



# Robust-stable scheduling in dynamic flow shops based on deep reinforcement learning

Felix Grumbach<sup>1</sup> · Anna Müller<sup>1</sup> · Pascal Reusch<sup>1</sup> · Sebastian Trojahn<sup>2</sup>

Received: 8 June 2022 / Accepted: 7 December 2022 / Published online: 29 December 2022  
© The Author(s) 2022

## Abstract

This proof-of-concept study provides a novel method for robust-stable scheduling in dynamic flow shops based on deep reinforcement learning (DRL) implemented with OpenAI frameworks. In realistic manufacturing environments, dynamic events endanger baseline schedules, which can require a cost intensive re-scheduling. Extensive research has been done on methods for generating proactive baseline schedules to absorb uncertainties in advance and in balancing the competing metrics of robustness and stability. Recent studies presented exact methods and heuristics based on Monte Carlo experiments (MCE), both of which are very computationally intensive. Furthermore, approaches based on surrogate measures were proposed, which do not explicitly consider uncertainties and robustness metrics. Surprisingly, DRL has not yet been scientifically investigated for generating robust-stable schedules in the proactive stage of production planning. The contribution of this article is a proposal on how DRL can be applied to manipulate operation slack times by stretching or compressing plan durations. The method is demonstrated using different flow shop instances with uncertain processing times, stochastic machine failures and uncertain repair times. Through a computational study, we found that DRL agents achieve about 98% result quality but only take about 2% of the time compared to traditional metaheuristics. This is a promising advantage for the use in real-time environments and supports the idea of improving proactive scheduling methods with machine learning based techniques.

**Keywords** Dynamic flow shop · Predictive scheduling · Proactive scheduling · Robust scheduling · Reinforcement learning · Simheuristics

## Introduction

Job scheduling is a major area of interest within the field of production planning. A production organization that can often be observed in practice is the job shop, where each production job has a set of operations to be processed succes-

sively. Each operation is processed without interruption by a continuously available production resource (e.g. a machine), and each resource can only process one operation at a time. In the job shop scheduling problem, the objective is to define operation sequences in such a way, that an objective function (e.g., makespan or flow time minimization) is optimized. Another common form is the flow shop, where every job has the same processing order regarding the resources. A specialized form of the flow shop is the permutation flow shop, where the sequencing decision can only be made at the first resource (cf. transfer lines) (Naderi & Ruiz, 2010).

However, in realistic manufacturing environments, further planning parameters must be considered, which can be independent of the production organization. Especially dynamic events such as machine failures endanger baseline schedules and can require re-scheduling procedures. This leads to a loss of time, additional manufacturing costs, missing production goals, customer dissatisfaction and stress for employees (Liu et al., 2017; Mailliez et al., 2021; Shen et al.,

---

✉ Felix Grumbach  
felix.grumbach@fh-bielefeld.de

Anna Müller  
anna.mueller@fh-bielefeld.de

Pascal Reusch  
pascal.reusch@fh-bielefeld.de

Sebastian Trojahn  
sebastian.trojahn@hs-anhalt.de

<sup>1</sup> Center for Applied Data Science (CfADS), Bielefeld University of Applied Sciences, Gütersloh, Germany

<sup>2</sup> Department of Economics, Anhalt University of Applied Sciences, Bernburg, Germany

2016). Dynamic events can be resource-dependent (internal events such as uncertain setup and processing times, machine failures, uncertain material delivery windows or unplanned rework) or customer-related (external events such as new unplanned orders, order cancellations, changed due dates or order priorities) (Rahmani & Heydari, 2014).

For this reason, there is a need for dynamic or stochastic scheduling methods that take into account uncertainties. Proactive scheduling, also called robust or predictive scheduling, plays a central role in the treatment of dynamic events in advance. Instead of a reactive approach based on subsequent or successive re-scheduling, uncertainties are considered *ex ante* while generating baseline schedules. These schedules should be as insensitive as possible to disruptions so that re-scheduling situations are prevented (de Vonder et al., 2007; Goren & Sabuncuoglu, 2008; Jorge Leon et al., 1994; Negri et al., 2020).

As critical indicators for proactive schedules, robustness and stability metrics have been studied widely. Robustness  $R$  is defined as the difference between expected performance  $P_E$  and real performance  $P_R$  (see Eq. 1) (Jorge Leon et al., 1994), whereby the performance corresponds to the main scheduling objective value (MSOV), e.g. makespan. If the difference is positive, the baseline schedule is too conservative. As a result, resources are not used optimally. If the difference is negative, the baseline schedule is too optimistic. This can lead to overload and time pressure. Due to the nature of minimization problems, an absolute robustness value is also used in some studies (see e.g. Al-Behadili et al., 2019; Goren & Sabuncuoglu, 2008). However, it is then not possible to take precise consideration of conservatism and optimism. The stability  $S$  indicates how precisely the operations  $o \in O$  are planned for the resources. A stability measure, often used in other studies, is defined as the sum of absolute deviations between expected operation completion times  $c_E$  and real operation completion times  $c_R$  (see Eq. 2) (Al-Behadili et al., 2019; Goren & Sabuncuoglu, 2008; Liu et al., 2017; Rahmani & Heydari, 2014).

$$R = P_E - P_R \quad (1)$$

$$S = \sum_{o \in O} |c_E^o - c_R^o| \quad (2)$$

Depending on the chosen performance indicator, there is a conflict in balancing robustness and stability: Improving one metric can worsen the other (Goren & Sabuncuoglu, 2008). It has been shown that exact robust-stable scheduling is an NP-complete problem even in single machine environments (Bougeret et al., 2019). Recent studies have demonstrated heuristic methods to generate balanced robust-stable schedules. We identified two main paradigms to measure robustness and stability: using MCE or using surrogate measures. There are also two main ways to balance

robustness and stability: by creating neighborhood solutions or by adding slack times to critical operations (see Section “Literature review”). The problem with these approaches is, that they either do not explicitly optimize robustness and stability or require too much computing time. This can be a crucial hurdle in reactive real-time environments.

In recent years, several researchers have identified DRL as an appropriate optimization method for scheduling problems (see e.g. Kardos et al., 2021; Park et al., 2021). DRL is a machine learning technique to solve decision problems with deep artificial neural networks. Through conditioning, an autonomous virtual agent learns a policy on how to interact with a sequentially influenceable environment in order to achieve objectives and maximize the outcome. The iterative process can be formulated as a Markov Decision Process, which can be summarized as follows: The agent observes an environment state  $S_i$  and applies action  $A_i$ . Based on the action effect, the environment is transferred to a new state  $S_{i+1}$  and a reward  $R_i$  is distributed to the agent (Morales & Zaragoza, 2012). As function approximators, deep neural networks are used to represent the policy gradually learned and to predict appropriate actions (output) for given state observations (input) (François-Lavet et al., 2018). Fully trained DRL agents achieve a very good compromise between result quality and computing time (Liu et al., 2020). However, after conducting a systematic literature review, we identify a notable lack of scientific approaches investigating, how DRL can be used to optimize and to balance robustness and stability in dynamic flow shops in the proactive stage. The contribution of this study is a proposal on how modern DRL techniques can be implemented to enable a good trade-off between robustness, stability and computing time (see Section “Trade-off between MSO, robustness and stability”).

This article is organized as follows: First, we present related studies with robust-stable scheduling approaches and highlight research gaps (see Section “Literature review”). After that, we show our method of how DRL can be used to dynamically modify operations by stretching or compressing processing times (see Section “Proposed approach”). We then carry out numerical experiments and benchmark different DRL models and metaheuristics in terms of computing time and result quality. In addition, the robustness/stability trade-off, the behavior and the scalability of DRL agents is examined (see Section “Computational study”). Finally, we summarize our results and present future research questions (see Section “Conclusions and future research”).

## Literature review

In recent years, a considerable amount of literature has been published on proactive or reactive scheduling in different

problem contexts such as project, production, crew or workforce planning. However, there is a relatively small body of literature that is concerned with pure proactive scheduling in production environments considering robustness and stability. The existing literature was analyzed with respect to the use of slack-based techniques, the use of simulation, the trade-off analysis of robustness and stability and the application of DRL for proactive scheduling problems. An overview of the reviewed papers is given in Table 1.

### Dynamic environments and uncertainty modeling

A great deal of previous research has focused on rather simple environments, such as single machine (SM) and parallel machine (PM) environments, flow shops (FS), permutation flow shops (PFS), job shops (JS) or flexible job shops (FJS) (Goren & Sabuncuoglu, 2008; Goren et al., 2011; Liu et al., 2017; Shen et al., 2016; Vieira et al., 2017; Wang et al., 2022; Wu et al., 2020). Few authors have used more complex environments like dual resource JS (DRJS) or dual resource FJS (DRFJS) (Soofi et al., 2021; Xiao et al., 2019). Most researchers considered the minimization of makespan ( $C_{max}$ ), flow time (F), total tardiness (T) resp. earliness/tardiness (E/T) as single or separate main scheduling objectives (MSO) (Davenport et al., 2001; Goren & Sabuncuoglu, 2008; Gonzalez-Neira et al., 2021; Jorge Leon et al., 1994; Liu et al., 2017; Rahmani & Heydari, 2014). Furthermore, the minimization of energy consumption (EC), setup time (ST), and idle time (IT) has been considered (Moratori et al., 2010; Sundstrom et al., 2017), some of which were taken into account simultaneously (Moratori et al., 2010; Salmasnia et al., 2014).

The greater part of the literature on proactive scheduling focuses on varying processing times (PT) (Juan et al., 2014; Rahmani & Heydari, 2014; Shen et al., 2016) or machine failures (MF) (Al-Behadili et al., 2019) separately or simultaneously (Negri et al., 2020). Gonzalez-Neira et al. (2021) and Vieira et al. (2017) considered varying setup times (ST) in addition to varying processing times. Sundstrom et al. (2017) considered internal delays (ID). It has been noted that external uncertainties (ex.) such as new job arrivals are only considered in the reactive stage (e.g. Moratori et al., 2010). Since we focus on proactive scheduling approaches, we did no further analysis on the considered external uncertainties. In the literature examined, uncertainties were often modeled by probability distributions (PD) (Al-Behadili et al., 2019; Davenport et al., 2001), intervals (Wang et al., 2022) or scenarios (Rahmani & Heydari, 2014; Soofi et al., 2021; Xiao et al., 2019). A scenario can be defined as a specific production situation from a finite set of possible situations that can occur with a certain probability or under certain conditions and is reflected in various parameters (e.g. processing times). Xiong et al. (2013) determined a failure probability (FB) for every

machine considering the machine's busy time and the total workload of all machines. Negri et al. (2020) used real-time sensor data (RTD) to model machine health conditions.

### Techniques to generate proactive schedules

One approach to solve a scheduling problem under uncertainty is to explicitly allow the insertion of slack or analyze a schedule's slack (Davenport et al., 2001; Liu et al., 2017; Moratori et al., 2010; Salmasnia et al., 2014; Sundstrom et al., 2017; Xiong et al., 2013). The idea of slack-based techniques is to add additional time to certain activities, so that the schedule can absorb the effects of the uncertainty and avoid re-scheduling (Davenport et al., 2001; Sundstrom et al., 2017). Davenport et al. (2001) introduced two slack-based techniques: time window slack (TWS) and focused time window slack (FTWS). TWS ensures that each operation will have at least a specified amount of slack. FTWS additionally takes into account where along the temporal horizon the operation is scheduled. Interestingly, the authors claimed that later scheduled operations need more slack to improve robustness. Unlike other studies, Hatami et al. (2018) applied slack-based techniques in backward scheduling. Their approach aims at setting a suitable amount of slack at the beginning of the schedule.

Another approach to generate proactive production schedules is by creating neighborhood solutions and choosing the best neighbor found (Al-Behadili et al., 2019; Goren & Sabuncuoglu, 2008; Shen et al., 2016). Neighbors are feasible solutions that have a small distance in the solution space relative to the initial solution (Shen et al., 2016). In practical scheduling heuristics, neighborhoods are generated by swapping operations in their topological order (Goren & Sabuncuoglu, 2008) or by flipping assigned resources in flexible environments. Up to now, a number of studies adopted a simheuristic approach to generate proactive schedules. Simheuristics integrate simulation into a metaheuristic-driven framework (Juan et al., 2015). Thus, they combine the effectiveness of metaheuristics and the simulation's capability of uncertainties. Negri et al. (2020) proposed a simheuristic approach that composes a genetic algorithm and a discrete event simulation (DES). Liu et al. (2017) proposed a hybridized evolutionary multi-objective optimization algorithm, where each phenotype of the population is evaluated by a simulation model. Another simheuristic approach was proposed by Gonzalez-Neira et al. (2021). Their approach can be split up into two phases. In the construction phase, a schedule that optimizes earliness/tardiness, is generated. In the second phase, the local search phase, interchanges between jobs are performed and evaluated through MCE. So far, several studies have used exact approaches for robust-stable scheduling (e.g. Davenport et al., 2001; Goren et al., 2011; Rahmani & Heydari, 2014; Salmasnia et al., 2014; Wu et al., 2020).

**Table 1** Related work matrix

Study	Environment	MSO	Uncertainty	Uncertainty modeling	Approach	Slack time	Robustness criterion	Simulation
Al-Behadili et al. (2019)	PFS	$C_{max}$	MF, ex.	PD	H		R, S	✓
Davenport et al. (2001)	JS	T	MF	PD	E	✓	R	✓
Gonzalez-Neira et al. (2021)	PFS	E/T	PT, ST	PD	H		EV, SD	✓
Goren and Sabuncuoglu (2008)	SM	T, F	MF	PD	H		R, S	
Goren et al. (2011)	JS	$C_{max}$	MF, PT	PD	E, H		S	
Hatami et al. (2018)	PFS	$C_{max}$	PT	PD	H		R	✓
Jorge Leon et al. (1994)	JS	$C_{max}$	PT, MF	PD	H		R	✓
Juan et al. (2014, 2015)	PFS	$C_{max}$	PT	PD	H		R	✓
Liu et al. (2017, 2020)	PFS	F	MF, ex.	PD	H	✓	R, S	✓
Moratori et al. (2010)	JS	T, ST, IT, F	ex.		H	✓	S	
Negri et al. (2020)	FS	$C_{max}$	MF, PT	PD, RTD	H		R	✓
Rahmani and Heydari (2014)	FS	$C_{max}$	PT, ex.	Scen.	E		MMR	
Salmasnia et al. (2014)	SM	$C_{max}$ , E/T	PT	PD	E	✓	R	
Shen et al. (2016)	FJS	$C_{max}$	PT	Scen.	H		R	✓
Soofi et al. (2021)	DRF-JS	$C_{max}$	MF	Scen.	H		AMS, RMS	✓
Sundstrom et al. (2017)	JS	EC, $C_{max}$	ID	PD	E	✓	R, S	
Vieira et al. (2017)	JS	$C_{max}$	ST, PT	PD	H		R	✓
Wang et al. (2020, 2022)	PM	$C_{max}$	PT	Inter-val	H		MMR	
Wu et al. (2020)	FS	$C_{max}$	PT	Scen.	E, H		RMS	✓
Xiao et al. (2017, 2019)	DRJS	$C_{max}$	PT	Scen.	H		R	✓
Xiong et al. (2013)	FJS	$C_{max}$	MF	FB	H	✓	R	

But due to the high computational time required for exact methods, most researchers propose heuristic approaches (see Al-Behadili et al., 2019; Goren et al., 2011; Soofi et al., 2021; Xiao et al., 2019; Wang et al., 2022).

### Techniques to evaluate proactive schedules

Most researchers evaluate proactive schedules by using robustness and stability measures described in Section “Introduction”. Rahmani et al. (2014), who model varying processing times through scenarios, evaluated a schedule by the expected robustness over all scenarios. This approach was also applied by Wang et al. (2022), who called it the min-max regret criterion (MMR). Wu et al. (2020) introduced a metric called robust makespan (RMS). RMS is defined as the maximum makespan of a schedule among all scenarios. Soofi et al. (2021) also use RMS and additionally consider the average makespan (AMS) over all scenarios. Gonzalez-Neira et al. (2021) additionally used two qualitative criteria to evaluate the schedule performance and address robustness through expected value (EV) and standard deviation (SD) of the earliness/tardiness. The greater part of the literature on proactive scheduling has utilised simulation to evaluate a schedule’s robustness or stability, such as Leon et al. (1994), Davenport et al. (2001) and Shen et al. (2016). Alternatively, simulation models can be integrated into the optimization algorithm, as described above.

Since simulation experiments require substantial computational times (Goren & Sabuncuoglu, 2008), some studies used surrogate measures for evaluation, such as Goren and Sabuncuoglu (2008), Goren et al. (2011), Liu et al. (2017) or Sundstrom et al. (2017). Surrogate measures are quickly observable metrics that are assumed to correlate well with stochastically ascertainable robustness or stability. Two well-known examples are the average total slack time (Jorge Leon et al., 1994) or sum of variances on the critical path (Goren et al., 2011). Classically, they are used to estimate how susceptible the critical path is to disturbances such as machine breakdowns or varying processing times. The disadvantage is that they only implicitly describe the actual robustness and stability, which can lead to undesirable side effects such as neglecting non-critical operations. Conclusively, surrogate measures have shown low effectiveness compared to simulation-based metrics (Xiao et al., 2017).

### Trade-off between MSO, robustness and stability

A number of studies optimized two or more objectives simultaneously. Al-Behadili et al. (2019) proposed an approach that optimizes makespan as MSO, robustness and stability. In general, a schedule is called stable, if the real operations do not deviate from the planned operations of the baseline schedule. It can be measured by the sum of absolute devia-

tions between expected operation completion times and real operation completion times (see Eq. 2) or by using surrogate measures, such as slack-based methods, which do not require simulation and therefore show more computational efficiency (see e.g. Davenport et al., 2001). But there are only few researchers who analyzed the trade-offs between MSO and robustness or between robustness and stability. Xiong et al. (2013) analyzed the trade-off between MSO and robustness. They developed a multi-objective evolutionary algorithm to minimize the estimated makespan and estimated robustness, that generates a set of feasible, pareto-optimal schedules. They suggested involving managers in the decision-making process by allowing them to choose the final schedule based on domain knowledge. In further research, Shen et al. (2016) also proposed a heuristic approach to optimize two MSOs and the robustness over different scenarios. They highlighted that the MSOs and robustness are seriously conflicted, and no solution can simultaneously optimize all three objectives.

Few studies have analyzed the trade-off between robustness and stability. Goren and Sabuncuoglu (2008) analyze the trade-off in a single-machine environment with random machine failures for three different MSO namely makespan, total flow time and total tardiness. They explored the effect on stability if robustness is optimized and vice versa. They discovered that stability is not considered during robustness optimization and that the schedule’s robustness worsens during stability optimization. Thus, the researchers suggested managing the trade-off by optimizing the weighted linear combination of robustness and stability depending on the MSO: In case of makespan minimization as MSO, practitioners should focus on stability optimization and in case of flow time or tardiness minimization, robustness optimization should be prioritized. Liu et al. (2017) managed the trade-off between robustness and stability by developing an algorithm that efficiently creates a set of pareto-optimal schedules. The visualization of these pareto fronts show, that increasing stability leads to lower robustness and vice versa. The paper focuses on the algorithm’s efficiency and draws no further conclusions about the actual trade-off. Sundstrom et al. (2017) systematically evaluated the triangular trade-off between MSO namely energy consumption, robustness and stability. They also used pareto fronts to analyze the trade-off. Each front depicts robustness and stability values for a constant energy consumption and shows the same results as Liu et al. (2017). Further analysis of the corresponding schedules revealed that slack at the end of the schedule would improve the robustness, while stability is improved by slack throughout the schedule (Sundstrom et al., 2017).

### DRL-based methods

Recent studies have been utilized DRL for scheduling with uncertainties. However, the usage of DRL in the field of

production scheduling is limited to the reactive phase of the approaches, such as in (Minguillon & Stricker, 2020; Shahrabi et al., 2017; Wang et al., 2020). However, examples of robust scheduling with DRL in the proactive phase can be found in other domains (Kenworthy et al., 2021; Su et al., 2018). Kenworthy et al. (2021) used a combination of DRL and Integer Programming (IP) to solve an aircraft crew-scheduling problem. The probabilities in the neural network output layer are used to assign coefficient weights to the variables in the IP. This approach enables an IP with fewer variables and constraints that can deal with stochastic flight durations. Their scheduling objective is to maximize the total amount of buffers. A buffer, in this case, is defined as the amount of time between successive flights of a particular pilot. Su et al. (2018) proposed a hybrid teaching-learning-based optimization algorithm that includes DRL techniques in the teaching phase. They apply their algorithm for aircraft carrier flight deck operations with stochastic durations to maximize the probability of completing within the limitative makespan and minimize the weighted sum of expected makespan and the makespan variance.

## Research gaps

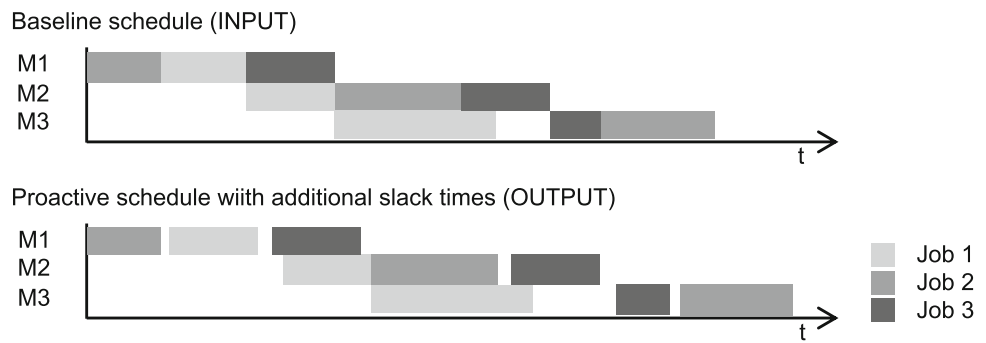
The following essential research gaps can be summarized from our literature review:

1. There exist a few exact robust-stable scheduling approaches such as (Davenport et al., 2001; Goren et al., 2011; Rahmani & Heydari, 2014; Salmasnia et al., 2014; Wu et al., 2020) and several heuristic approaches such as (Al-Behadili et al., 2019; Goren et al., 2011; Soofi et al., 2021; Xiao et al., 2019; Wang et al., 2022). To the best of our knowledge, there does not exist an approach that uses DRL in the proactive stage of production planning. We also cannot identify a DRL-based approach that directly balances robustness and stability in general. Since a fully trained DRL agent can be computationally efficient, can handle uncertain environments, and has proven good results in proactive scheduling in other domains, the use of DRL for robust-stable production scheduling should be scientifically investigated.
2. Many researchers used simulations to evaluate robustness and stability (Al-Behadili et al., 2019; Davenport et al., 2001; Soofi et al., 2021; Wu et al., 2020; Xiao et al., 2019), since the effectiveness is higher compared to surrogate measures (Xiao et al., 2017). Thus, there exist several approaches that integrate simulations directly into the heuristic search strategies (Gonzalez-Neira et al., 2021; Liu et al., 2017; Negri et al., 2020). These so called simheuristics improve the verification and validation processes (Juan et al., 2015) and thus, lead to better results. But a major disadvantage of simheuristics is their computational effort (Juan et al., 2015). To ensure stochastic precision in practical cases with a large number of planning objects and uncertainties, a correspondingly high number of MCEs must be carried out. Thus, the real-time applicability of simheuristics is rather low and more research is required too minimize the computational effort to generate proactive schedules without losing insights gained by simulation.
3. A few researchers have analyzed a schedule's robustness and its stability simultaneously (Al-Behadili et al., 2019; Goren & Sabuncuoglu, 2008; Liu et al., 2017; Sundstrom et al., 2017). The approaches of Goren and Sabuncuoglu (2008) and Liu et al. (2017) suggested two different ways how to manage the trade-off between robustness and stability. To the best of our knowledge, Sundstrom et al. (2017) are the only researchers, who further analyze, how prioritizing robustness and stability influence the schedule itself. Thus, we suggest further research on how proactive schedules are influenced by the trade-off and additionally, if the trade-off is influenced by the MSO.

## Proposed approach

This section describes the core components of our DRL-based method to generate robust-stable schedules proactively. The approach enables a non-iterative method with a metrics-oriented optimization: A DRL agent internalizes the behavior of a stochastic DES and is able to adjust operation processing times without performing computationally intensive MCE step by step. The method is aligned to a dynamic flow shop scenario considering uncertain processing times, machine failure probabilities and uncertain repair times. It is inspired by our industrial partner (see Sections “**Deterministic flow shop problem**”, “**Scenario with uncertainties**”). The method comprises the following sequential sub-steps: First, an opening optimization procedure is conducted, where a baseline schedule is generated deterministically without considering robustness and stability. Then, stability, robustness and other metrics are measured within DES-based MCE (see Section “**Robustness and stability evaluation**”). Based on the stochastic results, a DRL agent modifies the baseline schedule to improve robustness and stability. Here, either operations are extended by additional slack times (*stretching*) or operations are shortened in their plan duration (*compressing*) (see Sections “**Subsequent robustness and stability optimization**”, “**DRL design**”). Stretching corresponds to conservative planning, while compression corresponds to optimistic planning. For example, the agent can put additional slack times on the critical path of the schedule to stretch operations and catch endangering dynamic events (see Fig. 1).

**Fig. 1** Illustrated baseline schedule with 3 machines and 3 jobs (input for the proposed DRL-based approach) and proactive schedule with additional slack times (output from the DRL agent)



**Table 2** Notation for the deterministic baseline problem (flow shop scheduling problem)

Notation	Description
$n$	Number of jobs
$m$	Number of subsequent machines
$p_{i,k}$	Processing time of a job $i$ on machine $k$
$\aleph$	Big positive integer number
$\delta_{i,k}$	Start time of a job $i$ on machine $k$ (variable)
$\epsilon_{i,j,k}$	1, if job $i$ precedes job $j$ on machine $k$ , else 0 (variable)

### Deterministic flow shop problem

Before describing the dynamic problem with uncertainties and the robustness/stability optimization, the opening procedure’s deterministic baseline problem must be formalized. Inspired by the production organization by our corporate research partner, a flow shop scheduling problem with waiting times and infinite machine buffers is considered. In this setting, every production job has a number of operations equal to the number of machines. As a non-permutation setting, jobs can overtake each other in the production line. As MSO, the operation sequences should be set in such a way that either the makespan or the total flow time without release times is minimized. Due to the alignment as a feasibility study, the problem was simplified to ensure a better transferability of the proposed approach. For example, employees, sequence dependencies or setup operations were not taken into account (Table 2).

The baseline flow shop model for makespan minimization can be formulated as the following mixed integer linear program:

$$\min C_{max} = \max_{1 \leq i \leq n} \{\delta_{i,m} + p_{i,m}\} \tag{3}$$

s.t.

$$\delta_{i,k} + p_{i,k} \leq \delta_{i,k+1} \quad \forall i \in \{1, \dots, n\}, \forall k \in \{1, \dots, m - 1\} \tag{4}$$

$$\epsilon_{i,j,k}(\aleph + p_{j,k}) + \delta_{i,k} - \delta_{j,k} \geq p_{j,k} \tag{5}$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}, \forall k \in \{1, \dots, m\}$$

$$(1 - \epsilon_{i,j,k})(\aleph + p_{i,k}) + \delta_{j,k} - \delta_{i,k} \geq p_{i,k} \tag{6}$$

$$\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}, \forall k \in \{1, \dots, m\}$$

$$\delta_{i,k} \in \mathbb{N} \quad \forall i \in \{1, \dots, n\}, \forall k \in \{1, \dots, m\} \tag{7}$$

$$\epsilon_{i,j,k} \in \{0, 1\} \quad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, n\}, \forall k \in \{1, \dots, m\} \tag{8}$$

Equation 3 represents the objective function for makespan minimization. Equation 4 ensures that an operation of a job can only start, when the preceding operation on the previous machine is completed. Furthermore, wait times between operations are allowed with this modeling. Equations 5, 6 ensure that a machine can only process one operation at a time. For total flow time minimization (without release times), Eq. 3 must be replaced by the following objective function:

$$\min F = \sum_{i=1}^n \delta_{i,m} + p_{i,m} \tag{9}$$

### Scenario with uncertainties

To specify a dynamic problem with consideration of uncertainties, the scenario must be explained in more detail: the partner company manufactures custom industrial fittings, where the single production of complete systems involves the non-takt manufacturing steps (1) machining, (2) welding and (3) assembly. For each product category  $PC \in \{1, 2\}$  and manufacturing step  $k$ , we assume triangularly distributed uncertain processing times  $D_{k,PC}$ . Each uncertain processing time is represented as a tuple  $(a, c, b)$ , where  $[a, b]$  are the interval bounds and  $c$  is the mode ( $a \leq c \leq b$ ). In this way, asymmetric distributions can also be defined in a compact

**Table 3** Metrics for a single simulation run

Metric	Description
$R$	Robustness of a schedule measured in a simulation run (see Eq. 1)
$S$	Stability (see Eq. 2)
$C_{max}$	Makespan
$F$	Total flow time
$e_{i,k}$	End time of operation ( $i, k$ )
$TS_{i,k}$	Total slack of an operation

manner. The expected value  $E(X)$  and the standard deviation  $\sqrt{V(X)}$  for a triangular distribution tuple  $X$  can be defined as follows:

$$E(X) = \frac{a + b + c}{3} \quad (10)$$

$$\sqrt{V(X)} = \frac{\sqrt{(a-b)^2 + (b-c)^2 + (a-c)^2}}{6} \quad (11)$$

Moreover, each machine has a failure probability  $P(F_k)$ . As a simplified modeling, a failure can occur once per operation and requires a repair operation with a triangularly distributed machine-specific repair time  $Q_k$ . Equation 12 defines the expected processing time of an operation including a probable machine repair time. This value is used for each operation processing time  $p_{i,k}$  in the deterministic opening procedure to achieve an optimal or near optimal MSOV under realistic conditions (see Section “[Action space analysis](#)”).

$$E(D_{k,c}) + P(F_k)E(Q_k) \quad (12)$$

### Robustness and stability evaluation

After generating a baseline schedule deterministically with expected processing times, robustness and stability must be evaluated under stochastic conditions within DES-based MCE. The DES is used to dynamically simulate a baseline schedule according to the operation sequence set by the opening procedure. Dynamic simulation means that random processing times are set according to the processing time distributions and consideration of downtimes. During simulation, operations are started as soon as the previous operation is completed and the machine is available. The start time of an operation set by the opening procedure can therefore also be undershot. According to the formulated baseline problem (see Section “[Deterministic flow shop problem](#)”), waiting times between operations and jobs are possible. Moreover, station buffer upper bounds are not taken into account. The DES is designed as a process-based simulation in which each resource is designed as its own asynchronous process. In the beginning, a separate process is started for each machine  $M_1, \dots, M_m$ . A process runs until all associated operations are completed according to the baseline schedule. In each

**Table 4** Summary metrics for all simulation runs

Metric	Description
$\bar{R}$	Average robustness from all simulation runs
$\bar{S}$	Average stability
$\bar{C}_{max}$	Average makespan
$\bar{F}$	Average total flow time
$\bar{e}_{i,k}$	Average end time of an operation
$\bar{TS}_{i,k}$	Average total slack of an operation
$\bar{TS}$	Average total slack from all operations

process, the next specified operation is taken from the input buffer as it becomes available. Next, the operation is executed with a randomly set processing time according to the distributions. In addition, a machine failure can occur that must be remedied within the respective repair time. After the operation is completed, it is committed to the next machine’s buffer or final drain. The simulation ends when all processes are finished. In the context of MCE, the simulation is performed multiple times, with dynamic events being triggered randomly in each case. After a simulation run, the metrics described in Table 3 are measured. Considering all simulation experiments, the metrics described in Table 4 can be obtained.

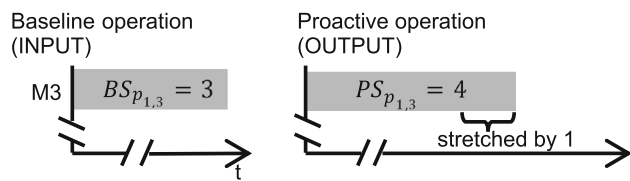
### Subsequent robustness and stability optimization

For a better understanding of the proposed DRL-based method, the subsequent dynamic problem is formalized in this section. The optimization model enables stretching and compressing operation durations to absorb dynamic events and to improve robustness and stability simultaneously. To evaluate a proactive schedule candidate  $PS$ , the optimization procedure integrates MCE (see Section “[Robustness and stability evaluation](#)”) to calculate an aggregated and normalized robustness/stability value  $\Lambda \in \mathbb{R}_{>0}$ .

$$\min \Lambda = \frac{|PS_{\bar{R}}|w}{|BS_{\bar{R}}|} + \frac{PS_{\bar{S}}(1-w)}{BS_{\bar{S}}} \quad (13)$$

s.t.





**Fig. 2** Stretching an operation before evaluating robustness and stability via MCE. Simplified example with 1 machine and 1 job

$$PS_{p_{i,k}} \in \mathbb{N} \quad \forall i \in \{1, \dots, n\}, \forall k \in \{1, \dots, m\} \quad (14)$$

The lower the  $\Lambda$  value, the better is the linear combination of robustness and stability in comparison to the related baseline schedule  $BS$ . Since robustness and stability are competing objectives, a linear weighting is applied using a robustness weight  $w \in [0, 1]$  and a stability weight  $1 - w$  for balancing (see Eq. 13). Stretching or compressing operations is enabled by manipulating operation plan durations  $p_{i,k}$  in a range as it is allowed by technological constraints. Every operation duration  $PS_{p_{i,k}}$  from the  $PS$  is overwritten with regard to improve robustness and stability (see Eq. 14, Fig. 2). The next section describes how a DRL agent can be implemented to solve this dynamic problem, taking into account the logistical specifics of the baseline problem (see Section “[Deterministic flow shop problem](#)”).

## DRL design

This section describes the general conception of the DRL agent based on the formalized dynamic problem in Section “[Subsequent robustness and stability optimization](#)”. We identified two state-of-the-art actor critic DRL algorithms for discrete action spaces: Proximal Policy Optimization (PPO) (see OpenAI, 2022c; Schulman et al., 2017) and Advantage Actor-Critic (A2C) (see Mnih et al., 2016; OpenAI, 2022b), which have been realized using an OpenAI Gym environment (see OpenAI, 2022a) in combination with the *Stable-Baselines3* framework (see Stable-Baselines3, 2022). Hyperparameter configuration including neural network architecture and documentation of the training process can be found in Section “[DRL policy learning benchmark \(PPO vs. A2C\)](#)”.

Algorithm 1 illustrates the essential training procedure including reward calculation. An episode always includes as many steps as there are operations in the schedule. The first step refers to the first operation on machine  $M_1$ , the last step to the last operation on machine  $M_m$ . For each training step, the current state  $\pi$ , the chosen action  $\zeta$  and the current modified proactive schedule candidate  $PS$  are passed as arguments. The state space contains various features that are intended to describe the baseline schedule and the current proactive schedule structure as generally as possible (see Table 5). Figure 3 shows the discrete action space utilized.

It provides three actions applicable for each scheduled operation  $(i, k)$ . Depending on the action chosen, the operation processing time  $p_{i,k}$  is stretched or compressed to improve robustness and stability.

In the step function essential for the episodic learning, the next operation is first initialized and then modified considering the chosen action, which is also overwritten in the  $PS$  candidate. Then, the  $PS$  is simulated once in a deterministic manner. In this way, current effects on robustness and stability are measured and updated in a new state space  $\pi^*$ . Based on this, the intermediate reward can be calculated, that is distributed after each step (see Algorithm 2). It is calculated according on how the agent set the processing time per operation. Setting optimistic processing times improves stability, if operations were previously stretched. On the other hand, conservative processing times improve robustness (see Section “[Action space analysis](#)”). Stability optimization through action 2 is rewarded directly but results in a penalty due to robustness degradation considering  $w$ . On the other hand, robustness improvement only results in a reward, which is slightly higher. In the last episode step, the  $PS$  is handed over to the MCE to calculate the robustness/stability value  $\Lambda$  (see Eq. 13). Based on this, a final penalty is distributed, where  $\Lambda$  is multiplied by a large number. This ensures that the final penalty has a significantly greater impact than the intermediate rewards. A value greater than 1 awards a higher penalty that increases linearly with degradation. For a value less than 1, the penalty decreases rapidly as the improvement increases. In this way, combined with appropriate discount factors, we were able to make the agent more greedy and reliable in terms of achievable end results. The specified reward and penalty scores are the result of a successive empirical analysis and will not be explained further.

### Algorithm 1 DRL training episode (pseudocode snippet)

```

1: function STEP( $\pi$ : state,  $\zeta$ : action,  $PS$ : proactive schedule candidate)
2:    $(i, k) \leftarrow getNextOperation()$ 
3:    $PS \leftarrow modifyOperation(\zeta, i, k, PS)$ 
4:    $\pi^* \leftarrow simulateSchedule(PS)$ 
5:    $rew \leftarrow interReward(\zeta, \pi, \pi^*)$  ▷ See Algorithm 2
6:   if Last scheduled operation then
7:      $rew \leftarrow rew + \begin{cases} (-100\Lambda)^{10}, & \text{if } \Lambda < 1 \\ -125\Lambda, & \text{else} \end{cases}$  ▷ See Eq. 13
8:   end if
9:   return  $\pi^*, rew$ 
10: end function

```

## Computational study

Numerical experiments were conducted and discussed to answer the following questions: (1) How does the plan dura-

**Table 5** DRL observation features with logistical specifics for non-permutation flow shops with waiting times and infinite machine buffers

No.	Description
1	Number of jobs
2	Current episode step
3	Current operation machine $k$ [dummy encoded]
4	Current operation product category $c$ [dummy encoded]
5	Determines, if the total slack of the current operation ( $i, k$ ) operation is above average ( $BS_{\overline{TS}_{i,k}} > BS_{\overline{TS}}$ ) [binary encoded]
6	Determines, if the current operation is on the critical path ( $BS_{\overline{TS}_{i,k}} = 0$ ) [binary encoded]
7	Number of succeeding jobs on the current machine
8	Sum of stretched and compressed operations on the machine $\left( \sum_{o \in O'} \begin{cases} 0, & \text{if action 1 has been chosen} \\ -1, & \text{if action 2 has been chosen } O' := \text{Machine operations} \\ 1, & \text{else} \end{cases} \right)$
9	Sum of stretched and compressed operations within the job
10	Current effect on robustness ( $PS_{MSOV} - BS_{MSOV} + BS_{\overline{R}}$ )
11	Current effect on the job end time ( $PS_{e_{i,m}} - BS_{\overline{e}_{i,m}}$ )
12	Current effect on the operation end time ( $PS_{e_{i,k}} - BS_{\overline{e}_{i,k}}$ )

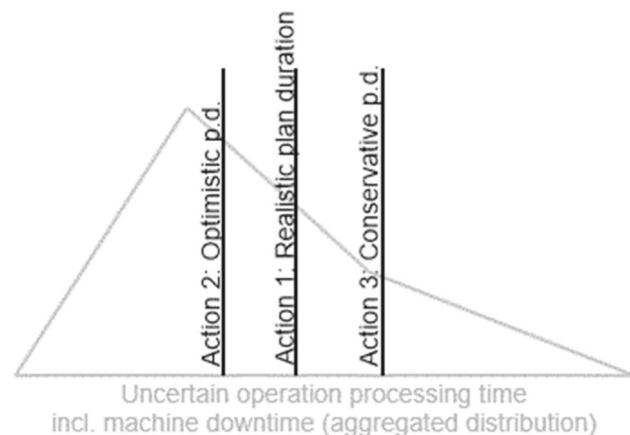
**Algorithm 2** Intermediate reward calculation (pseudocode snippet)

```

1: function INTERREWARD( $\zeta, \pi, \pi^*$ )
2:    $rew \leftarrow 0$ 
3:   if  $|\pi_{10}^*| < |\pi_{10}|$  then ▷ see Table 5
4:      $rew \leftarrow 6w$ 
5:   else
6:      $rew \leftarrow -4w$ 
7:   end if
8:   if  $\zeta_2$  has been selected then
9:      $rew \leftarrow 4(1 - w)$ 
10:  end if
11:  return  $rew$ 
12: end function

```

tion affect the trade-off between robustness and stability and how to define the action space (see Section “[Action space analysis](#)”)? (2) How well do DRL agents learn a policy to generate robust-stable schedules (see Section “[DRL policy learning benchmark \(PPO vs. A2C\)](#)”)? (3) How is the learned policy reflected in the agent’s decision behavior (see Section “[DRL agent behavior analysis](#)”)? (4) How well performs a DRL agent in comparison to a Simulated Annealing with Research Strategy (SARS) approach regarding result quality and computational efficiency (see Section “[DRL performance benchmark \(PPO vs. SARS\)](#)”)? (5) How performs the agent when varying problem size and uncertainty and how are robustness and stability affected (see Section “[Scalability investigation](#)”)? The general experimental setup for all experiments including test data description is documented in Section “[Experimental setup](#)”.



**Fig. 3** DRL discrete action space with three possible actions for selecting an operation plan duration within the aggregated uncertainty distribution. The actions have been determined by the experiments in Section “[Action space analysis](#)”. *Action 1* corresponds to plan duration  $PD_3$ ; *Action 2* corresponds to  $PD_2$ ; *Action 3* corresponds to  $PD_4$  (see Appendix Table 9). Illustrated representation not to scale

## Experimental setup

The experiments were conducted with the following hardware: GPU: Nvidia Quadro RTX3000 5980MB VRAM; CPU: Intel Core i7-10875H @ 2.3 GHz (16 CPUs); 16384MB RAM. Makespan ( $C_{max}$ ) and total flow time ( $F$ ) minimization were considered as separate MSOs. With respect to the scope, robustness and stability have been considered equally in the experiments ( $w = 0.5$ ). In fact, the weight  $w$  is an independent variable that has an effect on the planning result depending on the MSO Goren and Sabuncuoglu

(2008). However, results from equal weighting are discussed in relation to the agent performance (see Section “[DRL policy learning benchmark \(PPO vs. A2C\)](#)”). After intensive research, we could not identify suitable benchmark instances which can be used for our approach and uncertainty modeling without further modification. Furthermore, the uncertainties had to be scalable in order to analyze the degree of uncertainty. Consequently, like other authors (see e.g. Al-Behadili et al., 2019; Goren & Sabuncuoglu 2008; Liu et al., 2017), we generated suitable instances. In addition to instances based on the industrial scenario (see Section “[Scenario with uncertainties](#)”), we modified a well-known instance proposed by Taillard (1993) for deterministic flow shop scheduling benchmarks.

- **GMRT5x3.** Compact scenario-based test data with 5 jobs and 3 machines.
- **GMRT10x3.** Extended scenario-based test data with 10 jobs.

The GMRT instances were utilized to generally analyze the DRL agent learning performance and behavior (cf. initially mentioned experiment questions 1–4). For training and evaluating the DRL agent, 10 training and 10 test instances were generated per data set and per MSO, each with a random number of product categories. The COIN-OR Branch-and-Cut solver V2.10 has been used to generate optimal deterministic baseline schedules. The uncertainties considered for the subsequent proactive optimization method can be found in Appendix Tables 6, 7. According to experiment question 4, the proposed DRL-based approach has been compared against an iterative metaheuristic. Consequently, the number of experiments must be chosen such that the true  $\Lambda$  value of a neighbor is precisely approximated. It had to be prevented that random outliers are confused with local optima. We used the standard deviation of the means to measure the error of yielded  $\Lambda$  values Jacoboni and Lugli (1989). Here we set an upper limit of about  $\sigma_n \approx 0.005$ , which corresponds to about  $n = 10,000$  experiments and led to reliable results.

In order to indicate the reliability and scalability of the method, the following instances were used:

- **T20x5LV.** Modified Taillard instance with 20 jobs, 5 machines and low uncertainty variance (LV).
- **T20x5HV.** The same instance with a high uncertainty variance (HV).

According to experiment question 5, it was analyzed how the agent handles more extensive environments with varied uncertainties. In this case, the baseline schedules were generated using the Shortest Processing Time (SPT) dispatching rule. Due to the specifics of our scheduling and uncertainty

model, the data was modified as described in Appendix Table 8.

## Action space analysis

This analysis addressed the first experiment question and examined the impact of different plan durations (PD) utilizable as  $PS_{pi,k}$  in the subsequent robustness and stability optimization method (see Section “[Subsequent robustness and stability optimization](#)”). The aim was to investigate and to specify the scope of the DRL action space (see Section “[DRL design](#)”), which is important for a proper policy learning. In particular, it was analyzed which PD leads to which consequences in terms of robustness and stability. Moreover, general conclusions about the conflict between robustness and stability could be drawn. Five PD were considered, where  $PD_1$  is a very optimistic,  $PD_3$  a realistic and  $PD_5$  a very conservative time. All PD are within the bounds of the aggregated processing time distribution of an operation (see Appendix Table 9). Appendix Table 10 and Fig. 4 give a statistical overview of the effects.

What stands out in the results are the very similar trade-off patterns and the associated impact of different PD. According to our observations,  $PD_3$  achieved good stability values for  $C_{max}$  and  $F$  minimization. More optimistic or conservative values worsened the stability in all cases. The more conservative  $PD_4$ , on the other hand, led to better robustness. With  $PD_5$ , even positive robustness was achieved for all baseline schedules. This measurement confirms the conflict in a simultaneous robustness and stability optimization in this case: The more realistic the plan durations, the better the stability. If, on the other hand, the PD are chosen more conservatively, the robustness increases at the expense of stability. This interesting finding may be related to statistical scope per metric. With a single operation, it is most likely that the expected operation duration will occur (scope of stability). However, at the level of the overall schedule (scope of robustness), previously unknown causes lead to overly optimistic planning when expected values are utilized. Future studies should examine these causes in more detail, which would go beyond the scope of this work.

A closer inspection of the figure shows that the robustness values in the  $F$  case are significantly more sensitive to the PD chosen and more widely spread, which can also be observed in the different value ranges of the robustness axes. This result can be explained by the fact that the  $C_{max}$  objective function only depends on one value (end time of the last operation), whereby the  $F$  objective function contains the end time of each job. Therefore, instead of one, several operations must be considered in the  $F$  case. This is an indicator that stretching or compressing operation durations has to be done in a targeted manner. In this case, it is not sufficient to just stretch an operation at the end in order to achieve

apparent robustness. Conclusively, it can be confirmed that a corresponding optimization method for balancing robustness and stability can be utilized by choosing PD dynamically per operation and in a targeted manner. Since we locate the best trade-offs between  $PD_2$  and  $PD_4$ , these three PD have been included in the DRL action space. In terms of a feasibility study, this choice of actions may be appropriate. However, future work should examine the applicability of continuous action spaces, where even more targeted values can potentially be set. Subject to the three selectable actions defined, the agent's learning and prediction performance are presented and discussed in the next section. How and why the agent selects which action in which situation is examined in Section “[DRL agent behavior analysis](#)”.

### DRL policy learning benchmark (PPO vs. A2C)

The purpose of this experiment was to evaluate an A2C and a PPO model to compare two modern DRL algorithms regarding their learning and predicting performance (see experiment question 2). Separate models were trained for  $F$  minimization and  $C_{max}$  minimization as MSO. Each model was trained multiple times with GMRT5x3 and GMRT10x3 training samples. After extensive tests, some *Stable-Baselines3* standard parameters have been modified to improve policy learning (see Appendix Table 11). During successive modification and identification of proper hyperparameters, PPO was significantly more robust to adjustments. In contrast to A2C, PPO has generally shown good performance even with different settings. A2C sometimes delivered bad results, so choosing a proper parameter setting was very time-consuming. A possible explanation for this might be that A2C is more sensitive to the hyperparameters than PPO.

Figure 5 shows an overview of the agents' performances quantified by overall reward obtained during the training process for  $F$  and  $C_{max}$  minimization as MSO (left graph). In the first training half, all models experienced significant growth, which eventually slowed during the exploitation phase. Independently from the MSO, the learning curves share a similar pattern, where the PPO learning curve has a steeper slope and has significantly higher rewards in the final stages. Another interesting finding is, that both models consistently earn more reward for  $F$  minimization in comparison to  $C_{max}$  minimization. This result may be explained by the fact that the weight  $w = 0.5$  has an effect on the outcome. According to Goren and Sabuncuoglu (2008), greater priority should be given to stability in the context of  $C_{max}$  minimization. Furthermore, robustness is more insensitive in this case, which can lead to poorer results with this weighting. In summary, the observed difference in performance is complementary to the authors' recommendation.

Moreover, the obtained predictions (average  $\Lambda$  values) for the fully trained DRL models are visualized in form of a box

chart (right graph). Statistical details are added in Appendix Table 12. For all test samples and for both MSO, PPO generally achieved better average  $\Lambda$  values and lower standard deviations. This also reflects the patterns of rewards achieved in the training processes, with the improvement being more pronounced in the  $F$  case than in the  $C_{max}$  case. The standard deviation was less than 0.043 in each case, with PPO tending to have slightly less scatter with this metric. Therefore, in some cases, the  $\Lambda$  values deteriorated compared to the baseline schedule. Nevertheless, the average robustness and stability could be improved by the agents, whereby PPO has outperformed A2C. In addition, the PPO model was handier to train and better able to learn policy, because it was more independent of hyperparameters.

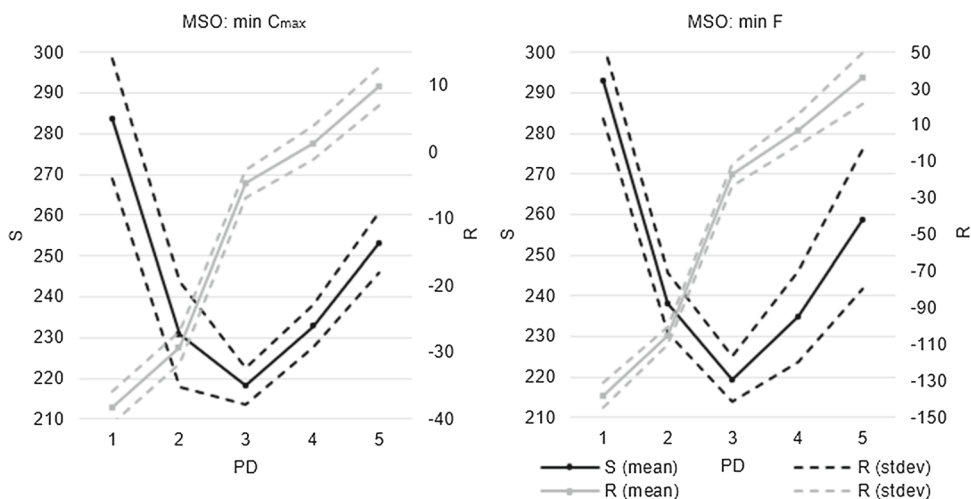
The results in this section indicate that DRL is a suitable method for robust-stable scheduling in dynamic manufacturing environments. However, with respect to DRL design and to the reduced and abstracted problem, further investigations must be carried out. First, this study was limited to the manipulation of processing times (stretching or compressing operations). With respect to the large scope, other approaches could not be considered. Consequently, more research is required to analyze and implement other paradigms of robust-stable scheduling, such as generating proactive schedules by creating neighborhood solutions through resource flipping or changing sequences by operation swapping. Second, it must be examined how an agent performs in different environments with different parameters. A related scalability analysis is carried out in Section “[Scalability investigation](#)”. In advance, the next section moves on to discuss the behavior of the PPO agent and to answer the question: Which action has been chosen in which situation and what are the consequences?

### DRL agent behavior analysis

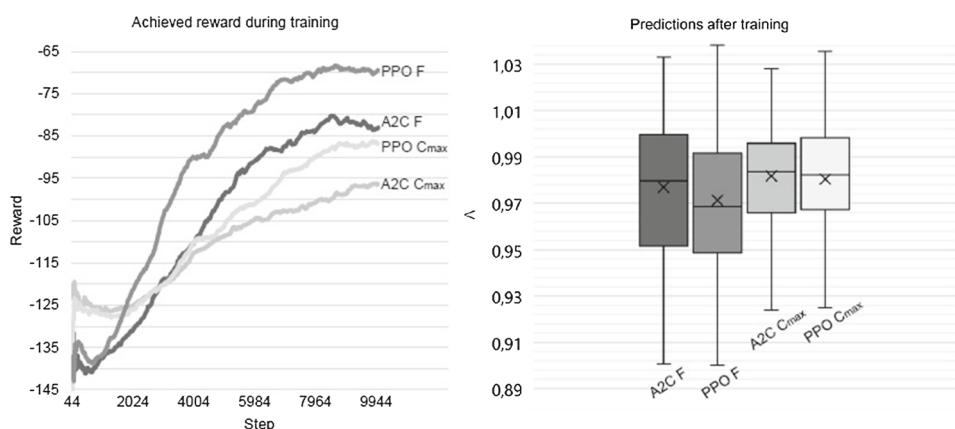
The following analysis examines the DRL agent's decision behavior and draws conclusions about the optimization metrics (see experiment question 3). It was analyzed, which actions based on which observations were chosen by the agent and how did this affect robustness and stability. The fully trained PPO agent was analyzed based on GMRT10x3 instances in the context of  $F$  minimization. From 500 episodes, all actions chosen by the agent were compared with the situational state observation.

Looking at Fig. 6, it is apparent that the agent did not choose the actions evenly: Stretching (applying  $PD_4$ ) was applied significantly more often, whereby the realistic value ( $PD_3$ ) was hardly retained. This result may be explained by the fact that the baseline schedules generated are always too optimistic. This corresponds to the expectation in accordance with the analysis in Section “[Action space analysis](#)”: The use of  $PD_3$  always implies a negative robustness. However,

**Fig. 4** Different PD and their effect on robustness and stability. Results observed by GMRT5x3 with  $F$  and  $C_{max}$  minimization as MSO. In both MSO cases, the robustness/stability conflict becomes transparent. Good stability is obtained with realistic expected values ( $PD_3$ ). Good robustness can be achieved with more conservative values ( $PD_4$ )



**Fig. 5** PPO vs. A2C benchmark for  $F$  and  $C_{max}$  minimization as MSO ( $w = 0.5$ ). Average rewards achieved during training over time (left graph) and best model’s predictions after training (right graph). All models were trained and tested with GMRT5x3 and GMRT10x3 samples. PPO outperforms A2C in terms of obtained rewards and predictions after training ( $\Delta$  values)



these results were predictable, since robustness and stability have to be balanced with  $w = 0.5$ . On the other side, the application of compressing (applying  $PD_2$ ) is particularly interesting. Overall viewed and in comparison to  $PD_3$  and  $PD_4$ ,  $PD_2$  has a negative impact on robustness and stability. Nevertheless, the agent selected compression in about 15% of all cases.

In a further analysis, we examined in which situations the agent tends to compress operations. For this purpose, we have generated a classification decision tree based on state-action tuples. Techniques based on decision trees are a suitable method to (approximately) explain the behavior of DRL agents (Ding et al., 2020). In order to reduce complexity, stretching actions were not taken into account. Figure 7 shows an excerpt of the first decision tree leaves. Even if the Gini coefficients are relatively even, the essential pattern is emerging that be interpreted as follows: If the agent assumes that (1) an operation will end earlier than planned and (2) that the operation tends to be on the critical path, the operation is compressed by the agent. Such a situation occurs especially, when predecessor operations have already been stretched. Thus, compressing the operation compensates the

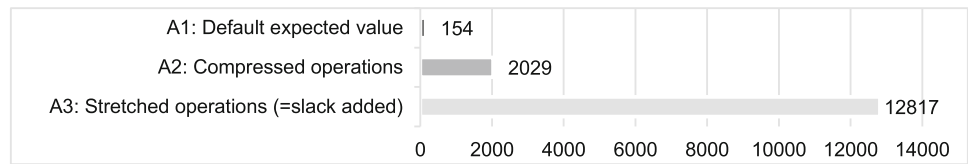
effect of an “overflow” of the current operation and all successor operations.

At first glance, this behavior seems trivial due to the reward design (see Section “DRL design”), but it may indicate a problem related to the stability metric. The stability metric used in this study and by most authors only considers operation end times (see Eq. 2). And if, in practice, target processing times are shortened in order to cushion other operations, this can lead to stress and pressure. This observation may support the hypothesis that the widely used stability metric is too imprecise. We suggest qualitatively, that stability should be characterized by the fact that re-scheduling procedures can be avoided as often as possible. The more stable the plan, the less often re-scheduling occurs or the less impact re-scheduling causes. In particular, the impact must be suitably quantified.

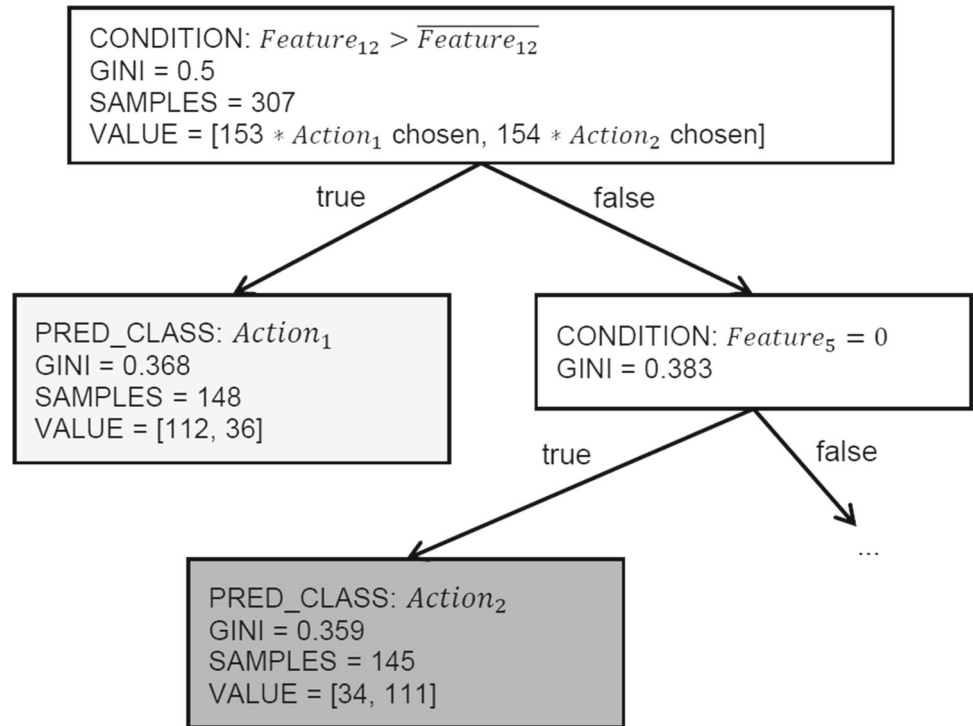
**DRL performance benchmark (PPO vs. SARS)**

In this experiment we analyzed the PPO agent performance in comparison to a SARS algorithm roughly based on Yu et al. (2021) (see experiment question 4). The SARS algorithm has been implemented as follows: In every iteration, a random

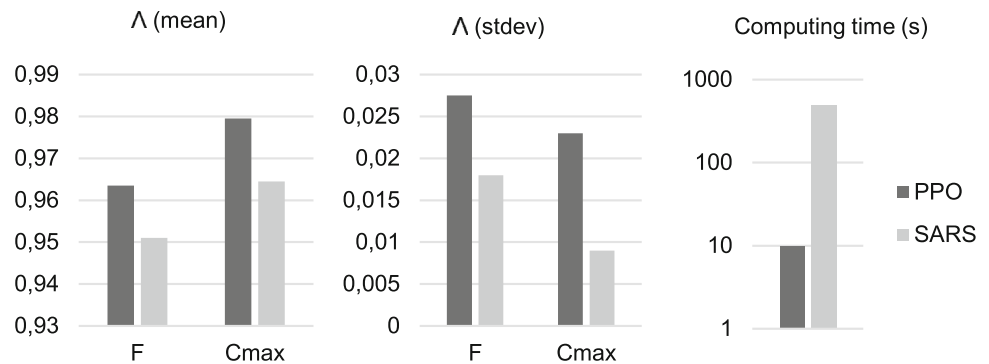
**Fig. 6** Actions chosen by the agent ( $F$  minimization as MSO in the context of GMRT10x3). The agent stretched most of the operations and compressed them less often. The default expected value (realistic operation duration) was hardly selected



**Fig. 7** Decision process whether operations are compressed (CART decision tree). The agent tends to choose compression when a critical path operation may end earlier than originally planned



**Fig. 8** PPO vs. SARS benchmark for  $F$  and  $C_{max}$  minimization as MSO ( $w = 0.5$ ) in the context of GMTR5x3 and GMRT10x3. In this analysis, average  $\Lambda$  values achieved, scatter and computing time are compared. The computing time is shown on a logarithmic axis. PPO achieves about 2% worse results than SARS, but only requires about 2% computing time

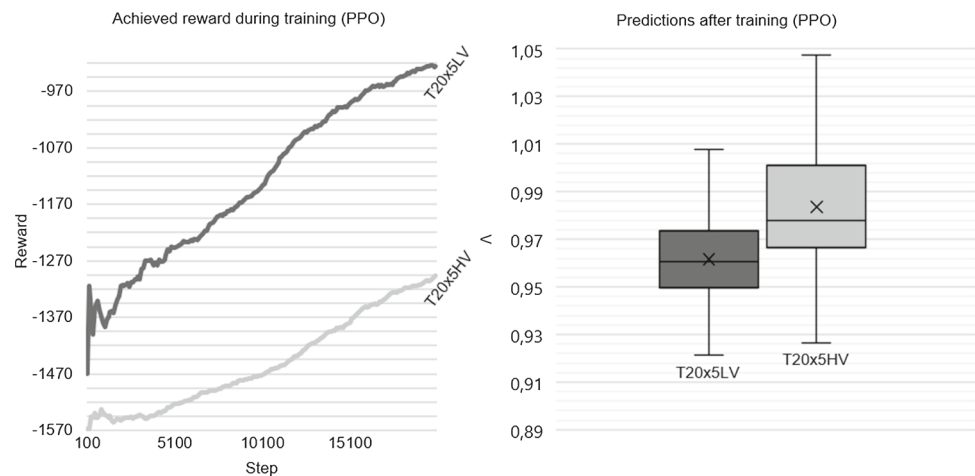


operation is randomly stretched or compressed according to the action space options (see Appendix Algorithm 3). The number of cooling steps was set to 50, which already required a computation time of about 8 minutes. This is due to the fact that the step-by-step evaluation performed 10,000 MCE each time, which significantly increased the computing time. All five steps were checked to see whether an improvement had occurred. If not, the current solution has been reverted to the best solution found. This performance analysis examined the relationship between average  $\Lambda$  value achieved, the related

scatter in the form of standard deviation, and approximate computation time in seconds.

Figure 8 shows the obtained results summarized for GMRT10x3 and GMRT5x3. Statistical details can be found in Appendix Table 13. What stands out in these results is the dominance of the DRL agent in terms of computing time. The DRL Agent required about 10 seconds to generate a proactive schedule and to evaluate via MCE. Without evaluation, the time could be further reduced to less than one second. The average  $\Lambda$  values and scatter obtained by the PPO agent were

**Fig. 9** PPO performance for  $F$  minimization as MSO ( $w = 0.5$ ) in the context of T20x5LV and T20x5HV data sets. Average rewards achieved during training over time (left graph) and best model's predictions after training (right graph). Lower uncertainty variances (T20x5LV) lead to better performances during and after training



only slightly worse in all cases. This can be an indication that the agent needs even better features or training sequences for better generalization. However, this potential for improvement is not considered in detail in this feasibility study. From the graph, it can be seen that in the case of  $C_{max}$  minimization, significantly poorer average results were also achieved with the SARS implementation. This supports the hypothesis that this is an effect of the weight  $w = 0.5$  and not attributed to the DRL agent or training process design (see Section “DRL policy learning benchmark (PPO vs. A2C)”). Interestingly, the scattering behaves in the opposite way: Both the agent and the metaheuristic each achieve a better standard deviation in the  $C_{max}$  case. This can be explained by the greater variability of the  $F$  objective, which has already been observed and discussed in Section “Action space analysis”. The more operations are stretched in terms of conservative, robust planning, the greater the scattering of the robustness achieved in the  $F$  case (see Fig. 4).

Based on the previous findings, these results support the idea that DRL is a viable and efficient method to generate proactive schedules, although there is room for improvement. The greatest advantage of the proposed DRL approach lies in the low computing time. After a proper training, it obtains about 98% of the result quality in about 2% of the time compared to traditional methods such as metaheuristics. This can be explained by the fact, that a DRL agent internalized the behaviour of the stochastic simulation model within the training to infer on robustness and stability. Conventional probabilistic methods such as metaheuristics have no way of storing experiences and hypotheses that lead to the situational selection of suitable actions. Instead, it is necessary to evaluate successively, which takes up a lot of computing time due to the complex MCE.

As mentioned in the previous parts, there are some disadvantages or improvement potentials that can be investigated in further studies. The central research question raised in this section is: How is it possible to outperform metaheuristics in

other performance criteria such as mean scores and scatter? In particular, it should be examined whether hyperparameters, training processes, actions, rewards and observations can be further improved. Finally, the scalability of the DRL design is evaluated in the next section.

### Scalability investigation

In order to assess the applicability of the proposed approach to other environments, repeated measurements of the learning and prediction performance were conducted (see Fig. 9 and Appendix Table 14). It was analyzed how the agent handles a larger problem input (more machines, more jobs) and how the extent of uncertainties affects robustness and stability. With respect to the large scope, we have focused on  $F$  minimization as MSO and PPO as DRL technology. With minor modifications, we were able to make the agent operational for a larger scaled environment. The training duration was increased to 20,000 steps and the final reward was distributed by a factor of ten. This ensured the higher weighting of the final reward compared to the sum of the intermediate rewards.

The charts below illustrate the average reward achieved during training (left graph) and the prediction quality after training completion (right graph). The agents experienced a learning curve in both scenarios (low and high uncertainty) and generated comparatively good proactive schedules after training. What is striking about the training performances is the significantly different reward scores obtained. At higher uncertainty variances (T20x5HV data set), the agent has a significantly lower slope of rewards over time, possibly influenced by the MCE: The greater the impact of uncertainties, the more experiments have to be carried out in order to calculate precise robustness and stability metrics. In addition, it can be assumed that the agent has to explore more during training and also experiences many deterioration effects. Likewise, the fully trained agent generated significantly bet-

ter proactive schedules in the case of low uncertainty. With regard to the  $\Delta$  values, better means, less scattering and fewer outliers could be achieved with T20x5LV. Interestingly, the results were even better than in the previous GMRT experiments. This is an indicator that the proposed method can also be scaled to larger flow shops. However, an increase in uncertainty results in a loss of obtainable robustness and stability. Conclusively, the degree of uncertainty has a greater effect than the number of planning objects such as machines or jobs.

A major disadvantage is still, that an agent must be trained for new situations. Future work should analyze how an agent can be better generalized for a varying number of planning objects. It is particularly important to design the reward and the observation space even more universally. Due to the need for more and more robustness, flexibility and responsiveness in innovative logistic systems (Jafari et al., 2022; Monostori, 2018) this also refers to other scheduling domains and associated MSO. In addition to related production organizations such as flexible job shops or more complex environments such as multi-resource shops, robust-stable scheduling can also be used, for example, in route planning, project management or for crew scheduling. From a mathematical point of view, it is also about allocating resources with tasks on the timeline. Consequently, the robustness and stability metrics defined as well as the design of the action space remain universally applicable. However, the observation space responsible for describing the environmental constraints has to be adapted for each specific problem in an extensive feature engineering process. Moreover, the proactive consideration of other types of dynamic events and especially external events such as new job arrivals is of scientific and practical relevance.

## Conclusions and future research

The presented research examined, how DRL can be applied in the proactive stage for robust-stable scheduling to absorb uncertainties in advance. For this purpose, a DRL concept was developed, where scheduled operations were stretched or compressed in their time in order to optimize the competing metrics, robustness and stability. The metrics were collected in the course of DES-based MCE for dynamic flow shops, whereby uncertain processing times and machine repair times were given by triangular distributions. The study was primarily set out to analyze the effectiveness, efficiency and scalability of DRL. After extensive numerical experiments, the findings suggest that DRL, and especially PPO, is a viable method to generate proactive schedules in near real-time. PPO comes about 98% close to SARS, but requires only 2% of its computing time after a successful training. This is a great advantage for the application in time-critical reactive

environments. Moreover, it could be shown that the DRL agents can also learn and predict successfully after varying jobs, machines and uncertainties. These findings will be of interest to researchers and practitioners and could be used to develop proactive methods by making them more efficient and intelligent.

The major restriction of this work was proof-of-concept DRL design focusing the proactive stage for  $F$  and  $C_{max}$  minimization as MSO. In summary, the following wide range of future research questions can be derived from the limitations of this study:

- **Aggregated robust-stable scheduling.** A limitation of the proposed approach is the decomposition of MSO optimization and proactive planning in two consecutive steps. It could be analyzed which positive and undesirable effects are related to an aggregation of both stages (see Section “[Proposed approach](#)”).
- **DRL-based re-scheduling.** This article does not include an analysis of how to combine proactive planning and reactive planning. Further work could analyze and develop a hybrid approach of predictive-reactive scheduling considering DRL on one or both stages (see Section “[Proposed approach](#)”).
- **Robustness/stability trade-off.** The conflict between robustness and stability must be formally explained in more detail (see Section “[Action space analysis](#)”).
- **Continuous actions.** Future work could improve the agent’s precision by utilizing continuous rather than discrete action spaces (see Section “[Action space analysis](#)”).
- **Neighborhood-based scheduling.** It would be interesting how robust-stable neighborhood solutions can be generated using DRL (see Section “[DRL policy learning benchmark \(PPO vs. A2C\)](#)”).
- **Balancing robustness and stability.** As far as we know, general situational rules for weighting robustness and stability with regard to MSO and practical requirements have not yet been established (see Section “[DRL policy learning benchmark \(PPO vs. A2C\)](#)”).
- **Stability metric improvement.** It could be identified that the frequently used stability metric in particular can lead to undesirable effects. Future research should develop methods that focus directly on minimizing re-scheduling situations or associated effects (see Section “[DRL agent behavior analysis](#)”).
- **Agent performance improvement.** Further investigations could analyze how DRL can obtain even better end results than traditional metaheuristics and how to reduce the prediction scatter (see Section “[DRL performance benchmark \(PPO vs. SARS\)](#)”).
- **Other scheduling contexts.** It would be interesting to evaluate the applicability of other scheduling models,



MSO or different types of dynamic events (see Section “Scalability investigation”).

- **Agent generalizability.** Further research is required to make the agents more reliable and independent of the environment (see Section “Scalability investigation”).
- **Practical application.** Due to the scope, no evaluation of practical use could be carried out in this work. In this context, future studies should address in-situ simulations, benchmarks and in-depth analysis of robust-stable methods in practical environments.

**Author Contributions** FG: conceptualization, methodology, formal analysis, investigation, writing—original draft, writing—review and editing; AM: investigation (related work analysis), writing—original draft, writing—review and editing; PR: funding acquisition, resources; ST: supervision.

**Funding** Open Access funding enabled and organized by Projekt DEAL. This work was supported as part of the joint research project *Human-centered Smart Service Lab/Predictive Scheduling* (with project number [EFRE-030018] of the European Regional Development Fund) which is funded by the federated state North Rhine-Westphalia, Germany. Project description available on <https://www.fh-bielefeld.de/forschung/forschungsprojekte/aktuelle-projekte-fb-3/reusch-predictive-scheduling> (German language).

**Code availability** Code with test instances and experiment results available on: <https://doi.org/10.17605/OSF.IO/SXM3Q>.

## Declarations

**Conflict of interest** The authors have no conflict of interest to declare that are relevant to the content of this article.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## Appendix

See Tables 6, 7, 8, 9, 10, 11, 12, 13, 14 and Algorithm 3.

**Table 6** GMRT uncertain processing times  $D_{k,PC}$  for machine  $k$  and product category  $c$  (triangularly distributed)

Constant	Value
$D_{1,1}$	(149, 150, 155)
$D_{2,1}$	(165, 167, 170)
$D_{3,1}$	(90, 95, 102)
$D_{1,2}$	(165, 168, 172)
$D_{2,2}$	(205, 207, 212)
$D_{3,2}$	(195, 198, 200)

**Table 7** GMRT failure probabilities  $P(F_k)$  and triangular machine repair time distributions  $Q_k$

Constant	Value
$P(F_1)$	0.05
$P(F_2)$	0.15
$P(F_3)$	0.1
$Q_1$	(35, 40, 48)
$Q_2$	(25, 27, 30)
$Q_3$	(30, 35, 40)

**Table 8** Uncertainty modeling for modified Taillard instances (LV: low uncertainty variance, HV: high uncertainty variance)

Uncertain parameter	Value
$D_{i,k}$	LV: $(p_{i,k} * 0.94, p_{i,k}, p_{i,k} * 1.06)$ HV: $(p_{i,k} * 0.88, p_{i,k}, p_{i,k} * 1.12)$
$P(F_k)$	LV: {0.04, 0.06, 0.1} HV: {0.07, 0.09, 0.15}
$Q_k$	LV: {(10, 15, 20), (8, 16, 24)} HV: {(20, 25, 35), (25, 32, 40)}

Product categories are not considered in comparison to the GMRT instances. Instead, every job has its own triangularly distributed uncertain processing time  $D_{i,k}$ . On initializing the modified test instances, machine failure probability  $P(F_k)$  and repair time  $Q_k$  have been selected randomly from the sets defined below

**Table 9** Plan duration (PD) parameters for the subsequent method to balance robustness and stability (see Section “Subsequent robustness and stability optimization”)

PD	Processing time expected value	Comment
$PD_1$	$E(D_{k,PC}) - \frac{\sqrt[2]{V(D_{k,PC})}}{2}$	Very optimistic
$PD_2$	$E(D_{k,PC})$	Optimistic
$PD_3$	$E(D_{k,PC}) + P(F_k)E(Q_k)$	Realistic
$PD_4$	$E(D_{k,PC}) + P(F_k)E(Q_k) + \frac{\sqrt[2]{V(D_{k,PC})}}{4} + P(F_k)\frac{\sqrt[2]{V(Q_k)}}{4}$	Conservative
$PD_5$	$E(D_{k,PC}) + P(F_k)E(Q_k) + \sqrt[2]{V(D_{k,PC})} + P(F_k)\sqrt[2]{V(Q_k)}$	Very conservative

Basis for the specification of the DRL action space.  $PD_3$  corresponds to the expected value including probable repair time.  $PD_2$  corresponds to the expected value without repair time. Within the bounds of the aggregated distribution, the further values were set even more optimistically or conservatively by subtracting or adding a proportion of the standard deviation

**Table 10** Effects of PD on robustness and stability, differentiated according to makespan and total flow time optimization

PD	MSO	Robustness			Stability		
		$R$	$R_\sigma$	$R_{min}$	$S$	$S_\sigma$	$ R  + S$
$PD_1$	$C_{max}$	- 38.35	2.38	- 44.64	283.8	14.63	322.15
$PD_2$	$C_{max}$	- 29.19	2.39	- 34.83	230.67	12.98	259.86
$PD_3$	$C_{max}$	- 4.68	2.1	- 9.43	218.06	4.37	222.74
$PD_4$	$C_{max}$	1.35	2.52	- 4.37	232.79	5.34	234.14
$PT P_5$	$C_{max}$	9.89	2.84	3.05	253.28	7.26	263.17
$PD_1$	$F$	- 137.79	6.93	- 150.99	293.06	9.5	430.85
$PD_2$	$F$	- 105.32	4.6	- 112.73	238.22	7.46	343.54
$PD_3$	$F$	- 16.97	5.88	- 25.88	219.39	5.59	236.36
$PD_4$	$F$	7.56	8.23	- 5.7	234.85	11.15	242.41
$PT P_5$	$F$	35.93	14.29	17.12	258.88	17.19	294.81

The data presented here was collected in the context of GMRT5x3

**Table 11** DRL hyperparameters for the PPO/A2C benchmark

Hyperparameter	Value
Overall training steps	$10^4$
Network update interval (steps)	22
Learning rate (linearly decreasing)	$10^{-4}$ (PPO) / $10^{-3}$ (A2C)
Discount factor	0.99
Activation function	Rectified Linear Unit (ReLU)
Network architecture	- Input layer: min. 12 neurons (depending on the feature encoding) - 1st hidden layer (shared): $2^9$ neurons - 2nd hidden layer (value net): $2^7, 2^6$ neurons - 2nd hidden layer (policy net): $2^6$ neurons - Output layer: 3 neurons (corresponding to $PD_2, PD_3, PD_4$ )

**Table 12** PPO vs. A2C benchmark for  $F$  and  $C_{max}$  minimization as MSO ( $w = 0.5$ )

Model	Data	MSO: min $F$		MSO: min $C_{max}$	
		$\Lambda$	$\Lambda_\sigma$	$\Lambda$	$\Lambda_\sigma$
PPO	GMRT5x3	.969	.027	.972	.03
PPO	GMRT10x3	.958	.028	.987	.016
PPO	GMRT5x3 GMRT10x3	.971	.029	.98	.023
A2C	GMRT5x3	.983	.043	.976	.028
A2C	GMRT10x3	.967	.031	.991	.017
A2C	GMRT5x3 GMRT10x3	.977	.035	.982	.026

Predictions by the fully trained DRL models were obtained for GMRT5x3 and GMRT10x3 (separate and mixed). Mean  $\Lambda$  values and standard deviations are presented for each model and MSO

**Algorithm 3** SARS candidate creation (pseudocode snippet)

```

 $o \leftarrow \text{choice}(O)$ 
2:  $a \leftarrow \text{choice}(A)$ 
    $o_{dur} \leftarrow \text{applyAction}(a)$ 
4:  $\Lambda \leftarrow \text{MCE}(O)$ 
   return  $\Lambda$ 

```

▷ Choose a random operation from the set of all scheduled operations  $O$   
 ▷ Get random action from action space  $A$   
 ▷ Stretch or compress the operation duration  
 ▷ Conduct MCE and measure robustness  $R$  and stability  $S$

**Table 13** SARS results for  $F$  and  $C_{max}$  minimization as MSO ( $w = 0.5$ ) in the context of GMTR5x3 and GMRT10x3

Method	Data	$t$	MSO: min $F$		MSO: min $C_{max}$	
			$\bar{\Lambda}$	$\Lambda_{\sigma}$	$\bar{\Lambda}$	$\Lambda_{\sigma}$
SARS	GMRT5x3	$\approx 500$	.955	.017	.948	.01
SARS	GMRT10x3	$\approx 500$	.947	.019	.981	.008

**Table 14** PPO after-training predictions on T20x5LV and T20x5HV instances for  $F$  minimization as MSO ( $w = 0.5$ )

Method	Data	$\bar{\Lambda}$	$\Lambda_{\sigma}$
PPO	T20x5LV	.961	.019
PPO	T20x5HV	.984	.027

The degree of uncertainty has an effect on the achievable robustness and stability

## References

- Al-Behadili, M., Ouelhadj, D., & Jones, D. (2019). Multi-objective biased randomised iterated greedy for robust permutation flow shop scheduling problem under disturbances. *Journal of the Operational Research Society*, 71(11), 1847–1859. <https://doi.org/10.1080/01605682.2019.1630330>
- Bougeret, M., Pessoa, A. A., & Poss, M. (2019). Robust scheduling with budgeted uncertainty. *Discrete Applied Mathematics*, 261, 93–107. <https://doi.org/10.1016/j.dam.2018.07.001>
- Davenport, A. J., Gefflot, C., & Beck, J. C. (2001). Slack-based techniques for robust schedules. In *Proceedings of the sixth european conference on planning (ECP-2001)*.
- de Vonder, S. V., Demeulemeester, E., & Herroelen, W. (2007). A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10(3), 195–207. <https://doi.org/10.1007/s10951-007-0011-2>
- Ding, Z., Hernandez-Leal, P., Ding, G. W., Li, C., & Huang, R. (2020). [arXiv:2011.07553](https://arxiv.org/abs/2011.07553).
- François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3–4), 219–354. <https://doi.org/10.1561/22000000071>
- Gonzalez-Neira, E. M., Montoya-Torres, J. R., & Jimenez, J.-F. (2021). A multicriteria simheuristic approach for solving a stochastic permutation flow shop scheduling problem. *Algorithms*, 14(7), 210. <https://doi.org/10.3390/a14070210>
- Goren, S., & Sabuncuoglu, I. (2008). Robustness and stability measures for scheduling: Single-machine environment. *IIE Transactions*, 40(1), 66–83. <https://doi.org/10.1080/07408170701283198>
- Goren, S., Sabuncuoglu, I., & Koc, U. (2011). Optimization of schedule stability and efficiency under processing time variability and random machine breakdowns in a job shop environment. *Naval Research Logistics (NRL)*, 59(1), 26–38. <https://doi.org/10.1002/nav.20488>
- Hatami, S., Calvet, L., Fernandez-Viagas, V., Framinan, J. M., & Juan, A. A. (2018). A simheuristic algorithm to set up starting times in the stochastic parallel flowshop problem. *Simulation Modelling Practice and Theory*, 86, 55–71. <https://doi.org/10.1016/j.simpat.2018.04.005>
- Jacoboni, C., & Lugli, P. (1989). The Monte Carlo method for semiconductor device simulation. *Springer Vienna*. <https://doi.org/10.1007/978-3-7091-6963-6>
- Jafari, H., Ghaderi, H., Malik, M., & Bernardes, E. (2022). The effects of supply chain flexibility on customer responsiveness: The moderating role of innovation orientation. *Production Planning & Control*. <https://doi.org/10.1080/09537287.2022.2028030>
- Jorge Leon, V., David Wu, S., & Storer, R. H. (1994). Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5), 32–43. <https://doi.org/10.1080/07408179408966626>
- Juan, A. A., Barrios, B. B., Vallada, E., Riera, D., & Jorba, J. (2014). A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times. *Simulation Modelling Practice and Theory*, 46, 101–117. <https://doi.org/10.1016/j.simpat.2014.02.005>
- Juan, A. A., Faulin, J., Grasman, S. E., Rabe, M., & Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives*, 2, 62–72. <https://doi.org/10.1016/j.orp.2015.03.001>
- Kardos, C., Laflamme, C., Gallina, V., & Sihm, W. (2021). Dynamic scheduling in a job-shop production system with reinforcement learning. *Procedia CIRP*, 97, 104–109. <https://doi.org/10.1016/j.procir.2020.05.210>
- Kenworthy, L., Nayak, S., Chin, C., & Balakrishnan, H. (2021). *NICE: Robust scheduling through reinforcement learning-guided integer programming*. [arXiv:2109.12171](https://arxiv.org/abs/2109.12171).
- Liu, C.-L., Chang, C.-C., & Tseng, C.-J. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *IEEE Access*, 8, 71752–71762. <https://doi.org/10.1109/access.2020.2987820>
- Liu, F., Wang, S., Hong, Y., & Yue, X. (2017). On the robust and stable flowshop scheduling under stochastic and dynamic disruptions. *IEEE Transactions on Engineering Management*, 64(4), 539–553. <https://doi.org/10.1109/tem.2017.2712611>
- Mailliez, M., Battaia, O., & Roy, R. N. (2021). Scheduling and rescheduling operations using decision support systems: Insights from emotional influences on decision-making. *Frontiers in Neuroergonomics*. <https://doi.org/10.3389/fnrgo.2021.586532>

- Minguillon, F. E., & Stricker, N. (2020). Robust predictive—Reactive scheduling and its effect on machine disturbance mitigation. *CIRP Annals*, 69(1), 401–404. <https://doi.org/10.1016/j.cirp.2020.03.019>
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). *Asynchronous methods for deep reinforcement learning*. arXiv:1602.01783.
- Monostori, J. (2018). Supply chains robustness: Challenges and opportunities. *Procedia CIRP*, 67, 110–115. <https://doi.org/10.1016/j.procir.2017.12.185>
- Morales, E. F., & Zaragoza, J. H. (2012). An introduction to reinforcement learning. *Decision theory models for applications in artificial intelligence* (pp. 63–80). IGI Global. <https://doi.org/10.4018/978-1-60960-165-2.ch004>
- Moratori, P., Petrovic, S., & Vazquez-Rodriguez, J. A. (2010). Fuzzy approaches for robust job shop rescheduling. In *International conference on fuzzy systems*. IEEE. <https://doi.org/10.1109/fuzzy.2010.5584722>.
- Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4), 754–768. <https://doi.org/10.1016/j.cor.2009.06.019>
- Negri, E., Pandhare, V., Cattaneo, L., Singh, J., Macchi, M., & Lee, J. (2020). Field-synchronized digital twin framework for production scheduling with uncertainty. *Journal of Intelligent Manufacturing*, 32(4), 1207–1228. <https://doi.org/10.1007/s10845-020-01685-9>
- OpenAI. (2022a). *Getting started with gym*. <https://gym.openai.com/docs/>. Accessed May 24, 2022.
- OpenAI. (2022b). *OpenAI baselines: ACKTR & A2C*. <https://openai.com/blog/baselines-acktr-a2c/>. Accessed May 24, 2022.
- OpenAI. (2022c). *Proximal policy optimization*. <https://openai.com/blog/openai-baselines-ppo/>. Accessed May 24, 2022.
- Park, K. T., Jeon, S.-W., & Noh, S. D. (2021). Digital twin application with horizontal coordination for reinforcement-learning-based production control in a re-entrant job shop. *International Journal of Production Research*. <https://doi.org/10.1080/00207543.2021.1884309>
- Rahmani, D., & Heydari, M. (2014). Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times. *Journal of Manufacturing Systems*, 33(1), 84–92. <https://doi.org/10.1016/j.jmsy.2013.03.004>
- Salmasnia, A., Khatami, M., Kazemzadeh, R. B., & Zegordi, S. H. (2014). Bi-objective single machine scheduling problem with stochastic processing times. *TOP*, 23(1), 275–297. <https://doi.org/10.1007/s11750-014-0337-9>
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms*. arXiv:1707.06347.
- Shahrabi, J., Adibi, M. A., & Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering*, 110, 75–82. <https://doi.org/10.1016/j.cie.2017.05.026>
- Shen, X.-N., Han, Y., & Fu, J.-Z. (2016). Robustness measures and robust scheduling for multi-objective stochastic flexible job shop scheduling problems. *Soft Computing*, 21(21), 6531–6554. <https://doi.org/10.1007/s00500-016-2245-4>
- Soofi, P., Yazdani, M., Amiri, M., & Adibi, M. A. (2021). Robust fuzzy-stochastic programming model and meta-heuristic algorithms for dual-resource constrained flexible job-shop scheduling problem under machine breakdown. *IEEE Access*, 9, 155740–155762. <https://doi.org/10.1109/access.2021.3126820>
- Stable-Baselines3. (2022). *Reliable reinforcement learning implementations*. <https://stable-baselines3.readthedocs.io>. Accessed May 24, 2022.
- Su, X., Han, W., Wu, Y., Zhang, Y., & Liu, J. (2018). A proactive robust scheduling method for aircraft carrier flight deck operations with stochastic durations. *Complexity*, 2018, 1–38. <https://doi.org/10.1155/2018/6932985>
- Sundstrom, N., Wigstrom, O., & Lennartson, B. (2017). Conflict between energy, stability, and robustness in production schedules. *IEEE Transactions on Automation Science and Engineering*, 14(2), 658–668. <https://doi.org/10.1109/tase.2016.2643621>
- Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2), 278–285. [https://doi.org/10.1016/0377-2217\(93\)90182-m](https://doi.org/10.1016/0377-2217(93)90182-m)
- Vieira, G. E., Kück, M., Frazzon, E., & Freitag, M. (2017). Evaluating the robustness of production schedules using discrete-event simulation. *IFAC-PapersOnLine*, 50(1), 7953–7958. <https://doi.org/10.1016/j.ifacol.2017.08.896>
- Wang, H., Sarker, B. R., Li, J., & Li, J. (2020). Adaptive scheduling for assembly job shop with uncertain assembly times based on dual q-learning. *International Journal of Production Research*, 59(19), 5867–5883. <https://doi.org/10.1080/00207543.2020.1794075>
- Wang, W., Gao, C., & Shi, L. (2022). Robust optimization on unrelated parallel machine scheduling with setup times. *IEEE Transactions on Automation Science and Engineering*. <https://doi.org/10.1109/tase.2022.3151611>
- Wu, C.-C., Gupta, J. N. D., Cheng, S.-R., Lin, B. M. T., Yip, S.-H., & Lin, W.-C. (2020). Robust scheduling for a two-stage assembly shop with scenario-dependent processing times. *International Journal of Production Research*, 59(17), 5372–5387. <https://doi.org/10.1080/00207543.2020.1778208>
- Xiao, S., Sun, S., & Jin, J. (2017). Surrogate measures for the robust scheduling of stochastic job shop scheduling problems. *Energies*, 10(4), 543. <https://doi.org/10.3390/en10040543>
- Xiao, S., Wu, Z., & Yu, S. (2019). A two-stage assignment strategy for the robust scheduling of dual-resource constrained stochastic job shop scheduling problems. *IFAC-PapersOnLine*, 52(13), 421–426. <https://doi.org/10.1016/j.ifacol.2019.11.092>
- Xiong, J., Ning Xing, L., & Wu Chen, Y. (2013). Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics*, 141(1), 112–126. <https://doi.org/10.1016/j.ijpe.2012.04.015>
- Yu, V. F., Mauludin, A., Redi, A. A. N. P., Lin, S.-W., & Yang, C.-L. (2021). Simulated annealing with restart strategy for the path cover problem with time windows. *Mathematics*, 9(14), 1625. <https://doi.org/10.3390/math9141625>

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.