European Journal of Operational Research 299 (2022) 75–86

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Permutation flow shop scheduling with multiple lines and demand plans using reinforcement learning

Janis Brammer[a,*], Bernhard Lutz[b], Dirk Neumann[b]

[a] Volkswagen AG, Berliner Ring 2, Wolfsburg 38440, Germany
[b] University of Freiburg, Rempartstr. 16, Freiburg 79098, Germany

ARTICLE INFO

Article history:
Received 5 November 2019
Accepted 7 August 2021
Available online 14 August 2021

Keywords:
Scheduling
Permutation flow shop problem
Reinforcement learning
Mixed-integer programming
Constraint programming

ABSTRACT

Existing studies on the permutation flow shop problem (PFSP) commonly assume that jobs are produced on a single line. However, manufacturers may speed up their production by employing multiple lines, where each line produces sub-parts of the final product; which must be assembled by a synchronization machine. This study presents a novel reinforcement learning (RL) approach for the PFSP with multiple lines and demand plans. Our approach differs from existing RL-based scheduling methods as we train the policy to directly generate the sequence in an iterative way, where actions denote the job type to be sequenced next. During cutoff time, we follow a multistart approach that generates sequences with the trained policy, which are subsequently optimized by local search. Our numerical evaluation based on 1050 problem instances with up to three production lines shows that our approach outperforms existing methods on the multi-line problems for short cutoff times, while there is a tie with existing methods for medium and long cutoff times. A further analysis suggests that our approach can also be applied to problems with imbalanced demand plans.

1. Introduction

In the permutation flow shop problem (PFSP), n jobs have to be processed on m machines, where jobs must be processed in a fixed order and are not allowed to overtake each other (Johnson, 1954). For each job, we are given the processing time on each machine. The objective is to find the optimal sequence of jobs to minimize the overall makespan. Shorter makespans imply higher sales numbers, which yields financial benefits for manufacturers. For more than two machines, PFSP is NP-complete (Garey, Johnson, & Sethi, 1976). This means that it is generally not possible to find an optimal solution in a feasible time. In order to find approximate solutions in real-world settings with short cutoff times, researchers have proposed several heuristic solutions (e.g., Komaki, Sheikh, & Malakooti, 2019; Nawaz, Enscore, & Ham, 1983; Ruiz & Maroto, 2005).

Although PFSP is one of the most studied problems in operations research (Fernandez-Viagas, Ruiz, & Framinan, 2017; Komaki et al., 2019), previous studies typically assumed all machines to be located on a single production line. However, manufacturers may speed up their production by employing multiple production lines, where sub-parts of a product are produced in parallel. We have encountered the PFSP with multiple production lines at a large European car manufacturer, where several lines produce sub-parts of the car components, which are finally assembled by a synchronization machine. In their use case, the car underbody is produced on a single line, while two side bodies are produced individually on two other lines. At the synchronization machine, all three parts are welded together to form the car body. The quantities to be produced for each job type are given by a demand plan. This problem extension occurs in the car industry, where jobs are not unique (Bautista & Alfaro, 2018). The multi-line problem was first introduced by Guimaraes, Ouazene, de Souza, & Yalaoui (2016), who proposed a mixed integer linear programming (MILP) formulation and several simplification strategies to reduce the multi-line problem to a one-line problem. Thereby, it becomes possible to apply established constructive heuristics like Johnson (1954)'s method and the NEH heuristic (Nawaz et al., 1983). In their recent study, Guimarães, Ouazene, de Souza, & Yalaoui (2019) improved on their MILP problem formulation and proposed a greedy randomized adaptive search procedure (GRASP), which outperforms the approaches of their previous study.

This study presents a novel reinforcement learning (RL) approach for the PFSP with multiple lines and demand plans. The RL policy generates the sequence incrementally, so that actions

* Corresponding author.
E-mail addresses: janis.brammer@volkswagen.de (J. Brammer), bernhard.lutz@is.uni-freiburg.de (B. Lutz), dirk.neumann@is.uni-freiburg.de (D. Neumann).

https://doi.org/10.1016/j.ejor.2021.08.007
0377-2217/

denotes the job type to be sequenced next. The state representation contains intermediate information about the simulated production process with respect to the current partial sequence. To guide the policy towards minimizing the makespan, the learning environment assigns negative rewards to actions that cause idle time on the synchronization machine. Once the policy is learned, sequences are generated quickly, which allows us to apply a multistart approach to continuously generate sequences from the trained policy as an initial solution for local search. For our numerical evaluation, we extend the dataset by Taillard (1993) to 1050 problems instances with up to three production lines and demand plans. We find that, on the multi-line problems, our approach outperforms existing methods for short cutoff times. For medium and long cutoff times, our approach is tied with existing methods. On the one-line problems, iterated greedy (Ruiz & Stützle, 2007) remains the dominant approach. In our main analyses, demand plans are sampled with equal probability for each job type, which yields rather balanced distributions. However, a subsequent analysis shows that the performance of our approach also persists on problems with more imbalanced demand plans.

The proposed RL approach hence differs from existing RL-based scheduling methods, e.g., for the traveling salesman problem (see Bello, Pham, Le, Norouzi, & Bengio, 2016; Bengio, Lodi, & Prouvost, 2021), where the state representation is given by the problem characteristics and the action space covers all possible sequences. Our approach can also be distinguished from existing RL methods for the PFSP, where RL selects the hyperparameters of an adaptive learning approach (Agarwal, Colak, & Eryarsoy, 2006), or dispatching rules like shortest or longest processing time (Zhang, Wang, Zhong, & Hu, 2013). The main advantage of our RL approach over established heuristics and MILP or CP solvers is that it builds on a pre-trained scheduling policy, which allows manufacturers to shift extensive computations before the actual sequence generation, which must be done in a comparably short cutoff time. Once the policy is learned, our approach can transfer knowledge from the pre-trained policy to similar but unseen problems. MILP and CP solvers and existing heuristics solve each problem individually during the available cutoff time without relying on any form of precomputation. We argue that the time needed for policy learning can be neglected as this can be done in advance for a fixed production system, providing the distribution of the demand plan is known.

The remainder of this paper is structured as follows. Section 2 formalizes the PFSP with multiple lines and demand plans. Section 3 describes our reinforcement learning approach by detailing action space, reward function, state representation, and the method for policy learning. Section 4 describes our numerical evaluation and Section 5 presents the results. Section 6 concludes and presents avenues for future research.

2. Permutation flow shop problem with multiple lines and demand plans

2.1. Problem formulation

We adopt the problem formulation proposed by Guimarães et al. (2019), where the permutation flow-shop problem is formalized as an MILP. The MILP formulation is based on the prior studies of Wagner (1959), Stafford (1988), and Tseng, Stafford, & Gupta (2004).[1] In contrast to the default PFSP, this extended version allows for multiple production lines $l = 1, \dots, v$ (Guimaraes et al., 2016). Each line l contains one or more machines $m^l_1, \dots, m^l_{n_l}$, where n_l denotes the number of machines in line l. All lines work independently until the job is transferred to the synchronization machine m^S, which can only start processing a job if it has been fully processed on all production lines. The job quantities to be produced are given by a demand plan $d = [d_1, \dots, d_n]$ (Bautista & Alfaro, 2018). The total number of jobs is denoted by $J = \sum_{i=1}^{n} d_i$. For each job type i, we are given the processing time $p^l_{i,k}$ on machine k of line l, and the processing time p^S_i on m^S. Each machine can only process one job at the same time and jobs must be processed in the same order on all production lines. Between each two machines, we assume an unlimited buffer, likewise for the final synchronization machine and its supplying lines. We follow Guimaraes et al. (2016) and Guimarães et al. (2019) in virtually inserting the synchronization machine as the last machine on each line: $m^S = m^l_{n_l+1},\ l = 1, \dots, v$. An overview of all parameters and variables is provided in Table 1.

The full optimization problem is formalized below. The objective is to minimize the total makespan of the sequence (1). Eq. (2) ensures that the decision variables are boolean. Eq. (3) ensures that the idle time variables are non-negative. Eq. (4) ensures that the sequence complies with the demand plan. Eq. (5) ensures that exactly one job type is sequenced in each sequence position. Eq. (6) ensures that the first job has no idle time after being processed on any machine. Eq. (7) ensures that the idle time of the synchronization machine while waiting to process the first job is equal to the largest processing time of the first job over all lines. Eq. (8) states that the first job may have to wait to be processed on the synchronization machine. Eq. (9) calculates the idle time for the first job, while waiting to be processed on the second and all subsequent machines of all lines. Eq. (10) presents the job-adjacency, machine linkage constraint (Tseng & Stafford, 2001). This constraint defines the idle time of m^l_{k+1} before processing the job at position $t+1$ for all lines. The machine idle time of m^l_{k+1} before processing the job at position $t+1$ equals the machine idle time of the previous machine m^l_k before processing the job at position $t+1$ plus the idle time of the job at position $t+1$ waiting to be processed on m^l_k plus the processing time of the job at position $t+1$ on m^l_k. However, machine m^l_{k+1} is not idle while the job at position t is waiting to be processed on m^l_{k+1} and while the job at position t is processed on m^l_{k+1}. Eqs. (11) and (12) ensure the synchronization, that is, all jobs have to be completed on all lines before they can be processed on m^S.

$$\text{Minimize } c_{\max} = c^1_J \tag{1}$$

Subject to:

$$z_{i,t} \in \{0,1\} \qquad t = 1, \dots, J;\ i = 1, \dots, n \tag{2}$$

$$y^l_{k,t}, x^l_{k,t} \geq 0 \qquad t = 1, \dots, J;\ k = 1, \dots, n_l + 1;\ l = 1, \dots, v; \tag{3}$$

$$\sum_{t=1}^{J} z_{i,t} = d_i \qquad i = 1, \dots, n \tag{4}$$

$$\sum_{i=1}^{n} z_{i,t} = 1 \qquad t = 1, \dots, J \tag{5}$$

$$y^l_{k,1} = 0 \qquad k = 1, \dots, n_l;\ l = 1, \dots, v \tag{6}$$

$$\sum_{r=1}^{n_l} \sum_{i=1}^{n} p^l_{i,r} z_{i,1} \leq x^S_1 \qquad l = 1, \dots, v \tag{7}$$

[1] Several recent studies have also modeled flowshop problems as an MILP (see Komaki et al., 2019).

Table 1
Parameters and variables.

Parameters	
$i = 1, \ldots, n$	Job types
$d = [d_1, \ldots, d_n]$	Demand plan defines the quantity to be produced for job type i
$J = \sum_{i=1}^{n} d_i$	Total number of jobs to be produced
$t = 1, \ldots, J$	Sequence positions
$l = 1, \ldots, v$	Production lines
n_l	Number of machines on line l
m_k^l	Machine k of line l
m^S	Synchronization machine virtually inserted as last machine on each line $m^S = m_{n_l+1}^l,\ l = 1, \ldots, v$
$p_{i,k}^l$	Processing time of job type i on m_k^l
p_i^S	Processing time of job type i on m^S
Variables	
$c_{\max}$	Completion time of the job at the last sequence position
c_t^l	Completion time of job at sequence position t on line l including processing by m^S
$z_{i,t}$	Binary decision variable. Equals 1 if a job of type i is assigned to the t^{th} position in the sequence, and 0 otherwise
$x_{k,t}^l$	Idle time on machine m_k^l before processing the job at position t
x_t^S	Idle time on machine m^S before processing the job at position t. $x_t^S = x_{n_l+1,t}^l,\ t = 1, \ldots, J;\ l = 1, \ldots, v$
$y_{k,t}^l$	Idle time of job at position t, after being processed by machine m_k^l and waiting for processing on machine m_{k+1}^l

$$x_1^S - \left(x_{n_l,1}^l + \sum_{i=1}^{J} p_{i,n_l}^l z_{i,1} \right) = y_{n_l,1}^l \qquad l = 1, \ldots, v \tag{8}$$

$$\sum_{r=1}^{k-1} \sum_{i=1}^{n} p_{i,r}^l z_{i,1} = x_{k,1}^l \qquad k = 2, \ldots, n_l;\ l = 1, \ldots, v \tag{9}$$

$$x_{k+1,t+1}^l = x_{k,t+1}^l + \sum_{i=1}^{n} p_{i,k}^l z_{i,t+1} + y_{k,t+1}^l - y_{k,t}^l - \sum_{i=1}^{n} p_{i,k+1}^l z_{i,t}$$
$$t = 1, \ldots, J-1;\ k = 1, \ldots, n_l;\ l = 1, \ldots, v \tag{10}$$

$$\sum_{u=1}^{t} \sum_{i=1}^{n} p_i^S z_{i,u} + \sum_{u=1}^{t} x_u^S = c_t^l \qquad t = 1, \ldots, J;\ l = 1, \ldots, v \tag{11}$$

$$c_t^l = c_t^{l-1} \qquad t = 1, \ldots, J;\ l = 2, \ldots, v \tag{12}$$

Theorem 1. *For PFSP with multiple lines and demand plans, minimizing the idle time on the synchronization machine also minimizes the overall makespan.*

Proof. We first show that the overall makespan $c_{\max}$ is equal to the sum of the overall working time of m^S and the overall idle time of m^S. Let P_{work}^S denote the overall working time of the synchronization machine m^S and P_{idle}^S denote the overall idle time of m^S. Combining (1), (11), and (12) leads to

$$c_{\max} = \sum_{u=1}^{J} \sum_{i=1}^{n} p_i^S z_{i,u} + \sum_{u=1}^{J} x_u^S \tag{13}$$

$$c_{\max} = P_{work}^S + P_{idle}^S \tag{14}$$

Second, we show that P_{work}^S is the same for any sequence given a fixed demand plan $d = [d_1, \ldots, d_n]$. Since each sequence only permutates the order in which jobs are processed, P_{work}^S is equal to

$$P_{work}^S = \sum_{i=1}^{n} d_i\, p_i^S. \tag{15}$$

Hence, P_{work}^S can be seen as constant and acts as a theoretical lower bound for the overall makespan of a given demand plan. This implies that the overall makespan only depends upon P_{idle}^S. □

2.2. Example

We provide an illustrative example with two production lines containing two machines each. There are $n = 3$ job types grouped according to the demand plan $d = [2, 1, 2]$. Let the sequence of job types be defined as $seq = [1, 2, 3, 3, 1]$. The resulting sequence diagram is presented in Fig. 1. The processing times $p_{i,k}^l$ of the job types on the machines are denoted as white time slots, e.g., $p_{1,2}^1 = 4$, $p_1^S = 5$. The arrows denote transfers of jobs between machines. Time slots highlighted in dark-gray denote idle times on the respective machines. The idling is caused by jobs that are not ready on one or more previous machines. The total idle time on m^S is 13 and the overall makespan of this sequence is $c_{\max} = P_{work}^S + P_{idle}^S = 17 + 13 = 30$.

3. Reinforcement learning approach

Reinforcement learning is a special type of machine learning where an agent interacts with an environment to learn a policy π^* by maximizing a numerical reward signal (Sutton, Barto et al., 1998). In the following, we describe the environment, action space, reward function, state representation, and the algorithm for policy learning.

3.1. Environment

The environment executes the action of the agent and returns the next state and a reward. In our study, the environment models the task of sequence generation for PFSP with multiple lines and demand plans. At each sequence position $t = 1, \ldots, J$, the agent receives the current state s_t of the flow shop. Based on the state s_t, the agent selects an action a_t, i.e., the job type to be sequenced at position t. After receiving an action a_t, the environment schedules the selected job type i at position t and simulates the processing of the partial sequence. Subsequently, the environment returns the next state s_{t+1} and the reward r_t for action a_t to the agent. The simulation ends when all jobs have been sequenced.

3.2. Actions

Actions allow the agent to interact with its environment. In each step, the agent selects an action $a_t \in A$, where A denotes the action space. In our environment, a_t denotes the job type to be sequenced at position t. Therefore, A is defined as the set of job types

$$A = \{1, \ldots, n\}. \tag{16}$$

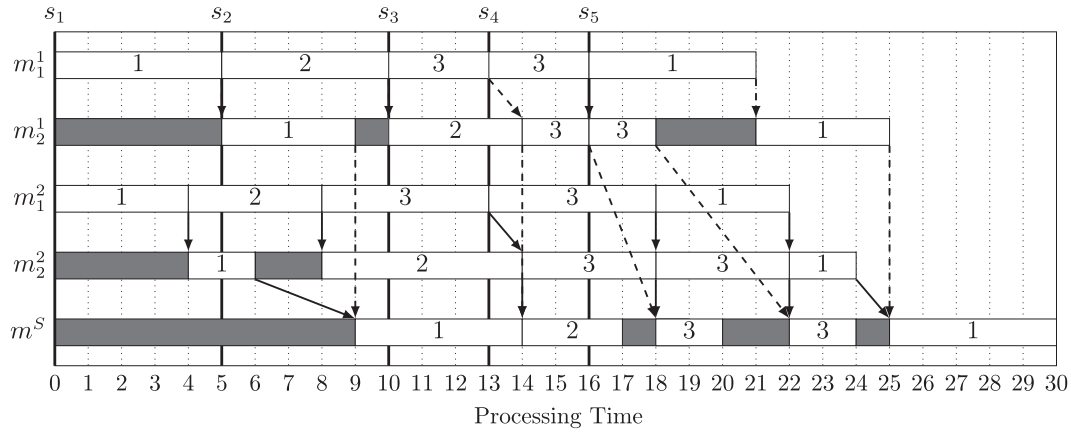


Fig. 1. Exemplary sequence diagram of PFSP with multiple lines.

While the sequence is generated, the number of valid actions decreases. Let $d_{i,t}$ denote the remaining quantity of job type i at sequence position t. For $t = 1$, we set $d_{i,1} = d_i$. We define an action a_t sequencing job type i at position t to be valid, if $d_{i,t} > 0$. If $d_{i,t} = 0$, a_t is invalid. Since the action space defines the dimension of the policy network, it cannot be changed during policy learning. Moreover, there is currently no straightforward way of preventing invalid actions in reinforcement learning (Zahavy, Haroush, Merlis, Mankowitz, & Mannor, 2018). We therefore pursue the strategy of ignoring invalid actions during the learning process. If the agent chooses an invalid action, it receives a zero reward $r_t = 0$ and the state is returned unchanged, i.e., $s_{t+1} = s_t$. Setting the discount factor γ to a value smaller than one, e.g., $\gamma = 0.99$, implicitly prevents situations in which the agent repeatedly performs invalid actions and consequently does not finish the learning task. However, invalid actions do not limit the practicability of approach as we can ensure that a sequence is valid by filtering invalid actions when applying the policy.

3.3. Reward

The reward function has a strong impact on the resulting policy (Sutton et al., 1998). For our problem, an intuitive approach would be to define the reward as the negative makespan of the generated sequence. However, the late reward signal requires the agent to discount the rewards over the whole sequence, which would lead to a less favorable learning process.

Therefore, we define a reward r_t for action a_t based on the negative idle time on the synchronization machine x_t^S. We assign a zero reward to invalid actions and a positive reward to valid actions. For this purpose, we need to shift the rewards of valid actions to a positive range by adding a value δ. In a pre-study, we observed that the 99th percentile of x_t^S equals approximately 197. The maximum value we encountered is equal to 615, which is caused by the idle time of the first job. We found that setting $\delta = 200$ yields the best results. The scaling factor $\lambda = 0.1$ was also determined in our pre-study. Corresponding details are provided in Appendix C of the supplementary material. Altogether, the reward r_t for action a_t sequencing job type i at position t is defined as

$$r_t = \begin{cases} 0.1 \cdot (-x_t^S + 200), & \text{if } d_{i,t} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

For example, consider the sequence diagram of Fig. 1. At s_3, job type $i = 3$ is sequenced and we obtain $x_3^S = 1$, since m^S is idle for one unit of processing time. According to (17), the agent receives $r_3 = 19.9$.

3.4. State representation

The state representation s_t contains the remaining job quantities $d_{i,t}$, the remaining processing times of all machines, and the reward per model with respect to the current partial sequence. To describe the state representation, we have to distinguish between the discrete time component given by the sequence position t and units of processing time on the machines. Recall that the agent receives state s_t before performing a_t, which denotes the job type to be sequenced at position t.

Based on this notion, we introduce two more variables as follows. First, $o_{k,t}^l$ and o_t^S denote the remaining processing times of m_k^l and m^S until the job at position $t-1$ is fully processed by m_k^l and m^S, respectively. To calculate the remaining processing times, we need to determine the elapsed units of production time. Here, we select the first machine of the first line m_1^1 as a reference. This means that we subtract the units of processing time after (m_1^1) has processed the job at position $t-1$. If a machine has already processed the job at position $t-1$, then $o_{k,t}^l$ equals the negative machine idle time. For $t = 1$, the remaining processing times are zero $o_{k,1}^l = 0,\ k = 1, \ldots, n_l;\ l = 1, \ldots, \nu,\ o_1^S = 0$. Second, $r_{i,t}$ denotes the reward that the agent would receive if job type i is sequenced at position t. Altogether, the state s_t is given by

$$s_t = (d_{1,t}, \ldots, d_{n,t}, o_{1,t}^1, o_{2,t}^1, \ldots, o_t^S, r_{1,t}, \ldots, r_{n,t}). \tag{18}$$

We provide an example of the state representation with respect to the sequence diagram shown in Fig. 1. At s_1, the agent sequenced job type $i = 1$ and subsequently receives $s_2 = (1, 1, 2, 0, 4, -1, 1, 9, 20, 20, 20)$. The quantity of job type $i = 1$ was reduced by one (initial demand plan $d = [2, 1, 2]$). The job at position $t = 1$ is fully processed by m_1^1 after five units of processing time and by m_2^1 after 9 units of processing time. Subtracting the reference time point (i.e., when m_1^1 has fully processed this job) yields $o_{1,2}^1 = 5 - 5 = 0$ and $o_{2,2}^1 = 9 - 5 = 4$. For m_1^2, we obtain a negative remaining processing time $o_{1,2}^2 = 4 - 5 = -1$, since m_1^2 has fully processed the job at position $t = 1$ after 4 units of processing time. Machine m_2^2 will finish processing the first job after 6 units of processing time, hence $o_{2,2}^2 = 6 - 5 = 1$. m^S will finish processing the first job after 14 units of processing time. Subtracting the reference time point yields $o_2^S = 14 - 5 = 9$. All three job types will not cause idle time at m_S if they are sequenced next. According to (17), $x_t^S = 0$ results in a reward of 20.

3.5. Policy learning

We employ proximal policy optimization (PPO) for policy learning as it presents a state-of-the-art policy gradient algorithm that

is easy to implement and tune (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017). In this approach, the policy $\pi_\theta(a_t|s_t)$ is learned by using a neural network, where the input is given by the state representation s_t and the output denotes the probability distribution over the actions a_t to perform in s_t. The weights of the network present the policy parameters θ. We opted for a policy gradient method instead of deep Q-learning (Mnih et al., 2015) due to the following reasons. First, policy gradient methods are guaranteed to converge to a local optimum. Second, policy gradient methods allow for the learning of a stochastic policy, i.e., a probability distribution over the set of actions. This implicitly handles the exploration vs. exploitation problem. Third, a stochastic policy is beneficial for our problem as it allows us to generate and evaluate multiple sequences.

We briefly describe the functionality of PPO according to Schulman et al. (2017). The pseudocode is provided in Algorithm 1. PPO alternates between generating trajectories

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

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

$(s_1, a_1, r_1), \ldots, (s_T, a_T, r_T)$ and updating θ. For each $t = 1, \ldots, T$, PPO calculates the discounted cumulative reward of s_t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where γ denotes the discount parameter (set to 0.99). Besides the probability distribution $\pi_\theta(a_t|s_t)$ over the actions a_t in state s_t, the PPO network also estimates $V^\theta(s_t)$ as the value of being in state s_t. Based on $V^\theta(s_t)$, PPO calculates the 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 greater probabilities to actions that lead to higher rewards than the current estimate $V^\theta(s_t)$, and vice versa. Finally, PPO updates the policy parameters θ to maximize the loss function $L(\theta)$.

$$\max_\theta L(\theta) = \sum_{t=1}^{T} \min\left\{ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t, clip\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon \right) \hat{A}_t \right\} - c_1 (R_t - V^\theta(s_t))^2 - c_2 \sum_{a_t \in A} \pi_\theta(a_t|s_t) \log_2 \pi_\theta(a_t|s_t) \quad (19)$$

The loss function $L(\theta)$ consists of three terms. The first term ensures that the policy parameters θ will be updated such that actions with positive advantage $\hat{A}_t$ are assigned higher probabilities, and vice versa. Here, PPO limits the extent of single 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 $[1-\varepsilon, 1+\varepsilon]$. We set ε to its default value of 0.20. The second term ensures that the update of θ also improves the estimate of $V^\theta(s_t)$. The third term ensures that the policy preserves randomness, so that the agent still explores random actions. The parameters c_1 and c_2 are set to their default values of 0.50 and 0.01 respectively. An overview of all PPO parameters is provided in Appendix A of the supplementary material.

4. Numerical evaluation setup

4.1. Dataset

Our numerical evaluation is based on the benchmark dataset by Taillard (1993). We extend this dataset to multiple production lines and demand plans as follows. The problem layout with dimensions $(J/M/n) = (100/10/10)$ denotes the *reference layout*. We evaluate seven variations from this layout, where we vary one dimension while keeping all others constant. For each layout, we generate five unique variations by sampling the processing times from a uniform distribution $[1, 99]$. For each variation, we sample ten demand plans from a multinomial distribution with equal probabilities for each job type (we later provide an analysis with more imbalanced distributions). This process is repeated for up to three production lines. For each line-setting, we keep the number of machines per line constant. An overview of all parameters is provided in Table 2. In total, our evaluation is based on 3 line-settings $\times$ 7 layouts $\times$ 5 processing time settings $\times$ 10 demand plans $=$ 1050 different problem instances.[2]

Table 2
Parameters of dataset.

Parameter	Values
Jobs in total J	20, 100, 500
Machines per line M	5, 10, 20
Number of job types n	5, 10, 20
Number of lines v	1, 2, 3

4.2. Cutoff time

All approaches are only given a limited cutoff time (20) to find a solution. The cutoff time (in milliseconds) depends on the sequence length J, the total number of machines, and the factor w (Ruiz & Stützle, 2008).

$$\text{cutoff time (ms)} = \begin{cases} J \cdot \frac{M \cdot v + 1}{2} \cdot w, & \text{if } v > 1, \\ J \cdot \frac{M}{2} \cdot w, & \text{if } v = 1. \end{cases} \quad (20)$$

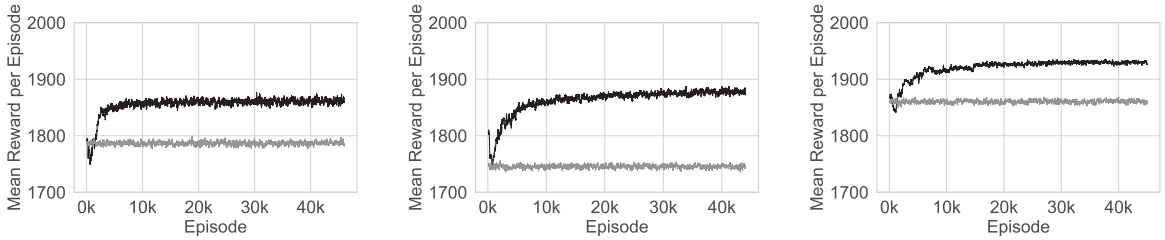
We carry out three experiments with short, medium, or long cutoff times according to $w \in \{30, 60, 90\}$. When the cutoff time has run out, the best current solution is considered the final solution of the respective approach.

4.3. Policy training

Our reinforcement learning approach is implemented in Python 3.6.8 using the framework OpenAIGym (Brockman et al., 2016). We train one policy for each problem layout with fixed processing times, which results in a total of 105 policies. During policy learning, we sample random demand plans from the aforementioned multinomial distribution. The total number of timesteps in policy learning is limited to five million (one timestep denotes sequencing one job). This corresponds to an average learning time of approximately 90 minutes per policy on our machine with an Intel(R) Xeon(R) CPU E5-2687W v2. We ignore the time for policy learning (target task time scenario) as it can be performed in advance for a fixed production system. The policy needs to be trained only once given the distribution of the demand plan. After the policy is learned, our approach can transfer knowledge from policy learning to similar but unseen problems (Taylor & Stone, 2009). We assume that adding or removing machines requires more time than retraining the policy (90 minutes), and that severe changes to the production do not occur without prior planning so that the policy can be retrained.

Fig. 2 shows the learning curve of our approach (in black) for the reference layout as the mean reward over the last 100 episodes. In each episode, the agent generates a sequence of length

[2] Our dataset is provided at https://data.mendeley.com/datasets/5txxwj2g6b/3 (Brammer, Lutz, & Neumann, 2021).



(a) One-line setting (b) Two-line setting (c) Three-line setting

Fig. 2. Learning curves of reinforcement learning approach.

100, so that the highest possible reward per episode is upper bounded by $(0+200)\times\ 0.1\times\ 100=2000$, see (17). The reward level differs between the line settings, since the processing times are different. We observe that the policies converge in all three line settings. As a comparison, we also plot the theoretic reward of Johnson (1954)'s method (and its application to multi-line problems) in gray.

Concerning invalid actions (i.e., when the agent attempts to sequence job type i with $d_{i,t}=0$), we observe that returning the same state and a zero reward makes the agent almost fully avoid invalid actions. A detailed analysis of the occurrence of invalid actions during the learning process is provided in Appendix B of the supplementary material. We find that the trained policy performs one invalid action out of 100 actions in total. Invalid actions are, however, not a limitation for real-world application, as we can filter invalid actions and select the best alternative valid action.

4.4. Sequence generation with trained policy during cutoff time

We evaluate two approaches denoted by *RL* and $RL+LS$ to generate sequences with the trained policy. The first approach *RL* uses the whole cutoff time to sample sequences and finally selects the sequence that minimizes $c_{\max}$. The trained policy $\pi^*(s_t)$ presents a probability distribution over the actions in state s_t. For each state s_t, we draw action $a_t=i$ from $\pi^*(s_t)$. If a_t is invalid ($d_{i,t}=0$), we select the best valid alternative. This ensures that all generated sequences are valid during real-world application.

Algorithm 2 Sequence generation with trained RL policy.

```
1: input: π* trained PPO policy, Env environment, d demand plan, cutoff time
2: J = Σ_{i=1}^{n} d_i, seq* = ∅
3: while cutoff time not run out do
4:     s_t = Env.reset()
5:     while |seq| < J do
6:         sample action a_t = i from π*(s_t)
7:         if d_{i,t} = 0, set a_t to best valid alternative from π*(s_t)
8:         s_t, seq = Env.step(a_t)
9:     optimize seq with local search using first improvement move (RL~+~LS only)
10:    if seq* = ∅ or c_max(seq) < c_max(seq*) then
11:        seq* = seq
12: output: seq*
```

The approach denoted by $RL+LS$ additionally optimizes the initial sequences from the policy using local search with first improve move (a swap is directly applied if it reduces $c_{\max}$) until no improvement can be found. Algorithm 2 describes both approaches, but line 9 is only executed for $RL+LS$.

4.5. Competing approaches

The competing approaches can be grouped into three categories. First, we employ the MILP solver Gurobi and ILOG CP Optimizer. Second, we implement several constructive heuristics using the multi-line to one-line simplification strategies proposed by Guimaraes et al. (2016). Third, we implement several iterative heuristics; namely, iterated greedy (IG), by Ruiz & Stützle (2007), GRASP by Guimarães et al. (2019), local search based on *NEH*, and ILOG CP Optimizer combined with local search.

4.5.1. Gurobi

We implement Gurobi 8.0.1 according to the MILP problem formulation from Section 2, which is based on Guimarães et al. (2019). If Gurobi cannot find an optimal solution within the cutoff time, its approximate solution is considered final. If Gurobi does not find any solution during cutoff time, we cannot compare it with other approaches. We then provide the number of instances with no solution for each line-setting.

4.5.2. Constraint programming

CP was shown to perform very well in scheduling problems (Meng, Zhang, Ren, Zhang, & Lv, 2020). We implement ILOG CP Optimizer 12.10 based on the problem formulation by Kizilay, Tasgetiren, Pan, & Gao (2019), which we extend to the multi-line problem with demand plans. To account for demand plans, we assign an individual index $j=1,\dots,J$ to each job. Therefore, we enumerate the jobs according to the demand plan. For instance, given the demand plan $d=[2,2,1]$, the two jobs of type $i=1$ are identified by $j=1,2$, while the job of type $i=3$ is identified by $j=5$. Note that the index j is generally not equal to the sequence position t, which is to be determined by the CP solver. Let $type(j)$ denote the type $i=1,\dots,n$ of job j. The decision variables $JobInt^l_{j,k}$ and $JobInt^S_j$ denote interval variables for job j on machines m^l_k and m^S with duration $p^l_{type(j),k}$ and $p^S_{type(j)}$. The CP model is given as

$$\min\left(\max_{j=1,\dots,J}(endOf(JobInt^S_j))\right) \tag{21}$$

$$endBeforeStart(JobInt^l_{j,k}, JobInt^l_{j,k+1}) \tag{22}$$

$$j=1,\dots,J;\ k=1,\dots,n_l-1;\ l=1,\dots,v$$

$$endBeforeStart(JobInt^l_{j,n_l}, JobInt^S_j) \qquad j=1,\dots,J;\ l=1,\dots,v \tag{23}$$

$$noOverlap(m^l_k) \qquad k=1,\dots,n_l+1;\ l=1,\dots,v \tag{24}$$

$$sameSequence(m^1_1, m^l_k) \qquad k=1,\dots,n_l+1;\ l=1,\dots,v. \tag{25}$$

The objective is to minimize the total makespan of the sequence, which corresponds to minimizing the maximal end of an interval $JobInt^S_j$ (21). Eq. (22) ensures that each job ends before the next one starts. Eq. (23) needs to be added for the multi-line problem as it ensures that the processing of jobs must be completed on all

lines before being processed on m^S. Finally, Eq. (24) ensures that no overlap of jobs occurs and Eq. (25) ensures that jobs are processed in the same sequence by all machines.

4.5.3. Constructive heuristics

We also consider the constructive heuristics Johnson's method and NEH based on the multi-line to single-line simplification strategies proposed by Guimaraes et al. (2016). However, due to spatial constraints, we only provide the results of the best performing simplification strategy for each constructive heuristic. Johnson's method finds the optimal solution for PFSP with two machines. Guimaraes et al. (2016) simplify the multi-line problem to a virtual two-machine flow shop problem as follows. For each job type, the processing time on the first virtual machine is set to the average (JOH_{avg}) or maximum ($JOH_{\max}$) processing time on all machines excluding the synchronization machine. The processing time of the second virtual machine is set to the processing time of the synchronization machine.

The idea of the NEH heuristic (Nawaz et al., 1983) is to insert jobs iteratively after sorting them in descending order based on their total processing time. Guimaraes et al. (2016) apply *NEH* to the multi-line problem by aggregating the processing times of each job type as the average (NEH_{avg}) or the maximum ($NEH_{\max}$) processing times of all parallel machines. Guimaraes et al. (2016) also apply *NEH* to each line separately (NEH_{sep}), from which the sequence that minimizes $c_{\max}$ is selected. We implement the datastructure proposed by Taillard (1990) to reduce the computational effort of *NEH* from $O(J^3M)$ to $O(J^2M)$.

4.5.4. Iterative heuristics

We implement several heuristics that iteratively optimize one or more initial sequences. The IG method proposed by Ruiz & Stützle (2007) creates an initial sequence based on *NEH*. Subsequently, it iterates over the phases "destruction" and "construction". In the destruction phase, *IG* randomly removes several jobs from the sequence. In the construction phase, *IG* uses *NEH* to reinsert the removed jobs to minimize $c_{\max}$. A local search using first improve move finally optimizes the sequence from the construction phase. A new sequence is accepted if it reduces $c_{\max}$. Otherwise, the new sequence may still be accepted with a certain probability. For the multi-line problems, *IG* is implemented with the best *NEH* simplification strategy.

The GRASP of Guimarães et al. (2019) was specifically developed for the multi-line PFSP. In each iteration, the metaheuristic generates a random sequence, which is then optimized by local search with best improvement move (only the best swap of a neighborhood is applied). We additionally implement GRASP with local search using first improvement move. The approaches are denoted by $GRASP_{best}$ and $GRASP_{first}$.

Lastly, we implement two baseline approaches to assess the influence of local search with first improvement move on the solution quality. $NEH + LS$ denotes local search based on *NEH* for the one-line problems, and the best *NEH* simplification strategy for the multi-line problems. $CP + LS$ denotes constraint programming combined with first improvement local search, where the cutoff time is split evenly over CP and LS.

4.6. Performance metric

To measure the performance of the presented approaches, we calculate the *Average Relative Percentage Deviation* (ARPD) from the best solution as suggested by Fernandez-Viagas et al. (2017). To define ARPD, we first need to define the relative percentage deviation (RPD). Let $c^{i,A}_{\max}$ denote the makespan of approach A on problem instance i, and $c^{i,A^*}_{\max}$ denote the makespan of the best relative solution on instance i. $RPD^{i,A}$ is defined as

$$RPD^{i,A} = \frac{c^{i,A}_{\max} - c^{i,A^*}_{\max}}{c^{i,A^*}_{\max}} \cdot 100. \tag{26}$$

Let $\mathcal{I}$ denote the set of all considered problem instances. The ARPD of approach A is defined as

$$ARPD^A = \frac{\sum_{i \in \mathcal{I}} RPD^{i,A}}{|\mathcal{I}|}. \tag{27}$$

If $ARPD^A = 0.20$, the sequences generated by A have on average 0.20% longer makespan than the best respective sequence. Note that the best relative solution is not necessarily an optimal solution.

For all approaches that involve randomness, ARPD is calculated based on the mean makespans over five evaluations with different seeds.

5. Results

We first evaluate all approaches based on short, medium, and long cutoff times (Ruiz & Stützle, 2008). Second, we evaluate all approaches against the (near-)optimal solutions found by Gurobi with an extended cutoff time of ten hours. Third, we assess the performance of all approaches on problems, where the demand plan follows a more imbalanced distribution.

5.1. Short cutoff time

Table 3 presents the results with short cutoff time ($w = 30$). The columns present the ARPD (27) of each approach for the respective problem layout. Each layout is denoted by sequence length/machines per line/number of job types. The specific variation from the reference layout (100/10/10) is underlined. For each layout, ARPD is calculated over a total of 50 instances. In the column "All", ARPD is calculated over all 350 instances from the seven layouts of the particular line setting.

Table 3 shows that $RL + LS$ (RL sequence generation combined with local search) outperforms all competing approaches in the multi-line settings. $RL + LS$ consistently achieves ARPD values below one, except from the 500/10/10 problems. Using only the trained RL policies to sample sequences (*RL*) leads to greater ARPD values, which indicates that it is beneficial to include the local search step. However, the solution quality also depends on the initial sequence. $RL + LS$ is superior to random sequence generation combined with best and first improvement local search ($GRASP_{best}$ and $GRASP_{first}$), as well as, the best NEH variation combined with local search ($NEH + LS$). Splitting the cutoff time evenly over constraint programming and local search ($CP + LS$) presents the best competing approach in the multi-line settings. $CP + LS$ performs, in particular, better than applying *CP* alone. The constructive heuristics (*JOH* and *NEH*) generally perform poorly, while *NEH* manages to show comparably better performance in the one-line setting. *IG* is the best approach in the one-line setting, but it also performs well in the multi-line settings. Gurobi performs very well if it finds a solution, but it does not find a solution for 221 out of 1050 instances.

5.2. Medium cutoff time

Table 4 presents the results with medium cutoff time ($w = 60$). Now, $RL + LS$ performs about equal to $CP + LS$ in the multi-line settings, while

IG still dominates all approaches in the one-line setting. Again, it is beneficial to include the local search step instead of using the whole cutoff time to sample sequences from the trained RL policy. $NEH + LS$ could reduce its ARPD value, but to a lesser extent

Table 3
Evaluation results (ARPD) with short cutoff time ($w = 30$).

	Reference	Jobs in total		Stations		Job types		All
	100/10/10	20/10/10	500/10/10	100/5/10	100/20/10	100/10/5	100/10/20	*
				One-line setting				
Cutoff [s]	15.0	3.0	75.0	7.5	30.0	15.0	15.0	*
Gurobi	**0.07**	**0.11**	–	**0.00**	–	**0.02**	–	–
CP	1.85	0.47	1.21	0.29	3.72	0.48	2.06	1.44
CP + LS	0.88	0.58	0.51	0.17	0.97	0.27	1.07	0.64
JOH	29.97	24.40	29.58	21.03	30.50	30.73	28.76	27.85
NEH	2.40	2.69	1.50	1.11	3.11	2.49	1.64	2.13
NEH + LS	0.71	1.65	0.83	0.26	0.67	0.63	0.53	0.75
IG	0.12	0.20	**0.00**	0.07	**0.06**	0.40	**0.05**	**0.13**
$GRASP_{best}$	5.97	1.33	2.06	2.43	7.66	2.61	6.54	4.09
$GRASP_{first}$	1.31	0.94	0.92	0.61	1.21	0.69	1.48	1.02
RL	0.48	0.58	1.59	0.12	1.07	0.62	1.50	0.85
RL + LS	0.25	0.32	0.87	0.03	0.40	0.45	0.58	0.41
				Two-line setting				
Cutoff [s]	31.5	6.3	157.5	16.5	61.5	31.5	31.5	*
Gurobi	–	1.70	**0.00**	**0.00**	–	**0.01**	–	–
CP	2.12	0.44	1.47	0.06	2.95	0.95	2.63	1.52
CP + LS	0.74	0.58	0.63	0.05	0.68	0.56	0.85	0.58
$JOH_{\max}$	27.19	20.65	27.66	11.47	25.31	25.24	23.90	23.06
$NEH_{\max}$	11.33	7.53	12.54	6.64	9.21	8.97	8.19	9.20
$NEH_{\max} + LS$	1.18	1.80	7.31	0.38	0.69	1.26	0.70	1.90
$IG_{\max}$	0.87	0.30	8.84	0.18	0.92	0.60	0.58	1.76
$GRASP_{best}$	4.98	1.07	1.73	1.43	4.89	2.53	5.85	3.21
$GRASP_{first}$	1.08	0.77	0.81	0.41	0.89	0.81	1.07	0.83
RL	0.88	0.89	3.57	0.34	1.67	0.79	2.75	1.56
RL + LS	**0.19**	**0.16**	1.97	0.12	**0.08**	0.44	**0.42**	**0.48**
				Three-line setting				
Cutoff [s]	46.5	9.3	232.5	24.0	91.5	46.5	46.5	*
Gurobi	–	1.43	–	**0.01**	–	0.14	–	–
CP	1.76	0.56	0.87	0.36	4.02	0.79	2.19	1.51
CP + LS	0.57	0.80	**0.20**	0.15	0.90	0.16	0.48	0.46
JOH_{avg}	17.37	13.04	18.07	13.52	21.64	15.04	20.36	17.01
NEH_{avg}	6.92	8.08	5.75	8.41	10.23	7.83	7.67	7.84
$NEH_{avg} + LS$	0.73	1.72	2.52	0.51	0.59	0.31	**0.45**	0.98
IG_{avg}	0.44	**0.22**	3.76	0.30	0.89	0.25	0.48	0.91
$GRASP_{best}$	3.60	0.87	1.12	1.80	5.27	1.73	4.86	2.75
$GRASP_{first}$	0.74	0.56	0.32	0.47	0.68	0.24	0.71	0.53
RL	0.84	0.91	1.48	0.53	2.20	0.57	2.66	1.31
RL + LS	**0.22**	0.24	1.07	0.11	**0.13**	**0.06**	0.50	**0.33**

Instances with no Gurobi solution: 28 in one-line, 68 in two-line, 125 in three-line setting.

than the multistart approaches $GRASP_{best}$ and $GRASP_{first}$. The performance of the constructive heuristics remains almost unchanged as they require little computational effort and can thus not benefit from longer cutoff times. However, both solvers benefit from more computational time. *CP + LS* now slightly outperforms *RL + LS* in the three-line setting and *Gurobi* reduced the number of instances with no solution to 116 out of 1050 instances.

5.3. Long cutoff time

Table 5 presents the results with long cutoff time ($w = 90$). Long cutoff times should favor the MILP and CP solvers as they become more likely to find optimal solutions. Again, *RL + LS* performs about equal to *CP + LS* in the multi-line settings and *IG* presents the best performing approach in the one-line settings. *RL + LS* performs best in four out of seven two-line layouts and comparably worse on the 500/10/10 problems, where *Gurobi* finds the best solutions. In the three-line settings, *RL + LS* performs best in the 100/20/10 problems and it achieves an ARPD below 0.10 for the 100/5/10 and 100/10/5 problems for which *Gurobi* finds the best solutions. However, even with the long cutoff time, *Gurobi* still does not find a solution for three one-line, 23 two-line, and 46 three-line problem instances. *CP* alone now performs slightly better than *CP + LS* in the one-line setting, but *CP + LS* remains superior in the multi-line settings.

Altogether, our analyses show that our approach *RL + LS* presents a better alternative to existing solutions in the multi-line problems given that the cutoff times are short. For medium and long cutoff times, our approach performs about equal in comparison to constraint programming combined with local search in the multi-line problems. Short cutoff times are beneficial for our approach as the trained RL policy quickly generates a good initial sequence, which can subsequently be optimized with local search. The quality of the initial solution matters as *NEH + LS*, $GRASP_{best}$, and $GRASP_{first}$ also apply local search, but only to the NEH solutions or random sequences. However, this yields longer makespans than applying local search to the initial sequences from the RL policy.

5.4. Comparison with (near-)optimal solution

In our previous analyses, the cutoff time was determined based on the problem complexity (Ruiz & Stützle, 2008). From a theoretical point of view, it seems interesting to analyze how close all

Table 4
Evaluation results (ARPD) with medium cutoff time ($w = 60$).

	Reference	Jobs in total		Stations		Job types		All
	100/10/10	20/10/10	500/10/10	100/5/10	100/20/10	100/10/5	100/10/20	*
One-line setting								
Cutoff [s]	30.0	6.0	150.0	15.0	60.0	30.0	30.0	*
Gurobi	**0.04**	**0.10**	–	**0.00**	–	**0.03**	–	–
CP	0.71	0.43	0.83	0.01	1.95	0.19	0.69	0.69
CP + LS	0.55	0.45	0.25	0.06	0.77	0.16	0.53	0.39
JOH	29.99	24.42	29.59	21.03	30.60	30.74	28.78	27.88
NEH	2.42	2.71	1.51	1.11	3.19	2.50	1.65	2.16
NEH + LS	0.64	1.66	0.65	0.24	0.62	0.63	0.51	0.71
IG	0.11	0.18	**0.01**	0.04	**0.09**	0.35	**0.04**	**0.12**
$GRASP_{best}$	4.47	1.03	2.04	1.28	6.00	1.54	5.00	3.05
$GRASP_{first}$	1.01	0.73	0.44	0.36	0.87	0.47	1.02	0.70
RL	0.46	0.55	1.56	0.11	1.11	0.63	1.45	0.84
RL + LS	0.19	0.29	0.64	0.02	0.35	0.41	0.46	0.34
Two-line setting								
Cutoff [s]	63.0	12.6	315.0	33.0	123.0	63.0	63.0	*
Gurobi	–	1.24	**0.00**	**0.00**	–	**0.00**	–	–
CP	1.32	0.37	1.24	**0.00**	2.16	0.36	1.76	1.03
CP + LS	0.59	0.35	0.42	**0.00**	0.59	0.25	0.61	0.40
JOH_{max}	27.37	20.72	27.72	11.47	25.59	25.35	24.48	23.24
NEH_{max}	11.49	7.59	12.60	6.64	9.46	9.06	8.70	9.36
$NEH_{max} + LS$	0.86	1.86	4.26	0.27	0.58	0.99	0.75	1.37
IG_{max}	0.64	0.21	6.97	0.14	0.69	0.53	0.66	1.40
$GRASP_{best}$	3.87	0.85	1.80	0.91	4.18	1.79	5.09	2.64
$GRASP_{first}$	0.83	0.53	0.63	0.29	0.60	0.72	0.89	0.64
RL	1.00	0.87	3.55	0.32	1.84	0.85	3.16	1.65
RL + LS	**0.18**	**0.13**	1.01	0.10	**0.14**	0.47	**0.56**	**0.37**
Three-line setting								
Cutoff [s]	93.0	18.6	465.0	48.0	183.0	93.0	93.0	*
Gurobi	–	1.12	–	**0.00**	–	0.09	–	–
CP	0.96	0.40	0.79	0.10	3.07	0.37	1.43	1.02
CP + LS	0.25	0.38	**0.16**	0.03	0.58	0.15	**0.28**	**0.26**
JOH_{avg}	17.49	13.11	18.26	13.52	21.95	15.09	20.85	17.18
NEH_{avg}	7.03	8.15	5.92	8.42	10.50	7.88	8.11	8.00
$NEH_{avg} + LS$	0.64	1.78	1.76	0.43	0.41	0.29	0.67	0.85
IG_{avg}	0.41	**0.14**	2.71	0.16	0.66	0.20	0.54	0.69
$GRASP_{best}$	2.49	0.55	1.34	1.09	4.00	0.96	3.86	2.04
$GRASP_{first}$	0.63	0.42	0.23	0.28	0.50	0.16	0.78	0.43
RL	0.91	0.93	1.58	0.53	2.40	0.59	3.00	1.42
RL + LS	**0.23**	0.21	0.75	0.09	**0.16**	**0.07**	0.59	0.30

Instances with no Gurobi solution: 6 in one-line, 38 in two-line, 72 in three-line setting.

approaches come to the (near-)optimal solutions found by Gurobi with an extended cutoff time of ten hours. Due to the large computational effort, we limit this analysis to the reference layout (100/10/10). All other approaches use problem specific cutoff times with $w = 90$.

The results are provided in Table 6. *Gurobi (10 hours)* denotes Gurobi with ten hours cutoff time and *Gurobi (normal)* denotes Gurobi with problem-specific cutoff time. Even after searching for ten hours, *Gurobi (10 hours)* did not always find an optimal solution. The number of non-optimal solutions is also provided in Table 6. Interestingly, the ARPD of *Gurobi (10 hours)* is not perfectly zero as it is outperformed by *IG* in one one-line instance, by *RL + LS* and *NEH + LS* in two two-line instances, and by *IG* in one three-line instance.[3] Compared to *Gurobi (10 hours)*, the solutions found by *RL + LS* deviate by up to 0.42 percent on average. *Gurobi* (normal), *CP*, *CP + LS*, *IG*, and $GRASP_{first}$ also achieve ARPD values below one.

[3] One-line instance: t11_100_10_10_2 (variation 2), two-line instances: t21_100_10_10_9 (variation 1) and t21_100_10_10_9 (variation 5), three-line instance: t31_100_10_10_0 (variation 5), see (Brammer et al., 2021).

5.5. Demand plans following more imbalanced distributions

Ultimately, we consider problems with more imbalanced demand plans. In our previous analyses, the demand plans followed a multinomial distribution with equal probabilities for each job type. We now consider two other multinomial distributions with linearly decreasing probabilities $p_i - p_{i+1} = const.$, or exponentially decreasing probabilities $\frac{p_i}{p_{i+1}} = 0.80$, for $i = 1, \ldots, n-1$ with $\sum_{i=1}^{n} p_i = 1$. While sampling the demand plans, we ensure that $d_i > 0$, $i = 1, \ldots, n$. Again, we limit our analysis to the reference layout with long cutoff times ($w = 90$).

Table 7 presents the results. *RL* denotes the RL approach as trained on the multinomial distribution with equal probabilities, while $RL_{lin/exp}$ denotes our approach as trained on the distributions with linearly or exponentially decreasing probabilities. Evidently, our approach without policy retraining (*RL + LS*) performs better than all competing approaches in the multi-line settings. In addition, Table 7 shows that our approach with the retrained policies ($RL_{lin/exp} + LS$) achieves an even smaller ARPD than *RL + LS*. This suggests that our RL approach is not limited to problems with balanced demand plans.

Table 5
Evaluation results (ARPD) with long cutoff time ($w = 90$).

	Reference	Jobs in total		Stations		Job types		All
	100/10/10	20/10/10	500/10/10	100/5/10	100/20/10	100/10/5	100/10/20	*
				One-line setting				
Cutoff [s]	45.0	9.0	225.0	22.5	90.0	45.0	45.0	*
Gurobi	**0.02**	**0.05**	0.05	**0.00**	–	**0.02**	–	–
CP	0.22	0.30	0.39	0.01	0.81	0.10	0.28	0.30
CP + LS	0.36	0.44	0.19	0.01	0.65	0.11	0.46	0.32
JOH	30.01	24.43	29.60	21.03	30.68	30.74	28.80	27.90
NEH	2.43	2.71	1.52	1.11	3.25	2.50	1.67	2.17
NEH + LS	0.64	1.67	0.56	0.24	0.64	0.63	0.52	0.70
IG	0.10	0.14	**0.01**	0.03	**0.09**	0.30	**0.05**	**0.10**
$GRASP_{best}$	2.78	0.77	2.04	0.83	4.39	1.04	3.53	2.20
$GRASP_{first}$	0.81	0.63	0.32	0.28	0.76	0.40	0.76	0.57
RL	0.47	0.51	1.54	0.11	1.15	0.62	1.43	0.83
RL + LS	0.17	0.28	0.56	0.02	0.31	0.37	0.38	0.30
				Two-line setting				
Cutoff [s]	94.5	18.9	472.5	49.5	184.5	94.5	94.5	*
Gurobi	0.51	1.19	**0.00**	**0.00**	–	**0.01**	–	–
CP	0.67	0.39	1.01	**0.00**	1.65	0.26	0.99	0.71
CP + LS	0.43	0.38	0.37	**0.00**	0.45	0.21	**0.55**	**0.34**
$JOH_{\max}$	27.43	20.77	27.74	11.47	25.75	25.38	24.70	23.32
$NEH_{\max}$	11.54	7.64	12.61	6.64	9.60	9.09	8.90	9.43
$NEH_{\max} + LS$	0.83	1.91	2.82	0.27	0.67	0.94	0.85	1.18
$IG_{\max}$	0.59	0.22	5.59	0.10	0.68	0.47	0.64	1.18
$GRASP_{best}$	2.46	0.66	1.77	0.44	2.75	1.02	3.48	1.80
$GRASP_{first}$	0.70	0.52	0.51	0.25	0.59	0.71	0.87	0.59
RL	1.02	0.88	3.51	0.31	1.94	0.85	3.28	1.69
RL + LS	**0.14**	**0.14**	0.82	0.09	**0.18**	0.47	0.56	**0.34**
				Three-line setting				
Cutoff [s]	139.5	27.9	697.5	72.0	274.5	139.5	139.5	*
Gurobi	–	1.04	–	**0.00**	–	**0.05**	–	–
CP	0.53	0.39	0.69	0.06	2.62	0.21	0.80	0.76
CP + LS	**0.14**	0.38	**0.10**	0.04	0.58	0.09	**0.26**	**0.23**
JOH_{avg}	17.52	13.16	18.35	13.52	22.13	15.11	20.99	17.25
NEH_{avg}	7.06	8.20	6.00	8.42	10.67	7.89	8.23	8.07
$NEH_{avg} + LS$	0.61	1.83	1.39	0.41	0.50	0.31	0.77	0.83
IG_{avg}	0.29	**0.12**	1.99	0.09	0.58	0.17	0.49	0.53
$GRASP_{best}$	2.10	0.57	1.38	0.91	3.47	0.59	3.36	1.77
$GRASP_{first}$	0.35	0.30	0.24	0.17	0.36	0.11	0.57	0.30
RL	0.94	0.95	1.63	0.52	2.52	0.60	3.06	1.46
RL + LS	0.21	0.20	0.55	0.08	**0.18**	0.07	0.52	0.26

Instances with no Gurobi solution: 3 in one-line, 23 in two-line, 46 in three-line setting.

Table 6
Evaluation results (ARPD) with respect to (near-)optimal solution.

	One-line setting	**Two-line setting**	**Three-line setting**
Cutoff [s]	36,000	36,000	36,000
Gurobi (10 hours)	**0.00**	**0.01**	**0.01**
Non-optimal solutions	2/50	9/50	3/50
	One-line setting	**Two-line setting**	**Three-line setting**
Cutoff [s]	45.0	94.5	139.5
Gurobi (normal)	**0.05**	0.79	–
CP	0.25	0.96	0.61
CP + LS	0.39	0.71	**0.22**
JOH	30.05	27.79	17.62
NEH	2.46	11.86	7.15
NEH + LS	0.67	1.11	0.69
IG	0.13	0.87	0.37
$GRASP_{best}$	2.84	2.78	2.20
$GRASP_{first}$	0.84	0.98	0.43
RL	0.50	1.31	1.02
RL + LS	0.20	**0.42**	0.29

Instances with no Gurobi (normal) solution: 2 in three-line setting.

Table 7
Evaluation results (ARPD) of other distributions in the demand plan.

Distribution	Linear			Exponential		
Setting	**One-line**	**Two-line**	**Three-line**	**One-line**	**Two-line**	**Three-line**
Cutoff [s]	45.0	94.5	139.5	45.0	94.5	139.5
Gurobi	0.36	–	–	0.30	–	–
CP	0.20	0.55	0.28	0.20	0.70	0.45
CP + LS	0.20	0.37	0.18	0.19	0.40	0.22
JOH	28.18	28.02	18.12	27.82	28.65	17.50
NEH	1.93	10.80	8.20	2.08	10.89	8.82
NEH + LS	0.42	0.68	0.48	0.50	0.74	0.37
IG	**0.05**	0.63	0.30	**0.10**	0.50	0.28
$GRASP_{best}$	1.97	2.80	1.62	2.04	2.89	1.33
$GRASP_{first}$	0.44	0.57	0.26	0.49	0.61	0.24
RL	1.51	2.88	1.01	1.31	2.99	1.13
RL + LS	0.34	0.33	0.12	0.31	0.23	0.13
$RL_{lin/exp}$	0.97	1.19	0.84	0.73	1.07	0.82
$RL_{lin/exp} + LS$	0.36	**0.27**	**0.12**	0.32	**0.15**	**0.11**

Instances with no Gurobi solution: linear distribution: 3 in two-line, 1 in three-line setting, exponential distribution: 3 in two-line, 3 in three-line setting.

6. Conclusion

We proposed a novel RL approach for the PFSP with multiple lines and demand plans. In contrast to existing RL-based scheduling methods, we train the policy to generate the sequence in an iterative way, where actions denote the job types to be sequenced next. Our numerical evaluation showed that our approach is superior to existing constructive and iterative heuristics, as well as, MILP and CP solvers for short cutoff times and tied with existing methods for medium and long cutoff times. We have also evaluated the performance of our approach against the (near-)optimal solutions found by Gurobi with an extended cutoff time of ten hours. Here, we have found that the sequences generated by our approach have up to 0.42% longer makespan on average than the (near-)optimal sequences. Finally, we have shown that the performance of our approach persists on problems with imbalanced demand plans following multinomial distribution with linearly or exponentially decreasing probabilities.

Our study offers several opportunities for future research. First, future research could study different optimization targets, such as the mean idle time per machine. If an immediate reward signal can be provided, this only requires small changes to the reward function. Second, it seems worth investigating whether the requirement that jobs need to be processed in the same order on all lines can be weakened, such that jobs only need to be processed in the same order on each individual line. This should reduce the total makespan at the cost of larger buffers at the end of each production line. Third, our approach could be applied to the general flow shop problem, where jobs are allowed to overtake each other. For this purpose, the action space had to be changed, such that action (i, m) corresponds to inserting job type i into machine m. Fourth, the proposed multistart approach that feeds initial solutions from RL to a metaheuristic like local search could be also extended to other scheduling problems like job-shop or mixel model sequencing.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2021.08.007

References

Agarwal, A., Colak, S., & Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research, 169*(3), 801–815.

Bautista, J., & Alfaro, R. (2018). Mixed integer linear programming models for flow shop scheduling with a demand plan of job types. *Central European Journal of Operations Research, 28*, 5–23.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. Available at https://arxiv.org/abs/1611.09940, last accessed April 16, 2021.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: A methodological tour d'Horizon. *European Journal of Operational Research, 290*(2), 405–421.

Brammer, J., Lutz, B., & Neumann, D. (2021). Permutation flow shop dataset. Mendeley Data, V3. https://data.mendeley.com/datasets/5txxwj2g6b/3.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., & Tang, J. et al. (2016). OpenAI Gym. Available at https://arxiv.org/abs/1606.01540, last accessed: February 13, 2021.

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. M. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research, 257*(3), 707–721.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research, 1*(2), 117–129.

Guimaraes, I. F., Ouazene, Y., de Souza, M. C., & Yalaoui, F. (2016). Semi-parallel flow shop with a final synchronization operation scheduling problem. *IFAC-PapersOnLine, 49*(12), 1032–1037.

Guimarães, I. F., Ouazene, Y., de Souza, M. C., & Yalaoui, F. (2019). Flowshop scheduling problem with parallel semi-lines and final synchronization operation. *Computers and Operations Research, 108*, 121–133.

Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly, 1*(1), 61–68.

Kizilay, D., Tasgetiren, M. F., Pan, Q.-K., & Gao, L. (2019). A variable block insertion heuristic for solving permutation flow shop scheduling problem with makespan criterion. *Algorithms, 12*(5), 100–130.

Komaki, G., Sheikh, S., & Malakooti, B. (2019). Flow shop scheduling problems with assembly operations: A review and new trends. *International Journal of Production Research, 57*(10), 2926–2955.

Meng, L., Zhang, C., Ren, Y., Zhang, B., & Lv, C. (2020). Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers and Industrial Engineering, 142*, 106347.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

Nawaz, M., Enscore, E. E., Jr., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91–95.

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–494.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*(3), 2033–2049.

Ruiz, R., & Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research, 187*(3), 1143–1159.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. Available at https://arxiv.org/abs/1707.06347, last accessed: February 13, 2021.

Stafford, E. F. (1988). On the development of a mixed-integer linear programming model for the flowshop sequencing problem. *Journal of the Operational Research Society, 39*(12), 1163–1174.

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*. Cambridge, MA: MIT Press.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research, 47*(1), 65–74.
Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278–285.
Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research, 10*(7), 1633–1685.
Tseng, F. T., Stafford, E. F., & Gupta, J. N. (2004). An empirical analysis of integer programming formulations for the permutation flowshop. *Omega, 32*(4), 285–293. http://www.sciencedirect.com/science/article/pii/S030504830300152X
Tseng, F. T., & Stafford, E. F., Jr. (2001). Two MILP models for the N× M SDST flowshop sequencing problem. *International Journal of Production Research, 39*(8), 1777–1809.
Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly, 6*(2), 131–140.
Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., & Mannor, S. (2018). Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in neural information processing systems* (pp. 3562–3573).
Zhang, Z., Wang, W., Zhong, S., & Hu, K. (2013). Flow shop scheduling with reinforcement learning. *Asia-Pacific Journal of Operational Research, 30*(05), 1350014.