–155.
[24] L. Canese, G.C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re and S. Spanò, Multi-agent reinforcement learning: a review of challenges and applications. *Appl. Sci.* **11** (2021) 4948.
[25] R. Liu, R. Piplani and C. Toro, A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem. *Comput. Oper. Res.* **159** (2023) 106294.
[26] Y. Zhang, H. Zhu, D. Tang, T. Zhou and Y. Gui, Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems. *Robot. Comput. Integr. Manuf.* **78** (2022) 102412.
[27] Z. Wang and W. Liao, Smart scheduling of dynamic job-shop based on discrete event simulation and deep reinforcement learning. *J. Intell. Manuf.* **35** (2024) 2593–2610.
[28] W. Chen, Z. Zhang, D. Tang, C. Liu, Y. Gui, Q. Nie and Z. Zhao, Probing an LSTM-PPO-based reinforcement learning algorithm to solve dynamic job-shop scheduling problem. *Comput. Ind. Eng.* **197** (2024) 110633.
[29] X. Wu, X. Yan, D. Guan and M. Wei, A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time. *Eng. Appl. Artif. Intell.* **131** (2024).
[30] V. Samsonov, M. Kemmerling, M. Paegert, D. Lütticke, F. Sauermann, A. Gützlaff, G. Schuh and T. Meisen, Manufacturing control in job shop environments with reinforcement learning, in Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) (2021) 589–597.

[31] Z. Yang, L. Bi and X. Jiao, Combining reinforcement learning algorithms with graph networks to solve dynamic job shop scheduling problems. *Processes* **11** (2023).
[32] M. Zhang, Y. Lu, Y. Hu, N. Amaitik and Y. Xu, Dynamic scheduling method for job-shop manufacturing systems by deep reinforcement learning with proximal policy optimization. *Sustainability* **14** (2022).
[33] Y. Pu, F. Li and S. Rahimifard, Multi-agent reinforcement learning for job shop scheduling in dynamic environments. Sustainability **16** (2024).
[34] Y. Zeng, Z. Liao, Y. Dai, R. Wang, X. Li and B. Yuan, Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism (2022).
[35] L. Song, Y. Li and J. Xu, Dynamic job-shop scheduling based on transformer and deep reinforcement learning. *Processes* **11** (2023) 3434.
[36] D. Lakshmipathy, M. Chandrasekaran, T. Balamurugan and P. Sriramya, A new GT heuristic for solving multi objective job shop scheduling problems. *AMM* (2014).
[37] M. Yousefi, M. Yousefi, D. Hooshyar and J.A. de Oliveira, An evolutionary approach for solving the job shop scheduling problem in a service industry. *Int. J. Adv. Intell. Inform.* (2015).
[38] M.R. Garey, D.S. Johnson and R. Sethi, The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1** (1976) 117–129.
[39] G. Mosheiov, Complexity analysis of job-shop scheduling with deteriorating jobs. *Discrete Appl. Math.* **117** (2002) 195–209.
[40] Y.N. Sotskov and N.V. Shakhlevich, NP-hardness of shop-scheduling problems with three jobs. *Discrete Appl. Math.* **59** (1995) 237–266.
[41] J. Du and J.Y.-T. Leung, Minimizing the number of late jobs on unrelated machines. *Oper. Res. Lett.* **10** (1991) 153–158.
[42] M. Chadi and H. Mousannif, Understanding reinforcement learning algorithms: the progress from basic Q-learning to proximal policy optimization. Preprint arXiv:2304.00026 (2023).
[43] C.J.C.H. Watkins and P. Dayan, Q-learning. *Mach. Learn.* **8** (1992) 279–292.
[44] M. Coggan, Exploration and exploitation in reinforcement learning. Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University (2004).
[45] O. Gies, B. Chaib-draa and É. Damas, Apprentissage de la coordination multiagent: Q-learning par jeu adaptatif.
[46] H. Dong, H. Dong, Z. Ding, S. Zhang and T. Chang, Deep Reinforcement Learning. Springer, Singapore (2020).
[47] Z. Zhang, Z. Pan and M.J. Kochenderfer, Weighted double Q-learning, in International Joint Conferences on Artificial Intelligence IJCAI (2017) 3455–3461.
[48] Y. Zhang, P. Sun, Y. Yin, L. Lin and X. Wang, Human-like autonomous vehicle speed control by deep reinforcement learning with double Q-learning, in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE (2018) 1251–1256.
[49] X. Chen, C. Wang, Z. Zhou and K. Ross, Randomized ensembled double q-learning: learning fast without a model. Preprint arXiv:2101.05982 (2021).
[50] H. Hasselt, Double Q-learning. *Adv. Neural Inf. Process Syst.* **23** (2010).
[51] M.E. Taylor, N.K. Jong and P. Stone, Transferring instances for model-based reinforcement learning, in Joint European Conference on Machine Learning and Knowledge Discovery in Databases (2008) 488–505.
[52] L.A. Celiberto Jr, J.P. Matsuura, R.L. De Màntaras and R.A. Bianchi, Using transfer learning to speed-up reinforcement learning: a cased-based approach, in 2010 Latin American Robotics Symposium and Intelligent Robotics Meeting (2010) 55–60.
[53] M.E. Taylor and P. Stone, Transfer learning for reinforcement learning domains: a survey. *J. Mach. Learn. Res.* **10** (2009).
[54] F.L. Da Silva and A.H.R. Costa, A survey on transfer learning for multiagent reinforcement learning systems. *J. Artif. Intell. Res.* **64** (2019) 645–703.
[55] N. Mebarki and A. Shahzad, Correlation among tardiness-based measures for scheduling using priority dispatching rules. *Int. J. Prod. Res.* **51** (2013) 3688–3697.
[56] N. Mebarki, *Une Approche d'ordonnancement Temps Réel Basée Sur La Sélection Dynamique de Règles de Priorité*. Ph.D. thesis (1995). http://www.theses.fr/1995LYO10043.
[57] M. Belmamoune, L. Ghomri and Z. Yahouni, PYTHON code for generating instances for: Tardy jobs minimization in a dynamic job-shop environment using double q-learning algorithm. https://github.com/abirmanal27/Dynamic-job-shop-instances/tree/main.