Received 15 December 2023, accepted 10 January 2024, date of publication 17 January 2024, date of current version 25 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3355269

APPLIED RESEARCH

Reinforcement Learning for Two-Stage Permutation Flow Shop Scheduling—A Real-World Application in Household Appliance Production

ARTHUR MÜLLER[1], FELIX GRUMBACH[2], AND FIONA KATTENSTROTH[3]
[1]Fraunhofer IOSB-INA, 32657 Lemgo, Germany
[2]Center for Applied Data Science (CfADS), Bielefeld University of Applied Sciences, 33619 Gütersloh, Germany
[3]Miele & Cie.KG, 33332 Gütersloh, Germany

Corresponding author: Arthur Müller (arthur.mueller@iosb-ina.fraunhofer.de)

This work was supported by the Ministry of Economic Affairs, Industry, Climate Action and Energy of the State of North Rhine-Westphalia, Germany, "SUPPORT" Project 005-2111-0026.

ABSTRACT Solving production scheduling problems is a difficult and indispensable task for manufacturers with a push-oriented planning approach. In this study, we tackle a novel production scheduling problem from a household appliance production at the company Miele & Cie. KG, namely a two-stage permutation flow shop scheduling problem (PFSSP) with a finite buffer and sequence-dependent setup efforts. The objective is to minimize idle times and setup efforts in lexicographic order. In extensive and realistic data, the identification of exact solutions is not possible due to the combinatorial complexity. Therefore, we developed a reinforcement learning (RL) approach based on the Proximal Policy Optimization (PPO) algorithm that integrates domain knowledge through reward shaping, action masking, and curriculum learning to solve this PFSSP. Benchmarking of our approach with a state-of-the-art genetic algorithm (GA) showed significant superiority. Our work thus provides a successful example of the applicability of RL in real-world production planning, demonstrating not only its practical utility but also showing the technical and methodological integration of the agent with a discrete event simulation (DES). We also conducted experiments to investigate the impact of individual algorithmic elements and a hyperparameter of the reward function on the overall solution.

INDEX TERMS Reinforcement learning, production scheduling, permutation flow shop scheduling problem.

I. INTRODUCTION

Production scheduling problems are widely studied in academic literature and solving these problems has a significant impact on the success of manufacturing companies. A particular case are permutation flow shop scheduling problems, which were first introduced by [1]. PFSSPs typically involve a sequence of jobs that must be processed by multiple machines in a specific order. Methods for solving PFSSPs include heuristics, numerical methods, and metaheuristics. In recent years, RL has attracted more and more attention as an alternative approach for successfully solving scheduling problems [2], [3]. However, many approaches do not move beyond the academic context due to their abstraction from real-world requirements, as [4] and [5] point out.

Our paper addresses a specific PFSSP, a so called two-stage PFSSP, that occurs in a household appliance production of Miele. A two-stage PFSSP encompasses two distinct production stages with multiple machines in at least one of the stages. Thus, it is not only necessary to plan the sequence of jobs within the first stage, but also to consider how this sequence affects the material flow in the second stage.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

TABLE 1. Comparison of recent RL approaches for solving the PFSSP.

Ref.	Problem	Method	Objective	Benchmarks					Comparison				
				Synthetic Data	Taillard [6]	Carlier [7]	Reeves [8]	Heller [9]	Class. Heuristics	Metaheuristics	Math. Optimizer	Constr. Progr.	RL
[10]	PFSSP	REINFORCE with heuristics as baseline	total tardiness	✓					✓				
[11]	distributed PFSSP	multi-agent RL	makespan		✓					✓			
[12]	PFSSP	Actor-Critic RL + heuristic	makespan	✓	✓				✓				
[13]	dynamic PFSSP	Double Deep Q-Network (DQN)	total tardiness	✓					✓				
[14]	distr. PFSSP with prev. maintenance	DQN with diminishing greedy rate	makespan	✓						✓			✓
[15]	PFSSP	Policy-Based RL + Graph Isomorphism Network	late work		✓				✓	✓			
[16]	large-scale PFSSP	GA based on RL	makespan			✓	✓			✓			
[17]	PFSSP with multiple lines	PPO + local search	makespan		✓				✓	✓	✓	✓	✓
[18]	PFSSP	Q-learning + heuristics	makespan		✓				✓	✓			✓
[19]	PFSSP	Artificial Bee Colony + Q-learning	makespan		✓	✓	✓	✓		✓			
[20]	PFSSP	Knowledge guided RL	makespan	✓					✓	✓	✓		✓

Due to this interdependence, two-stage PFSSPs impose more solving difficulties than traditional PFSSPs. The concrete problem considers one machine in the first stage and multiple machines in the second stage. Moreover, it has a finite buffer connecting the stages, as well as sequence-dependent setup efforts in the first stage, and machine shifts. The interdependence also arises from a tradeoff between objectives: The goal is to minimize idle times in the second stage and the setup effort required for changing product types in the first stage. To the best of our knowledge, this problem has not yet been addressed in the literature.

Since the problem's complexity precludes an exact solution and RL generally outperforms heuristics for multi-objective problems [4], we utilize RL for our solution. To this end, we formulated the PFSSP as a Markov decision process (MDP). Furthermore, we developed an RL approach comprising algorithmic elements such as action masking, reward shaping, and curriculum learning to incorporate domain-specific knowledge. For training the agent, we utilized the professional DES software FlexSim to create a simulation model of the environment. Furthermore, we conduct experiments with realistic data sets from production to benchmark our solution against a state-of-the-art GA. The results show that our RL approach performs better, especially on complex problem instances. In addition, we investigate how each algorithmic element contribute to the success of the solution and how weighting factors in the reward function can be used to lexicographically optimize the two objectives.

The paper is motivated by the research gap identified in the literature. Recent work on solving PFSSPs using RL is shown in Table 1. Our work differs from the literature by having a PFSSP variant that has never been considered before. Moreover, the literature mainly considers the optimization of only one objective. Mostly this is the makespan, e.g., in [12], [14], [16], [17], [18], [19], [20], and [21]. However, based on the real-world requirements of the household appliance production, we consider the optimization of two objectives, also, in a lexicographic way. Furthermore, most of the papers considered combine RL with heuristics, metaheuristics or other methods to generate feasible and near optimal schedules. We focus, however, on an approach where RL is used exclusively. One advantage of this is a shorter runtime, since a trained RL agent generates a schedule in one iteration, whereas combined methods usually require several iterations. This is especially relevant for runtime-intensive simulation models. In order to evaluate the agent in real-world usage, we draw the problem instances from real-world production. This differs from the literature reviewed, which mostly employs either the Taillard data set [6] or synthetic data. As a methodology for benchmarking our approach, we use a metaheuristic, as in the majority of the literature reviewed.

II. PROBLEM FORMULATION

The real-world problem can be defined as a non-linear Mixed Integer Program (MIP) that precisely formalizes all relevant constraints and objectives. It is a two-stage PFSSP with multiple second stage assembly stations, finite buffers, sequence-dependent setup efforts and station-related shift windows. Each production job comprises two sequential operations, corresponding to the two consecutive stages, for manufacturing one product. As displayed in Figure 5, a single pre-assembly station (PAS) in the first stage fills a central buffer with semi-finished products (SFP). The SFP are limited to a small set of generic types, which are afterward converted to specific products. At the following order penetration point, each SFP must be assigned to a predefined non-identical final assembly station (FAS) in the second stage. Under consideration of a static buffer capacity limit, it is possible that the buffer initially contains a number of SFP of different types. Thus, some jobs can skip the first stage by directly assigning a SFP from the buffer to them. As a further complication, the PAS must be suitably set up for each SFP type, which leads to sequence-dependent setup efforts. Moreover, the stations can have different shift windows, which can have an effect on the material flows. On the FAS, some jobs have predecessor relationships to other jobs on the same FAS. This means that they cannot be released until the predecessor jobs have been completed. Johnson [1] demonstrated that a specialization of the problem for makespan minimization without sequence-dependent setup efforts, finite buffers, work shifts and release conditions can be solved exactly and efficiently in polynomial time. However, the approach in this form cannot be applied to our extended problem, nor were we able to identify suitable efficient algorithms in the existing literature. Considering the given conflicting objectives and further constraints, we assume that the problem is complex and that traditional algorithms are unlikely to solve it within an acceptable runtime.

The problem consists of the following parameters:

TABLE 2. MIP constants.

Parameter	Description
J	Set of job indices $J = \{1, ..., n\}$
K	Set of station indices $K = \{1, ..., 5\}$, where $k = 1$ is the PAS and $k > 1$ is a FAS (see Figure 5)
S_k	Set of work shifts of a station k containing shift triples (s_1, s_2, s_3). s_1 is the unique shift identifier, s_2 is the shift begin timestamp and s_3 is the shift end timestamp, $s_2, s_3 \in \mathbb{N}$ (seconds).
$s_{max} \in \mathbb{N}$	Latest possible timestamp (in seconds) from all work shifts
$\mathring{\tau}$	Set of all SFP types
τ_i	SFP type of job i
$b_{max} \in \mathbb{N}$	Maximum number of SFP in the central buffer
$b_{\tau,0} \in \mathbb{N}$	Initial amount per type τ in the central buffer
$p_{i,k} \in \mathbb{R}^+$	Processing time of job i on station k
$q_{\tau,\tau'} \in \mathbb{R}^+$	Sequence-dependent setup effort at station $k = 1$, when type τ directly precedes type τ'
$m_{i,k} \in \{0, 1\}$	1, if job i is assigned to FAS k
$r_{i,j} \in \{0, 1\}$	1, if job j can be released on the FAS after job i has been completed

The following variables are required to solve the MIP:

TABLE 3. MIP variables.

Variable	Description
$\alpha_{i,k} \in \mathbb{R}^+$	Start time of a job i on the assigned station k
$\beta_i \in \{0, 1\}$	1, if the SFP of job i is directly assigned from the initial buffer content ($\rightarrow$ the PAS is skipped and the FAS can start immediately).
$\gamma_{i,\hat{s}} \in \{0, 1\}$	1, if job i is realized in shift $\hat{s}$.

The following auxiliary definitions allow a better modeling of the constraints: ζ is a binary value, which determines if a job i precedes job j on station k ($\rightarrow$ if the completion time of i is lower or equal the start time of j).

$$\zeta_{i,j,k} = \begin{cases} 1, \text{ if both jobs are assigned to station } k \text{ and the} \\ \quad \text{completion time of } i \text{ is lower or equal the start} \\ \quad \text{time of } j \\ 0 \end{cases}$$

Based on this, ζ^* is a binary value, which determines direct preceding jobs on the PAS.

$$\zeta^*_{i,j} = \begin{cases} 1, \text{ if } i \text{ directly precedes } j \text{ on station } k = 1 \\ 0 \end{cases}$$

λ determines the actual processing time required for job i on the PAS. When the SFP of a job is initially available in the buffer, it is not assigned to the PAS and the processing time is zero. It should be noted that a setup operation can take place separately from the actual processing and therefore does not affect the processing time on the PAS.

$$\lambda_i = (1 - \beta_i)\ p_{i,1}$$

Finally, C determines the completion time of a job i on a station k.

$$C_{i,k} = \begin{cases} \alpha_{i,1} + \lambda_i, \text{ if } k = 1 \\ m_{i,k}(\alpha_{i,k} + p_{i,k}) \end{cases}$$

Based on the real use case, the primary objective is to minimize the sum of idle times from all FAS (1). In the course of lexicographical optimization, the second objective is to minimize sequence-dependent setup efforts on the PAS (2).

$$\min \sum_{k \in K: k>1} \left(\max_{i \in J} \left\{ C_{i,k} \right\} - \sum_{i \in J} m_{i,k}\ p_{i,k} \right) \tag{1}$$

$$\min \sum_{i \in J} \left((1 - \beta_i) \sum_{j \in J \setminus i} \zeta^*_{j,i}\ q_{\tau_j,\tau_i} \right) \tag{2}$$

The first constraint (3) ensures that a station can process only one job at a time:

$$\zeta_{i,j,k} + \zeta_{j,i,k} > 0$$
$$\forall i \in J, \forall j \in J \setminus i, \forall k \in K : k = 1 \vee m_{i,k} m_{j,k} = 1 \tag{3}$$

The assigned FAS can only start when the job is available in the buffer:

$$\alpha_{i,k} \geq \left(\alpha_{i,1} + \lambda_i \right) \left(1 - \beta_i \right)$$
$$\forall i \in J, \forall k \in K : k > 1 \wedge m_{i,k} = 1 \tag{4}$$

Only those jobs can directly start on the FAS, if their types are initially available in the buffer:

$$\sum_{i \in J: \tau_i = \tau} \beta_i \leq b_{\tau,0} \quad \forall \tau \in \mathring{\tau} \tag{5}$$

Jobs assigned to stations must be realized in exactly one shift:

$$1 = \left(\sum_{\hat{s} \in S_k} \gamma_{i,\hat{s}_1}\right) + \begin{cases} \beta_i, \text{ if } k = 1 \\ 0 \end{cases}$$
$$\forall i \in J, \forall k \in K : k = 1 \vee m_{i,k} = 1 \quad (6)$$

The start time of a job must be after the shift's begin timestamp:

$$\alpha_{i,k} \geq \sum_{\hat{s} \in S_k} \gamma_{i,\hat{s}_1} \quad \hat{s}_2 \quad \begin{cases} (1-\beta_i), \text{ if } k = 1 \\ 1 \end{cases}$$
$$\forall i \in J, \forall k \in K : k = 1 \vee m_{i,k} = 1 \quad (7)$$

The completion time of a job must be before the shift's end timestamp:

$$C_{i,k} \quad \begin{cases} (1-\beta_i), \text{ if } k = 1 \\ 1 \end{cases} \leq \sum_{\hat{s} \in S_k} \gamma_{i,\hat{s}_1} \quad \hat{s}_3$$
$$\forall i \in J, \forall k \in K : k = 1 \vee m_{i,k} = 1 \quad (8)$$

Jobs can then be released when the predecessor jobs have been completed.

$$\alpha_{j,k} \geq \alpha_{i,k} + p_{i,k}$$
$$\forall i \in J, \forall j \in J : r_{i,j} = 1, \forall k \in K : k > 1 \wedge m_{i,k} m_{j,k} = 1 \quad (9)$$

The capacity of the central buffer cannot be exceeded regarding its upper bound of SFP:

$$\sum_{t'=0}^{t} \sum_{i \in J} \begin{cases} 1, \text{if} \alpha_{i,1} \leq t' \\ 0 \end{cases} - \sum_{k \in K : k > 1} \begin{cases} m_{i,k}, \text{ if } \alpha_{i,k} \leq t' \\ 0 \end{cases} \leq b_{max}$$
$$\forall t \in \{0, \ldots, s_{max}\} \quad (10)$$

III. METHOD

A. REINFORCEMENT LEARNING

Reinforcement Learning is a type of Machine Learning, where an agent is trained to behave optimally in an environment [22]. At each time step t, the agent selects an action a_t based on a representation of the relevant information about the environment called state s_t. When taking an action, the agent receives a reward r_{t+1}, which represents the desirability of the action taken in terms of the optimization goal. The behavior of the agent is determined by its policy $\pi(a_t|s_t)$, which maps states to actions. Policies are often modelled as neural networks which are parameterized with θ. RL problems are typically formulated as a Markov decision process (MDP), which is a 5-tuple that encompasses:

1) A set of states $\mathcal{S}$,
2) a set of actions $\mathcal{A}$,
3) a transition probability function $\mathcal{T}(s_{t+1}|s_t, a_t)$,
4) a reward function $R(s_t, a_t)$ determining the reward r_{t+1},
5) and a discount factor $\gamma \in [0, 1)$ for discounting future rewards.

Over the last decade, several RL algorithms have been developed. One of the most widespread is the Proximal Policy Optimization (PPO) algorithm developed by OpenAI [23], which is a so-called policy-based method. We select PPO as basis for our RL solution, as it has been shown in preliminary experiments to be more robust and less hyperparameter sensitive than other RL algorithms (see also [24]).

The goal of PPO is to learn parameters θ that maximize the expected cumulative discounted reward:

$$\theta \leftarrow \text{argmax}_\theta \mathbb{E}[\Sigma_t \gamma^t R(s_t, a_t)|\pi_\theta]$$

PPO achieves this by maximizing a clipped surrogate objective:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \quad (11)$$

where $\hat{A}_t$ is the estimated advantage at time step t (calculated by a generalized advantage estimator [25]), $r_t(\theta)$ is the likelihood ratio $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$, and ϵ is a small positive scalar that limits the update step size. This objective ensures moderate updates of θ, so that π_θ is not deviating too much from $\pi_{\theta_{\text{old}}}$.

B. MDP DEFINITION

Crucial for the successful application of RL to solve this optimization problem is an adequate formulation of the MDP. For ease of understanding, we explain and specify the MDP using the concrete problem instance at Miele. The household appliance production consists of a PAS where $|\mathring{\tau}| = 8$ different SFP types can be produced. After being produced, these are stored in a buffer with a static capacity of $b_{max} = 2900$. Four final assembly stations pick SFPs out of the buffer to produce the final products. The planning period is a week with five working days. The MDP is described in the following.

1) STATE SPACE

The state space must include all information necessary for the agent to fulfill the demands of the final assembly stations. I.e. there must always be enough SFPs available in the buffer so that the demanded household appliances can be produced and the final assembly stations are not idle (1). We therefore introduce several features that condense the information about when and how much SFPs are demanded at the current time step. These features cover all relevant parameters of the MIP.

- $u_{\tau,24}(t)$: The amount of SFPs of type τ that is required from all FAS in the next 24 hours.
- $u_{\tau,end}(t)$: The amount of SFPs of type τ that is required from all FAS until the end of the planning period.
- $b_\tau(t)$: The amount of SFPs of type τ in the buffer.
- $v_{\tau,24}(t) = u_{\tau,24}(t) - b_\tau(t)$: The amount of SFPs of type τ still to be produced, so that the FAS demand is covered in the next 24 hours.

- $v_{\tau,end}(t) = u_{\tau,end}(t) - b_\tau(t)$: The amount of SFPs of type τ still to be produced, so that the FAS demand is covered until the end of the planning period.
- $t_\tau(t)$: This represents the duration for which the demand of SFP type τ in the buffer is still covered.

All variables are normalized and clipped between 0 and 1. We incorporate $v_{\tau,24}$, $v_{\tau,end}$, and t_τ of all SFP types into the state space. If the amount of an SFP of type τ in the buffer exceeds the amount required until the end of the planning period (i.e. $b_\tau > u_{\tau,end}$), $v_{\tau,end}$ is set to 0 and t_τ is set to 1. Similarly, $v_{\tau,24}$ is also set to 0 if there are enough SFPs of the required type in the buffer for the next 24 hours. Additionally, we add $b_u(t)$ to the state space, which represents the sum of all SFPs in the buffer divided by b_{max}.

The second objective is to minimize the setup efforts, as seen in (2). In order for the agent to take the setup efforts into account, the last produced SFP type $\tau'(t)$ is included into the state space as one-hot-encoded vector.

Taking all components into account, the state is represented by the following vector:

$$\boldsymbol{s}_t = [v_{0,24}(t), \ldots, v_{7,24}(t), v_{0,end}(t), \ldots, v_{7,end}(t), \\ t_0(t), \ldots, t_7(t), b_u(t), \tau'(t)]$$

2) ACTION SPACE

Since there are eight different SFP types, we define the action space to be discrete, where an action a is an integer with $a \in \mathring{\tau} = \{0, 1, \ldots, 7\}$. When the RL agent selects an action, a predefined amount o_1 of the selected SFP type will be produced on PAS 1. In this way, the production schedule is not given from the start, but is built up successively as the agent interacts with the environment. In retrospect, the production plan is then created by stringing together the actions. o_1 is held constant over the whole planning period and for each SFP type. Since for each SFP type the same processing time is given, a constant o_1 ensures equidistant time steps for the agent-environment interactions. Otherwise, a more sophisticated approach such as a so-called Semi-MDP would be needed to cope with varying production amounts and therefore non-equidistant time steps [26], [27]. We define o_1 to be 50 based on domain expertise. Smaller values allow the agent to produce more fine-grained and thus potentially better production plans, but they prolong training because more steps would be required to achieve the same number of episodes.

Furthermore, we use action masking to integrate domain knowledge into the agent in order to shorten the training time and increase the probability of successful training. Action masking is a technique known in reinforcement learning approaches for computer games [28] or traffic signal control [29], among others. This involves masking undesirable or non-permissible actions at each decision point of the agent, so that the agent cannot select them. We use action masking to ensure that only SFP types are produced for which there is a need until the end of the planning horizon ($v_{\tau,end} > 0$). This restriction ensures that the agent does not select any unrequired SFPs and is therefore not wasting production capacity for required SFPs.

We define the mask as a vector $\boldsymbol{m}_a(t) = [m_{a,0}(t), m_{a,1}(t), \ldots, m_{a,7}(t)]$, where each element is defined as

$$m_{a,\tau}(t) = \begin{cases} 1, & \text{if } v_{\tau,end}(t) > 0 \\ 0 \end{cases} \tag{12}$$

The values for the mask are computed immediately before each decision point of the agent. For undesirable actions, the probability that the agent will choose that action is set to 0.

3) REWARD SHAPING

In RL, objectives have to be converted into a reward function. With regard to the objective functions 1 and 2, the obvious approach is to punish high idle times in the FAS stage as well as high setup efforts in the PAS stage. The agent would then learn to minimize idle times and setup efforts in order to evade punishment. For this purpose, we introduce $d(t)$, which is the sum of the idle times of all FAS between the last and the current decision time step t. The reward function is then

$$r_1(t) = -\alpha_{idle} d(t) - \alpha_{se} q_{\tau,\tau'}(t), \tag{13}$$

where $q_{\tau,\tau'}(t)$ is the sequence-dependent setup effort in the last time step and α_{idle} and α_{se} represent the weights of the respective objectives.

An alternative approach for the reward function is achieved by *reward shaping*. Reward shaping is used to guide the learning process of an agent towards the desired behavior [30], [31]. It involves modifying the reward function through integrating of domain expertise to make it easier for the agent to learn the optimal policy. This shortens the training time and increases the probability of successful training.

Therefore, instead of directly minimizing idle times, we shape the reward to focus on avoiding critical demands so that idle times do not occur in the first place. The advantage with using criticality is that it provides finer granularity in evaluating the agent's performance, which helps guide the learning process in the desired direction.

To this end, we define a reward function for each SFP type $r_{v/t,\tau}$ that punishes critical requirements more than non-critical. Critical in this context means that the ratio $v_{\tau,24}(t)/t_\tau(t)$ - the required amount of SFP type τ relative to the remaining time of how long the demand is covered - is high. The higher this value, the more likely idle times in a FAS will occur because the required quantity of type τ cannot be produced in time. Conversely, a smaller ratio indicates a less critical demand. The fewer SFPs are required (low value of $v_{\tau,24}$) or the more time is available for production (high value of t_τ), the more this ratio moves towards its low point of 0.

We define a section-wise linear function consisting of 3 sections (see Figure 1) to map the criticality of an SFP type to a continuous range of values. In the first range $[0 - 0.036)$ representing the non-critical situation, the reward decreases from 0 (no demand for this SFP type) to -2. The second range $[0.036 - 0.072)$ represents the critical situation. The reward

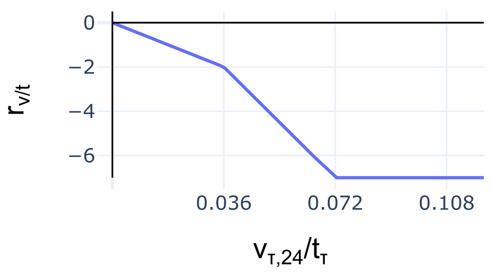


FIGURE 1. Reward Function for punishing the criticality of an SFP type.

drops from -2 to -7 with increasing ratio $v_{\tau,24}(t)/t_\tau(t)$. All ratios ≥ 0.072 reflect a highly critical situation, reflected by a reward of -7. The sum of these rewards is denoted as $r_{v/t}$:

$$r_{v/t} = \sum_{\tau \in \mathring{\tau}} r_{v/t,\tau}$$

Additionally, we define a penalty value of -1.5 for all demands that are required in less than $t_{mgn} = 30min$.

$$r_{mgn,\tau}(t) = \begin{cases} -1.5, & \text{if } t_\tau(t) < t_{mgn} \\ 0 \end{cases} \tag{14}$$

The sum of these rewards for all SFP types is denoted as r_{mgn}:

$$r_{mgn} = \sum_{\tau \in \mathring{\tau}} r_{mgn,\tau}$$

This motivates the agent to keep a sufficiently large stock in the buffer for all SFP types, so that a safety margin in terms of time is maintained.

To account for setup effort, we use $q_{\tau,\tau'}(t)$ as we did in (13). The total reward of this alternative approach using reward shaping is thus given by:

$$r_2(t) = r_{v/t} + r_{mgn} - \alpha_{se} q_{\tau,\tau'}(t) \tag{15}$$

Both approaches, r_1 and r_2, will be compared in the experiments.

4) EPISODE LENGTH

An episode ends when the buffer covers the demand for SFPs for the remaining planning period, but at the latest at s_{max}. However, if the capacity of the buffer is reached due to misplanning (see (10)) and at the same time the FAS cannot produce any more due to the lack of the required SFPs in the buffer, the episode is ended prematurely.

C. CURRICULUM LEARNING

The two objectives (minimization of idle times and setup efforts) conflict with each other. For instance, to minimize setup effort, there should ideally be no change of SFP types on station 1. This, in turn, would mean that the requirements of the SFPs would not be met, resulting in idle times. Conflicting optimization goals pose a challenge for the learning process. To guide the agent in the learning process, we use curriculum learning [32], [33], i.e. the agent is exposed to a sequence of tasks with increasing complexity. Thereby we divide the learning problem into 3 tasks from easy to hard. The agent starts with the easiest task until it masters it, and then learns the next harder task. The resulting algorithm is presented in Algorithm 1.

Since minimizing idle time is the main objective, this is trained in the first task. The punishment for setup effort from (15) is removed for this purpose:

$$r_{2,easy}(t) = r_{v/t} + r_{mgn} \tag{16}$$

To further simplify this task, the original action mask in equation 12 is constrained to produce only the 3 most critical SFPs. We define the mask as $\boldsymbol{m}_{easy}(t) = [m_{easy,0}(t), m_{easy,1}(t), \ldots, m_{easy,7}(t)]$, where each element is defined as:

$$m_{easy,\tau}(t) = \begin{cases} 1, & \text{if } v_{24,\tau}/t_\tau \text{ among the greatest 3 values} \\ 0 \end{cases} \tag{17}$$

At the end of each episode, the sum of all idle times $\sum_t d(t)$ is added to a Queue D with a capacity of 100. The agent learns the first task until the mean of D, i.e. the average sum of idle times of the last 100 episodes, is less than $100s$. Then, this task is considered as solved. After that, it switches to the next task. In task 2, the original action mask (12) is used. Thus, the agent must now learn from a larger set of SFPs to choose those that minimize idle times. In the last task, the penalty of setup effort is added to the reward function, resulting again in the original reward function (15). For reducing the training time, we use a parallel algorithm that leverages the computing resources of multiple CPUs on a machine.

It is theoretically possible that the initial buffer allocation is so poor that $mean(D) < 100s$ will never be reached. In this case, this condition would have to be relaxed. However, from the practical experience of domain experts, it is known that this case rarely occurs.

D. GENETIC ALGORITHM (GA)

In order to benchmark the performance of the RL agent in a representative way, a suitable metaheuristic algorithm was employed. Here, we utilized a Non-dominated Sorting GA (NSGA-II), which is a widely used multi-objective GA in the current scheduling literature as well as in many real-world applications [34], [35]. The GA was implemented in the Python framework pymoo, which provides a set of modern, suitably preconfigured metaheuristics for multi-objective optimization [36]. Appropriate and modified GA operators were used for fair comparability and are described in the following subsections.

1) SCHEDULE ENCODING AND EVALUATION

A solution candidate of a schedule (=individual) is represented as a sequential vector of SFP types $\tau \in \mathring{\tau}$ (see Table 2) to be processed on the PAS from left to right. The individual's SFP sequence includes all SFP types required by the FAS to complete all production jobs. In order to evaluate (encode) an

Algorithm 1 Parallelized PPO With Action Masking and Curriculum Learning

```
1:  Initialize parameters $\theta$.
2:  Initialize $Task = 1$ and Queue $D$ with length 100.
3:  for each iteration do
4:      for each actor do
            {parallelized across CPUs}
5:          Collect set of trajectories by running policy $\pi_\theta$ in
            the environment
6:      end for
7:      for each trajectory do
8:          At the end of the episode, append sum of idle times
            $\sum_t d(t)$ to $D$
9:      end for
10:     Update $\theta$ to maximize $L(\theta)$
11:     if mean($D$) $< 100$ then
12:         $T \leftarrow T + 1$
13:     end if
14:     if $T == 1$ then
15:         Use $m_{easy}$ and $r_{2,easy}$
16:     else if $T == 2$ then
17:         Use $m_a$ and $r_{2,easy}$
18:     else if $T \geq 3$ then
19:         Use $m_a$ and $r_2$
20:     end if
21: end for
```

individual and to determine its fitness, the simulation model introduced in Section IV-A1 is utilized. Upon completing the full simulation of the SFP sequence, the individual's fitness value, which corresponds to the objective function value (see Eq. 1 and 2), can be calculated by obtained simulation metrics.

2) INITIALIZATION

Due to the limited capacity of the central buffer, it was not possible to generate feasible individuals for the base population by randomly permuting the SFP sequence. As a result, the number of viable sequences for processing the SFP on the PAS is restricted and must be determined with consideration of this bottleneck. For this purpose, let the matrix M be the starting point for generating a feasible individual. Each row corresponds to a specific FAS, where the columns (from left to right) determine the predefined sequence based on $r_{i,j}$ (see Table 2). As also depicted in Figure 2, the individual is generated as follows:

1) For a diversity of possible sequences, M undergoes a line-wise shuffling using a randomly generated permutation matrix P: $M \leftarrow M \times P$
2) M is column-wise flattened to create a valid sequence for the PAS, where the buffer capacity cannot be exceeded. However, the setup effort is still high when using this vector.
3) From left to right: Remove all types from the sequence, which are initially available in the buffer and therefore not required to be processed on the PAS.
4) The sequence is clustered according to a randomly selected group size n, grouping identical SFP types and reducing setup efforts. The clustering starts with the first element of the sequence, followed by moving the next $n - 1$ elements of the same type in front of the current element. Afterwards, the next element in the sequence is selected and the process is repeated.

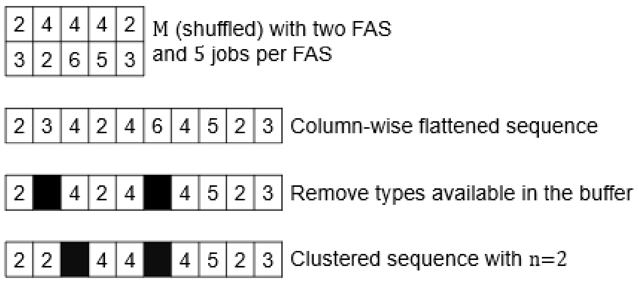

2 4 4 4 2
3 2 6 5 3
M (shuffled) with two FAS and 5 jobs per FAS
2 3 4 2 4 6 4 5 2 3
Column-wise flattened sequence
2 4 2 4 4 5 2 3
Remove types available in the buffer
2 2 4 4 4 5 2 3
Clustered sequence with n=2

FIGURE 2. An exemplary creation of an individual (SFP sequence) for the base population, taking into account the bottleneck of the central buffer and the clustering of identical SFP types to minimize PAS setup efforts.

3) CROSSOVER AND MUTATION

We used the well-known Job Order Crossover and Swap Mutation as operators to create neighbor individuals. Figure 3 illustrates the essential principle of the operators. Job Order Crossover involves combining two parent individuals: First, a random selection of SFP types from the first parent is transferred positionally to the offspring individual (see blue markings). Then, the free positions of the offspring are filled up with the remaining SFP types from the second parent in the order of the second parent (see red markings). [37] The Swap Mutation only changes the structure of a single individual by interchanging two random genes in their position (see green markings) [38].

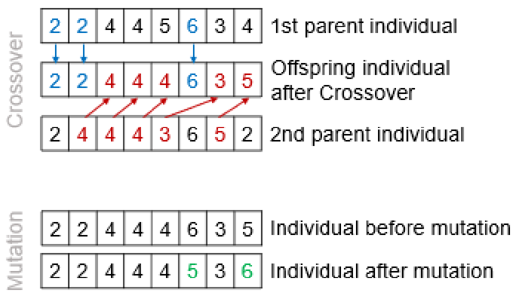


FIGURE 3. Simplified representation of Job Order Crossover and Swap Mutation.

IV. EXPERIMENTS

A. EXPERIMENTAL SETUP

1) PROBLEM INSTANCE, DATA, AND SIMULATION MODEL

Our approach is evaluated on realistic data from Miele production for 5 different weeks ($w1$, $w2$, $w3$, $w4$, $w5$). Each data set contains the following information for one week:

- The initial amount of SFPs in the buffer ($b_{\tau,0}$) for each type.
- A setup efforts matrix to determine $q_{\tau,\tau'}$.
- All work shifts S_k (including maintenance and break times) for the scheduling horizon for each station k.
- A set of jobs J with precedence constrains (determined by $r_{i,j}$). An example for all jobs assigned to a final assembly station for week 2 is shown in Figure 4.
- All processing times for all jobs on each station ($p_{i,k}$).

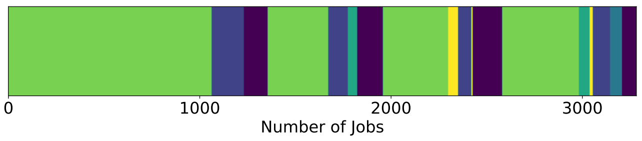


FIGURE 4. Example of all jobs of an FAS for week 2. Each color represents a different type.

As a basis for training the RL agent, a high-level simulation model of the household appliance production at Miele was developed with the DES FlexSim. Only the relevant parts and dependencies for the planning problem (see Sect. II) were modeled. Prior to each training episode or inference, the model is initialized with a week-specific data set. At each step, FlexSim collects all the data necessary to compute the state vector and reward and passes it to the agent. Based on these inputs, the agent decides which SFP type to produce next and sends that decision back to FlexSim. This is how the agent determines the production sequence in the PAS. A screenshot of the simulation model and its integration into the RL workflow is depicted in Figure 5.

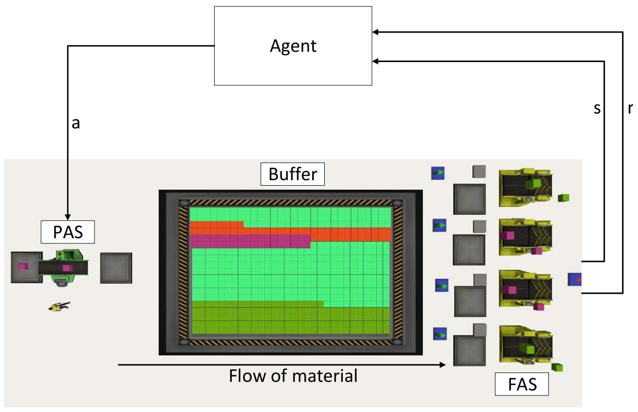


FIGURE 5. High-Level Simulation Model in FlexSim with conceptual integration of RL.

2) SOFTWARE ARCHITECTURE AND REINFORCEMENT LEARNING IMPLEMENTATION

We use a PPO algorithm from the Ray RLlib module [39], that we modify according to Algorithm 1 for training the RL agent. Since Ray RLlib requires Python, we encapsulated the FlexSim simulation model as a Python class. This class inherits from the Gymnasium module [40] and represents the training environment for the RL agent. FlexSim offers an RL connector that is used to communicate with the environment class via a socket connection.

For training, the environment class is handed over to Ray RLlib. The training process is parallelized with several environment instances on multiple CPUs, speeding up data collection. For inference, the trained agent π^* is packed in an HTTP Server. At every decision event, the simulation model queries the HTTP Server for the next action. The software architecture is shown in Figure 6.

To generate a schedule S^* for a week, the sampled actions are recorded sequentially in S^*. To do this, an environment *env* is first initialized with the data belonging to a week, such as $b_{\tau,0}$, S_k, etc. The environment provides the current state s_t and action mask m_a for each decision point, which are fed into π^* in order to sample the next action a_t. a_t is appended to $S*$ and fed back into the environment until the planning period is over or the buffer provides enough SFPs for all jobs (done=True). Algorithm 2 illustrates the generation of a schedule.

Algorithm 2 Schedule Generation With Trained PPO Agent
```
1: **Input** Trained PPO agent $\pi^*$, Environment $env$, $S^* = \emptyset$
2: $s_t$, $m_a$ = $env$.reset()
3: **while** not done **do**
4:     sample $a_t$ from $\pi^*(s_t)$ using action mask $m_a$
5:     append $a_t$ to $S^*$
6:     $s_t$, $m_a$, $done$ = $env$.step($a_t$)
7: **end while**
```

B. BENCHMARK WITH GENETIC ALGORITHM

In this experiment, the RL approach is benchmarked against GA in III-D. The hyperparameters for the RL algorithm were determined in preliminary experiments (Table 4). RL is trained with 700k steps, where one step is defined as producing o_1 SFPs. To ensure fair comparison, GA is stopped after the same number of steps. Both algorithms search for the optimal sequence of SFPs to be produced for five realistic data sets in order to lexicographically minimize idle times first and subsequently setup efforts. In this process, each trial is repeated 10 times to evaluate the stability and variance of the performance of the algorithms. The results are shown in Table 5. Here, an attempt is evaluated as failed if the algorithm was not able to minimize the idle time to 0. The average of the setup efforts $\overline{SE}$ refers only to the successful attempts of a week. For a more detailed evaluation of the setup efforts see Figure 7.

The results show that RL is significantly more robust in finding feasible solutions than GA. In fact, with RL all trials were successful, whereas with GA 38% of all trials failed. However, there is a significant fluctuation here between weeks. For example, the number of failed trials for week 1 is 8, whereas for week 2 it is 0. A possible explanation for this could be different complexity of the problem instances, where it might be more difficult to minimize the conflicting objectives. Further research should be undertaken to investigate the impact of the instance's structure on the problem's complexity.

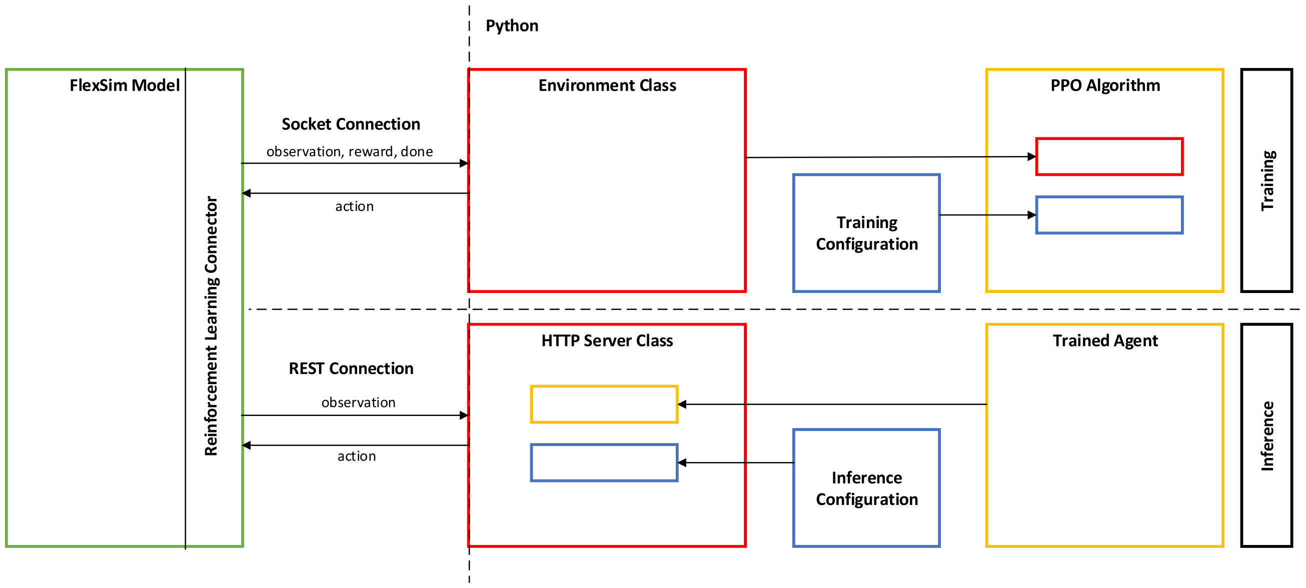


FIGURE 6. Software Architecture for training and inference of Ray RLlib RL agents with FlexSim.

TABLE 4. Hyperparameter for PPO algorithm. Nomenclature follows Ray RLlib.

Parameter	Value
`lambda`	0.95
`clip_param`	0.1
`lr`	1.7×10^{-4}
`gamma`	0.95
`train_batch_size`	1024
`sgd_minibatch_size`	32
`num_sgd_iter`	15
`fcnet_hiddens`	[256, 256]
`fcnet_activation`	relu

TABLE 5. Performance of RL and GA.

	RL		NSGA2	
week	N_{fail}	$\overline{SE}$	N_{fail}	$\overline{SE}$
1	0	227.7	8	956.5
2	0	144.0	0	197.3
3	0	140.1	1	221.1
4	0	127.0	6	204.0
5	0	140.0	4	132.2

Furthermore, the average setup efforts in weeks 1-4 are in some cases significantly smaller with RL than with the successful trials with GA. For example, with RL at week 3 the setup effort is on average 36.6% smaller compared to GA, at week 1 even by 76.1%. Likewise, the variance of the setup efforts for weeks 1-4 is smaller with RL, as can be seen from the boxplots. At week 5, no successful solution could be learned with GA 4 times. The remaining 6 trials, on the other hand, show an average of 5.6% smaller setup effort compared to the RL results.

Overall, it can be concluded that our proposed RL approach leads much more robustly to successful solutions with on average less setup effort compared to state-of-the-art metaheuristics.

C. ABLATION STUDIES

In the following, we examine the impact of reward shaping (RS), action masking (AM), and curriculum learning (CRCL) on the performance of the RL solution. For this purpose, we conducted several experiments with modified variants of our algorithm, in which we successively disabled some of these elements to quantify their impact on the success rate of the training. We tested the following variants:

- `AM+CRCL+RS`: Original variant, serving as baseline.
- `CRCL+RS`: Here, no AM was used, i.e., the agent could choose any action at any decision point. Accordingly, the curriculum learning protocol was adapted so that the first task was skipped.
- `AM+RS`: In this case, no Curriculum Learning was used. However, the action space of the agent has been restricted to the extent that only those SFPs may be produced for which there is a need until the end of the planning horizon (see Eq. 12).
- `RS`: No use of action masking or curriculum learning.
- `AM+CRCL`: The natural approach using idle time and setup effort directly as punishment is chosen as the reward function (see Eq. 13).

Each variant is thereby trained 10 times each for week 1 and 2. According to the success rate of GA, week 1 and 2 represent a difficult and a rather easy planning problem, respectively. The results are shown in Figure 8.

Omitting AM or CRCL leads to a 40% reduction in the successful training rate at week 1. This highlights the ability of AM and CRCL to enable the agent to find better policies in challenging planning problems. For week 2, omitting CRCL leads to a reduction of only 20% due to the lower problem complexity, while omitting AM has no effect on the success rate at all. The success rate for the `RS` variant is also noteworthy. Here, omitting both AM and CRCL at week

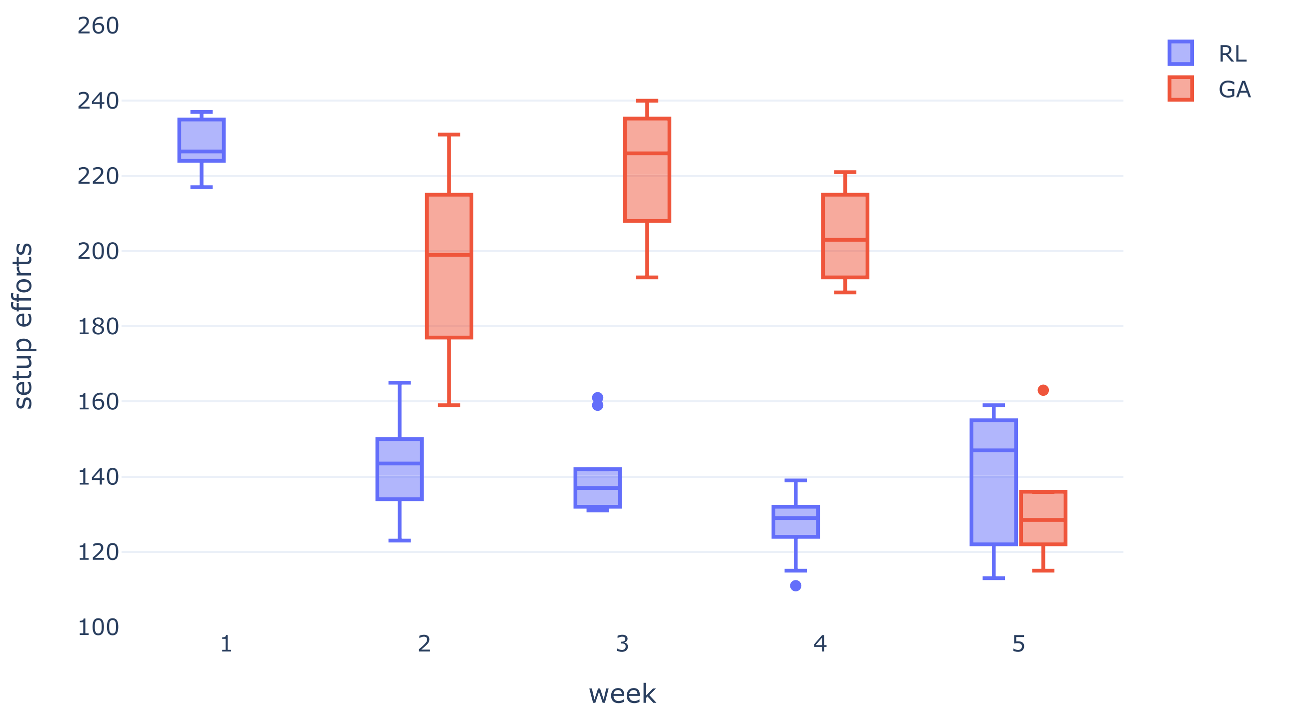


FIGURE 7. Setup efforts as boxplots for all successful trials. The results for GA at week 1 are not shown here because they are significantly larger than the other values.

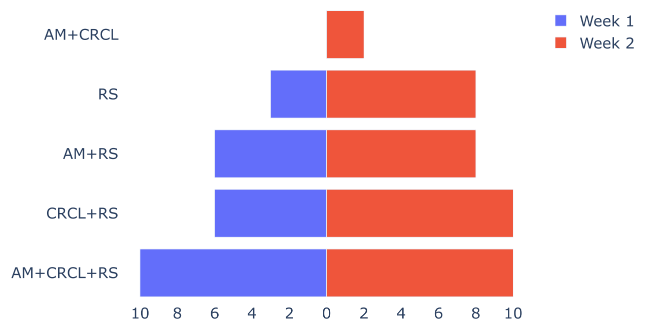


FIGURE 8. Number of successful trials for different ablations. AM=action Masking, CRCL=curriculum learning, RS=reward shaping.

1 leads to a 70% reduction in the success rate compared to the original variant. This emphasizes that also the interaction of AM and CRCL helps the agent to find good policies and not only the single algorithmic elements themselves. The highest impact on the success rate, however, is from reward shaping, as can be seen in the results of `AM+CRCL`. In this case, for week 1, no successful agent could be trained in any of the 10 trials. For week 2, the success rate also dropped significantly to only 20%.

It can be concluded from the ablation studies that integrating domain knowledge through RS, AM and CRCL is necessary to generate successful and performant schedules. This is even more true the more complex the data set is. Notably, these algorithmic building blocks are necessary to overcome the solving difficulties of the competing objectives in the two interdependent stages. At the same time, however, this also shows that these building blocks are sufficient to provide the agent with enough assistance during training. Thus, a purely RL-based approach to solving this optimization problem is possible, whereas many rather runtime-intensive approaches in the literature combine RL with other methods, as shown in section I.

D. SETUP EFFORTS WEIGHTING FACTOR

To mitigate the challenge of conflicting objectives, curriculum learning was applied, so that the agent first learns to minimize the idle times and then the setup efforts. However, balancing these objectives through the weight α_{se} in the reward function is also crucial, since it determines the trade-off between minimizing idle times and setup efforts. Therefore, in this experiment we investigate the impact of this weight by varying it from 0 to 16 in steps of 0.5 and training 5 runs for each of these values for weeks 1 and 2.

The average sum of the idle times and setup efforts per α_{se} are shown in Figure 9. First, it is noticeable that between 0 - the agent focuses only on minimizing idle times - and an only slightly higher value of 0.5, there is a significant reduction in setup efforts. For instance, in week 1, setup efforts are reduced from an average of 1829 to 343.

As α_{se} increases, setup efforts continue to decrease, as expected, since the agent gives more weight to minimizing them. However, an excessively high α_{se} leads the agent to prefer accepting idle times rather than planning in a way that ensures the final assembly lines always have an adequate supply of SFPs. This effect is observed at week 1 starting at an α_{se} of 9.5, where idle times are on average 1041s. As α_{se} increases, this effect occurs more severely, so that at

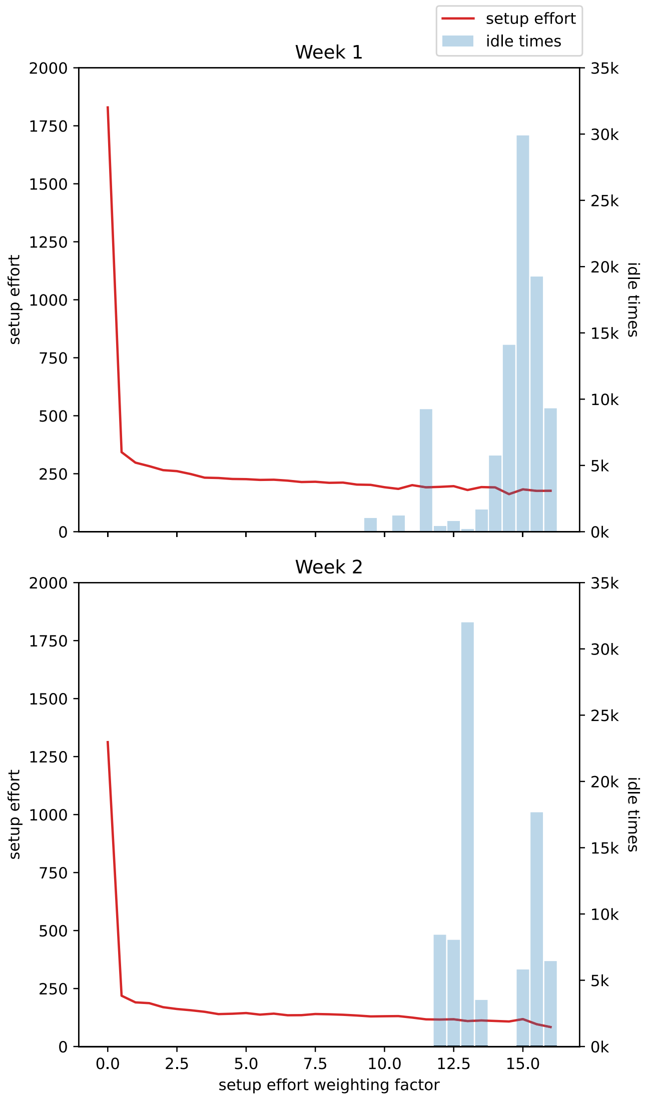


FIGURE 9. **Setup efforts and idle times for different α_{se}.**

$\alpha_{se} = 15$ there are more than $8h$ of idle times. For week 2, this effect occurs somewhat later (from $\alpha_{se} = 12$), but with the same magnitude in terms of the resulting idle times, e.g. ca. $8.9h$ at $\alpha_{se} = 13$.

This effect is thus dependent on the complexity of the data set. Therefore, it would be ideal to fine-tune α_{se} for each week to achieve the lowest possible setup effort. However, it is worth noting that there is a fairly wide range of values for α_{se} in which setup effort remains low and no idle time occurs, while further increasing α_{se} results in only a small reduction in setup effort. This mitigates the need for fine-tuning. For example, doubling α_{se} from 3.5 to 7 only results in a reduction of setup effort by only ca. 19 respectively 8.1% for week 1.

V. CONCLUSION AND FUTURE RESEARCH

In this paper, we addressed a scheduling problem occurring in one of Miele's household appliance productions: a two-stage PFSSP with a finite buffer, sequence-dependent setup efforts, and work shifts. The objective is to minimize the idle times and setup efforts in lexicographic order. For this purpose, the problem was formulated as a Markov decision process and then solved with RL. We developed an RL approach that integrates domain knowledge through reward shaping, action masking and curriculum learning.

Experiments on realistic data show the superiority of our approach over a state-of-the-art GA. They also demonstrated that incorporating domain knowledge is critical to successful planning on complex data sets. In addition, we investigated adjusting the setup effort weighting parameter to ensure minimization of idle time and setup effort in a lexicographic order. Moreover, we developed a software architecture to connect the DES to the Ray RLlib framework. Our work thus provides a successful example of the applicability of RL in real-world production planning.

In the future, we plan to further develop the system for real-world use, addressing three aspects in particular: **robustness**, **explainability**, and **universality**.

In real-world production, sporadic disturbances (e.g. because of machine failures) are unavoidable. Therefore, our approach has to be extended so that the agent can generate sequences that avoid idle times and keep setup efforts as minimal as possible despite occurring disturbances. The approach should make it possible to integrate historical data on disturbances such as machine failures in order to make the plans only as robust as necessary so that the actual optimization goals are not unnecessarily underprioritized.

Furthermore, the agent's decisions are to be made transparent so that production planners working with this system can understand and evaluate the decisions. And finally, an agent is to be developed that can perform well over a variety of different problem instances with varying complexities in a *zero-shot* or *few-shot* learning manner. Such an universal agent would save training time, since currently for each week a new agent has to be trained.

ACKNOWLEDGMENT

The authors would like to thank Felix Möhlmann from FlexSim Germany for his advice and support in building the simulation model.

REFERENCES

[1] S. M. Johnson, "Optimal two- and three-stage production schedules with setup times included," *Nav. Res. Logistics Quart.*, vol. 1, no. 1, pp. 61–68, Mar. 1954.
[2] C. Shyalika, T. Silva, and A. Karunananda, "Reinforcement learning in dynamic task scheduling: A review," *Social Netw. Comput. Sci.*, vol. 1, no. 6, p. 306, Nov. 2020.
[3] M. Panzer and B. Bender, "Deep reinforcement learning in production systems: A systematic literature review," *Int. J. Prod. Res.*, vol. 60, no. 13, pp. 4316–4341, Jul. 2022.
[4] B. M. Kayhan and G. Yildiz, "Reinforcement learning applications to machine scheduling problems: A comprehensive literature review," *J. Intell. Manuf.*, vol. 34, no. 3, pp. 905–929, Mar. 2023.
[5] F. Grumbach, N. E. A. Badr, P. Reusch, and S. Trojahn, "A memetic algorithm with reinforcement learning for sociotechnical production scheduling," *IEEE Access*, vol. 11, pp. 68760–68775, 2023.
[6] E. Taillard, "Benchmarks for basic scheduling problems," *Eur. J. Oper. Res.*, vol. 64, no. 2, pp. 278–285, Jan. 1993.
[7] J. Carlier, "Ordonnancements a contraintes disjonctives," *RAIRO-Oper. Res.*, vol. 12, no. 4, pp. 333–350, 1978.
[8] C. R. Reeves, "A genetic algorithm for flowshop sequencing," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 5–13, Jan. 1995.

[9] J. Heller, ''Some numerical experiments for an M × J flow shop and its decision-theoretical aspects,'' *Oper. Res.*, vol. 8, no. 2, pp. 178–184, Apr. 1960.

[10] C.-X. Wu, M.-H. Liao, M. Karatas, S.-Y. Chen, and Y.-J. Zheng, ''Real-time neural network scheduling of emergency medical mask production during COVID-19,'' *Appl. Soft Comput.*, vol. 97, Dec. 2020, Art. no. 106790.

[11] J. F. Ren, C. M. Ye, and Y. Li, ''A new solution to distributed permutation flow shop scheduling problem based on Nash Q-learning,'' *Adv. Prod. Eng. Manage.*, vol. 16, no. 3, pp. 269–284, Sep. 2021.

[12] Z. Pan, L. Wang, J. Wang, and J. Lu, ''Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling,'' *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 7, no. 4, pp. 983–994, Aug. 2023.

[13] S. Yang and Z. Xu, ''Intelligent scheduling for permutation flow shop with dynamic job arrival via deep reinforcement learning,'' in *Proc. IEEE 5th Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Mar. 2021, pp. 2672–2677.

[14] Q. Yan, W. Wu, and H. Wang, ''Deep reinforcement learning for distributed flow shop scheduling with flexible maintenance,'' *Machines*, vol. 10, no. 3, p. 210, Mar. 2022.

[15] Z. Dong, T. Ren, J. Weng, F. Qi, and X. Wang, ''Minimizing the late work of the flow shop scheduling problem with a deep reinforcement learning based approach,'' *Appl. Sci.*, vol. 12, no. 5, p. 2366, Feb. 2022.

[16] X. Gao, S. Yang, and L. Li, ''Optimization of flow shop scheduling based on genetic algorithm with reinforcement learning,'' *J. Phys., Conf. Ser.*, vol. 2258, no. 1, Apr. 2022, Art. no. 012019.

[17] J. Brammer, B. Lutz, and D. Neumann, ''Permutation flow shop scheduling with multiple lines and demand plans using reinforcement learning,'' *Eur. J. Oper. Res.*, vol. 299, no. 1, pp. 75–86, May 2022.

[18] Z. He, K. Wang, H. Li, H. Song, Z. Lin, K. Gao, and A. Sadollah, ''Improved Q-learning algorithm for solving permutation flow shop scheduling problems,'' *IET Collaborative Intell. Manuf.*, vol. 4, no. 1, pp. 35–44, Mar. 2022.

[19] H. Li, K. Gao, P.-Y. Duan, J.-Q. Li, and L. Zhang, ''An improved artificial bee colony algorithm with Q-learning for solving permutation flow-shop scheduling problems,'' *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 5, pp. 2684–2693, May 2023.

[20] Z. Pan, L. Wang, C. Dong, and J.-F. Chen, ''A knowledge-guided end-to-end optimization framework based on reinforcement learning for flow shop scheduling,'' *IEEE Trans. Ind. Informat.*, early access, 2023.

[21] J. Ren, C. Ye, and F. Yang, ''Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network,'' *Alexandria Eng. J.*, vol. 60, no. 3, pp. 2787–2800, Jun. 2021.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, ''Proximal policy optimization algorithms,'' 2017, *arXiv:1707.06347*.

[24] F. Grumbach, A. Müller, P. Reusch, and S. Trojahn, ''Robust-stable scheduling in dynamic flow shops based on deep reinforcement learning,'' *J. Intell. Manuf.*, vol. 2022, pp. 1–12, Dec. 2022.

[25] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, ''High-dimensional continuous control using generalized advantage estimation,'' in *Proc. 4th Int. Conf. Learn. Represent.*, Jun. 2015, pp. 1–14.

[26] L. Zhang, C. Yang, Y. Yan, and Y. Hu, ''Distributed real-time scheduling in cloud manufacturing by deep reinforcement learning,'' *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 8999–9007, Dec. 2022.

[27] Z. Ling, X. Wang, and F. Qu, ''Reinforcement learning-based maintenance scheduling for resource constrained flow line system,'' in *Proc. IEEE 4th Int. Conf. Control Sci. Syst. Eng. (ICCSSE)*, Aug. 2018, pp. 364–369.

[28] A. Kanervisto, C. Scheller, and V. Hautamäki, ''Action space shaping in deep reinforcement learning,'' in *Proc. IEEE Conf. Games (CoG)*, Aug. 2020, pp. 479–486.

[29] A. Müller and M. Sabatelli, ''Safe and psychologically pleasant traffic signal control with reinforcement learning using action masking,'' in *Proc. IEEE 25th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2022, pp. 951–958.

[30] M. E. Taylor and P. Stone, ''Transfer learning for reinforcement learning domains: A survey,'' *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Jul. 2009.

[31] B. Badnava, M. Esmaeili, N. Mozayani, and P. Zarkesh-Ha, ''A new potential-based reward shaping for reinforcement learning agent,'' in *Proc. IEEE 13th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Mar. 2023, pp. 1–6.

[32] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, ''Curriculum learning,'' in *Proc. Int. Conf. Mach. Learn.*, Montreal, QC, Canada, Jun. 2009, pp. 41–48.

[33] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, ''Curriculum learning for reinforcement learning domains: A framework and survey,'' *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 1–13, Jan. 2020.

[34] R. Washington, D. Garmatyuk, S. Mudaliar, and R. M. Narayanan, ''Many-objective RadarCom signal design via NSGA-II genetic algorithm implementation and simulation analysis,'' *Remote Sens.*, vol. 14, no. 15, p. 3787, Aug. 2022.

[35] W. Zheng, Y. Liu, and B. Doerr, ''A first mathematical runtime analysis of the non-dominated sorting genetic algorithm II (NSGA-II): (Hot-off-the-press track at GECCO 2022),'' in *Proc. Genetic Evol. Comput. Conf. Companion*. New York, NY, USA, Jul. 2022, pp. 53–54.

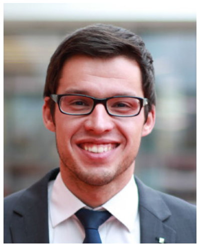
[36] J. Blank and K. Deb, ''Pymoo: Multi-objective optimization in Python,'' *IEEE Access*, vol. 8, pp. 89497–89509, 2020.

[37] I. Ono, M. Yamamura, and S. Kobayashi, ''A genetic algorithm for job-shop scheduling problems using job-based order crossover,'' in *Proc. IEEE Int. Conf. Evol. Comput.*, Mar. 1996, pp. 547–552.

[38] S.-H. Chen and M.-C. Chen, ''Operators of the two-part encoding genetic algorithm in solving the multiple traveling salesmen problem,'' in *Proc. Int. Conf. Technol. Appl. Artif. Intell.*, Nov. 2011, pp. 331–336.

[39] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, ''RLlib: Abstractions for distributed reinforcement learning,'' in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 7, 2018, pp. 4768–4780.

[40] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. de Cola, T. Deleu, M. Goulão, A. Kallinteris, K. G. Arjun, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, ''Gymnasium,'' Zenodo, Farama Found., Greenbelt, MD, USA, Mar. 2023. Accessed: Jul. 8, 2023. [Online]. Available: https://zenodo.org/record/8127025, doi: 10.5281/zenodo.8127026.



ARTHUR MÜLLER received the B.Sc. degree in electrical engineering and information technology from the Ostwestfalen-Lippe University of Applied Sciences and Arts, Lemgo, Germany, in 2012, and the M.Sc. degree from Gottfried Wilhelm Leibniz University, Hannover, Germany, in 2015. He is currently pursuing the Ph.D. degree with the University of Groningen, The Netherlands. He is also a Research Associate with the Industrial Automation Branch, Fraunhofer Institute of Optronics, System Technologies, and Image Exploitation, Lemgo. His research interest includes the application of reinforcement learning algorithms in real-world use cases.

FELIX GRUMBACH received the B.Sc. degree in information systems from the Bielefeld University of Applied Sciences, Bielefeld, Germany, in 2015, and the M.Sc. degree from the University of Hagen, Germany, in 2020. He is currently pursuing the Ph.D. degree with the Doctoral Center for Social, Health and Economic Sciences, Saxony-Anhalt, Germany. He is also a Research Associate with the Center for Applied Data Science, Bielefeld University of Applied Sciences. His research interest includes the robust and holistic optimization of complex production processes with the help of machine learning techniques.

FIONA KATTENSTROTH received the B.Eng. degree in industrial engineering from the Bielefeld University of Applied Sciences, Germany, in 2019, and the M.Eng. degree from the Ostwestfalen-Lippe University of Applied Sciences and Arts, Lemgo, Germany, in 2020. She is currently pursuing the Ph.D. degree with the University Paderborn, Germany, in cooperation with the Fraunhofer Institute for Mechatronic Systems Design. She is also a Simulation Engineer with Miele & Cie.KG Company, Gütersloh, Germany. Her research interest includes the longterm use of discrete-event simulation for industrial use cases.