



Stochastic mixed model sequencing with multiple stations using reinforcement learning and probability quantiles

Janis Brammer¹ · Bernhard Lutz² · Dirk Neumann²

Received: 13 September 2020 / Accepted: 31 August 2021 / Published online: 28 September 2021
© The Author(s) 2021

Abstract

In this study, we propose a reinforcement learning (RL) approach for minimizing the number of work overload situations in the mixed model sequencing (MMS) problem with stochastic processing times. The learning environment simulates stochastic processing times and penalizes work overloads with negative rewards. To account for the stochastic component of the problem, we implement a state representation that specifies whether work overloads will occur if the processing times are equal to their respective 25%, 50%, and 75% probability quantiles. Thereby, the RL agent is guided toward minimizing the number of overload situations while being provided with statistical information about how fluctuations in processing times affect the solution quality. To the best of our knowledge, this study is the first to consider the stochastic problem variation with a minimization of overload situations.

Keywords Scheduling · Mixed model sequencing · Reinforcement learning · Metaheuristics · Combinatorial optimization

1 Introduction

In the mixed model sequencing (MMS) problem, different models need to be sequenced for production on an assembly line that consists of multiple stations. Each station is operated by a human worker that requires specific processing times depending on the model. A work overload occurs if a worker cannot complete a

✉ Janis Brammer
janis.brammer@volkswagen.de

Bernhard Lutz
bernhard.lutz@is.uni-freiburg.de

Dirk Neumann
dirk.neumann@is.uni-freiburg.de

¹ Volkswagen AG, Berliner Ring 2, 38440 Wolfsburg, Germany

² University of Freiburg, Rempartstr. 16, 79089 Freiburg, Germany

workpiece on time, before it leaves the boundaries of the station. While the deterministic problem is well studied (see e.g., Boysen et al. 2009), little research has been done on the problem variation with stochastic processing times. Processing times may vary due to, e.g., different worker skills, fluctuations in the accuracy of tools, or lower material quality, which can disrupt schedules based on deterministic information.

The few existing studies that have considered MMS problems with stochastic processing times have focused on minimizing the expected work overload time. “Work overload time” refers to the amount of work that is required by utility workers to assist at cycles, for which a work overload is foreseeable. The underlying assumption is that utility workers work side-by-side with regular workers, so that the processing speed can be doubled and the cycle is fully processed before it leaves the boundaries of the station. However, large European car manufacturers often apply a different policy, whereby utility workers exclusively process a cycle when a work overload is foreseeable (Boysen et al. 2011). This is due to the fact that it is often not possible to perform certain tasks side-by-side as there may not be enough space for both workers to process a workpiece. Utility workers may also not be flexible enough to instantly assist at the exact right moment that is sufficient to successfully process a cycle. In the alternative policy implemented by European car manufacturers, the utility worker first has to walk to the respective station, which causes setup costs that dominate the costs of utility work (Boysen et al. 2011). Accordingly, the goal of this problem variation is to minimize the number of work overload situations.

To the best of our knowledge, this study is the first to consider the stochastic MMS problem with the objective of minimizing work overload situations. We propose a reinforcement learning (RL) approach that generates the sequence iteratively, where actions denote the model to be sequenced next. The learning environment simulates normally distributed processing times (Dong et al. 2014; Mosadegh et al. 2017, 2020; Özcan et al. 2011) and penalizes actions that lead to work overloads with negative rewards. To make the RL agent account for stochastic processing times, we define a state representation that contains several variables which indicate if a work overload occurs given that the processing times are equal to their respective 25%, 50%, or 75% quantiles. We find that including more quantiles further reduces the number of overload situations, but the marginal decrease is underproportionate to the number of quantiles, while the time required for policy learning increases linearly. The proposed approach can theoretically be applied to all mixed model assembly lines provided that the distributions of the models’ processing times are known.

The remainder of this paper is structured as follows. Section 2 provides an overview of related work. Section 3 formalizes the stochastic MMS problem and describes the characteristics of suitable solutions. Section 4 describes our RL approach by detailing environment, state and action space, reward function, and the method for policy learning. Section 5 presents the setup of our numerical evaluation and Sect. 7 presents the results. Finally, Sect. 8 describes several directions for how our approach could be extended.

2 Related work

MMS problems have been studied in multiple variations; for instance, with U-shaped (instead of straight) assembly lines (Li et al. 2012), two-sided assembly lines (Chutima and Naruemitwong 2014), heterogeneous worker types (Aroui et al. 2017; Cortez and Costa 2015), and alternative optimization targets, including worker idle times (Bautista et al. 2016; Mosadegh et al. 2020) or product rate variation (Chutima and Naruemitwong 2014). The MMS problem should be distinguished from car sequencing (Parrello et al. 1986), where work overload is implicitly minimized by satisfying handcrafted sequencing rules. Each rule has the form $N_o : H_o$, which stipulates that, among N_o sequence positions, a maximum of H_o vehicles with option o may occur. Options can denote, for instance, a sun roof, entertainment system, or a specific motor. The more rules are satisfied, the lower the work overload. A literature review on car sequencing studies is provided by Solnon et al. (2008). Compared to car sequencing, MMS provides more flexibility in finding an optimal solution, and, thus, sequences generated with MMS cause less work overload (Golle et al. 2014). However, applying MMS requires car manufacturers to spend significant effort in collecting the processing times for each model and station, but the practicability of MMS in industry has already been demonstrated in a case study (Bautista et al. 2012).

Prior research has studied the pure MMS problem and the integrated balancing and sequencing problem (Agrawal and Tiwari 2008; Boysen et al. 2007; Dong et al. 2014; Özcan et al. 2011), where the tasks to be performed for each model must first be assigned to stations given the precedence relation, such that a given criterion is optimized for a (near-)optimal sequence. Both problems exist since they are relevant in regard to different time horizons. The balancing decision has a mid-term horizon (e.g., several months) as it specifies work content and material usage for each station, while the exact daily demands are generally not known in advance. The sequencing problem instead has a short-term horizon (e.g., one shift) as it specifies the order in which a given demand of models is produced (Boysen et al. 2009). Comprehensive literature reviews of the balancing and sequencing problems, including the considered problem variations, are provided by Boysen et al. (2007) and Boysen et al. (2009). Boysen et al. (2009) have also identified stochastic processing times as a research gap.

So far, only a few studies have relaxed the strict assumption of deterministic processing times toward stochastic processing times. Table 1 provides an overview of existing studies on the stochastic MMS problem. The seminal study by Zhao et al. (2007) proposed an approach based on Markov chains to calculate the expected work overload time. In a nutshell, this approach approximates the current positions of the workers within their stations by dividing the interval of possible positions into several subintervals. For each subinterval, the expected overload time is calculated as the average of those overload times that would occur if the worker was located at the exact lower or upper interval boundaries. Based on the method by Zhao et al. (2007) and Dong et al. (2014) proposed a simulated annealing approach for the stochastic balancing and sequencing problem

Table 1 Existing studies on mixed model sequencing problems with stochastic processing times

Study	Problem type	Approach	Optimization target
Zhao et al. (2007)	Sequencing	Markov chain	Overload time
Agrawal and Tiwari (2008)	Balancing and sequencing	Ant colony optimization	Workload balance, risk of line stoppage
Özcan et al. (2011)	Balancing and sequencing	Genetic algorithm	Workload balance
Dong et al. (2014)	Balancing and sequencing	Simulated annealing	Overload time
Mosadegh et al. (2017)	Sequencing	Greedy, modified Dijkstra's algorithm	Overload time
Mosadegh et al. (2020)	Sequencing	Simulated annealing	Overload time, idle time
This study	Sequencing	Reinforcement learning	Overload situations

with U-shaped lines and a minimization of overload time. Mosadegh et al. (2017) developed a method similar to Dijkstra's algorithm that solves the stochastic MMS problem for one stochastic station. In their recent study, Mosadegh et al. (2020) considered the MMS problem with multiple stochastic stations. The authors presented an approach based on simulated annealing, for which the parameters are selected with Q-learning.

Studies on the balancing and sequencing problem with stochastic processing times generally focus on balancing workload. Agrawal and Tiwari (2008) proposed an ant colony algorithm to simultaneously minimize variation of workload and risk of line stoppage. The risk of line stoppage for a given station is calculated as the probability quantile of the z-normalized processing time with mean and variance processing time over all tasks. Özcan et al. (2011) presented a genetic algorithm to minimize the absolute deviation of workloads (ADW), such that ADW is not exceeded according to a given confidence level.

This study proposes a RL approach for the stochastic MMS problem with a minimization of overload situations. To the best of our knowledge, our study is the first to consider the stochastic problem with the objective of minimizing overload situations instead of overload time.

3 Stochastic mixed model sequencing with a minimization of overload situations

Mixed model assembly lines allow manufacturers to exploit the advantages of flow-production, while offering a large diversified product portfolio (Boysen et al. 2009). In mixed model assembly lines, the workers process a cycle while walking within the boundaries of their station. After a workpiece is completed, the worker walks back toward the beginning of the station to process the next cycle. Consequently, the starting position of a cycle depends on the processing time of the previous cycle. If the processing times are deterministic, the approach used for sequence generation

Table 2 Variables of problem definition

Variable	Description
T	Sequence length
K	Number of stations
M	Number of different models
$y_{k,t}$	Binary variable that equals 1 if an overload occurs at station k for cycle t
$x_{m,t}$	Binary variable that equals 1 if model m is sequenced at position t
d	Demand plan $[d^1, \dots, d^m]$ that specifies the quantity d^m to be produced for model m
$w_{k,t}$	Position of worker at station k before processing cycle t
$p_{k,m}$	Stochastic processing time of model m at station k
l_k	Length of station k , one unit of station length corresponds to one unit of processing time
$b_{k,t}$	Processing time of the model scheduled at sequence position t for station k
c	Cycle time which species that a new cycle enters the conveyor belt every c units of processing time

can calculate the starting positions of the workers for all stations and cycles exactly and without uncertainty. However, processing times in real-world production may vary due to, e.g., different skills and experience among the workers, inaccurate tools, or faulty materials. Longer processing times increase the probability of work overloads for the current and all subsequent cycles at a given station. Therefore, the sequence must be generated in a way that accounts for stochastic processing times.

We consider the MMS problem with normally distributed processing times as they are a frequent choice in the literature (Dong et al. 2014; Mosadegh et al. 2017, 2020; Özcan et al. 2011). A normal distribution also reflects our idea that processing times in mixed model assembly lines should be close to their mean with high probability, while deviations from the mean are less likely. At the same time, a large positive deviation from the mean should have the same probability as a large negative deviation with the same absolute value.

In the following, we provide a mathematical problem formulation of the stochastic MMS problem with a minimization of overload situations. Subsequently, we describe the characteristics of suitable solutions.

3.1 Problem formulation

The formulation of the stochastic MMS problem with a minimization of overload situations is based on the deterministic problem introduced by Boysen et al. (2011). Table 2 provides an overview of all variables. The problem is formalized below from (1) to (13). The goal (1) is to minimize the total number of work overloads $y_{k,t}$ over all stations $k = 1, \dots, K$ and sequence positions $t = 1, \dots, T$. The sequence is modeled as several binary variables $x_{m,t}$ that equal 1 if model m is sequenced at position t . (2) and (3) ensure that $y_{k,t}$ and $x_{m,t}$ are binary. (4) ensures that each sequence position is filled with exactly one model. (5) ensures that the sequence complies with the demand plan. (6) ensures that the starting positions of the workers at the stations are non-negative. (7) sets the initial start position of all workers to zero and (8) resets the start position

after the sequence is completed. (9) specifies that the processing times are normally distributed with mean $\mu_{k,m}$ and standard deviation $\sigma_{k,m}$. (10) enforces that the sampled processing times are non-negative and less or equal to the station length l_k . The constraint $p_{k,m} \leq l_k$ is necessary since we assume closed stations, which implies that it must be possible to process each cycle within the boundaries of the respective station. One could, for instance, assume that the constraint $p_{k,m} \leq l_k$ has been considered in the balancing phase. (11) defines a variable for the processing time of the cycle t at station k . (12) and (13) ensure that the workers only move within the boundaries of their station. Besides, (13) incorporates the cycle time into the optimization problem, which specifies that a new cycle enters the conveyor belt at every c units of processing time. If $y_{k,t} = 1$, the constraints (12) and (13) are weakened, so that it becomes easier to find a solution.

$$\text{Minimize } z = \sum_{t=1}^T \sum_{k=1}^K y_{k,t} \tag{1}$$

Subject to:

$$y_{k,t} \in \{0, 1\}, \quad \forall k = 1, \dots, K; t = 1, \dots, T, \tag{2}$$

$$x_{m,t} \in \{0, 1\}, \quad \forall m = 1, \dots, M; t = 1, \dots, T, \tag{3}$$

$$\sum_{m=1}^M x_{m,t} = 1, \quad \forall t = 1, \dots, T, \tag{4}$$

$$\sum_{t=1}^T x_{m,t} = d^m, \quad \forall m = 1, \dots, M, \tag{5}$$

$$w_{k,t} \geq 0, \quad \forall k = 1, \dots, K; t = 2, \dots, T, \tag{6}$$

$$w_{k,1} = 0, \quad \forall k = 1, \dots, K, \tag{7}$$

$$w_{k,T+1} = 0, \quad \forall k = 1, \dots, K, \tag{8}$$

$$p_{k,m} \sim N(\mu_{k,m}, \sigma_{k,m}), \quad \forall k = 1, \dots, K; m = 1, \dots, M, \tag{9}$$

$$0 \leq p_{k,m} \leq l_k, \quad \forall k = 1, \dots, K; m = 1, \dots, M, \tag{10}$$

$$b_{k,t} = \sum_{m=1}^M p_{k,m} \cdot x_{m,t}, \quad \forall k = 1, \dots, K; t = 1, \dots, T, \tag{11}$$

$$w_{k,t} + b_{k,t} - l_k \cdot y_{k,t} \leq l_k, \quad \forall k = 1, \dots, K; t = 1, \dots, T, \tag{12}$$

$$w_{k,t+1} \geq w_{k,t} + b_{k,t} - l_k \cdot y_{k,t} - c, \quad \forall k = 1, \dots, K; t = 1, \dots, T. \tag{13}$$

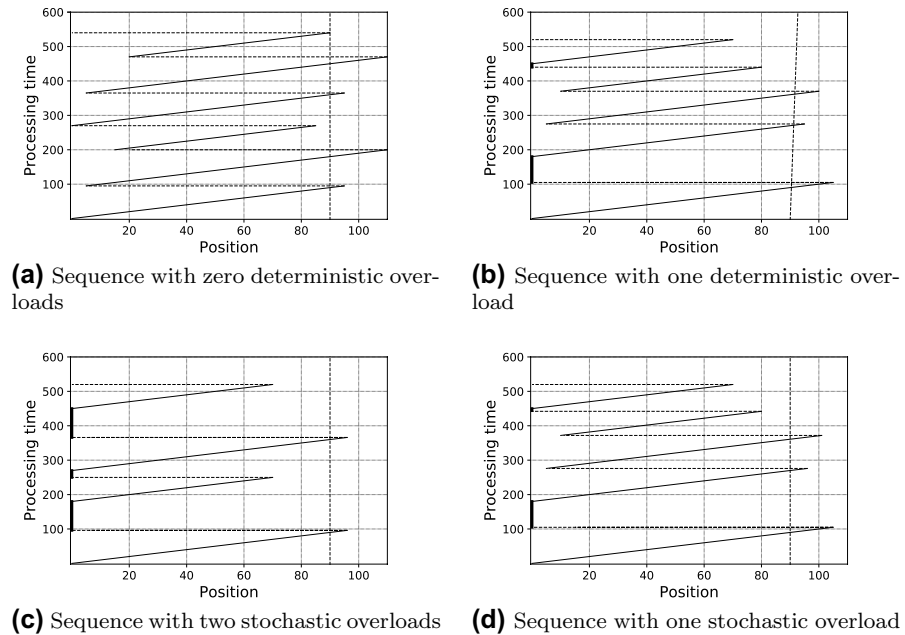


Fig. 1 Worker movements. The x-axis denotes the position of the worker with respect to the processing time on the y-axis

3.2 Characteristics of suitable solutions

We now explain what characterizes suitable solutions for the stochastic MMS problem with a minimization of overload situations. If sequences are optimized based on deterministic overloads, the resulting sequences entail tight schedules for the workers. The cycles will be completed when the workers are close the boundaries of the station, which implies that they can easily be violated if the processing times are larger than their mean. A good solution for the stochastic problem instead provides more buffer times to account for variations in the processing times. However, such buffers come at the cost of a greater number of deterministic overloads.

We provide an example to better illustrate this idea. We consider an assembly line that consists of only one station of length 110. Let the demand plan be given as $d = [2, 2, 2]$ and the means of the processing times be given as $\mu_{1,1} = 95$, $\mu_{1,2} = 105$ and $\mu_{1,3} = 70$. The cycle time is set to $c = 90$. There are two sequences given as $seq_1 = [1, 2, 3, 1, 2, 3]$ and $seq_2 = [2, 2, 1, 1, 3, 3]$. Figure 1 plots the movements of the workers for a) seq_1 and b) seq_2 if the processing times are deterministic and equal to their means. The x-axis denotes the worker position in the interval $[0, 110]$ and the y-axis denotes how much processing time has passed. seq_1 causes no deterministic overload, while seq_2 causes one deterministic overload as model $m = 2$ (with $\mu_{1,2} = 105$) is sequenced twice in a row. In particular, we observe that seq_1 always makes the worker finish a cycle close to the station boundary, while seq_2 leaves more buffers. We now assume that the processing time for model $m = 1$ increases from 95 to

96. As shown in Fig. 1c and d, seq_1 now leads to two overloads at cycles 2 and 5, while seq_2 still causes only one overload at cycle 2.

4 Reinforcement learning approach

The goal of RL is to learn a policy $\pi_\theta(a_t|s_t)$ with parameters θ that specifies which action a_t to perform at state s_t (Sutton and Barto 1998). For this purpose, the RL agent interacts with an environment and maximizes the discounted sum of rewards r_t over the learning episode. In this study, one episode corresponds to the generation of a complete sequence of one problem instance. The discrete time t reflects the current sequence position $t = 1, \dots, T$. The RL agent is trained to create the sequence incrementally. At each sequence position t , the agent evaluates the current state s_t and decides on the model $m \in \{1, \dots, M\}$ to sequence next. Accordingly, the action space A is given by the set of models

$$A = \{1, \dots, M\}. \quad (14)$$

4.1 Environment and state representation

The environment simulates the production process, including movements of workers and handling of work overloads. At each sequence position t , the environment provides the agent with the current state s_t . After the agent decided on its next action a_t , the environment simulates this action, updates the current state, and emits the reward r_t to the agent. This continues until the learning episode is finished. After this, the environment is reset and another learning episode starts. The learning process is completed when the policy has converged to a stable state.

The generated sequence must comply with the demand plan $[d^1, \dots, d^M]$, i.e., the number of cycles where model m is produced must be equal to d^m . To guide the agent toward generating valid sequences, we include the current remaining quantities d_t^1, \dots, d_t^M into the state representation s_t . If the agent decides to sequence model m in state s_t , then d_t^m is decreased by one in the next state s_{t+1} . When the sequence is empty at $t = 1$, the remaining quantities are equal to the demand plan: $d_1^m = d^m, \forall m = 1, \dots, M$.

The state representation also provides the agent with information about whether or not work overloads will occur if the processing times are equal to particular probability quantiles. For this purpose, we define a deterministic function $o_t^m(q) : [0, 1] \rightarrow \{0, 1\}$ that equals 1 if sequencing model m at sequence position t leads to at least one work overload $y_{k,t}$ given that the processing times of all stations are equal to their respective q -quantile with $q \in [0, 1]$.¹ Consider the

¹ We also performed an evaluation where $o_t^m(q)$ was defined as the total number of overloads that occur at the next sequence position. However, this resulted in inferior performance when compared to an approach that defined $o_t^m(q)$ as a binary function.

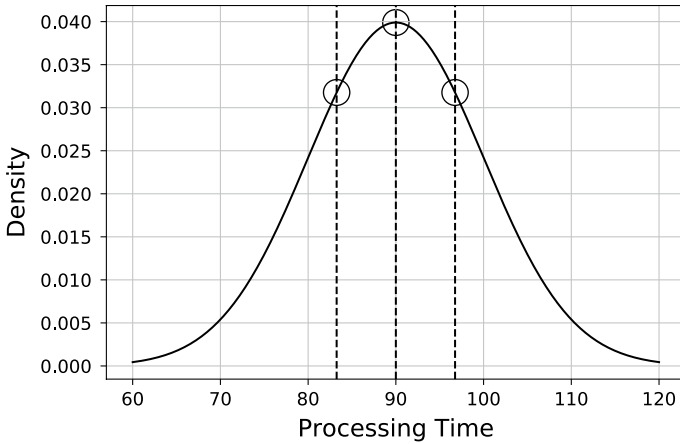


Fig. 2 Probability density function of a normal distribution $\mathcal{N}(90, 10)$. The quantiles $q_{0.25}$, $q_{0.50}$, and $q_{0.75}$ are highlighted

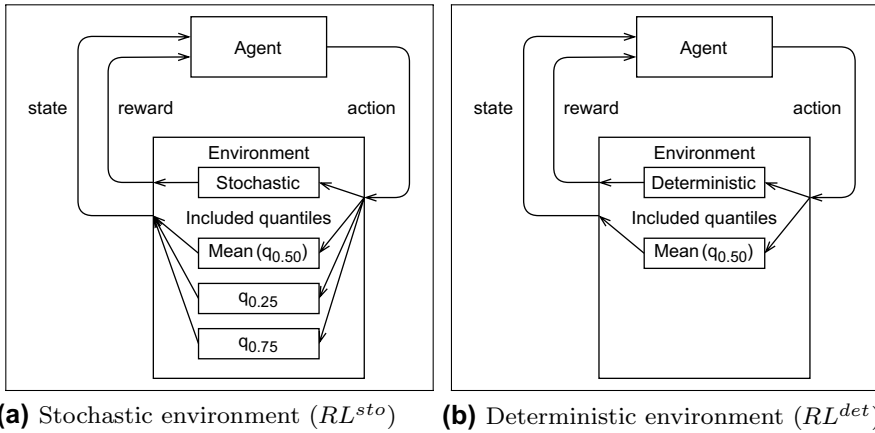


Fig. 3 Illustration of environments used in this study

following example. Let the worker of a station k with length $l_k = 120$ be at position $w_{k,t} = 30$, and the processing time of model m be distributed as $p_{k,m} \sim \mathcal{N}(90, 10)$. The worker has $l_k - w_{k,t} = 120 - 30 = 90$ units of processing time left to process cycle t consisting of model m . The worker will process the cycle in time if the processing time is less than or equal to the mean which denotes the 50% quantile: $o_t^m(q) = 0, \forall q \leq 0.50$. If the processing time is greater than the mean, a work overload will occur as there is not enough time to fully process the cycle, which implies $o_t^m(q) = 1, \forall q > 0.50$.

Based on the function $o_t^m(q)$, we implement and evaluate two environments RL^{sto} and RL^{det} as illustrated in Fig. 3. Both environments include the remaining quantities d_t^1, \dots, d_t^M in the state representation, but they differ in the reward function and the number of probability quantiles for which $o_t^m(q)$ is provided. The

stochastic environment RL^{sto} is illustrated in Fig. 3a. The reward is calculated based on stochastic processing times $p_{k,m}$ that are drawn from their respective distribution $\mathcal{N}(\mu_{k,m}, \sigma_{k,m})$. The state representation of RL^{sto} contains $o_t^m(q)$ for the 25%, 50%, and 75% quantiles.

$$s_t^{sto} = (d_t^1, \dots, d_t^M, \\ o_t^1(0.25), \dots, o_t^M(0.25), \\ o_t^1(0.50), \dots, o_t^M(0.50), \\ o_t^1(0.75), \dots, o_t^M(0.75)). \quad (15)$$

It should be noted that the state representation does not directly depend on the number of stations on the assembly line. Accordingly, a trained policy can also be applied to similar production systems with additional stations (we later provide a corresponding analysis to assess how this affects the performance).

Figure 2 shows the probability density function of a normal distribution with $\mu = 90$ and $\sigma = 10$. The 50% quantile is equal to the mean, whereas the 25% and 75% quantiles are equal to 83.25 and 96.75, respectively. We later present an analysis, where we study the effects of including $o_t^m(q)$ for more than three quantiles. The results show that this additional state information further reduces work overload situations, but the marginal benefit decreases with the number of quantiles.

As a baseline, we also implement a purely deterministic environment RL^{det} as shown in Fig. 3b. In RL^{det} , the reward is calculated deterministically based on the mean processing times. Accordingly, the state representation of RL^{det} only contains $o_t^m(q)$ for the 50% quantile.

$$s_t^{det} = (d_t^1, \dots, d_t^M, \\ o_t^1(0.50), \dots, o_t^M(0.50)). \quad (16)$$

4.2 Reward

The reward signal should guide the RL agent toward generating sequences that minimize the number of work overloads. It thus seems intuitive to reward the agent with the negative sum of work overloads at the end of each learning episode. However, this implies that the reward had to be discounted over hundreds of actions, which results in a less efficient learning process (Sutton and Barto 1998). Instead, we provide an immediate reward as the negative sum of work overloads $y_{k,t}$ that are caused by cycle t .

The agent may decide to sequence a model m , for which the remaining quantity to be produced is zero ($d_t^m = 0$). In this case, the action $a_t = m$ is invalid and the agent is punished with a negative reward of -10 . This value was determined based on a pre-study. The results of the pre-study can be found in Appendix 2 of the supplementary material. Handling invalid actions during the training process is still a challenging problem in RL (Zahavy et al. 2018). The action space cannot be altered as

this would affect the structure of the neural network. For the purpose of this study, we handle invalid actions by returning a negative reward of -10 and the same state $s_{t+1} = s_t$. This will implicitly make the agent avoid invalid actions, yet a full prevention is not guaranteed. An exemplary plot about the number of invalid actions over the learning process is provided in Appendix 3 of the supplementary material. Altogether, the reward r_t for action $a_t = m$ is defined as

$$r_t = \begin{cases} -\sum_{k=1}^K y_{k,t}, & \text{if } d_t^m > 0, \\ -10, & \text{otherwise.} \end{cases} \quad (17)$$

During real-world application, we can always ensure that the generated sequences are valid. If the agent attempts to sequence a model m with $d_t^m = 0$, we simply choose the next best valid action according to the policy $\pi_\theta(a_t|s_t)$. Invalid actions thus only affect the learning process, but they do not limit the applicability of our approach in real-world production.

4.3 Policy learning

RL alternates between generating trajectories $(s_1, a_1, r_1), \dots, (s_T, a_T, r_T)$ in terms of state-action-reward tuples with the current policy $\pi_\theta(a_t|s_t)$, and updating the policy parameters θ based on the generated data.

In this study, we implement proximal policy optimization (PPO, Schulman et al. 2017) for policy learning. PPO is a policy gradient method (Williams 1992), where the policy π_θ is learned with a neural network, so that the weights of the network denote the policy parameters θ . The neural network receives a state s_t as input and outputs a stochastic vector of size $|A|$. $\pi_\theta(a_t|s_t)$ hence equals the probability that the agent will perform action a_t in state s_t . During policy learning, actions leading to higher rewards will be assigned higher probabilities, whereas actions leading to lower rewards will be assigned lower probabilities. PPO is easy to implement and tune (Schulman et al. 2017) and is considered a state-of-the-art policy gradient method for RL (Zheng et al. 2018).

We briefly describe the functionality of PPO. All explanations are based on Schulman et al. (2017). Algorithm 1 provides the pseudocode of policy learning with PPO. Besides the action probabilities $\pi_\theta(a_t|s_t)$, the policy network also updates a value estimate $V^\theta(s_t)$ of state s_t . This value denotes the expected reward that the agent will receive from s_t to the end of the learning episode. Given a trajectory $(s_1, a_1, r_1), \dots, (s_T, a_T, r_T)$, PPO first calculates $R_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ as the sum of discounted rewards that the agent will receive between s_t and s_T , where γ denotes the discount parameter (set to 0.99). In addition, PPO calculates the estimated advantage of performing a_t in s_t as $\hat{A}_t = R_t - V^\theta(s_t)$. The rationale of the advantage estimate is that PPO aims at assigning higher probabilities to actions that lead to higher rewards than the current estimate $V^\theta(s_t)$. Finally, PPO updates its policy by maximizing the loss function $L(\theta)$.

$$\max_{\theta} L(\theta) = \sum_{t=1}^T L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 L_t^H(\theta) \quad (18)$$

The loss function $L(\theta)$ consists of three terms. The first term $L_t^{CLIP}(\theta)$ ensures that the policy parameters θ will be updated such that actions with positive advantage \hat{A} are assigned higher probabilities, and vice versa. However, PPO limits the extent of policy updates by clipping the ratio between new and old probability $\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ to the range $[1 - \epsilon, 1 + \epsilon]$. ϵ is set to 0.20 per default.

$$L_t^{CLIP}(\theta) = \min \left\{ \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, \text{clip} \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right\} \quad (19)$$

The second term $L_t^{VF}(\theta) = (R_t - V^{\theta}(s_t))^2$ denotes the error in predicting the value of state s_t . Including this term with a negative sign ensures that the update of θ also improves the estimate $V^{\theta}(s_t)$.

The third term $L_t^H(s_t)$ denotes the entropy (Shannon 1948) of the policy in state s_t : $L_t^H(s_t) = -\sum_{a_t \in \mathcal{A}} \pi_{\theta}(a_t|s_t) \log_2 \pi_{\theta}(a_t|s_t)$. Higher entropy values indicate that the probability distribution preserves randomness in the sense that the agent can still explore random actions instead of solely relying on the best action according to the current policy. The parameters c_1 (set to 0.50) and c_2 (set to 0.01) indicate the weight of the corresponding loss terms. All parameters except the number of time steps are set to their default values as stated in Appendix 1 of the supplementary material. We implement our RL approach in Python 3.6.8 using the PPO implementation from the RL framework ‘‘Stable Baselines’’ in version 2.7.0.

Algorithm 1 Policy learning with proximal policy optimization (Schulman et al. 2017)

- 1: **input:** policy π_{θ} , discount parameter γ
 - 2: initialize state value estimates $V^{\theta}(s_t)$, $\theta_{old} = \theta$
 - 3: **while** total number of timesteps not reached **do**
 - 4: Use current policy π_{θ} to generate trajectory $(s_1, a_1, r_1), \dots, (s_T, a_T, r_T)$
 - 5: **for all** $t = 1, \dots, T$ **do**
 - 6: $R_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ // calculate value of s_t
 - 7: $\hat{A}_t = R_t - V^{\theta}(s_t)$ // calculate advantage estimate
 - 8: Update θ by maximizing $L(\theta)$ based on $(s_1, a_1, R_1, \hat{A}_1), \dots, (s_T, a_T, R_T, \hat{A}_T)$
 - 9: $\theta_{old} = \theta$
-

We train the policies for 50 million time steps, where one time step corresponds to one action. This results in a total of approximately 500,000 learning episodes. Figure 4 shows the learning curves as the mean overloads per episode for both RL approaches RL^{sto} and RL^{det} on a problem instance with sequence length $T = 100$. As a baseline, the dashed lines denote the number of overloads for a simple greedy heuristic. Both policies converge to a stable state. After approximately 100,000 learning episodes, RL^{sto} and RL^{det} outperform the greedy heuristic (Boysen et al. 2011). As expected, the number of overloads per episode is smaller in the deterministic than in the stochastic environment. Recall that in the deterministic environment, the processing times always follow the mean, whereas processing times in the stochastic environment are drawn from their respective normal distributions.

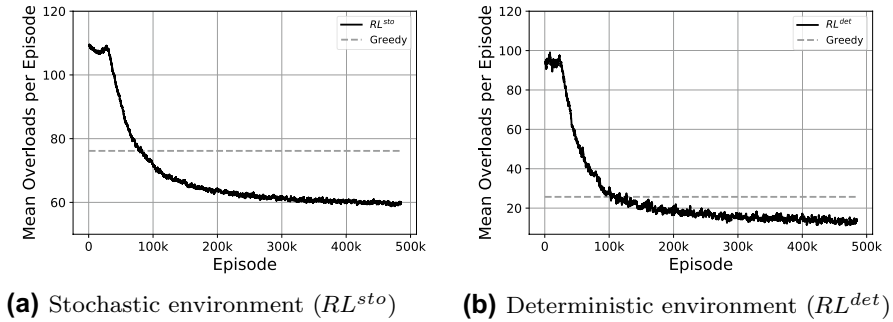


Fig. 4 Learning curves for both reinforcement learning environments

5 Numerical evaluation

5.1 Dataset

We use the dataset created by Boysen et al. (2011) that contains 216 problem layouts. The layouts can be divided into small and large layouts as shown in Table 3. The numbers of models and stations range from 5 to 30, while the sequence length varies between 15 and 300. The station length is either constant, or drawn from the uniform distributions $\mathcal{U}[85, 125]$ or $\mathcal{U}[85, 145]$. The cycle time c is fixed to 90, which means that a new cycle enters the assembly line at every 90 units of processing time. The deterministic processing times of the stations were created in two steps. First, an average processing time p_m of model m is randomly drawn from the interval $[0.75 \cdot c, c]$. Second, p_m was used to define the uniform interval from which to draw the deterministic processing time of each station $\mathcal{U}[0.5 \cdot p_m, \min\{l_k, 1.5 \cdot p_m\}]$.

Combining all settings from Table 3 yields 216 unique problem layouts with fixed deterministic processing times. For each layout, we generate 10 demand plans based on a multinomial distribution with T trials and equal probability $\frac{1}{M}$ for each model. This results in a total of 2160 deterministic problems.²

5.2 Evaluation procedure

We train one RL policy for each layout, which results in a total of 216 policies. Training one policy requires approximately ten hours on our server infrastructure that consists of two Intel(R) Xeon(R) Gold 6150 CPUs and 755 gigabytes of main memory. The time needed for policy learning is neglected as this can be done in advance for a fixed production system. To assess the performance of our approach on unseen problems, we perform three further analyses with variations in the

² The dataset can be downloaded from https://github.com/InformationSystemsFreiburg/stochastic_mms_dataset/.

Table 3 Descriptives about problem layouts

Variable	Description	Small	Large
M	Number of models	5, 10, 15	20, 25, 30
K	Number of stations	5, 10, 15	20, 25, 30
T	Sequence length	15, 20, 25	100, 200, 300
l_k	Station length constant	110, 150	
	Station length interval	[85, 125], [85, 145]	
c	Cycle time	90	

standard deviation of processing times, additional stations on the assembly line, and additional models in the demand plan.

For the evaluation, we consider normally distributed processing times (Dong et al. 2014; Mosadegh et al. 2017, 2020; Özcan et al. 2011) with means $\mu_{k,m}$ and joint standard deviation σ .

$$p_{k,m} \sim \mathcal{N}(\mu_{k,m}, \sigma) \text{ with } 0 \leq p_{k,m} \leq l_k. \quad (20)$$

The means $\mu_{k,m}$ are set to the deterministic processing times of the instances in our dataset. We require that $0 \leq p_{k,m} \leq l_k$ as negative processing times are not possible and there are no buffer areas between two adjacent stations. If $p_{k,m} > l_k$, it would be impossible to process the affected cycles within the boundaries of the station. The standard deviation σ is set to 10 in our main analysis, but we later also evaluate $\sigma \in \{5.0, 7.5, 12.5, 15\}$. To calculate the number of stochastic overloads, we evaluate 100 stochastic variations of each (deterministic) problem. In each variation, we sample the processing times from their respective distributions.

Algorithm 2 describes the evaluation procedure. We iterate over all competing approaches and problem instances. Each approach is provided a fixed cutoff time of 300 seconds to search for a solution (Boysen et al. 2011; Joly and Frein 2008; Lopes et al. 2020; Miltenburg 2002). The generated sequence is then evaluated based on 100 variations with random processing times. For this purpose, it is crucial that the sampled processing times are the same for all sequences. More precisely, if two sequences have the same model m at sequence position t , then the processing times of this model at all stations must be the same for both sequences. We therefore set a random seed (line 6) before evaluating each deterministic instance based on 100 stochastic variations. Work overloads are calculated based on the sampled processing times (lines 14, 15). The result of each stochastic variation $j = 1, \dots, 100$ is stored as a tuple $(i, \text{approach}, j, O)$ (line 16); all results are finally aggregated according to the performance metrics (line 17).

To calculate the number of overload situations, we assume that the processing times are known before the actual working steps. The number of work overloads is then calculated using the method for the deterministic problem by Boysen et al. (2011). Once a work overload is foreseeable for cycle t , the worker of the affected station walks directly to the beginning of their station to process cycle $t + 1$. The alternative would be to assume a probability threshold α until which the workers still

try to process the cycle while hoping to be successful in time. However, if a work overload is signaled too late, the utility worker may not have enough time remaining to process the cycle.

We simulate 100 stochastic variations to ensure that the sample distribution accurately reflects the population distribution. The sample means $\hat{\mu}_{k,m}$ and the sample standard deviation $\hat{\sigma}$ usually vary from the population means $\mu_{k,m}$ and the population standard deviation σ . Therefore, we calculate the 95% confidence intervals of sample mean and standard deviation for one station based on $T \in \{15, 100, 300\}$ and $\sigma = 10$. The confidence intervals of the sample mean equal $\hat{\mu}_{k,m} \pm 5.06$ for $T = 15$, $\hat{\mu}_{k,m} \pm 1.96$ for $T = 100$, and $\hat{\mu}_{k,m} \pm 1.13$ for $T = 300$. When evaluating 100 stochastic variations (i.e., 99 degrees of freedom), the confidence intervals change to $\hat{\mu}_{k,m} \pm 0.51$ for $T = 15$, $\hat{\mu}_{k,m} \pm 0.12$ for $T = 100$, and $\hat{\mu}_{k,m} \pm 0.11$ for $T = 300$. In addition, we calculate the 95% confidence interval of the sample standard deviation $\hat{\sigma}$. For one stochastic variation, the confidence intervals are $[0.73, 1.58] \times \sigma$ for $T = 15$, $[0.89, 1.16] \times \sigma$ for $T = 100$, and $[0.93, 1.09] \times \sigma$ for $T = 300$. Again, when evaluating 100 stochastic variations, the confidence intervals narrow down to $[0.97, 1.04] \times \sigma$ for $T = 15$, $[0.99, 1.01] \times \sigma$ for $T = 100$ and $[0.99, 1.01] \times \sigma$ for $T = 300$. Altogether, when performing 100 variations, sample mean and sample standard deviation deviate less than one unit of processing time from population mean and population standard deviation.

Algorithm 2 Evaluation procedure

```

1: input: instances  $\mathcal{D}$ , approaches  $\mathcal{A}$ , standard deviation  $\sigma$ 
2:  $results = \emptyset$ 
3: for all approach  $\in \mathcal{A}$  do
4:   for all  $i \in \mathcal{D}$  do
5:     seq = apply approach on instance  $i$  within cutoff time 300s
6:     set random seed to ensure comparability of different sequences
7:     for  $j = 1, \dots, 100$  do // 100 stochastic variations
8:       initialize overload matrix  $O \in \mathbb{N}^{T \times K}$  with zeros
9:       for all  $t = 1, \dots, T$  do
10:         $m = seq[t]$ 
11:        for all stations  $k = 1, \dots, K$  do
12:           $p_{k,m} = \text{sample from } \mathcal{N}(\mu_{k,m}, \sigma)$ 
13:           $p_{k,m} = \min(\max(0, p_{k,m}), l_k)$  // ensure  $0 \leq p_{k,m} \leq l_k$ 
14:           $y_{k,t}, w_{k,t+1} = \text{simulate processing of cycle } t \text{ at station } k \text{ with } p_{k,m}$ 
15:           $O_{k,t} = y_{k,t}$ 
16:        results = results  $\cup (i, \text{approach}, j, O)$ 
17: output: performance metrics based on results

```

5.3 Competing approaches

We evaluate our approach against several competing approaches from the literature as shown in Table 4. First, we implement several stochastic approaches, including the hyper simulated annealing (HSA) approach by Mosadegh et al. (2020), which

was developed for the stochastic MMS problem with a minimization of work overload time, and a multiple scenario approach [MSA, (Bent and Van Hentenryck 2004)]. Second, we evaluate several deterministic approaches based on the mean processing times. This includes a genetic algorithm, simulated annealing (Aroui et al. 2017), local search (Cortez and Costa 2015), tabu search (Boysen et al. 2011), and Gurobi version 8.0.1.

However, simply applying deterministic approaches to stochastic problems is likely to result in poor performance (Pinedo and Weiss 1987). We therefore modify the deterministic problems by artificially increasing the mean processing times

$$\mu_{k,m}^* = \mu_{k,m} + \lambda\sigma \quad (21)$$

with $\lambda \in \{0, 0.25, 0.50, 0.75, 1\}$. This should yield sequences with more deterministic overloads, but with less stochastic overload situations as the increased processing times cause larger buffer times between two subsequent cycles. For each approach, we report the results of the λ value that performed best. Accordingly, these approaches have a small advantage in the sense that we perform an ex-post selection of parameters.

In the following, we briefly explain each competing approach.

5.3.1 Greedy heuristic

Several competing approaches (HSA, TS, LS, GA, SA) use the greedy heuristic (Boysen et al. 2011) to create an initial sequence. The greedy heuristic generates the sequence incrementally. At each sequence position, the heuristic chooses the next model based on the number of work overloads that will occur if this model is sequenced next according to the deterministic processing times. Ties are broken according to (1) the maximum sum of processing times over all stations, (2) the maximum processing time at any station, and (3) the lowest index m .

5.3.2 Hyper simulated annealing

We implement hyper simulated annealing (HSA) according to Mosadegh et al. (2020). HSA was originally designed to minimize the expected work overload time, which does not necessarily also minimize the number of overload situations. However, we still implement this approach as it provides a state-of-the-art solution for mixed model assembly lines with stochastic processing times. The idea of HSA is to combine simulated annealing (SA) with Q-learning. The metaheuristic improves an initial sequence through a series of mutations. At each sequence position, one out of 16 actions is either selected randomly (with probability ϵ) or based on the current Q-learning policy. In this context, an action defines the next three mutation operators. There is a total of six mutation operators, which range from simple swaps (e.g., change models at positions (i, j)) to complex crossover operators (Kim et al. 1996). After the sequence is updated, the approach updates the Q-table and lowers the temperature of the SA. A new sequence is accepted if it outperforms the old sequence in terms of the performance metric, i.e., expected overload time. In addition, an equal

Table 4 Overview of all approaches

<i>Reinforcement learning approaches</i>	
RL^{sto}	RL trained in stochastic environment with 25%, 50%, and 75% probability quantiles
RL^{det}	RL trained in deterministic environment only with 50% probability quantile
<i>Stochastic approaches</i>	
HSA^T	Hyper simulated annealing (Mosadegh et al. 2020) with a minimization of work overload time
MSA	Multiple scenario approach (Bent and Van Hentenryck 2004)
<i>Deterministic approaches</i>	
HSA	Hyper simulated annealing (Mosadegh et al. 2020)
SA	Simulated annealing (Aroui et al. 2017)
GA	Genetic algorithm (Aroui et al. 2017)
LS	Local search (Cortez and Costa 2015)
TS	Tabu search (Boysen et al. 2011)
Gur	Mixed-integer linear programming solver Gurobi (version 8.0.1)

Initial sequences of metaheuristics are generated with the greedy heuristic (Boysen et al. 2011)

or worse sequence can still be accepted with a certain probability that depends on the temperature.

We implement HSA with two different fitness criteria. The first variation (HSA^T) minimizes the expected overload time, so that the approach is implemented like in the study by Mosadegh et al. (2020). The second variation (HSA) is based on the number of deterministic overload situations.

5.3.3 Multiple scenario approach

We implement a multiple scenario approach (MSA) according to Bent and Van Hentenryck (2004). The idea of MSA is to solve N deterministic instances individually within the cutoff time and aggregate the resulting sequences with a consensus function. For this purpose, we create a matrix $C \in \mathbb{N}^{M \times M}$ that stores how often model i is followed immediately by model j over all sequences $seq_n, n = 1, \dots, N$. Let $I(condition)$ denote the indicator function that equals 1 if $condition$ is true, and 0 otherwise. The matrix C is defined as

$$C[i, j] = \sum_{n=1}^N \sum_{t=1}^{T-1} I(seq_n[t] = i \wedge seq_n[t + 1] = j), \forall i, j = 1, \dots, M. \tag{22}$$

The first sequence position is chosen randomly. All other positions are determined based on the consensus function $f(t)$. Ties are broken according to the lower model index m .

$$f(t) = \operatorname{argmax}_m (C[t, m]) \text{ with } d^m > 0, t = 2, \dots, T. \tag{23}$$

We implement MSA with $N = 10$ based on four variations. We evaluate Gurobi or SA as underlying method to generate sequences for each problem instance. In addition, we introduce a flag that indicates whether we allow for one model to be sequenced at two subsequent sequence positions. If the flag is active, the sequences may exhibit blocks of the same model. Otherwise, the sequences become more diverse as two subsequent positions must be different. In a pre-study, we find that MSA based on SA and the flag active performs best. The results of the pre-study are provided in Appendix 4 of the supplementary material.

5.3.4 Simulated annealing

We implement simulated annealing according to Aroui et al. (2017). In each iteration, the metaheuristic modifies the current sequence with a random flip, swap, or slide operation, each of which is performed with equal probability. If these operations decrease the number of overloads, the modified sequence is accepted as the new best sequence. Otherwise, the sequence still has a certain chance to be accepted as the new solution, which depends on the current temperature. With each iteration, the temperature “cools down” exponentially according to a fixed cooling factor. Therefore, the metaheuristic can explore random solutions which appear worse during early iterations but reach a better overall solution during later iterations.

5.3.5 Genetic algorithm

We implement a genetic algorithm according to Aroui et al. (2017). We initialize the start population with the initial sequence and nine randomly generated sequences. In each iteration, the metaheuristic improves the population by applying crossover and mutation operations to the sequences. An elitism procedure sorts the current population in ascending order according to the number of overloads. Subsequently, it copies the sequence with the fewest overloads to the last five positions of the population. Then, the algorithm performs crossover operations on the first four positions of the population to generate four child sequences based on two random pairs of parents. The child sequences are usually invalid in the sense that they no longer comply with the demand plan. The metaheuristic, therefore, employs a repair function that fixes the child sequences. If a child yields fewer overloads than a parent, the parent is replaced by the child.

5.3.6 Local search

We implement the local search metaheuristic according to Cortez and Costa (2015). The idea of local search is to perform swaps around sequence positions that lead to a high number of work overloads. In each iteration, the metaheuristic randomly selects a sequence position based on a probability distribution ψ that weights the probabilities according to the number of overloads. Subsequently, local search iterates over the sequence positions and evaluates swapping sequence positions (i, j) that contain different models. If a swap yields an improved sequence, the improved

sequence becomes the new best sequence, the probability distribution ψ is updated, and the procedure is repeated. Otherwise, the swap is undone.

5.3.7 Tabu search

We implement the tabu search metaheuristic according to Boysen et al. (2011). In each iteration, the metaheuristic evaluates the neighborhood of the current best sequence, which refers to the set of all swaps (i, j) of different models. For each neighbor, the number of overloads is calculated and the neighbor with the lowest number of overloads becomes the new best sequence. To prevent back-and-forth swaps, the swap that created the next best sequence is added to a tabu list. The tabu list is defined as a first-in-first-out (FIFO) queue of capacity $(T/16)$: when the capacity is exceeded, the oldest entry is removed.

5.3.8 Gurobi

Ultimately, we implement the mixed-integer linear programming solver Gurobi in version 8.0.1 based on the problem formulation from Sect. 3.1 but without the constraints (9) and (10). Instead, the processing times $p_{k,m}$ are set to their mean values $\mu_{k,m}$ or $\mu_{k,m} + \lambda\sigma$ to artificially increase the processing times. Gurobi is implemented using the Python package “gurobipy” (version 8.0.1). All parameters except the cut-off time are set to their default values.

6 Performance metrics

We provide the results based on three performance metrics. First, and as our primary metric, we provide the mean number of stochastic work overload situations over all problem instances. Let P denote the number of problem instances and $O_{j,a}^{sto}$ the number of stochastic overloads for approach a on problem j (recall that $O_{j,a}^{sto}$ is calculated as the average over 100 stochastic variations of processing times). Our primary metric is defined as

$$\frac{1}{P} \sum_{j=1}^P O_{j,a}^{sto}. \tag{24}$$

Second, we include the number of deterministic overloads that would occur if the processing times were deterministic and equal to the mean of their respective distribution. Let $O_{j,a}^{det}$ denote the number of work overloads according to the mean processing times for approach a on problem instance j . The mean number of deterministic overloads is given as

$$\frac{1}{P} \sum_{j=1}^P O_{j,a}^{det}. \tag{25}$$

Third, we provide the average expected overload time over all instances as calculated with the method by Zhao et al. (2007). This allows us to assess the robustness of our results in regard to how minimizing the number of overload situations influences the expected overload time. Let $O_{j,a}^T$ denote the expected overload time for approach a on instance j . The average overload time is given as

$$\frac{1}{100P} \sum_{j=1}^P O_{j,a}^T. \quad (26)$$

We scale the result by 100 in order to facilitate the presentation of results.

In addition, we provide several metrics (mean deviation to best, percentage of best solutions, and calculation time) for informational purposes, see Appendix 5 of the supplementary material.

7 Results

We now describe the results of our numerical evaluation. We start with our main analysis, after which we provide the results of three analyses based on unseen problems with variations in the standard deviation of processing times, additional stations on the assembly line, and problems with additional models.³ Finally, we analyze the trade-off between the performance of our approach and the required time for policy learning when using more probability quantiles for the state representation.

7.1 Main analysis

We first analyze the performance of all approaches on a total of 2160 stochastic problem instances. Figure 5 presents the results for small, large, and all instances. The black bars denote the number of stochastic overloads, the white bars denote the number of deterministic overloads, and the gray bars denote the expected overload time (Zhao et al. 2007). The numerical results are provided in Appendix 6 of the supplementary material. The optimal λ values are indicated below the corresponding approaches.

Figure 5 shows that our approach as trained in the stochastic environment (RL^{sto}) is superior to other methods in regard to the number of stochastic overloads. Compared to the best competing approach SA, the relative reduction of stochastic overloads is $\frac{7.32-6.58}{7.32} \approx 10.11\%$ on the small instances, $\frac{117.57-109.11}{117.57} \approx 7.20\%$ on the large instances, and $\frac{62.44-57.84}{62.44} \approx 7.37\%$ on all instances. Training the RL policy based on the deterministic problems also performs well, but RL^{det} is outperformed by LS, GA, and SA on all instances.

³ All datasets are provided at https://github.com/InformationSystemsFreiburg/stochastic_mms_dataset/.

The number of stochastic overloads is always significantly greater than the number of deterministic overloads. All t -tests are statistically significant with $p < 0.001$. Regarding the approaches that solve modified deterministic problems with artificially increased processing times, we find a U-shaped relation between λ and the solution quality. That is, small increases in deterministic processing time have a positive impact on the number of stochastic overloads, while large increases have a detrimental effect (see Appendix 7 of the supplementary material).

We also consider the expected overload time to assess the amount of utility work that is required to resolve work overloads. HSA^T should perform best as it is the only approach that specifically minimizes the expected overload times. However, this only holds on the small instances, whereas HSA^T achieves larger overload time than RL^{sto} on the large instances, and when considering all instances. One possible explanation is that Mosadegh et al. (2020) evaluated their approach based on comparably smaller problems with up to six models.

So far, we have only considered the expected values. As the results are aggregated over multiple instances of different size, we cannot directly assess the variation of the stochastic performance metrics. We now consider the variation of stochastic overloads and overload time for a fixed medium-sized problem instance with 100 cycles, 20 stations, and 20 models. Figure 6 presents the boxplots for a) stochastic overloads and b) overload time over 1000 samples of processing times for RL^{sto} and the best competing approach SA with $\lambda = 0.25$. Compared to SA , RL^{sto} achieves a lower average and a better worst-case scenario in the number of stochastic overloads. Regarding overload time, the variation is similar but the worst-case slightly favors SA .

In summary, our main analysis indicates that RL^{sto} is able to minimize the number of overload situations, while keeping the expected overload time similar to competing approaches.

7.2 Variation of standard deviation

Next, we study variations in the standard deviations of stochastic processing times. We assume that the variations occur during actual production and they are not known in advance. For the evaluation, we evaluate all approaches based on the same sequences as generated in the main analysis, but we sample the processing times based on the altered normal distribution. We specifically do not retrain the RL policies to ensure that this analysis is based on unseen probability distributions.

Figure 7 presents the results for $\sigma \in \{5.0, 7.5, 12.5, 15.0\}$. The numerical results are provided in Appendix 6 of the supplementary material. As expected, stochastic overloads and expected overload time increase for greater standard deviations. For $\sigma = 5$, RL^{sto} is outperformed by LS , GA , and, in particular, SA , which performs best. The relative difference of SA over RL^{sto} is $\frac{38.52-36.33}{38.52} \approx 5.69\%$. However, it should be noted that the evaluation with $\sigma = 5$ is also the closest to deterministic processing times. For all analyses with $\sigma > 5$, RL^{sto} is superior to the competing approaches. The relative improvements of RL^{sto} over the best competing

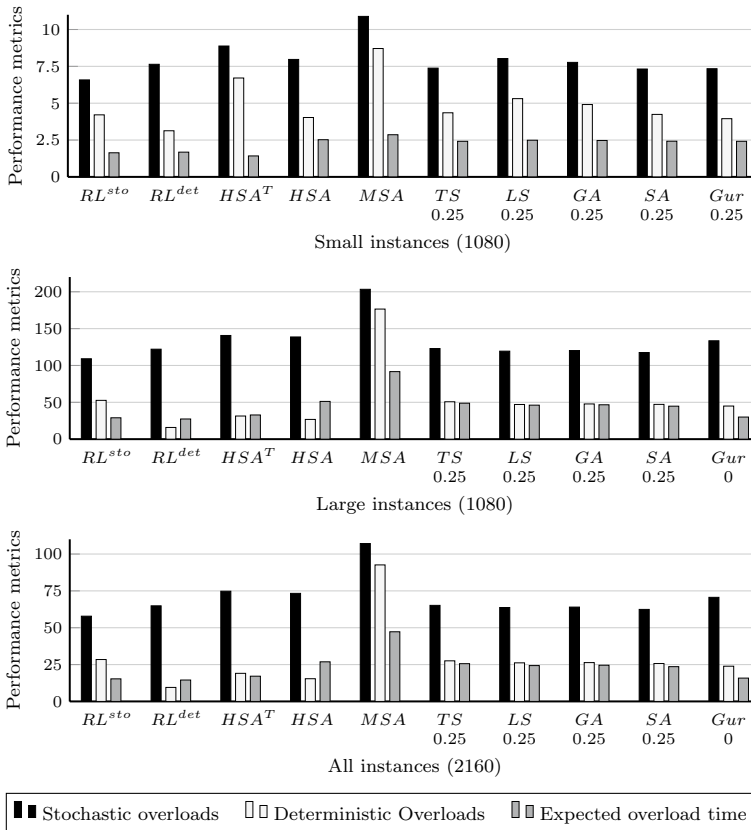


Fig. 5 Main results

methods are $\frac{48.89-47.04}{48.89} \approx 3.78\%$ for $\sigma = 7.5$, $\frac{75.79-70.44}{75.79} \approx 7.06\%$ for $\sigma = 12.5$, and $\frac{89.73-84.27}{89.73} \approx 6.08\%$ for $\sigma = 15$.

7.3 Additional stations

We now perform another analysis, where we evaluate the performance of our approach in regard to unseen stations on the assembly line. For this purpose, we modify the problem instances to contain up to five additional stations. The processing times of the additional stations have similar processing times to the existing stations (see Sect. 5.1). All additional stations are added at the end of the assembly line without loss of generality. Due to computational constraints, we limit this analysis to a subset of 216 problem instances. Again, we do not retrain the RL policies to ensure that the problems are truly unseen but we adjust the implementation of $o_t^m(q)$ according to the new environment to apply the RL policies. All competing approaches are directly applied to the modified problems.

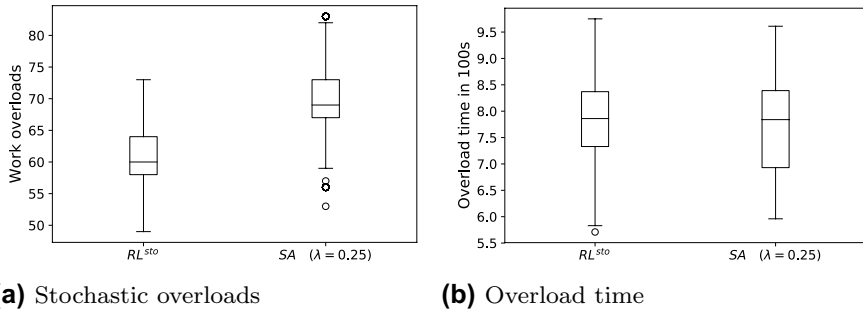


Fig. 6 Variation of stochastic overloads and overload time

Figure 8 presents the results of RL^{sto} against the best competing approach SA (with $\lambda = 0.25$). The numerical values including all approaches are provided in Appendix 8 of the supplementary material. Evidently, the outperformance of RL^{sto} persists when the assembly line contains up to three additional stations. However, RL^{sto} is outperformed by SA on problems with more than three additional stations.

7.4 Additional models

We also analyze the sensitivity of our approach toward additional but unseen models. For this purpose, we generate new problem instances with up to five new models, which have similar processing times to those of the existing models. Again, all competing approaches are applied to the new problem instances, but we do not retrain the RL policies to ensure that the models are truly unseen. To apply the trained RL policy to unseen models, we simplify the new problem to a known problem, where the demand plan solely contains known models. For each new model m^* , the function $sim(m^*)$ returns the known model with the lowest Manhattan distance between the vectors of processing times. Ties are broken according to the lower index m .

$$sim(m^*) = \operatorname{argmin}_{m=1, \dots, M} \sum_{k=1}^K |p_{k,m} - p_{k,m^*}|. \tag{27}$$

When generating the sequence, we have to decide between sequencing the new model m^* or the known model $sim(m^*)$ whenever the policy wants to sequence $sim(m^*)$. If $d^{m^*} = d^m$, we alternate between sequencing m^* and $sim(m^*)$, starting with m^* . If $d^{m^*} > d^m$, we first sequence the new models until $d_t^{m^*} = d_t^m$; then we alternate between m^* and $sim(m^*)$. And, if $d^{m^*} < d^m$, we first alternate between both models until $d_t^{m^*} = 0$; then we sequence the remaining known models.

The results of RL^{sto} and SA (with $\lambda = 0.25$) are shown in Fig. 9. The numerical values are provided in Appendix 9 of the supplementary material. We find that RL^{sto} is superior to SA for up to two additional models.

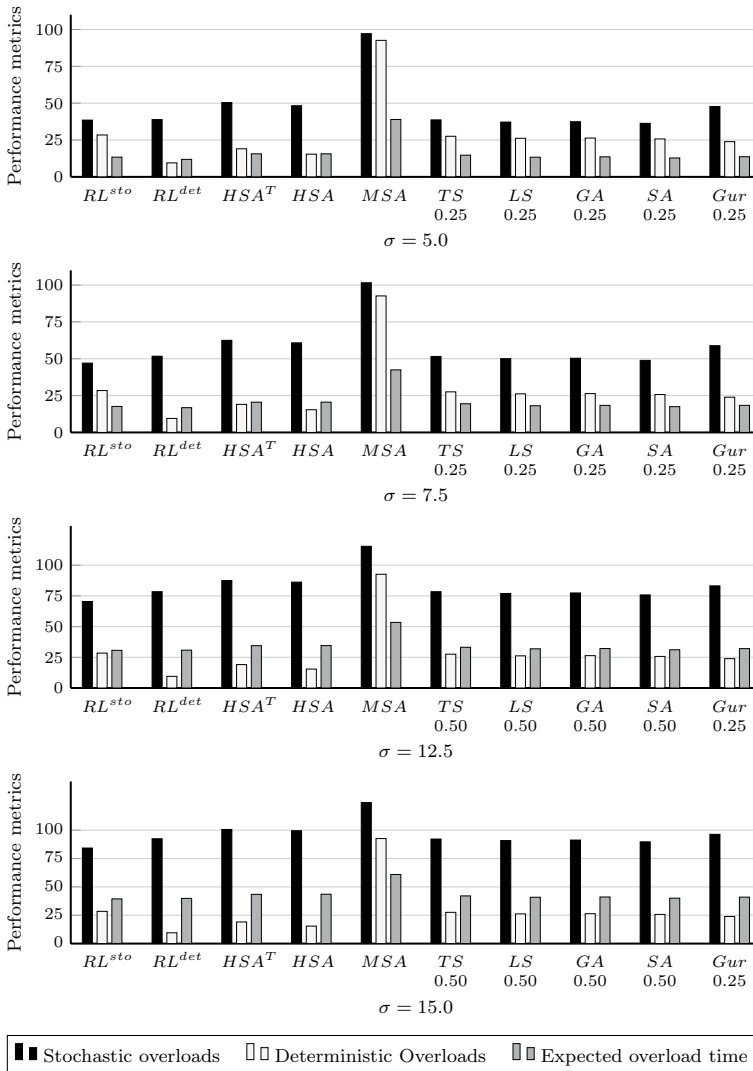


Fig. 7 Analysis with different standard deviations

7.5 Additional state information

Ultimately, we analyze how the performance of our approach changes if we extend the state representation by including more probability quantiles of processing times. Recall that the function $\sigma_t^m(q) \rightarrow \{0, 1\}$ indicates if scheduling model m at sequence position t will result in a work overload when the processing times of all stations are equal to their q -quantile. In the current implementation, we included $\sigma_t^m(q)$ for $q \in \{0.25, 0.50, 0.75\}$. However, it seems interesting to analyze how performance and policy learning time change if we include $\sigma_t^m(q)$ for more than three quantiles.

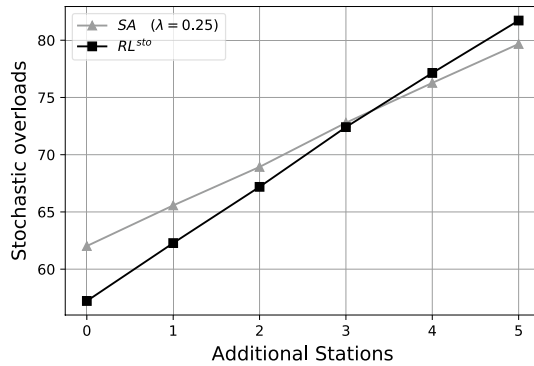


Fig. 8 Analysis with additional stations

Therefore, we retrain the policy in a modified environment, where the state representation contains $o_t^m(q)$ for 0, 1, 3, 5, 7, or 9 quantiles. The respective quantiles are determined through an even split over the interval $[0, 1]$, while ensuring that the 0.50-quantile (i.e., the mean) is always included. For instance, the environment with five quantiles calculates $o_t^m(q)$ for $q \in \{0.16, 0.33, 0.50, 0.66, 0.83\}$. Due to computational constraints, we limit this analysis to a single instance with sequence length 100, 20 stations, and 20 models.

Figure 10a plots the stochastic overloads for RL^{sto} depending on the number of quantiles. The performance of SA is also indicated as a baseline. As expected, the number of overloads decreases with the number of quantiles, but the marginal reduction of overloads becomes smaller the more quantiles are included. At the same time, increasing the dimensionality of the state representation increases the time that is required for training the policy. Figure 10b shows that the time for policy learning increases linearly with the number of quantiles.

Table 5 presents the numerical values of Fig. 10a and b. We observe that the improvement from our current implementation RL_3^{sto} to RL_5^{sto} is less than one work overload. Hence, we argue that including $o_t^m(q)$ for three probability quantiles provides a reasonable trade-off between performance and the time required for policy learning.

8 Extensibility

The main idea behind our approach is to incorporate deterministic information based on several probability quantiles into the state of an RL approach to solve the stochastic MMS problem. There appear to be several ways that this approach could be extended to similar stochastic problems. First, our method could be generalized to stochastic MMS problems with different probability distributions (e.g., uniform). For this purpose, the function $o_t^m(q)$ must be defined with respect to the quantiles

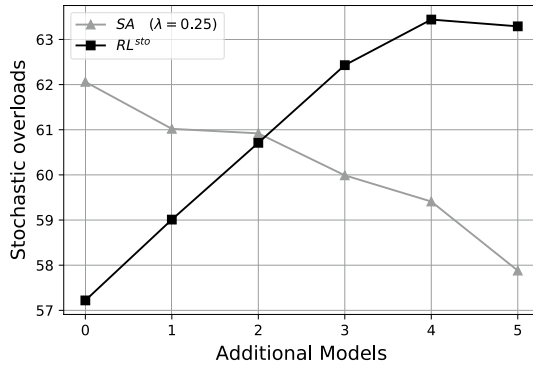


Fig. 9 Analysis with additional models

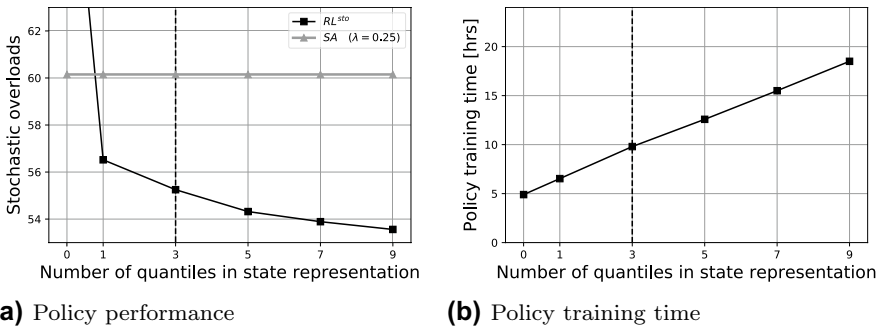


Fig. 10 Stochastic overloads and policy training time for RL^{sto} and different numbers of probability quantiles in the state representation. The dashed lines highlight the number of quantiles that was used for our previous analyses

of the respective distribution. It also seems straightforward to consider other MMS problem variations, e.g., with U-shaped assembly lines or different worker types.

Second, our approach can be applied to stochastic MMS problems with other objectives like minimize overload or idle time. For instance, to minimize the expected work overload time, the reward function needs to be changed to the negative sum of work overload times that occurred at cycle t .

Third, the stochastic MMS problem can be considered through the lens of robust optimization (e.g., Ben-Tal et al. 2009; Gabrel et al. 2014). Manufacturers may, for instance, only be able to handle a given number of overloads simultaneously or a maximum number of overloads during one shift. However, such constraints are generally not considered in the literature as they strongly depend on company-specific risk preferences regarding feasibility of solutions. It thus seems interesting to extend the problem formulation with additional constraints and combine our approach with methods from safe reinforcement learning (see e.g., Garcia and Fernández 2015).

Fourth, our approach may also be extended to other stochastic scheduling problems. The main challenge is to find a meaningful output for the function $\sigma_t^m(q)$ that provides the crucial information for the state. For instance, for stochastic job-shop

Table 5 Analysis of additional state information

	RL_0^{sto}	RL_1^{sto}	RL_3^{sto}	RL_5^{sto}	RL_7^{sto}	RL_9^{sto}	SA $\lambda = 0.25$
Stochastic overloads	72.70	56.52	55.25	54.32	53.89	53.56	60.15
Deterministic overloads	50.90	29.80	31.50	29.80	31.00	30.60	25.90
Overload time	15.95	14.36	13.73	13.77	13.76	13.67	14.22
Policy training time [h]	4.90	6.53	9.80	12.58	15.50	18.50	–

Best values are highlighted in bold

Deterministic overloads are equal for RL_1^{sto} and RL_5^{sto} due to small sample size

or flow-shop problems, $o_i^m(q)$ could be defined in regard to the number of machines that will become idle if job m is sequenced next given that all processing times are equal to their q -quantile.

Electronic supplementary material The online version of this article (doi:<https://doi.org/10.1007/s00291-021-00652-x>) contains supplementary material, which is available to authorized users.

Funding Open Access funding enabled and organized by Projekt DEAL.

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/>.

References

- Agrawal S, Tiwari M (2008) A collaborative ant colony algorithm to stochastic mixed-model U-shaped disassembly line balancing and sequencing problem. *Int J Prod Res* 46:1405–1429
- Aroui K, Alpan G, Frein Y (2017) Minimising work overload in mixed-model assembly lines with different types of operators: a case study from the truck industry. *Int J Prod Res* 55:6305–6326
- Bautista J, Alfaro R, Batalla-García C (2016) Grasp for sequencing mixed models in an assembly line with work overload, useless time and production regularity. *Prog Artif Intell* 5:27–33
- Bautista J, Cano A, Alfaro R (2012) Models for MMSP-W considering workstation dependencies: a case study of Nissan's Barcelona plant. *Eur J Oper Res* 223:669–679
- Ben-Tal A, El Ghaoui L, Nemirovski A (2009) Robust optimization. Princeton University Press, Princeton
- Bent RW, Van Hentenryck P (2004) Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Oper Res* 52:977–987
- Boysen N, Fliedner M, Scholl A (2007) A classification of assembly line balancing problems. *Eur J Oper Res* 183:674–693
- Boysen N, Fliedner M, Scholl A (2009) Sequencing mixed-model assembly lines: survey, classification and model critique. *Eur J Oper Res* 192:349–373
- Boysen N, Kiel M, Scholl A (2011) Sequencing mixed-model assembly lines to minimise the number of work overload situations. *Int J Prod Res* 49:4735–4760
- Chutima P, Naruemitwong W (2014) A pareto biogeography-based optimisation for multi-objective two-sided assembly line sequencing problems with a learning effect. *Comput Ind Eng* 69:89–104

- Cortez PM, Costa AM (2015) Sequencing mixed-model assembly lines operating with a heterogeneous workforce. *Int J Prod Res* 53:3419–3432
- Dong J, Zhang L, Xiao T, Mao H (2014) Balancing and sequencing of stochastic mixed-model assembly U-lines to minimise the expectation of work overload time. *Int J Prod Res* 52:7529–7548
- Gabrel V, Murat C, Thiele A (2014) Recent advances in robust optimization: an overview. *Eur J Oper Res* 235:471–483
- Garcia J, Fernández F (2015) A comprehensive survey on safe reinforcement learning. *J Mach Learn Res* 16:1437–1480
- Golle U, Rothlauf F, Boysen N (2014) Car sequencing versus mixed-model sequencing: a computational study. *Eur J Oper Res* 237:50–61
- Joly A, Frein Y (2008) Heuristics for an industrial car sequencing problem considering paint and assembly shop objectives. *Comput Ind Eng* 55:295–310
- Kim YK, Hyun CJ, Kim Y (1996) Sequencing in mixed model assembly lines: a genetic algorithm approach. *Comput Oper Res* 23:1131–1145
- Li J, Gao J, Sun L (2012) Sequencing minimum product sets on mixed-model U-lines to minimise work overload. *Int J Prod Res* 50:4977–4993
- Lopes TC, Michels AS, Lüders R, Magatão L (2020) A simheuristic approach for throughput maximization of asynchronous buffered stochastic mixed-model assembly lines. *Comput Oper Res* 115(104):863
- Miltenburg J (2002) Balancing and scheduling mixed-model U-shaped production lines. *Int J Flex Manuf Syst* 14:119–151
- Mosadegh H, Fatemi Ghomi S, Süer G (2017) Heuristic approaches for mixed-model sequencing problem with stochastic processing times. *Int J Prod Res* 55:2857–2880
- Mosadegh H, Fatemi Ghomi S, Süer G (2020) Stochastic mixed-model assembly line sequencing problem: mathematical modeling and Q-learning based simulated annealing hyper-heuristics. *Eur J Oper Res* 282:530–544
- Özcan U, Kellegöz T, Toklu B (2011) A genetic algorithm for the stochastic mixed-model U-line balancing and sequencing problem. *Int J Prod Res* 49:1605–1626
- Parrello BD, Kabat WC, Wos L (1986) Job-shop scheduling using automated reasoning: a case study of the car-sequencing problem. *J Autom Reason* 2:1–42
- Pinedo M, Weiss G (1987) The largest variance first policy in some stochastic scheduling problems. *Oper Res* 35:884–891
- Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. [arxiv: 1707.06347](https://arxiv.org/abs/1707.06347). Accessed 23 September 2020
- Shannon CE (1948) A mathematical theory of communication. *Bell Syst Tech J* 27:379–423
- Solnon C, Cung VD, Nguyen A, Artigues C (2008) The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the Roadef'2005 challenge problem. *Eur J Oper Res* 191:912–927
- Sutton RS, Barto AG et al (1998) Introduction to reinforcement learning. MIT Press, Cambridge
- Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn* 8:229–256
- Zahavy T, Haroush M, Merlis N, Mankowitz DJ, Mannor S (2018) Learn what not to learn: action elimination with deep reinforcement learning. *Adv Neural Inform Process Syst* 31:3562–3573
- Zhao X, Liu J, Ohno K, Kotani S (2007) Modeling and analysis of a mixed-model assembly line with stochastic operation times. *Naval Res Logist* 54:681–691
- Zheng Z, Oh J, Singh S (2018) On learning intrinsic rewards for policy gradient methods. *Adva Neural Inform Process Syst* 31:4644–4654

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