POLITECNICO DI TORINO
Repository ISTITUZIONALE

Online Single-Machine Scheduling via Reinforcement Learning

Original
Online Single-Machine Scheduling via Reinforcement Learning / Li, Yuanyuan; Fadda, Edoardo; Manerba, Daniele; Roohnavazfar, Mina; Tadei, Roberto; Terzo, Olivier. - ELETTRONICO. - 952:(2022), pp. 103-122.

Availability:
This version is available at: 11583/2896776 since: 2021-04-28T17:44:51Z

Publisher:
Springer

Published
DOI:

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

16 July 2022

Online Single-Machine Scheduling via Reinforcement Learning

Yuanyuan Li and Edoardo Fadda and Daniele Manerba and Mina Roohnavazfar and Roberto Tadei and Olivier Terzo

Abstract Online scheduling has been an attractive field of research for over three decades. Some recent developments suggest that Reinforcement Learning (RL) techniques can effectively deal with online scheduling issues. Driven by an industrial application, in this paper we apply four of the most important RL techniques, namely *Q-learning*, *Sarsa*, *Watkins's Q(λ)*, and *Sarsa(λ)*, to the online single-machine scheduling problem. Our main goal is to provide insights into how such techniques perform in the scheduling process. We will consider the minimization of two different and widely used objective functions: the total tardiness and the total earliness and tardiness of the jobs. The computational experiments show that Watkins's Q(λ) performs best in minimizing the total tardiness. At the same time, it seems that the RL approaches are not very effective in minimizing the total earliness and tardiness over large time horizons.

Yuanyuan Li
ESSEC Business School, 95000 Cergy, France e-mail: yuanyuan.li@essec.edu

Edoardo Fadda
Politecnico di Torino, Department of Control and Computer Engineering, corso Duca degli Abruzzi 24, 10129 Torino, Italy e-mail: edoardo.fadda@polito.it

Deniele Manerba
Department of Information Engineering, University of Brescia, 25123 Brescia, Italy e-mail: daniele.manerba@unibs.it

Roberto Tadei
Politecnico di Torino, Department of Control and Computer Engineering, corso Duca degli Abruzzi 24, 10129 Torino, Italy e-mail: roberto.tadei@polito.it

Olivier Terzo
LINKS Foundation - Advanced Computing and Applications, via Pier Carlo Boggio 61, 10138 Torino, Italy e-mail: olivier.terzo@linksfoundation.com

1 Introduction

Production scheduling is one of the most important aspects to address in many manufacturing companies (see [2]). The optimization problems arising within production scheduling can be of *static* or *dynamic* type (see [16]). In contrast with the static case, in which specifications and requirements are fully and deterministically known in advance, in the dynamic one, additional information (e.g., new orders, changes of available resources) may arrive during the production process itself. In this paper, we will consider the latter case, commonly called *online scheduling*, mainly fostered by our experience on an industrial project (Plastic and Rubber 4.0[1]) in which frequent occurrences of unexpected events call for more dynamic and flexible scheduling (see [22]).

We will mainly focus on online single-machine scheduling problems with release dates, where preemption is allowed. Let us consider a set $\mathcal{J}$ of jobs that are released over time. As soon as a job arrives, it is added to the end of a waiting queue. For each job $j \in \mathcal{J}$, let d_j be its due date and c_j its completion time. A job is early if its completion time is shorter than its due date. On the contrary, a job is tardy if its completion time is larger than its due date. When the completion time is equal to the due date, the job is on time. The goal of the problem is to arrange the queue's jobs to minimize a specific objective function. In this work, we will consider the minimization of two different objective functions: total tardiness ($\Gamma 1$) and the total earliness and tardiness ($\Gamma 2$) of the jobs. The two objectives are calculated as:

- $\Gamma 1 = \sum_{j \in \mathcal{J}} T_j$,
- $\Gamma 2 = \sum_{j \in \mathcal{J}} (E_j + T_j)$,

where T_j and E_j represent the tardiness and the earliness, respectively, and are computed as $T_j := \max\{0, c_j - d_j\}$ and $E_j := \max\{0, d_j - c_j\}$. They are among the most widely used objectives in scheduling, focusing on meeting jobs due dates. In particular, the minimization of the second objective characterizes the Just-In-Time principle in production.

The motivation to study a single-machine problem relies on the fact that, in plastic and rubber manufacturing, transforming raw material into a final product goes through one or two machines. On the other hand, even those manufacturing require multiple-machine scheduling problems. Each machine represents a chain's primary block. Thus improper usage of a machine can slow down the whole production process.

The easiest way to deal with scheduling in a dynamic context is by using the so-called *dispatching rules*. These rules prioritize jobs waiting for being processed and then select the job with a greedy evaluation whenever a machine gets free (see Section 2 for more details). While most dispatching rules schedule on a local view basis, other smarter approaches can provide better results in the long run. For instance, Reinforcement Learning (RL) is a continuing and goal-directed learning

[1] Plastic&Rubber 4.0. Piattaforma Tecnologica per la Fabbrica Intelligente (Technological Platform for Smart Factory), URL: `https://www.openplast.it/en/homepage-en/`

paradigm, and it represents a promising approach to deal with online scheduling. The potential of RL on online scheduling has been revealed in several works (see, e.g., [13], [31], [39]). However, while most works compare a single RL algorithm with commonly-used dispatching rules, they do not compare different RL algorithms. A research question naturally arises: how do different RL algorithms perform on online scheduling?

Motivated by investigating the applicability of RL algorithms on online single-machine scheduling in detail, in this work, we will compare the following approaches' performance:

- a random assignment (*Random*) which simply selects a job randomly;
- one of the most popular dispatching rules, namely the *earliest due date* (*EDD*) rule;
- four RL approaches, namely *Q-learning*, *Sarsa*, *Watkins's Q(λ)*, and *Sarsa(λ)*.

Furthermore, we will test the algorithms under different operating conditions (e.g., the frequency of job arrivals).

Therefore, we contribute the literature on two different aspects: getting insights on the compared methods and giving practitioners suggestions on selecting the best method against the specific situation. Notice that comparing and evaluating different algorithms against various aspects and performance indicators is a commonly adopted research methodology (see, e.g., [3], [6], [7], [8], [9], [10], and [14]). The specific comparison of RL algorithms can be found, for instance, in the game field. In [35], the authors compared two RL algorithms (*Q-learning* and *Sarsa*) through the simulation of bargaining games. Even though the two algorithms present slight differences, they might have essentially different simulation results, as reflected in our experiment (see Section 4).

Finally, we also propose some preliminary results obtained by the use of *Deep Q Network* (*DQN*), which utilizes the power of neural networks to approximate the value function (see [25] for a review about DQN). However, our experiments will show that *DQN* is better suited for high-dimensional inputs. In contrast, with smaller input settings, *DQN* has a longer training time and obtains results that are far from the performance of *Watkins's Q(λ)*.

The rest of the paper is organized as follows. Section 2 is dedicated to a general overview of RL techniques, while Section 3 introduces and reviews some previous works using RL approaches on scheduling problems. Section 4 describes the algorithmic framework for the online single-machine problem. Section 5 defines the simulation procedure, and the simulation results from three different types of experiments (Section 6). Finally, in Section 7, the paper concludes with a summary of the findings and some future lines.

2 Reinforcement Learning

RL is a branch of Machine Learning that improves automatically through experience. It comes from three main research branches: the first relates to learning by trial-and-error, the second relates to optimal control problems, and the last links to temporal-difference methods (see [33]). The three approaches converged together in the late eighties to produce the modern RL.

RL approaches can be applied to scenarios in which a decision-maker called *agent* interacts with a set of *states* called *environment* by means of a set of possible *actions*. A *reward* is given to the agent in each specific state. In this paper, we consider a discrete time system, i.e. defined over a finite set $\mathcal{T}$ of time steps with its cardinality being called *time horizon*. As shown in Figure 1, at each time step $t \in \mathcal{T}$, an agent in state S_t takes action A_t, then, the environment reacts by changing into state S_{t+1} and by rewarding the agent of R_{t+1}. The interaction starts from an initial state, and it continues until the end of the time horizon. Such a sequence of actions is named an *episode*. In the following, $\mathcal{E}$ will represent the set of episodes.

Each *state* of the system is associated with a *value function* that estimates the expected future reward achievable from that state. Each state-action pair (S_t, A_t) is associated with a so-called *Q-function* $Q(S_t, A_t)$ that measures the future reward achievable by implementing action A_t in state S_t. The agent's goal is to find the best *policy*, which is a function mapping the set of states to the set of actions, maximizing the cumulative *reward*. If exact knowledge of the Q-function is available, the best policy for each state is defined by $\max_a Q(S_t, a)$.

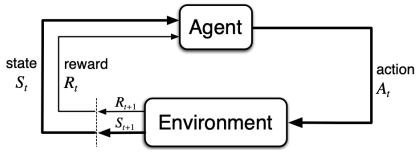

Fig. 1 The agent-environment interaction in RL [33].

To estimate the value functions $Q(s, a)$ and discover the optimal policies, three main classes of RL techniques exist Monte Carlo (MC)-based, Dynamic Programming (DP)-based methods, and temporal-difference (TD)-based methods. Unlike DP-based methods, which require complete knowledge of all the possible transitions, MC-based techniques only require some experience and the possibility to sample from the environment randomly. TD-based methods are a sort of combination of MC-based and DP-based ones: they sample from the environment like in MC-based methods and perform updates based on current estimates like DP-based ones. Moreover, TD-based techniques are also appreciated for being flexible, easy to implement, and computationally fast. For these reasons, in this paper, we will consider only RL algorithms belonging to the TD-based methods. Even if several TD-based RL algorithms have been introduced in the literature, the most used

are *Sarsa* (an acronym for State-Action-Reward-State-Action), *Q-learning* and their variations, e.g. the *Watkins's Q(λ)* method and the *Sarsa(λ)* (see [36]).

3 Literature Review

Since the first research on scheduling problem was performed in the mid-1950s, many articles have been published in the literature, considering different problem variants and solution approaches.

The manufacturing industries sometimes include a machine bottleneck, which affects, in some cases, all the jobs. Studies on single machine scheduling problems have been gaining importance for a long time since this bottleneck's management is crucial. The excellent surveys by Pinedo [28], Adamu and Adewumi [1], and the work proposed by Leksakul and Techanitisawad [18] have detailed the literature on the theory and applications about this problem in the past several decades.

In the manufacturing environment, various objectives can be considered to use the resources and provide good customer service efficiently. Scheduling against due dates has received considerable attention to meet principles like Lean Management, Just-in-Time, Simultaneous Engineering, etc. For example, the Just-in-Time principle states that jobs are expected to be on time since both late and early processing may negatively influence the manufacturing costs. While late processing does not meet customer expectations, early processing increases inventory costs and causes possible wastes since some products have a limited lifetime. One of the pioneers addressing minimizing the sum of earliness and tardiness (also referred to as the sum of deviations from a common due date) was [20]. Ying [38] addressed a single-machine problem against common due dates concerning earliness and tardiness penalties. He proposed a recovering beam search algorithm to solve this problem. Behnamian et al. [17] considered the problem of parallel machine scheduling to minimize both makespan and total earliness and tardiness. Fernandez-Viagas et al. [11] studied the problem of scheduling jobs in a permutation flow shop to minimize the sum of total tardiness and earliness. They developed and compared four heuristics to deal with the problem. More recently, the two-machine permutation flow shop scheduling problem to minimize total earliness and tardiness has been addressed by two branch-and-bound algorithms utilizing lower bounds and dominance conditions [30].

Total tardiness minimization is another common criterion in the scheduling literature where only the tardiness penalties are considered. Koulamas [21] surveyed theoretical developments, exact and approximation algorithms for the single-machine scheduling problem with the aim of total tardiness minimization. In [26], single machine scheduling with family setup and resource constraints to minimize total tardiness minimization was addressed. A mathematical formulation and a heuristic solution approach were presented. Recently, Silva et al. [24] studied the single machine scheduling problem that minimizes the total tardiness. They presented two algorithms to deal with the situation in which the processing time is uncertain.

As for the scheduling modes, research on online scheduling is one of the popular streams. Since this problem has been an active field for several decades, an in-depth analysis of the literature review is beyond the present paper's scope. Thus, in this section, we recall some of the most traditional approaches to online scheduling, and we review the main applications of RL to this problem.

Differently from tailored algorithms (heuristic and exact methods), which might require effort in implementation and calibration over a broad set of parameters, dispatching rules are widely adopted for online scheduling for their simplicity (see, e.g., [19]). For instance, the *earliest due date* (*EDD*) dispatching rule is one of the most commonly used ones in practical applications [34]. *EDD* schedules first the job with the earliest due date. Again, in [15], the authors propose a deterministic greedy algorithm known as *list scheduling* (*LS*), which assigns each job to the machine with the smallest load. For more details, we refer the reader to the work [27] that classified over one hundred dispatching rules. In [5], the authors designed a deterministic algorithm and a randomized one for online machine sequencing problems using Linear Programming techniques. At the same time, in [23], the authors proposed an algorithm to make jobs artificially available to the online scheduler by delaying the release time of jobs.

In online scheduling, a decision-maker is regularly scheduling jobs over time, attempting to reach the overall best performance. Therefore, it is reasonable that RL represents one of the possible techniques to exploit such a setting.

In [13], the authors interpreted job-shop scheduling problems as sequential decision processes. They try to improve the job dispatching decisions of the agent by employing an RL algorithm. Experimental results on numerous benchmark instances showed the competitiveness of the RL algorithm. More recently, in [39], the authors modeled the scheduling problem as a Markov Decision Process and solved it through a simulation-based value iteration and a simulation-based *Q-learning*. Their results clearly showed that such RL algorithms could achieve better performance concerning several dispatching heuristics, disclosing RL application's potential in the field. In the context of an online single-machine environment, in [37], the authors compared the performance of *neural fitted Q-learning* techniques using combinations of different states, actions, and rewards. They proved that taking only the necessary inputs of states and actions is more efficient.

While all the discussed works revealed RL's competitiveness on scheduling problems, a further comparison of the performance among various RL algorithms is still missing in the scheduling literature. With the knowledge of the available studies showing RL's potential and the demand from the industrial application, we are motivated to compare different RL approaches' performance on online scheduling for getting more insights. In particular, we carry out experimental studies on four of the most commonly used model-free RL algorithms, namely *Q-learning*, *Sarsa*, *Watkins's Q(λ)*, and *Sarsa(λ)*. Our comparison methodology is inspired by [37], in which the best configuration for minimizing maximal lateness is pursued. In our work, instead, we propose two different objective functions to minimize: the total tardiness and the total earliness and tardiness. Moreover, another significant difference with their work lies in the way we evaluate the results. While they used the result

from one run, our results come from 50 runs with different random seeds, and two different time step sizes are tested (the interaction between agent and environment is checked in each step). We further test a neural network-based RL technique showing that it is unnecessary to use such a combination when the state space is limited.

4 Reinforcement Learning Algorithms for Online Scheduling

In this section, we describe the algorithmic framework used to deal with our online single-machine scheduling problem. In particular, we provide several variants based on different RL techniques.

4.1 States, actions, and rewards

To be approached by RL techniques, we define our problem setting along the lines used in [37]. In particular:

- *state*: a state is associated with each possible length of the jobs in the waiting queue;
- *action*: if not all the jobs are finished, the action is either to select one new job from a specific position of the waiting queue and start processing it (we recall that preemption is allowed) or to continue processing the job which has been already assigned to the machine in the previous step;
- *reward*: since RL techniques aim at maximizing rewards while our problem seeks to minimize its objective function (either the total tardiness or the total earliness and tardiness), we set the reward of a state as the opposite value of the considered measure.

When the action implies selecting a job from a certain position in the waiting queue, it is important to decide the order in which jobs are stored inside the queue. Therefore, we implemented three possible ordering of jobs that provide very different scheduling effects:

- jobs are unsorted (*UNSORT*), i.e., they have the same order as the arrivals;
- jobs are sorted by increasing value of due time (*DT*);
- all unfinished jobs are sorted by increasing the value of the sum of due time and processing time (*DT+PT*).

For instance, by using *DT*, if the action is to select a job in the second position of the queue, the job with the second earliest due time will be processed.

4.2 RL algorithms adopted

We have decided to implement four different RL algorithms, namely *Q-learning*, *Sarsa*, *Watkins's Q(λ)*, and *Sarsa(λ)*. They are described in the following. Here are some notations used:

- s state;
- a action;
- $\mathcal{S}$ set of nonterminal states;
- $\mathcal{A}(s)$ set of actions possible in state s;
- S_t state at t;
- A_t action at t;
- R_t reward at t.

4.2.1 Q-learning

Q-learning is a technique that learns the value of an optimal policy independently of the agent's action. It is largely adopted for its simplicity in the analysis of the algorithm and for the possibility of early convergence proofs by directly approximating the optimal action-value function (see [36] and [33]). The updating rule for the estimation of the Q-function is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]. \quad (1)$$

The $Q(S_t, A_t)$ function estimates the quality of state-action pair. At each time step t, the reward R_{t+1} from state S_t to S_{t+1} is calculated and $Q(S_t, A_t)$ is updated accordingly. The coefficient α is the learning rate ($0 \leq \alpha \leq 1$); it determines the extent that new information overrides the old information. Furthermore, γ is the discount factor determining the importance of future reward and finally, $\max_a Q(S_{t+1}, a)$ is the estimation of best future value.

The values of the Q-function are stored in a look-up table called *Q-table*. Figure 2 displays an example of Q-table storing Q-function values for states from 0 to 10 (in row) and actions from selecting *Job 1* to *Job 5* (in column). By overlooking the

Q Table		Actions				
		Select Job 1	Select Job 2	Select Job 3	Select Job 4	Select Job 5
States	0	0	0	0	0	0
	.	.	.	.	.	.
	.	.	.	.	.	.
	.	.	.	.	.	.
	5	-20	-15	-34	-14	-31
	.	.	.	.	.	.
	.	.	.	.	.	.
	.	.	.	.	.	.
	10	-15	-21	-22	-16	-23

Fig. 2 An example of Q table.

actual policy being followed in deciding the next action, *Q-learning* simplifies the analysis of the algorithm and enabled early convergence proofs.

4.2.2 Sarsa

Sarsa is a technique that updates the estimated Q-function by following the experience gained from executing some policies (see [32] and [33]). The updating rule for the estimation of the Q-function is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]. \quad (2)$$

The structure of formula (2) is similar to (1). The only difference is that (2) considers the actual action implemented in the next step A_{t+1}, instead of the generic best action $\max_a Q(S_{t+1}, a)$.

As for *Q-learning*, also in *Sarsa* the values of the Q-function are stored in a *Q table*. Despite the more expensive behaviour with respect to *Q-learning*, *Sarsa* may provide better online performances in some scenarios (as shown by the *Cliff Walking* example in [33]).

4.2.3 Watkins's Q(λ)

Watkins's Q(λ) is a well-known variant of *Q-learning*. The main difference with respect to classical *Q-learning* is the presence of a so-called *eligibility trace*, i.e. a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. A trace is initialized when a state is visited or an action is taken, and then the trace gets decayed over time according to a decaying parameter λ (with $0 \leq \lambda \leq 1$). Let us call $e_t(s, a)$ the trace for a state-action pair (s, a). Let us also define an indicator parameter $\mathbb{1}_{xy}$ that takes value 1 if and only if x and y are the same, and 0 otherwise. Then, for any (s, a) pair (for all $s \in \mathcal{S}$, $a \in \mathcal{A}$), the updating rule for the estimation of the Q-function is:

$$Q_{t+1}(s, a) = Q_t(s, a) + \alpha \delta_t e_t(s, a) \quad (3)$$

where

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t) \quad (4)$$

and

$$e_t(s, a) = \gamma \lambda e_{t-1}(s, a) + \mathbb{1}_{sS_t} \mathbb{1}_{aA_t} \quad (5)$$

if $Q_{t-1}(S_t, A_t) = \max_a Q_{t-1}(S_t, a)$, and $\mathbb{1}_{sS_t} \mathbb{1}_{aA_t}$ otherwise.

As the reader can notice, by plugging Eq. (4) into Eq. (3), we obtain an equation similar to (1) but with the additional eligibility term that increments the value of δ_t if the state and action selected by the algorithm are one of the eligibility states. In the rest of the paper we use *Q(λ)* referring to *Watkins's Q(λ)*.

4.2.4 Sarsa(λ)

Similarly to *Q(λ)*, the *Sarsa(λ)* algorithm represents a combination between *Sarsa* and eligibility traces to obtain a more general method that may learn more efficiently. Here, for any (s, a) pair (for all $s \in \mathcal{S}$, $a \in \mathcal{A}$), the updating rule for the estimation of the Q-function is:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \tag{6}$$

where

$$\delta_t = R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) \tag{7}$$

and

$$e_t(s,a) = \gamma\lambda e_{t-1}(s,a) + \mathbb{1}_{sS_t}\mathbb{1}_{aA_t} \tag{8}$$

Unlike Eq. (5), there is no other condition (set the eligibility traces to 0 whenever a non-greedy action is taken) added. A more in-depth discussion about the interpretation of the formulas is given in [33].

5 Simulation Setting

In order to perform the comparison under interest, we create an online scheduling simulation procedure as described in Algorithm 1.

We first update Q tables through a training phase then use the Q tables to select actions in the test phase.

The arrival time of job j are distributed according to an exponential distribution, i.e., $X_j \sim exp(r)$ with the rate parameter valued $r = 0.1$. It is simulated in this way: at the first time step, a random number of jobs (from 1 to 6 jobs) and an interval time (following the exponential distribution) are generated. Once a job is generated (simulating the job's arrival), it will immediately be put into the waiting queue. Then at the next time step, if the interval time is passed, new jobs will be generated and put into the waiting queue; meanwhile, a new interval time will be created. Otherwise, nothing is created. Then the same procedure repeats till reaching a final state.

For the settings regarding RL algorithms:

- In the policy, $\epsilon = 0.1$ enabling highly greedy actions while keeping some randomness in job selections;

Algorithm 1 Online scheduling simulation through RL algorithms

Require: $|\mathcal{E}|$ number of episodes; $|\mathcal{T}|$ number of time-steps;
Initialize $Q(s, a) = 0, \forall\, s \in \mathcal{S}, a \in \mathcal{A}$;
for $\eta \leftarrow 1$ to $|\mathcal{E}|$ **do**
  Initialize S
  for $t \leftarrow 1$ to $|\mathcal{T}|$ **do**
    if new jobs arrive **then**
      Update waiting list L
    end if
    if L is not empty **then**
      Take A_t in S_t, observe R_t, S_{t+1}
      Calculate A_{t+1} and update Q_t
      $S_t \leftarrow S_{t+1}$, $A_t \leftarrow A_{t+1}$
    end if
  end for
end for

- In the value function, $\alpha = 0.6$, i.e., there is a bit higher tendency to explore more possibilities while a bit lower in keeping exploiting old information, whereas $\gamma = 1.0$, which means it strives for a long-term high reward;
- In the eligibility traces, λ is 0.95, a high decaying value leads to a longer-lasting trace.

It is worth noting that all the algorithms considered are heuristics. They focus on finding a good solution quickly by finding a balance between the solution space's intensified and diversified explorations. Nevertheless, the direct implantation of the algorithms above does not ensure enough diversification. For this reason, it is common to use a ϵ-greedy method. Thus, with probability ϵ, exploration is chosen, which means the action is chosen uniformly at random between the available ones. Instead, with probability $1 - \epsilon$, exploitation is chosen by taking the actions with the highest values greedily. After knowing how to balance exploration and exploitation, we need to define a learning method for finding out policies leading to higher cumulative rewards.

In an episode, we start a new schedule by initializing state S and terminates when either reaching the maximum steps or no jobs to process. To simulate real-time scheduling, for each episode, we check the arrivals of new jobs and update the waiting queue if there are, then we choose the action A, and calculate the reward R and the next state S' accordingly. The Q-functions are updated according to the exact RL algorithms used. The same procedure is carried out in both training and test phases except that in the test. The Q-table is not initialized with empty values but obtained from the training phase.

Let us show how the total tardiness value evolves, for an example in which *Q-learning* is used to schedule the jobs. In Fig. 3, the graph on the bottom shows that the reward increases and reaches the maximum and holds steady after 80 episodes. Accordingly, the objective value (the total tardiness) decreases with more noticeable fluctuations and drops more slowly after 80 episodes. While the reward keeps stable, total tardiness continues dropping to around 40000. To summarize, using total

tardiness as a goal is useful, but it is still challenging to represent the trend of this objective value adequately.

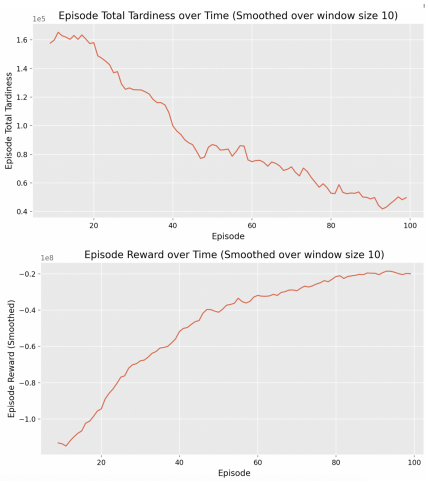

Fig. 3 The changes to reward and the objective value (total tardiness) of 100 episodes.

6 Experimental Results and Discussion

In this section, we propose three different experimental results. Section 6.1 compares the performance among random assignment (*Random*), *EDD*, and the four RL approaches implemented for both minimizing the total tardiness and the total earliness and tardiness. Section 6.2 investigates the possible impact of different operating conditions (i.e., frequency of jobs arrivals) on the RL approaches. Finally, Section 6.3 compares *Q(λ)* and *DQN*.

The algorithms have been implemented in Python 3.6. To avoid possible ambiguities, we locate the related code in a public repository[2]. All the experiments are carried out on an *Intel Core i5* CPU@2.3GHz machine equipped with 8GB RAM and running *MacOS* v10.15.4 operating system.

6.1 RL algorithms vs *Random* and *EDD*

To check if considering different time horizons leads to different results, we consider two experiments in which the time horizon $\mathcal{T}$ is set to 2500 and 5000, respectively.

[2] `https://github.com/Yuanyuan517/RL_OnlineScheduling.git`

For each of the settings, we ran 50 tests with different random seeds. For each algorithm Θ, we call $\Gamma_{\zeta\Theta}$ the objective value achieved in simulation ζ . Furthermore, we define $\rho_{\zeta\Theta}$ to be the percentage gap between the objective value achieved by the best algorithm and algorithm Θ during run ζ, i.e.,

$$\rho_{\zeta\Theta} = \frac{\Gamma_{\zeta\Theta}}{\min_{\zeta\Theta} \Gamma_{\zeta\Theta}}. \tag{9}$$

To compare the different algorithms, we consider the average value of $\rho_{\zeta\Theta}$ concerning all the runs.

The simulation results with the algorithms (under different job orders, time horizons) for the total tardiness and the total earliness and tardiness minimization are displayed in Table 1 and 2, respectively. Note that avg($\rho_{\zeta\Theta}$) and std($\rho_{\zeta\Theta}$) represent respectively the mean value and standard deviations of $\rho_{\zeta\Theta}$. The best value among

		$\lvert\mathcal{T}\rvert$=2500		$\lvert\mathcal{T}\rvert$=5000	
Algorithm	**Jobs order**	avg($\rho_{\zeta\Theta}$)	std($\rho_{\zeta\Theta}$)	avg($\rho_{\zeta\Theta}$)	std($\rho_{\zeta\Theta}$)
Random	-	2.59	0.50	3.06	0.69
EDD	-	7.67	1.76	9.19	1.47
Q-learning	*UNSORT*	2.15	0.43	2.04	0.35
Q-learning	*DT*	1.45	0.28	1.29	0.20
Q-learning	*DT+PT*	1.44	0.30	1.25	0.18
Sarsa	*UNSORT*	2.55	0.53	2.47	0.39
Sarsa	*DT*	1.65	0.40	1.76	0.36
Sarsa	*DT+PT*	1.66	0.47	1.68	0.33
Sarsa(λ)	*UNSORT*	4.42	0.93	5.04	0.93
Sarsa(λ)	*DT*	7.04	1.35	7.73	1.34
Sarsa(λ)	*DT+PT*	3.08	1.03	7.70	1.33
Q(λ)	*UNSORT*	2.04	0.42	2.01	0.40
Q(λ)	*DT*	**1.11**	**0.18**	1.13	0.17
Q(λ)	*DT+PT*	1.19	0.26	**1.09**	**0.14**

Table 1 Simulations of the algorithms with different settings and considering the total tardiness minimization.

all the combinations of algorithms and jobs order policies for each time horizon is highlighted in bold font.

While in [37] the authors show that *EDD* gets a better result than RL in minimizing the maximum tardiness, as shown in Table 1, all the implemented RL algorithms outperform *EDD* in minimizing the total tardiness. This result is exciting and probably depends on whether the learning paradigm is more tailored to optimize min-sum problems than min-max ones. Also, it can be seen that the size of running time steps influences the result on job order but does not affect the algorithm. For the case with 2500 steps, the configuration *Q(λ)* plus *DT* gets the best result, instead for 5000 steps, the configuration *Q(λ)* plus *DT+PT* outperforms the others.

Table 2 Simulations of the algorithms with different settings and considering the total earliness and tardiness minimization.

		$\|\mathcal{T}\|$=2500		$\|\mathcal{T}\|$=5000	
Algorithm	**Jobs order**	avg($\rho_{\zeta\Theta}$)	std($\rho_{\zeta\Theta}$)	avg($\rho_{\zeta\Theta}$)	std($\rho_{\zeta\Theta}$)
Random	-	5.85	9.95	19.34	42.96
EDD	-	4.17	**1.86**	**6.24**	**2.99**
Q-learning	*UNSORT*	5.33	9.33	12.62	30.34
Q-learning	*DT*	3.95	6.71	10.20	23.10
Q-learning	*DT+PT*	3.72	6.20	9.91	22.20
Sarsa	*UNSORT*	5.72	9.62	17.45	39.43
Sarsa	*DT*	4.43	7.87	16.34	37.85
Sarsa	*DT+PT*	4.46	8.13	13.77	33.89
Sarsa(λ)	*UNSORT*	10.77	17.78	36.59	75.93
Sarsa(λ)	*DT*	13.29	23.84	49.71	111.14
Sarsa(λ)	*DT+PT*	6.04	9.87	55.77	116.36
Q(λ)	*UNSORT*	4.68	8.25	15.45	34.97
Q(λ)	*DT*	3.89	6.66	10.21	23.98
Q(λ)	*DT+PT*	**3.29**	5.62	9.23	21.36

Besides, we find with the sorting choice *DT+PT* that all algorithms get smaller average values except for the configuration *Q(λ)* with 2500 steps. Comparatively, a randomly sorting job leads to a much worse result.

Instead, as reported in Table 2, *EDD* outperforms the other algorithms in minimizing the total earliness and tardiness, both in terms of both the mean and the standard deviation for the larger time horizon. Moreover, it achieves the smallest standard deviation for both time horizons. However, the configuration using *Q(λ)* and *DT+PT* gets the smallest mean for the case with 2500 time steps. Taking into account the three job's ordering, it can be noticed that all the algorithms in combination with *UNSORT* have the worst results in terms of both the mean and the standard deviation, except for the algorithm *Sarsa(λ)* (which instead performs very poorly with the sorting choice *DT*).

Finally, it can be noticed that the mean and the standard deviation obtained by the algorithms in minimizing the total earliness and tardiness are larger than those achieved in Table 1. Unlike the total tardiness minimization's objective, the total earliness and tardiness may not be well addressed by the proposed RL algorithms. Considering the measure of jobs, earliness can negatively affect the effectiveness of RL algorithms. A possible reason can be found in the test environment settings. In the experiments, the due date is calculated by first taking a random value, namely the processing time of the job, from an exponential distribution $X \sim Exp(\iota)$ where

$$\iota = \frac{1}{7 \times \max_{j \in \mathcal{J}}\{processingTimeJob_j\}},$$

and adding that value to the current simulation time. Reminding that the tardiness of a job j is defined as $T_j := \max\{0, c_j - d_j\}$ where $c_j = startTime_j + processingTimeJob_j$, then the more the jobs accumulated as time running, the big-

ger the difference between the start time and due date for a job. Hence, more delays will occur, which might cause the simulation results in favor of tardiness calculation.

6.2 *Q(λ)* performance against different job arrival rates

We carried out another test against different frequencies of job arrivals (controlled by the rate parameter r) by considering the two best RL algorithm combinations resulted from the previous tests, i.e., *Q(λ)* plus *DT* and *Q(λ)* plus *DT+PT*. To understand whether the value of r affects the performance, we experimented with 2 more values, i.e. $r = \{0.05, 0.2\}$ in addition to the previous one $r = 0.1$. Tables 3 and 4 show the results of this test in the case of minimization of total tardiness and total earliness and tardiness, respectively. Note that the results have been normalized by following Eq. (9) with 50 tests and $|\mathcal{T}| = 2500$ for each test.

Jobs order	r	avg($\rho_{\zeta\Theta}$)	std($\rho_{\zeta\Theta}$)
DT	0.05	**1.14**	**0.18**
DT+PT	0.05	1.17	0.55
DT	0.10	**1.10**	**0.17**
DT+PT	0.10	1.17	0.26
DT	0.20	1.17	0.28
DT+PT	0.20	**1.12**	**0.24**

Table 3 Experiments on the rate parameter with best settings from *Q(λ)* concerning the total tardiness minimization.

Jobs order	r	avg($\rho_{\zeta\Theta}$)	std($\rho_{\zeta\Theta}$)
DT	0.05	**1.74**	**0.83**
DT+PT	0.05	1.94	0.90
DT	0.10	3.89	6.66
DT+PT	0.10	**3.29**	**5.62**
DT	0.20	5.97	6.66
DT+PT	0.20	**5.75**	**6.37**

Table 4 Experiments on the rate parameter with best settings from *Q(λ)* concerning the total earliness and tardiness minimization.

As shown in the Table 3, with small values of r (e.g., 0.05, 0.10), i.e., when jobs arrive much less frequently than the last one, the version with jobs ordered by *DT* performs better. When jobs arrive much more frequently, the version sorted by *DT+PT* wins. Hence, a careful selection of algorithms and settings according to the operating conditions matters.

Table 4 shows results on comparing the total earliness and tardiness with the same settings as the ones of Table 3. However, even with a different objective, the results for $r = 0.05$ and $r = 0.20$ are similar: the version using DT (for the former) and $DT + PT$ (for the latter) perform better. The difference lies on $r = 0.10$, which gets better performance with $DT + PT$ instead of DT in Table 3. Thus, a combination of factors (settings, operating conditions, and objective) is clearly necessary to be considered when selecting the RL algorithm.

6.3 Comparison between *Q(λ)* and *DQN*

Finally, in this section, we compare a four-layer DQN and $Q(\lambda)$ plus DT+PT, which is the best performing RL algorithm. Figure 4 shows such a comparison in the total tardiness minimization, while Figure 5 is dedicated to the case minimizing the total earliness and tardiness. We run 50 tests and $|\mathcal{T}| = 5000$ in each test. The horizontal axis represents the total tardiness and the vertical axis shows the probability the objective value falls in. Note that the brown area indicates the overlapping between $Q(\lambda)$ and DQN.

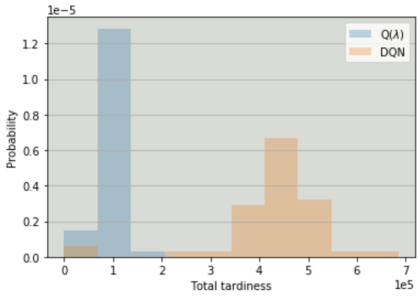

Fig. 4 Comparison between $Q(\lambda)$ and DQN on the total tardiness of 50 runs with different seeds representing different schedules.

From Figure 4, we can see $Q(\lambda)$ has much higher probability with smaller objective value, which indicates $Q(\lambda)$ outperforms DQN. Taking into account the time spent in training DQN is almost 10 times of $Q(\lambda)$, $Q(\lambda)$ is a better option, especially for guaranteeing a flexible and adaptive scheduling in realtime.

The results in Figure 5 are very similar to the previous ones. Compared to $Q(\lambda)$, DQN has a much higher probability with a bigger objective, which stands for its poor performance.

7 Conclusions and Future

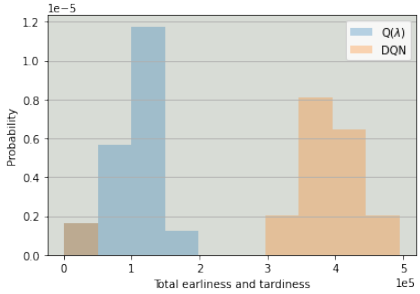

Fig. 5 Comparison between $Q(\lambda)$ and DQN on the total earliness and tardiness of 50 runs with different seeds representing different schedules.

Research Directions

In this paper, we compared four RL methods, namely *Q-learning*, *Sarsa*, *Watkins's Q(λ)*, and *Sarsa(λ)*, with *EDD* and random assignment on an online single-machine scheduling problem with two different objectives, as the total tardiness and the total earliness and tardiness minimization. The experiments show that:

- better scheduling performance in minimizing the total tardiness is achieved by the RL method *Watkins's Q(λ)*, especially when the action concerns the selection of jobs sorted by due date for the smaller time horizon ($|\mathcal{T}| = 2500$) and the selection of jobs sorted by due date and processing time for bigger time horizon ($|\mathcal{T}| = 5000$).
- considering the measure of earliness may negatively affect the performance of RL algorithms. In minimizing the total earliness and tardiness, *Watkins's Q(λ)* with the sorting choice *DT+PT* performs better for the small-time horizon in terms of mean values. In contrast, *EDD* can get better results for the large-time horizon.
- when considering different frequencies of jobs arrival, the combination of *Q(λ)* and job orders have different performances in various operating conditions with different objectives.
- slight differences in algorithms and objectives can profoundly change the results.

Besides, with limited input, using DQN is too costly for extended running time and energy spent adjusting parameters to guarantee a good result. In addition to the numerical results explicitly presented in the paper, according to our previous experience, RL algorithms also do not perform well on a single job-related objective (e.g., maximum tardiness [37]). These indicate careful analysis should be done from different viewpoints (running time, operating conditions, average results from multiple experiments) for making a wiser selection of algorithms.

Furthermore, with multiple machines, more transitions must be considered, which need more representational state information. Thus it will be impossible to store values of all state-action pairs in a Q-table. DQN may take a leading role then. As indicated by the work [12], unpredictable changes may happen at different places in

the state-action space, and more care should be taken to avoid instabilities of DQN. One technique that can achieve this goal is the so-called *kernel function* (see [4]), which builds a future research avenue. Another possibility is creating an algorithm selection framework, as explored in work by Rice [29]. In particular, by mapping from the problem characteristics to the appropriate algorithms considered in the framework, we can achieve an automatic selection of the best one to use.

Acknowledgment

This research was partially supported by the Plastic and Rubber 4.0 (P&R4.0) research project, POR FESR 2014-2020 - Action I.1b.2.2, funded by Piedmont Region (Italy), Contract No. 319-31. The authors acknowledge all the project partners for their contribution.

References

[1] Adamu MO, Adewumi A (2014) A survey of single machine scheduling to minimize weighted number of tardy jobs. Journal of Industrial and Management Optimization 10:219–241

[2] Brucker P (2010) Scheduling Algorithms, 5th edn. Springer Publishing Company, Incorporated

[3] Castrogiovanni P, Fadda E, Perboli G, Rizzo A (2020) Smartphone data classification technique for detecting the usage of public or private transportation modes. IEEE Access 8:58377–58391, DOI 10.1109/ACCESS.2020.2982218

[4] Cerone V, Fadda E, Regruto D (2017) A robust optimization approach to kernel-based nonparametric error-in-variables identification in the presence of bounded noise. In: 2017 American Control Conference (ACC), IEEE, DOI 10.23919/acc.2017.7963056, URL `https://doi.org/10.23919`

[5] Correa JR, Wagner MR (2009) Lp-based online scheduling: from single to parallel machines. Mathematical Programming 119(1):109–136

[6] Fadda E, Plebani P, Vitali M (2016) Optimizing monitorability of multi-cloud applications. In: Nurcan S, Soffer P, Bajec M, Eder J (eds) Advanced Information Systems Engineering. CAiSE 2016. Lecture Notes in Computer Science, Springer, Cham, vol 9694, pp 411–426, DOI https://doi.org/10.1007/978-3-319-39696-5_25

[7] Fadda E, Perboli G, Squillero G (2017) Adaptive batteries exploiting on-line steady-state evolution strategy. In: Squillero G, Sim K (eds) Applications of Evolutionary Computation. EvoApplications 2017. Lecture Notes in Computer Science, Springer, Cham., vol 10199, pp 329–341, DOI https://doi.org/10.1007/978-3-319-55849-3_22

[8] Fadda E, Manerba D, Tadei R, Camurati P, Cabodi G (2019) KPIs for Optimal Location of charging stations for Electric Vehicles: the Biella case-study. In: Ganzha M, Maciaszek L, Paprzycki M (eds) Proceedings of the 2019 Federated Conference on Computer Science and Information Systems, IEEE, Annals of Computer Science and Information Systems, vol 18, pp 123–126, DOI 10.15439/2019F171, URL http://dx.doi.org/10.15439/2019F171

[9] Fadda E, Manerba D, Cabodi G, Camurati P, Tadei R (2021) Evaluation of Optimal Charging Station Location for Electric Vehicles: An Italian Case-Study, pp 71–87. DOI 10.1007/978-3-030-58884-7_4

[10] Fadda E, Manerba D, Cabodi G, Camurati PE, Tadei R (2021) Comparative analysis of models and performance indicators for optimal service facility location. Transportation Research Part E: Logistics and Transportation Review 145, DOI https://doi.org/10.1016/j.tre.2020.102174, URL http://www.sciencedirect.com/science/article/pii/S1366554520308176

[11] Fernandez-Viagas V, Dios M, Framinan JM (2016) Ecient constructive and composite heuristics for the permutation flowshop to minimise total earliness and tardiness. Computers and operations research 75:38–48

[12] François-Lavet V, Fonteneau R, Ernst D (2015) How to discount deep reinforcement learning: Towards new dynamic strategies. arXiv preprint arXiv:151202011

[13] Gabel T, Riedmiller M (2008) Adaptive reactive job-shop scheduling with reinforcement learning agents. International Journal of Information Technology and Intelligent Computing 24(4):14–18

[14] Giusti R, Iorfida C, Li Y, Manerba D, Musso S, Perboli G, Tadei R, Yuan S (2019) Sustainable and de-stressed international supply-chains through the synchro-net approach. Sustainability 11:1083, DOI 10.3390/su11041083

[15] Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell System Technical Journal 45(9):1563–1581, DOI 10.1002/j.1538-7305.1966.tb01709.x

[16] Graves SC (1981) A review of production scheduling. Operations Research 29(4):646–675, DOI 10.1287/opre.29.4.646

[17] JBehnamiana, Ghomi SF, Zandieh M (2009) A multi-phase covering pareto-optimal front method to multi-objective scheduling in a realistic hybrid flow-shop using a hybrid metaheuristic. Expert systems with applications 36:11057–11069

[18] K Leksakul AT (2005) An application of the neural network energy function to machine sequencing. Computational Management Science 2:309—-338

[19] Kaban A, Othman Z, Rohmah D (2012) Comparison of dispatching rules in job-shop scheduling problem using simulation: a case study. International Journal of Simulation Modelling 11(3):129–140, DOI 10.2507/IJSIMM11(3)2.201

[20] Kanet J (1981) Minimizing the average deviation of job completion times about a common due date. Naval Research Logistics Quarterly 28:643–651

[21] Koulamas C (2010) The single-machine total tardiness scheduling problem: Review and extensions. European Journal of Operational Research 202:1–7

[22] Li Y, Carabelli S, Fadda E, Manerba D, Tadei R, Terzo O (2020) Machine learning and optimization for production rescheduling in industry 4.0. The International Journal of Advanced Manufacturing Technology pp 1–19, DOI 10.1007/s00170-020-05850-5

[23] Lu X, Sitters R, Stougie L (2003) A class of on-line scheduling algorithms to minimize total completion time. Operations Research Letters 31(3):232–236, DOI 10.1016/S0167-6377(03)00016-6

[24] Marco Silve NM Michael Poss (2020) Solution algorithms for minimizing the total tardiness with budgeted processing time uncertainty. European Journal of Operational Research 283:70–82

[25] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602

[26] Oliver Herr G (2016) Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints. European Journal of Operational Research 248:123–135

[27] Panwalkar SS, Iskander W (1977) A survey of scheduling rules. Operations Research 25(1):45–61, DOI 10.1287/opre.25.1.45

[28] Pinedo M (2012) Scheduling: Theory, Algorithms, and Systems. Springer, New York, NY, USA

[29] Rice JR (1976) The algorithm selection problem. In: Advances in computers, vol 15, Elsevier, pp 65–118

[30] Schaller J, Valente J (2019) Branch-and-bound algorithms for minimizing total earliness and tardiness in a two-machine permutation flow shop with unforced idle allowed. Computers and Operations Research 109:1–11

[31] Sharma H, Jain S (2011) Online learning algorithms for dynamic scheduling problems. In: 2011 Second International Conference on Emerging Applications of Information Technology, pp 31–34

[32] Singh S, Jaakkola T, Littman ML, Szepesvári C (2000) Convergence results for single-step on-policy reinforcement-learning algorithms. Machine learning 38(3):287–308, DOI 10.1023/A:1007678930559

[33] Sutton RS, Barto AG (2018) Reinforcement learning: An introduction. MIT press

[34] Suwa H, Sandoh H (2012) Online scheduling in manufacturing: A cumulative delay approach. Springer Science & Business Media

[35] Takadama K, Fujita H (2004) Toward guidelines for modeling learning agents in multiagent-based simulation: Implications from q-learning and sarsa agents. In: International Workshop on Multi-Agent Systems and Agent-Based Simulation, Springer, pp 159–172, DOI 10.1007/978-3-540-32243-6_13

[36] Watkins CJCH (1989) Learning from delayed rewards. Thesis Submitted for Ph.D., King's College, Cambridge

[37] Xie S, Zhang T, Rose O (2019) Online single machine scheduling based on simulation and reinforcement learning. In: Simulation in Produktion und Logistik 2019, Simulation in Produktion und Logistik 2019

[38] Ying KC (2008) Minimizing earliness–tardiness penalties for common due date single-machine scheduling problems by a recovering beam search algorithm. Computers and Industrial Engineering 55:494–502

[39] Zhang T, Xie S, Rose O (2017) Real-time job shop scheduling based on simulation and markov decision processes. In: 2017 Winter Simulation Conference (WSC), IEEE, pp 3899–3907, DOI 10.1109/WSC.2017.8248100