



Intelligent scheduling and reconfiguration via deep reinforcement learning in smart manufacturing

Shengluo Yang & Zhigang Xu

To cite this article: Shengluo Yang & Zhigang Xu (2021): Intelligent scheduling and reconfiguration via deep reinforcement learning in smart manufacturing, International Journal of Production Research, DOI: [10.1080/00207543.2021.1943037](https://doi.org/10.1080/00207543.2021.1943037)

To link to this article: <https://doi.org/10.1080/00207543.2021.1943037>



Published online: 19 Jul 2021.



Submit your article to this journal [↗](#)



Article views: 89




View related articles [↗](#)



View Crossmark data [↗](#)



Intelligent scheduling and reconfiguration via deep reinforcement learning in smart manufacturing

Shengluo Yang ^{a,b,c} and Zhigang Xu^{a,b}

^aChinese Academy of Sciences, Shenyang Institute of Automation, Shenyang, People's Republic of China; ^bChinese Academy of Sciences, Institutes for Robotics and Intelligent Manufacturing, Shenyang, People's Republic of China; ^cUniversity of Chinese Academy of Sciences, Beijing, People's Republic of China

ABSTRACT

To realise the intelligent decision-making of dynamic scheduling and reconfiguration, we studied the intelligent scheduling and reconfiguration **with dynamic job arrival** for a reconfigurable flow line (RFL) using deep reinforcement learning (DRL), for the first time. The system architecture of intelligent scheduling and reconfiguration in smart manufacturing is proposed, and the mathematical model is established to **minimise total tardiness cost**. In addition, a DRL system of scheduling and reconfiguration is proposed by designing state features, actions, and rewards for scheduling and reconfiguration agents. Moreover, the advantage actor-critic (A2C) is adapted to solve the studied problem. The training curve shows the A2C-based agents have effectively learned to generate better solutions for unseen instances. The test results show that the A2C-based approach outperforms two traditional meta-heuristics, iterated greedy (IG) and genetic algorithm (GA), in solution quality and CPU times by a large margin. Specifically, the A2C-based approach outperforms IG and GA by 57.43% and 88.30%, using only 0.46‰ and 2.20‰ CPU times of IG and GA. **The trained model can generate a scheduling or reconfiguration decision within 1.47 ms, which is almost instantaneous and can satisfy real-time optimisation.** Our work shows a promising prospect of using DRL for intelligent scheduling and reconfiguration.

ARTICLE HISTORY

Received 6 January 2021
Accepted 7 June 2021

KEYWORDS

Deep reinforcement learning; dynamic scheduling and reconfiguration; A2C; reconfigurable manufacturing system (RMS); intelligent scheduling; dynamic job arrival



1. Introduction


Mass customisation, flexible demand, and agile manufacturing are raising challenges for traditional manufacturing systems during the last decade. The reconfigurable manufacturing system (RMS), which can rapidly **adjust its structure and production configurations** to meet new production requirements, is proposed to tackle those challenges (Bortolini, Galizia, and Mora 2018). The RMS has several characteristics, such as modularity, customisation, scalability, etc. Haddou Benderbal, Dahane, and Benyoucef (2017) studied the modularity assessment of an RMS. Campos Sabioni, Daaboul, and Le Duigou (2021) concurrently optimised the configuration of an RMS and its **modular production design**. Dou et al. (2020) studied the integrated configuration design and scheduling in an RMS. Usually, the production scheduling and workshop reconfiguration should be optimised collaboratively in an RMS. In recent years, new technologies, such as cloud manufacturing (Lin et al. 2019b), digital twin (Fang et al. 2019), and machine learning, can help realise collaborative control and real-time optimisation

of production systems. Thus, it is urgent and critical to study the real-time and collaborative optimisation of scheduling and reconfiguration for an RMS in smart manufacturing.

The production scheduling has been studied extensively, typically for the permutation flow shop scheduling problem (PFSP) (Fernandez-Viagas, Ruiz, and Framinan 2017). In PFSP, a set of jobs are processed on several machines sequentially, and the job sequence is maintained for all machines. Many meta-heuristics (Ta, Billaud, and Bouquard 2015; Pagnozzi and Stützle 2016; Schaller and Valente 2019; Fernandez-Viagas, Molina-Pariente, and Framinan 2020) have been proposed to **minimise the total tardiness of jobs for static orders**. In recent years, several studies are carried out using meta-heuristics for the dynamic scheduling problems considering new job arrival (Liu, Jin, and Price 2018; Rahman, Janardhanan, and Nielsen 2019).

In addition to the meta-heuristics, the DRL was also used to optimise production scheduling in recent years. Scheduling problems can be modelled as a Markov

CONTACT Zhigang Xu  zgxu@sia.cn  No. 135 Chuangxin Road, Hunnan District, Shenyang City, Liaoning Province, China

 Supplemental data for this article can be accessed here. <https://doi.org/10.1080/00207543.2021.1943037>

Decision Process (MDP), where an intelligent agent successively decides which job to be processed next (Luo 2020). Besides, a scheduling decision can be made instantly after the DRL agent is trained. Thus, the DRL can provide an alternative approach for scheduling problems. The DRL has been used to solve several scheduling problems, particularly the job shop scheduling problem (JSP). Waschneck et al. (2018) optimised the JSP using a deep Q network (DQN). Luo (2020) studied the dynamic JSP with new job insertions using a DQN. Lin et al. (2019b) studied the JSP using multi-class DQN under the edge computing framework. Zhang et al. (2020) used the DRL to learn priority dispatching rules for JSP. They exploited the disjunctive graph representation and proposed a Graph Neural Network based scheme for JSP. Liu, Chang, and Tseng (2020) studied the JSP using an actor-critic algorithm. Wang et al. (2020) solved the assembly JSP with uncertain assembly times using dual Q-learning. Except for JSP, other production scheduling problems are also studied using DRL. Zhang et al. (2013) studied the flow shop scheduling problem (FSP) using a TD(λ) algorithm. Wu et al. (2020) studied the dynamic dispatching of re-entrant production systems using a deep neural network (DNN). Shi et al. (2020) studied the scheduling of a transfer robot in an automated production line using DRL.

For the collaborative optimisation of scheduling and reconfiguration in an RMS, several studies have been carried out using meta-heuristics. Dou et al. (2021) studied the integrated configuration design and scheduling in an RMS using a multi-objective particle swarm optimization (PSO) to minimise total cost and tardiness. Barenji et al. (2016) studied the scheduling problems of the manufacturing flow lines using the Prometheus methodology. Ivanov et al. (2021) proposed a control approach to schedule jobs in an RMS considering dynamic structural-logical constraints. Besides, as one of the common types of RMS, an RFL is also studied. RFL consists of a set of workstations that contain one or more reconfigurable machines capable of processing a given set of tasks (Yelles-Chaouche et al. 2020). Dou, Li, and Su (2016b) studied the bi-objective optimisation of integrated scheduling and reconfiguration to minimise total cost and tardiness for an RFL using NSGA-II.

From the above literature review, we can know that production scheduling has been studied extensively by meta-heuristics and DRL. Besides, the collaborative optimisation of scheduling and reconfiguration are receiving increasing attention in recent years. However, no research efforts have used the DRL to solve the intelligent scheduling and reconfiguration for an RMS considering dynamic job arrival.

This paper studied the intelligent scheduling and reconfiguration for an RFL with dynamic job arrival using DRL in smart manufacturing. The aim of our study is to dynamically schedule jobs arriving stochastically and reconfigure the RFL at a proper time, based on real-time production status and order information. The objective is to minimise the total tardiness cost of all jobs. Our study can realise the real-time optimisation and intelligent decision-making of scheduling and reconfiguration for an RFL with dynamic job arrival. In particular, the contributions are as follows.

- (1) We studied the intelligent scheduling and reconfiguration for an RFL with dynamic job arrival using DRL for the first time.
- (2) We proposed a system architecture of intelligent scheduling and reconfiguration based on DRL in smart manufacturing.
- (3) We modelled the DRL system of scheduling and reconfiguration by designing state features, actions, and rewards for both scheduling and reconfiguration agents.
- (4) We proposed procedures of using DRL and meta-heuristics to solve the collaborative optimisation of scheduling and reconfiguration with dynamic job arrival.

The rest of this paper is organised as follows. Section 2 presents the system architecture of intelligent scheduling and reconfiguration and formulates the mathematical model. Section 3 models the DRL system of scheduling and reconfiguration. Section 4 and 5 present one DRL algorithm, A2C, and two meta-heuristics, IG and GA, to solve the studied problem. Section 6 trains the A2C and compares the trained agent with IG, GA, and single dispatching rules. Section 7 concludes the paper and gives suggestions for future work.

2. Problem description and formulation

We present an architecture of intelligent scheduling and reconfiguration based on DRL in smart manufacturing. Additionally, the mathematical model is established to minimise total tardiness cost.

2.1. System architecture

The architecture of intelligent scheduling and configuration in smart manufacturing is illustrated in Figure 1. Figure 1 shows the system contains the cloud centre, manufacturing execution system (MES), controller layer, and device layer. The workshop processes n jobs coming

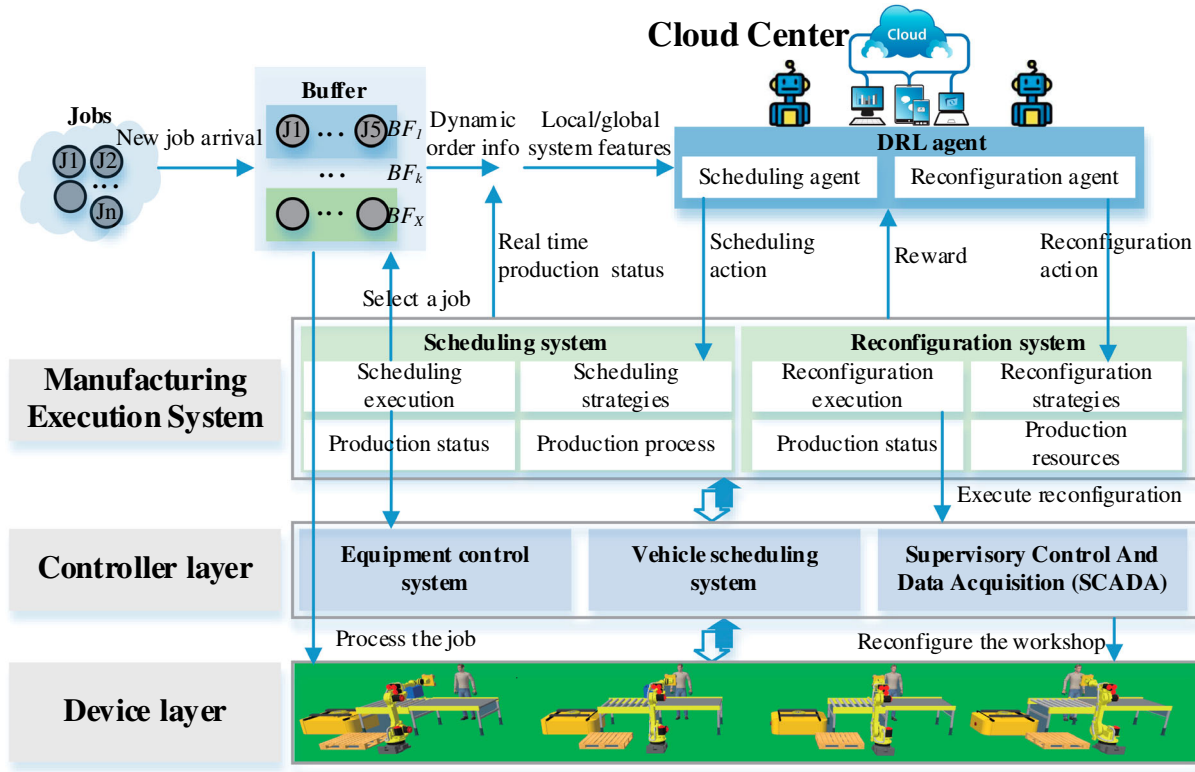


Figure 1. The architecture of intelligent scheduling and reconfiguration based on DRL in smart manufacturing.

to the system dynamically. The n jobs can be processed under X types of production modes. The scheduling agent in the cloud centre determines which job to be processed next for idle machines. The reconfiguration agent determines which production mode to be reconfigured next for the workshop at an appropriate time.

The production mode can be regarded as a certain state of the workshop. Each production mode corresponds to a specific workshop layout and configurations of production resources, such as a certain number of movable machines, vehicles, and workers. The jobs belonging to the same production mode have similar technological processes and can be processed in the same production line without any alteration. When the production mode changes, the workshop reconfigures through changing the workshop layout and reallocating the production resources. Each job coming to the system corresponds to one production mode k , $k = 1, 2, \dots, X$. The production mode of a job is determined by engineers in the design phase based on the technological process. Jobs are stored in their corresponding buffer BF_k , $k = 1, 2, \dots, X$, if the job cannot be processed immediately. We assume the production resources of a workshop can only fulfil the production requirements of one production mode at a time. This assumption is common for many actual workshops. For example,

a body shop can only produce one type of automobile simultaneously due to the limited number of welding robots.

The RFL is studied in this paper. Similar to the permutation flow shop, we simplify the problem by assuming the job sequence is maintained for all machines in the RFL. Therefore, only the first machine M_1 requires the scheduling or reconfiguration decision, and this requirement happens when M_1 finishes a job. In addition, like the usual assumptions in scheduling literature (Yang and Xu 2020), the processing for a machine cannot be interrupted during the operation, and infinite buffers exist between machines.

The whole system works as follows. When M_1 finishes a job, M_1 becomes idle, and the current scheduling execution status is sent to MES. The MES decides whether to reconfigure the workshop or not by a reconfiguration judgment mechanism. If the mechanism decides not to reconfigure, the idle machine calls the scheduling agent in the cloud centre to generate a scheduling action. The scheduling agent extracts current local system features, including real-time production status and dynamic order information, generates a scheduling action, and sends the action to the MES. The MES determines a scheduling strategy based on the action, uses the strategy to select a candidate job, and calls the controller layer to transport the job to the workshop from buffer. Then the

M_1 begins to process the job until finished in M_1 . The finished jobs will be delivered to the buffer of the next machine. A job becomes a finished job when completed on all machines.

If the reconfiguration judgment mechanism decides to reconfigure when a machine finishes a job, the reconfiguration agent generates a reconfiguration action based on current system features and order information and sends the reconfiguration action to MES. The MES selects a reconfiguration strategy based on the action, determines a new production mode to reconfigure, and calls the controller layer to execute the reconfiguration in the device layer. First, a 'complete WIP' operation is performed to make jobs that have already been processed and not completed, i.e. work-in-process (WIP), completed by all machines. After that, all machines become idle and are ready for reconfiguration. Then, the workshop reconfigures to the new production mode by reallocating production resources and changing the workshop layout with reconfiguration setup time required. The workshop can reconfigure from one production mode to any other production modes directly. During the transition of production modes, the constraints are not considered, and the reconfiguration setup time is assumed to be fixed. Actually, the setup time may vary between different production modes. However, compared with the processing time during the production of a production mode, the difference in setup time is negligible. New jobs are not allowed to enter the workshop during the period of 'complete WIP' and reconfiguration setup. Finally, the workshop is ready to accept and process new jobs under the new production mode. Idle machines will call the scheduling agent to determine a job to be processed next. After a scheduling or reconfiguration step, a reward is sent to update the corresponding agent.

2.2. Mathematical model

This section formulates the mathematical model of the studied problem. As mentioned above, a set of jobs arrive following a Poisson distribution, and each job corresponds to one of the X production modes. Only one production mode is processed at a time. The RFL selects and processes a job under the current production mode, and reconfigures to other modes at an appropriate time.

The notations used are listed below.

Indices:

j	index of a job, $1 \leq j \leq n$
i	index of a machine, $1 \leq i \leq m$
k	index of a production mode, $1 \leq k \leq X$

Parameters:

AT_j	arrival time of job j
t_{ij}	processing time of job j on machine i
d_j	due date of job j
α_j	unit (per second) tardiness cost of job j

Variables:

C_{ij}	completion time of job j on machine i
x_{jk}	1 if job j belongs to mode k ; 0 otherwise.
y_k	1 if mode k is the current production mode; 0 otherwise.

The objective is to minimise the total tardiness cost of all jobs. Based on the formulation of PFSP with total tardiness criteria (Wang et al. 2018) and other related works (Kazemi, Mazdeh, and Rostami 2017; Yang and Xu 2020), the mathematical model of our problem is formulated as follows:

$$\text{Minimize } \sum_{j=1}^n \alpha_j \times \max\{0, C_j - d_j\} \quad (1)$$

Subject to:

$$d_j = \left[CP \left(1 - TF - \frac{RDD}{2} \right), CP \left(1 - TF + \frac{RDD}{2} \right) \right], \quad j = 1, 2, \dots, n \quad (2)$$

$$CP = \frac{1}{m} \times \sum_{j=1}^n \sum_{i=1}^m t_{ij} \quad (3)$$

$$C_j = C_{ij}, \quad i = m, \quad \forall j = 1, 2, \dots, n \quad (4)$$

$$C_{ij} = \max\{C_{(i-1)j}, C_{i(j-1)}\} + t_{ij}, \quad i = 1, 2, \dots, m, \quad j = 1, 2, \dots, n \quad (5)$$

$$C_{ij} - t_{ij} \geq AT_j, \quad i = 1, \forall j = 1, 2, \dots, n \quad (6)$$

$$\sum_{k=1}^X x_{jk} = 1, \quad \forall j = 1, 2, \dots, n \quad (7)$$

$$\sum_{k=1}^X y_k = 1 \quad (8)$$

$$C_{0j} = 0, C_{i0} = 0 \quad (9)$$

The objective function (1) minimises the total tardiness cost of all jobs. Following (Kazemi, Mazdeh, and Rostami 2017; Yang and Xu 2020), the due date d_j is generated from a uniform distribution, as shown in constraint (2). In this constraint, the TF stands for tardiness factor, and RDD denotes the relative range of due dates. Both TF and RDD are constant. The CP, determined by constraint

(3), is an indicator of the total processing times of all jobs. Constraint (4) calculates the completion time of a job in the system. Constraint (5) determines the completion time of a job on a machine. Constraint (6) ensures a job can be processed only after it arrives. Constraint (7) guarantees that each job belongs to only one production mode. Constraint (8) ensures only a production mode is processed at a time. Constraint (9) provides the initial values of some variables.

3. System modelling

In this section, we modelled the DRL system of scheduling and reconfiguration by designing reward, actions, and state features for scheduling and reconfiguration agents. When an action is required for the two agents, the agent generates an action based on current state features. After the action is executed, a reward is returned to the agent to update its parameters.

We modelled the DRL system by designing the reconfiguration and scheduling agents independently. The reasons for independent design are as follows. First, it can reduce the dimensions of state features and action spaces and lower learning difficulty. In addition, if a unified agent is designed, the times for making decisions for scheduling and reconfiguration will be equal, and the agent needs to spend many epochs to learn to lower the frequency of reconfiguration. Moreover, the separate designation is scalable, which can incorporate other decision agents, such as a delivery agent, when necessary.

3.1. Reconfiguration agent – RCF

When the reconfiguration judgment unit decides to reconfigure, the reconfiguration agent RCF generates a reconfiguration action based on current state features, gets the reward of the last reconfiguration action, and learns from history transactions every T steps. The reward, actions, and state features for RCF are designed as follows.

3.1.1. Reward

Recall that the objective of the studied problem is to minimise the total tardiness cost of all jobs in the system. Every reconfiguration action should help to reduce the increase of total tardiness cost. In other words, for each reconfiguration action, the increase of total tardiness cost during this reconfiguration step should be minimised. The goal of a DRL agent is to maximise the cumulated reward of an episode. Thus, the reward of each reconfiguration step is defined as the inverse of newly added tardiness cost in the system per second during this step.

During a step, the tardiness cost is generated from buffer (BF) jobs, WIP, and finished jobs. Let FNS denotes finished jobs during the current reconfiguration step. Within a reconfiguration step, each completed job will be changed from WIP to FNS. The FNS is cleared when a current reconfiguration step ends. Denote t_S and $t_{S'}$ as the beginning and end times of the current reconfiguration step, respectively. Therefore, the reconfiguration reward of time step $[t_S, t_{S'}]$ is calculated by Equations (10) to (16).

$$R = -\frac{1}{t_{S'} - t_S} (TP_{BF} + TP_{WIP} + TP_{FNS}) \quad (10)$$

$$TP_{BF} = \sum_{k=1}^X \sum_{j=1}^{n_k} \alpha_j z_{jS'} [t_{S'} - \max(t_S, d_j)] \quad (11)$$

$$TP_{WIP} = \sum_{j=1}^{n_{WIP}} \alpha_j z_{jS'} [t_{S''} - \max(t_S, d_j)] \quad (12)$$

$$z_{jS'} = \begin{cases} 1, & d_j < t_{S'} \\ 0, & \text{else} \end{cases} \quad (13)$$

$$t_{S''} = \begin{cases} C_j, & \text{if job } j \text{ has been completed at } t_{S'} \\ t_{S'}, & \text{else} \end{cases} \quad (14)$$

$$TP_{FNS} = \sum_{j=1}^{n_{FNS}} \alpha_j z_{jC} [C_j - \max(t_S, d_j)] \quad (15)$$

$$z_{jC} = \begin{cases} 1, & d_j < C_j \\ 0, & \text{else} \end{cases} \quad (16)$$

where TP_{BF} , TP_{WIP} , and TP_{FNS} are the newly added tardiness cost of BF jobs, WIP, and FNS, respectively, during this step. The n_k , n_{WIP} , and n_{FNS} denote the number of jobs in BF_k, WIP, and FNS, respectively. The z_{jC} and $z_{jS'}$ indicate whether job j is overdue at its completion time C_j and time $t_{S'}$. The $t_{S''}$ is the real end time when calculating the tardiness cost for a WIP within the time step $[t_S, t_{S'}]$.

3.1.2. Reconfiguration judgment

Whenever the first machine M_1 finishes a job, the reconfiguration judgment is triggered to determine whether to reconfigure. Only when it decides to reconfigure, the RCF decides a reconfiguration action. The reconfiguration judgment mechanism reduces the frequency of reconfiguration manually. Otherwise, the RCF will spend many episodes to learn to lower the reconfiguration frequency to an extent.

Let k' , $k' = 1, 2, \dots, X$, denotes the current production mode, BF_{k'} denotes the buffer of current production mode k' , t_c denotes the current system time. The ψ_j denotes current unit tardiness cost of job j , where 'current' means at the current time t_c . ψ_j is calculated by

Equation (17).

$$\psi_j = \begin{cases} \alpha_j, & \text{if } t_c > d_j \\ 0, & \text{else} \end{cases} \quad (17)$$

The reconfiguration judgment unit decides to reconfigure at the following three conditions.

- (1) When $BF_{k'}$ is empty.
- (2) When no overdue jobs are left in $BF_{k'}$, and $\psi_{k'}$ is less than the P th percentile of Ψ .

Where $\psi_{k'}$ is the average of current unit tardiness cost of k' , Ψ is a set.

$$\psi_{k'} = \frac{1}{n_{k'}} \sum_{j=1}^{n_{k'}} \psi_j, \quad k' = 1, 2, \dots, X \quad (18)$$

$$\psi = \{\psi_1, \psi_2, \dots, \psi_X\} \quad (19)$$

- (3) When another n_{jud} jobs in $BF_{k'}$ have been processed, and $\psi_{k'}$ is less than the P percentile of ψ .

Where $n_{jud} = \max\{n_{prc}, \text{round}(n/(X * n_{rm}))\}$. This equation restricts that the workshop decides to reconfigure after at least n_{prc} new jobs are processed. The n_{rm} denotes the assumed reconfiguration times for each production mode. After tuning, the n_{prc} , n_{rm} , and P are set to 3, 3, and 70.

3.1.3. Actions

Since no rules perform well for all production status (Luo 2020), RCF should choose proper actions in different production status. The action space should offer candidate actions for all possible conditions. Moreover, action space should be efficient and lean.

Four actions (A_1 - A_4) are designed for RCF. Each action corresponds to a strategy for selecting a candidate production mode k , $k = 1, 2, \dots, X$. Under each strategy, only jobs in BF_k , $k = 1, 2, \dots, X$ are considered. If more than one production modes are best under a strategy, the first best mode will be selected. If the new production mode equals the current mode k' , the workshop will not reconfigure.

The four RCF actions (A_1 - A_4) are as follows.

- (1) A_1 : select the production mode k , which has the largest total ψ_j .

$$A_1 = \arg \max_k \left(\sum_{j=1}^{n_k} \psi_j \right), \quad k = 1, 2, \dots, X$$

Where ψ_j denotes the current unit tardiness cost of job j and is calculated by Equation (17).

This action is apparent because the mode k results in the maximum tardiness cost every second. This mode k should be processed first to reduce the increase of total tardiness cost.

- (2) A_2 : select the production mode k , which has the largest average ψ_j .

$$A_2 = \arg \max_k \left(\frac{1}{n_k} \sum_{j=1}^{n_k} \psi_j \right), \quad k = 1, 2, \dots, X$$

A_2 has advantages when a production mode k does not fulfil the maximum total ψ_j due to the small number of jobs, but the average ψ_j is large.

- (3) A_3 : select a production mode with the largest number of jobs in BF_k , $k = 1, 2, \dots, X$.

$$A_3 = \arg \max_k (n_k), \quad k = 1, 2, \dots, X$$

A_3 has advantages when all jobs are not overdue, or the number of jobs is so large that it becomes the dominant condition.

- (4) A_4 : select a production mode k , which has the minimum average safe time.

$$A_4 = \arg \min_k \left(\frac{1}{n_k} \sum_{j=1}^{n_k} ST_j \right), \quad k = 1, 2, \dots, X$$

Where ST_j is the safe time of job j and is calculated by Equation (20).

$$ST_j = d_j - \sum_{i=1}^m t_{ij} - t_c \quad (20)$$

3.1.4. State features

Recall that a DRL agent selects an action based on current state status. The state features should contain all necessary information for evaluating actions. The state features should be lean and directly related to the information useful for selecting actions to reduce noises.

Four state features (Ft_1, Ft_2, \dots, Ft_4), which are directly related to the action spaces, are designed for RCF. These features are as follows.

- (1) $Ft_1 = \{n_1, n_2, \dots, n_k, \dots, n_X\}$. The number of jobs in each BF_k , $k = 1, 2, \dots, X$.
- (2) $Ft_2 = \left\{ \sum_{j=1}^{n_1} \psi_j, \sum_{j=1}^{n_2} \psi_j, \dots, \sum_{j=1}^{n_k} \psi_j, \dots, \sum_{j=1}^{n_X} \psi_j \right\}$

Total current unit tardiness cost of jobs in each BF_k , $k = 1, 2, \dots, X$.

$$(3) \quad Ft_3 = \left\{ \frac{1}{n_1} \sum_{j=1}^{n_1} \psi_j, \frac{1}{n_2} \sum_{j=1}^{n_2} \psi_j, \dots, \frac{1}{n_k} \sum_{j=1}^{n_k} \psi_j, \dots, \frac{1}{n_X} \sum_{j=1}^{n_X} \psi_j \right\}$$

Average current unit tardiness cost of jobs in each BF_k , $k = 1, 2, \dots, X$.

$$(4) \quad Ft_4 = \left\{ \frac{1}{n_1} \sum_{j=1}^{n_1} ST_j, \frac{1}{n_2} \sum_{j=1}^{n_2} ST_j, \dots, \frac{1}{n_k} \sum_{j=1}^{n_k} ST_j, \dots, \frac{1}{n_X} \sum_{j=1}^{n_X} ST_j \right\}$$

Average safe time of jobs in each BF_k , $k = 1, 2, \dots, X$.

Note that each state feature is an array. For each state feature, four statistical characteristics, including the maximum, minimum, average, and standard deviation, are calculated to reflect its data characteristics. Thus, the total dimensions of state space for RCF is $4 \times 4 = 16$.

The min-max normalisation is used for state features. For state feature Ft_x , its rescaled state feature Ft'_x is calculated by Equation (21).

$$Ft'_x = \frac{Ft_x - \min(Ft_x)}{\max(Ft_x) - \min(Ft_x)} \quad (21)$$

Where Ft_x is the set of all Ft_x without rescaling in the first ten epochs.

3.2. Scheduling agent – SCD

The scheduling agent SCD generates an action to choose a candidate job within the current production mode k' , based on current state features. The rewards, state features, and actions for SCD are designed as follows.

3.2.1. Reward

Since the SCD selects a job from current production mode k' to process, each action of SCD should make the total tardiness cost of jobs belonging to k' increase as little as possible. The SCD should minimise the unit tardiness cost of the current mode k' during each step. The tardiness cost of mode k' is resulted from jobs in $BF_{k'}$ and WIP. Thus, the reward of SCD during the time step $[t_s, t_{s'}]$ is calculated as follows.

$$r = -\frac{1}{t_{s'} - t_s} (tp_{BF_{k'}} + tp_{WIP}) \quad (22)$$

$$tp_{BF_{k'}} = \sum_{j=1}^{n_{k'}} z_{js'} \alpha_j [t_{s'} - \max(t_s, d_j)] \quad (23)$$

$$tp_{WIP} = \sum_{j=1}^{n_{WIP}} z_{js'} \alpha_j [t_{s'} - \max(t_s, d_j)] \quad (24)$$

$$z_{js'} = \begin{cases} 1, & d_j < t_{s'} \\ 0, & \text{else} \end{cases} \quad (25)$$

$$t_{s'} = \begin{cases} C_j, & \text{if job } j \text{ has been completed at } t_{s'} \\ t_{s'}, & \text{else} \end{cases} \quad (26)$$

where the $tp_{BF_{k'}}$ and tp_{WIP} are the newly added tardiness cost during this step from jobs in $BF_{k'}$ and WIP. The $n_{k'}$ and n_{WIP} denote the number of jobs in $BF_{k'}$ and WIP. The $z_{js'}$ indicate whether job j is overdue at time $t_{s'}$. The $t_{s'}$ is the real end time when calculating the tardiness cost for a WIP within the time step $[t_s, t_{s'}]$.

3.2.2. Actions

Eight actions (a_1 - a_8) are proposed for SCD to choose a job to be processed next from the current buffer $BF_{k'}$. The eight actions are as follows.

- (1) $a_1 = \arg \max_j (\psi_j), \quad j \in BF_{k'}$
- (2) $a_2 = \arg \max_j (\alpha_j), \quad j \in BF_{k'}$
- (3) $a_3 = \arg \min_j (ST_j), \quad j \in BF_{k'}$
- (4) $a_4 = \arg \min_j (d_j), \quad j \in BF_{k'}$
- (5) $a_5 = \arg \min_j (AT_j), \quad j \in BF_{k'}$

The a_5 corresponds to the well-known first-in-first-out (FIFO) strategy.

- (6) $a_6 = \arg \min_j (\sum_{i=1}^m t_{ij}), \quad j \in BF_{k'}$

The a_6 selects a job with the shortest processing time (SPT).

- (7) $a_7 = \arg \max_j (\sum_{i=1}^m t_{ij}), \quad j \in BF_{k'}$

The a_7 selects a job with the longest processing time (LPT).

- (8) $a_8 = \arg \max_j (u'_j), \quad j \in BF_{k'}$

Where u'_j is the estimated utilisation rate of job j and is calculated as follows.

$$u'_j = 1 - \left(\sum_{i=1}^m WT'_{ij} / \sum_{i=1}^m t_{ij} \right), \quad j \in BF_{k'} \quad (27)$$

The WT'_{ij} is the estimated wait time of machine i when waiting for job j . Each job j in $BF_{k'}$ is assumed to be processed under current machine status to obtain the WT'_{ij} .

3.2.3. State features

As mentioned above, the state features should be directly related to the action spaces. Based on the action spaces, eight state features for SCD – ft_1, ft_2, \dots, ft_8 – are designed as follows.

- (1) $ft_1 = n_{k'}$. The number of jobs in $BF_{k'}$.
- (2) $ft_2 = \{\psi_j\}, j = 1, 2, \dots, n_{k'}$. Current unit tardiness cost of jobs in $BF_{k'}$.
- (3) $ft_3 = \{\alpha_j\}, j = 1, 2, \dots, n_{k'}$. Unit tardiness cost α_j of jobs in $BF_{k'}$.

- (4) $ft_4 = \{ST_j\}, j = 1, 2, \dots, n_{k'}$. Safe time ST_j of jobs in $BF_{k'}$.
- (5) $ft_5 = \{d_j\}, j = 1, 2, \dots, n_{k'}$. Due date d_j of jobs in $BF_{k'}$.
- (6) $ft_6 = \{AT_j\}, j = 1, 2, \dots, n_{k'}$. Arrival time AT_j of jobs in $BF_{k'}$.
- (7) $ft_7 = \{\sum_{i=1}^m t_{ij}\}, j = 1, 2, \dots, n_{k'}$. Total processing time of jobs in $BF_{k'}$.
- (8) $ft_8 = \{u'_j\}, j = 1, 2, \dots, n_{k'}$. Estimated utilisation rate u'_j of jobs in $BF_{k'}$.

Since ft_2, \dots, ft_8 are arrays, four statistical characteristics, including the maximum, minimum, average, and standard deviation, are calculated for them. These seven features result in $7 \times 4 = 28$ dimensions. Besides, another two characteristics are calculated for ft_2 , i.e. the number of zero values and non-zero values. Thus, for SCD, the total dimensions of state space are $1 + 7 \times 4 + 2 = 31$. Similar to RCF, the min-max normalisation is also used for state features of SCD.

4. A2C

In this section, we provide the procedure of solving dynamic scheduling and reconfiguration using DRL and adapt the A2C to solve the studied problem. As shown in Algorithm 1, the SCD generates a scheduling action a to choose a new job under the current production mode, when a job is finished in M_1 . The new job, denoted as job j , will be processed on all machines to obtain its

completion time on all machines, i.e. $[C_{1j}, C_{2j}, \dots, C_{mj}]$. However, the system time t_c is only pushed forward to the C_{1j} . Then, the WIP is updated by changing the jobs finished by all machines from WIP to FNS. The new job arrival operation, which adds the newly arrived jobs to buffer, will be performed when the current system time t_c changes.

The RCF generates a reconfiguration action A to choose a new production mode, when the reconfiguration judgment unit decides to reconfigure. Before reconfiguring, the WIPs under the old production mode are processed until finished on all machines. The system time t_c is pushed forward to the maximum completion time of WIPs, and all WIPs become finished jobs, FNS. After all WIPs are finished, all machines become idle. Then, the workshop reconfigures to the new production mode by moving and regrouping machines and other production resources using a reconfiguration setup time, t_{setup} . Finally, the workshop under the new production mode is ready to perform a new round of production.

The transitions of SCD and RCF are stored separately. The RCF updates every T steps, and the SCD updates when a reconfiguration occurs.

A2C is adapted to solve the studied problem. Equipped with two networks, A2C maintains a policy $\pi_\theta(a_t|s_t)$ using an actor network and estimates a value function $V_\phi(s_t)$ using a critic network (Mnih et al. 2016). The procedure of A2C is shown in Algorithm 2. In algorithm 2, the $H(\pi_\theta(s_j; \theta))$ is the entropy of $\pi_\theta(s_j; \theta)$, which improves exploration by discouraging premature convergence. The entropy $H(\pi_\theta(s_t))$ is calculated by

Algorithm 1 procedure of solving dynamic scheduling and reconfiguration with new job arrival using DRL

```

1: Initialize actor and critic network of SCD  $\pi_\theta, V_\phi$ 
2: Initialize actor and critic network of RCF  $\pi_\vartheta, V_\varphi$ 
3: for episode = 1:  $EP$  do
4:    $t_c = 0$ ; Get state of SCD and RCF  $s, S$ 
5:   Generate a reconfiguration action  $A, A = \pi_\vartheta(S)$ 
6:   Choose a production mode  $k'$  based on  $A$ 
7:   while step  $\leq n$  do
8:     Generate a scheduling action  $a, a = \pi_\theta(s)$ 
9:     Choose a job  $j$  under mode  $k'$  based on  $a$ , process  $j$  in all machines, obtain  $[C_{1j}, C_{2j}, \dots, C_{mj}]$ , set  $j$  as WIP
10:     $t_c = C_{1j}$  // Only push forward  $t_c$  to the time when  $M_1$  finishes job  $j$ 
11:    Update WIP by removing jobs whose  $C_{mj} < t_c$ ; perform new job arrival at  $t_c$ 
12:    Obtain current state  $s'$ , reward  $r$ , and reconfigure judgment
13:    Store transition  $\{s, a, r, s'\}$ ; update current state  $s = s'$ 
14:    if reconfigure then:
15:      Get current state  $S'$ , return reward  $R$ ; store transition  $\{S, A, R, S'\}$ 
16:      Generate a reconfiguration action  $A, A = \pi_\vartheta(S)$ ,  $S = S'$ 
17:      Choose a production mode  $k$  based on  $A$ , complete WIP, set  $t_c = \max(C_{mj}), j \in WIP$ 
18:       $t_c = t_c + t_{setup}, k' = k$ ; perform new job arrival at  $t_c$ 
19:      Update SCD  $\pi_\theta, V_\phi$  using data in transitions
20:      Update RCF  $\pi_\vartheta, V_\varphi$  every  $T$  steps using data in transitions
21:      Get current state  $s$ 
22:    end if
23:  end while
24: end for

```

Algorithm 2 procedure of A2C

```

1: Initialize actor and critic network  $\pi_\theta, V_\phi$ 
2: for episode = 1:  $EP$  do
3:   while step  $\leq n$  do
4:     Run policy  $\pi_\theta$  for  $T$  steps, collecting  $\{s_t, a_t, r_t, s_{t+1}\}$ 
5:     Calculate discounted reward  $dr_t$  of every step  $t$   $dr_t = \begin{cases} r_t + \gamma V_\phi(s_{t+1}), & \text{if } t \text{ is the } T\text{th step} \\ r_t + \gamma dr_{t+1}, & \text{otherwise} \end{cases}$ 
6:     Update actor-network  $\pi_\theta$  using gradient  $\nabla_\theta \log \pi_\theta(a_t|s_t)(dr_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t; \theta))$ 
7:     Update critic-network  $V_\phi$  using gradient  $\nabla_\phi (dr_t - V_\phi(s_t))^2$ 
8:   end while
9: end for

```

Equation (28).

$$H(\pi_\theta(s_t)) = - \sum_{a_t \in A} \pi_\theta(a_t|s_t) \log \pi_\theta(a_t|s_t) \quad (28)$$

5. Meta-heuristics

In this section, we adapt two meta-heuristics, IG and GA, to solve the dynamic scheduling and reconfiguration with new job arrival. No meta-heuristics have been proposed in the current literature to solve the studied problem. Note that the several existing literatures (Yuan et al. 2016; Dou, Li, and Su 2016a; Dou et al. 2020) in solving scheduling and reconfiguration are multi-objective and did not consider the dynamic job arrival. These algorithms are not appropriate for our problem. According to the reviews of Fernandez-Viagas, Ruiz, and Framinan (2017), IG is the most efficient, and GA is one of the most widely used meta-heuristics in solving PFSP. Besides, IG and GA are also extensively used in solving other scheduling problems. Thus, we adapt IG and GA to solve the studied problem.

For statistic scheduling problems, the jobs in the solution sequence are processed successively. However, for the dynamic scheduling problem, the job sequence should be optimised again when a new job arrives.

Besides, the workshop should reconfigure when the current production mode changes. The procedure of using meta-heuristics to solve this problem is provided in Algorithm 3.

In Algorithm 3, the π_A denotes the arrival sequence of all jobs in a training/test instance sorted by the arrival time. The π_A contains two columns, job index j and arrival time of job j , AT_j . At the current system time t_c , The jobs in π_A satisfying $AT_j > t_c$ will be released from π_A to the buffer. The π_T denotes the current job sequence to be processed. The π_T is obtained through the initialisation procedure of meta-heuristics using jobs arriving at the system at time zero. The jobs in the job sequence π_T are processed successively.

In the current literature, the rescheduling is activated once a new job arrives (Liu, Jin, and Price 2017, 2018), or only activated several times, such as five times (Valledor et al. 2018), for all newly arrived jobs. If the rescheduling is only activated several times, the CPU time for computing will be reduced several times due to the lower optimisation frequency. However, the newly arrived jobs before the rescheduling point cannot be included and optimised in the current job sequence timely, leading to the objective value get worse, especially for the rush orders/jobs. In our experiment, the rescheduling is activated only at the decision points having new jobs arrived.

Algorithm 3 procedure of solving dynamic scheduling and reconfiguration with new job arrival using meta-heuristics

```

1: Get  $\pi_A; t_c = 0; \pi_T = \text{Initialization} // \pi_A$ : arrival sequence,  $\pi_T$ : current job sequence
2: Sample a production mode  $k'$  from  $\pi_T$  randomly
3: while  $\text{sizeof}(\pi_T) > 0$  do
4:   At  $t_c$ , check  $\pi_A$  to see if any new job already arrived but not released
5:   if (new job arrived) then: // rescheduling
6:     Release the newly arrived jobs in  $\pi_A$  to the buffer, insert these jobs to  $\pi_T$ 
7:     Optimize  $\pi_T$  for several iterations using meta-heuristics, return the best job sequence  $\pi_T'$ , set  $\pi_T = \pi_T'$  // offline, the  $t_c$  does not increase during the optimization
8:   end if
9:   Remove the first job  $j$  in  $\pi_T$ , get its production mode  $k$ 
10:  if  $k \neq k'$  then // Reconfigure
11:    Complete WIP, set  $t_c = \max(C_{mj}), j \in WIP$ 
12:     $t_c = t_c + t_{\text{setup}}, k' = k$ 
13:  end if
14:  Process job  $j$  until finished, obtain  $[C_{1j}, C_{2j}, \dots, C_{mj}]$ , set  $j$  as WIP;  $t_c = C_{1j}$ 
15:  Update WIP by removing jobs whose  $C_{mj} < t_c$ 
16: end while

```

In this way, the rescheduling times are less than the number of newly arrived jobs, and at each decision point, the current job sequence is optimal with all jobs in the current system considered and rescheduled. When choosing the next job to be processed in M_1 , the system checks the arrival sequence π_A to see whether any new job already arrived but not released, i.e. check if any job j in π_A satisfies $AT_j < t_c$. If any new job is arrived and not released, the rescheduling is activated. Each newly arrived job will be inserted into the current job sequence π_T . Then, the π_T will be optimised by meta-heuristics for several iterations to find an optimal solution. When the production mode changes, the workshop will reconfigure to the new production mode before processing the selected job.

The optimisation time of meta-heuristics and the decision time of DRL algorithms are not added to the system time t_c . The decision time of trained DRL agents is negligible. However, the optimisation time of meta-heuristics is relatively large. If the two kinds of times are added to the system time t_c , the optimisation process of meta-heuristics will change the system state and increase the objective value. This cannot provide a fair comparison for objective value between the DRL and meta-heuristics. In addition, since the optimisation time varies between different computers, the objective value will also vary if the optimisation time is added.

5.1. IG

In this section, we adapt IG to solve the studied problem. A two-level solution representation for solving scheduling and reconfiguration is proposed. Also, the

key operations of IG, including initialisation, destruction, reconstruction, and local search, are illustrated.

IG begins with an initial solution and performs destruction, reconstruction, and local search on the solution sequence for several iterations to get an optimal or near-optimal solution. IG has been widely used in solving scheduling problems, such as PFSP (Ruiz and Stutzle 2007; Karabulut 2016; Pan, Ruiz, and Alfaro-Fernández 2017; Fernandez-Viagas, Valente, and Framinan 2018), distributed permutation flowshop scheduling problem (DPFSP) (Ribas, Companys, and Tort-Martorell 2019; Ruiz, Pan, and Naderi 2019), distributed assembly permutation flowshop scheduling problem (DAPFSP) (Hatami, Ruiz, and Andrés-Romano 2015; Pan et al. 2019; Yang and Xu 2020), etc.

A two-level solution representation, which contains job and production mode information, is designed, as shown in Figure 2. Based on the solution representation, the initial sequence is generated using initial jobs arriving at time zero. First, the production mode sequence is generated randomly. Then, the initial sequence is obtained by putting jobs together, following the production mode sequence. As shown in Figure 2(a), the production sequence is $\{1, 0, 2\}$, and jobs are grouped according to the production sequence. The initial sequence is used as the current job sequence π_T .

The jobs in π_T are processed successively. When inserting the newly arrived jobs to π_T , each new job will be placed at the last position of the job's production mode in π_T . For example, in Figure 2(a), if the newly arrived job is j_{13} and its production mode is 0, the sequence $[13, 0]$ will be placed behind $[12, 0]$. As illustrated in Algorithm 3, after the insertion of all newly arrived jobs, the π_T will

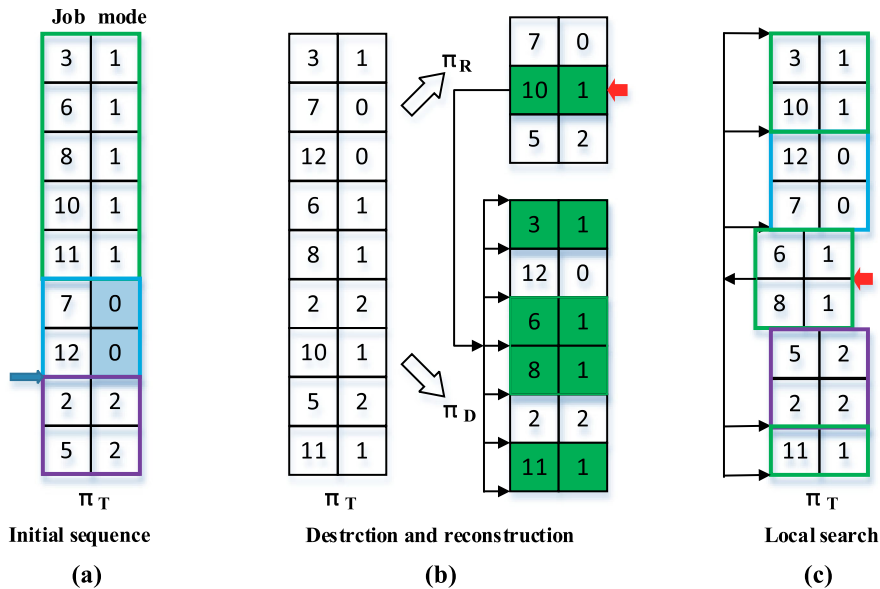


Figure 2. Solution representation of IG.

be optimised for several iterations by meta-heuristics. For IG, the π_T is optimised for several iterations through destruction, reconstruction, local search, and acceptance judgment.

The destruction operation removes several jobs from π_T randomly. First, a production mode k is selected randomly. Then, a job belonging to k is removed from π_T and inserted into the reconstruction sequence π_R . This operation continues until $|\pi_T| * p_{des}$ jobs are destructed. Where $|\pi_T|$ denotes the number of jobs in π_T and p_{des} is the proportion of destructed jobs. After the destruction, π_R with size $|\pi_T| * p_{des}$ is obtained, and the left solution sequence in π_T is denoted as π_D .

The reconstruction inserts jobs in π_R to π_D . A job j in π_R is randomly selected, removed, and tested at all possible positions in π_D resulted from j 's production mode. Job j is finally inserted at the position with the minimum objective, i.e. total tardiness cost. As shown in Figure 2(b), the job j_{10} is selected and tested in all possible positions, resulted from production mode $k = 1$.

In local search, an insertion neighbourhood search (INS) (Ruiz and Stutzle 2007) is used to search for the best position for every production mode group. Each production mode is searched successively. For a production mode k , its sequence groups π_k is obtained. As shown in Figure 2(c), for $k = 1$, $\pi_1 = \{[[3, 1], [10, 1]], [[6, 1], [8, 1]], [11, 1]\}$. Then, every group in π_k is removed and inserted in all possible positions resulted by all remaining groups in π_T . As shown in Figure 2(c), the group $[[6, 1], [8, 1]]$ is removed and tested in all possible positions resulted by other groups. Finally, the selected group is placed in the position with the minimum objective value.

For the acceptance criteria, we also use a simulated annealing-like criterion as adapted in (Hatami, Ruiz, and Andrés-Romano 2015; Pan et al. 2019). In our experiment, worse solutions are accepted with probability τ .

5.2. GA

GA is a population-based evolutionary algorithm and has been widely used to solve scheduling problems (Ta, Billaud, and Bouquard 2015; Rahman, Sarker, and Essam 2016; Xu et al. 2016; Liao and Fu 2019). GA evolves through crossover, mutation, and selection for several iterations based on a population.

The initial solution and current job sequence π_T are generated using the same way illustrated in 5.1 IG. Jobs in π_T are processed successively. When a new job arrives, the newly arrived jobs are appended to π_T by the same approach illustrated in 5.1 IG. Then, a population with size N is generated by randomly permuting π_T for N times. $N = \min(N', |\pi_T| * (|\pi_T| - 1))$, where N' is the given population size. The population evolves several iterations. After evolution, the best individual in the population will replace π_T .

In the crossover, the N solution sequences in the population are regarded as parent sequences and paired randomly. The two parent sequences in a pair perform a crossover with a probability p_c . A crossover point is randomly chosen when performing the crossover, as shown in Figure 3(a). The job sequences are switched between the two parent sequences, P_1 and P_2 , resulting in two child sequences, C_1 and C_2 . C_1 and C_2 need to be repaired because some job sequences may be duplicated. As shown in Figure 3(a), the job sequence $[6, 1]$ and $[3, 1]$ in P_{22}

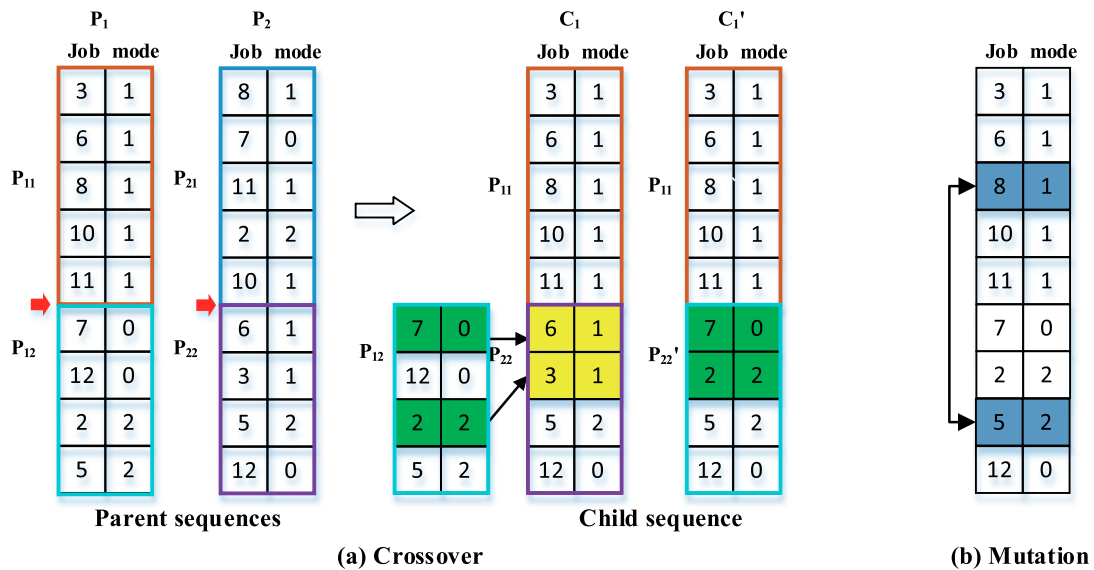


Figure 3. Illustration of crossover and mutation for GA.

are duplicated, as they have appeared in P_{11} . The duplicate sequences are replaced by sequences that appear in P_{12} but do not appear in P_{22} , i.e. sequences $[7, 0]$ and $[2, 2]$.

In the mutation, every individual in the population with a probability p_m exchange two of its genes randomly. As shown in Figure 3(b), the gene $[8, 1]$ and $[5, 2]$ are selected and switched.

In the selection, a new population with size N is generated by selecting individuals from the current population. An individual l will be selected with probability p_l and can be selected repeatedly, where p_l is calculated as follows.

$$p_l = \frac{f_l}{\sum_{l=1}^N f_l}, l = 1, 2, 3, \dots, N$$

where f_l denotes the fitness, i.e. total tardiness cost, of individual l .

6. Numerical experiments

In this section, the A2C-based reconfiguration and scheduling agents are trained on a large range of training instances. The trained model is then saved and used to make decisions for dynamic scheduling and reconfiguration on a new set of test instances. To verify the performance of the A2C-based approach, we compare it with two traditional meta-heuristics, IG and GA, on the test instances. In addition, to verify whether the trained agent has learned to choose appropriate actions for different situations or not, we compare the trained agent with some single dispatching rules.

6.1. Training process of A2C

Since the studied problem is a novel one, no benchmark instances are available in the literature. Following Luo (2020), we generated a set of instances randomly based on production configurations of n , m , and X , as shown in Table 1. Thus, a total number of $5 \times 4 \times 4 = 80$ instances are generated. Among those instances, 70% of them are selected randomly as training instances, and the remaining 30% instances are used as test instances.

Table 1. Parameter settings of production configurations.

Parameter	Value
Number of jobs (n)	{50, 80, 100, 120, 150}
Number of machines (m)	{3, 4, 6, 8}
Number of production modes (X)	{3, 4, 6, 8}
The proportion of initial jobs (p_{init})	10%
Interval time between two successive jobs	E(1/25)
Processing time of a job on a machine (t_{ij})	U[1, 50]
Setup time of reconfiguration (t_{setup})	60
Unit tardiness cost of a job (α_j)	U[0, 5]

For each instance, n_{init} jobs arrive at time zero, where $n_{init} = \max(n \cdot p_{init}, 3)$. The remaining $n - n_{init}$ jobs arrive following a Poisson distribution, and the average arrival time between two successive jobs is 25 s. For all parameters, the random seed is set to 1.

The parameters TF and RDD in Equation (2) are all set to 0.5. Parameters in IG are set as $p_{desJ} = 0.2$, $\tau = 0.05$; in GA, $p_c = 0.8$, $p_m = 0.1$, $N' = 50$.

All algorithms are coded with Python 3.8, and the experiments are performed on a PC with Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz and 12.0 GB of RAM. After a huge amount computing for calibration, the value of each hyperparameter is determined, as shown in Table 2. In Table 2, the entropy coefficient β decreases from 0.005 to 0.0005 in the first 70% training epochs, i.e. the first 3500 epochs, and keeps 0.0005 for the last 30% epochs.

The A2C-based scheduling and reconfiguration agents are trained for 5000 epochs. In each epoch, the SCD and RCF are first trained on all training instances and then evaluated on all test instances. The average total tardiness cost on all test instances during training epochs is shown in Figure 4. Note that the total tardiness cost in our training and test experiments is rescaled by dividing 1000. Figure 4 shows that the average total tardiness cost decreases dramatically, indicating that the A2C-based agents have learned to choose appropriate actions to generate a better solution. For the first 2000 training epochs, the total tardiness cost decreased from 125.54 to 80.30, realising a 36.04% improvement. The average total tardiness cost remains relatively stable in the last 1000 epochs, indicating the agents can provide high-level solutions stably. Note that the training curve is not fully converging. This may be due to the exploration mechanism of DRL. In other DRL-based scheduling literature (Zhang et al. 2020; Luo 2020), the training curve also not fully converges.

Figures 5 and 6 present the average episode reward obtained by SCD and RCF during training epochs to reflect the learning effect further. Each test instance in a training epoch is regarded as an episode. The average episode reward in these two figures is the average value of episode rewards on all test instances in a training epoch. As shown in these two figures, the average

Table 2. Parameter settings of A2C.

Parameter	Value
Number of hidden layers	3
Number of neurons in each hidden layer	100
Number of training epochs	5000
Discount factor (γ)	0.98
Update step iteration (T)	5
Learning rate of actor and critic for RCF	1E-5, 5E-5
Learning rate of actor and critic for SCD	6E-6, 2E-5
Range of entropy coefficient (β)	0.005 ~ 0.0005

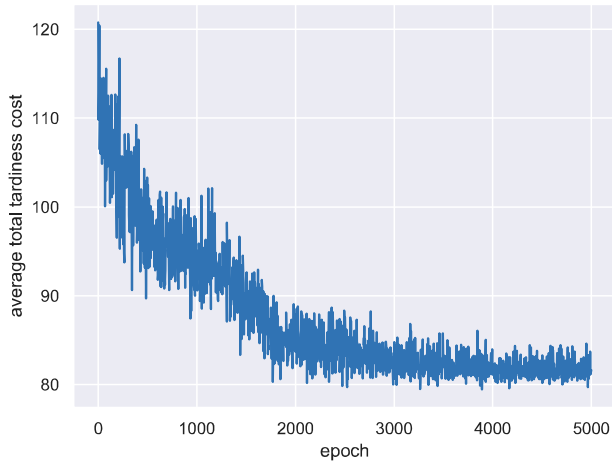


Figure 4. The average total tardiness cost on all test instances during training epochs.

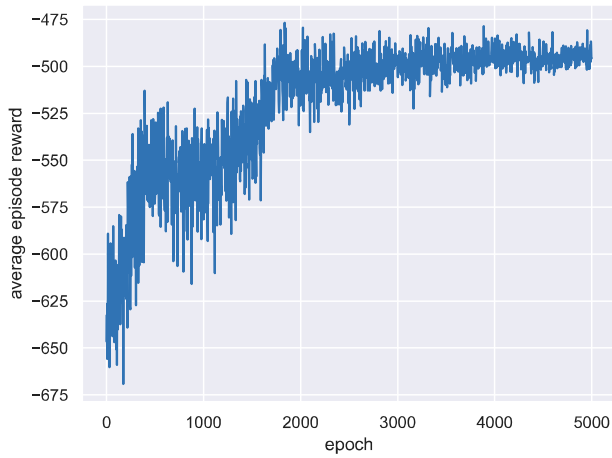


Figure 5. The training curve of scheduling agent considering average episode reward on all test instances during training epochs.

episode reward increases significantly. Since DRL agents aim to maximise episode reward, the two figures indicate that both SCD and RCF have efficiently learned how to choose an appropriate action at decision points. The learning effects validate that the system modelling of scheduling and reconfiguration are reasonable, and the DRL can be trained to solve the dynamic scheduling and reconfiguration problems.

6.2. Comparison with meta-heuristics

After 7 days of training, the trained model of A2C-based agents is saved and compared with IG and GA on the test instances. Each test instance is calculated 10 times repeatedly. IG and GA are tested at two searching iteration levels, i.e. 50 and 300 iterations. The average total tardiness cost of all algorithms is shown in Figure 7. Figure 7 shows that the A2C obtains the best results, followed by

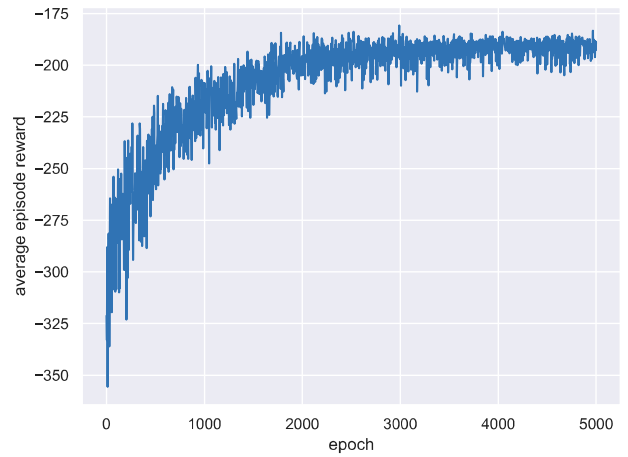


Figure 6. The training curve of reconfiguration agent considering average episode reward on all test instances during training epochs.

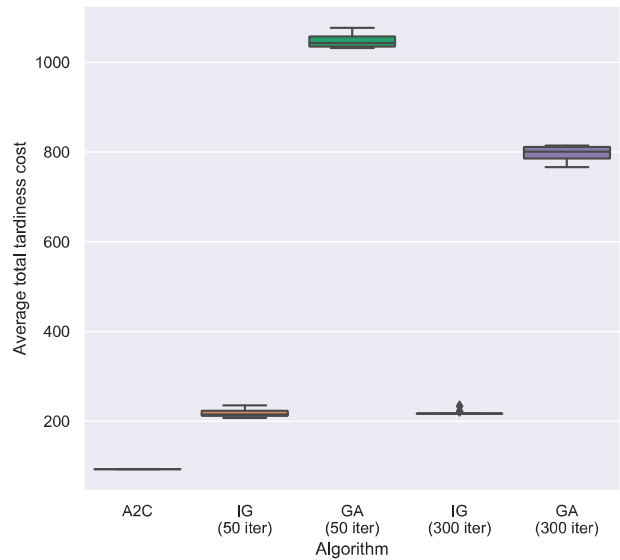


Figure 7. Algorithm comparison in terms of average total tardiness cost on all test instances.

IG, and GA with 50 iterations generates the worst results. IG outperforms GA by approximately three times. This is in line with the review of Fernandez-Viagas, Ruiz, and Framinan (2017), which shows that IG and IG-based algorithms are efficient in solving PFSP. The average total tardiness cost of GA decreased greatly when the iterations increased from 50 to 300, indicating that GA requires more optimising iterations to search for a better solution. However, the solutions of IG do not improve at a statistical level when the iteration increases, indicating the solutions of IG may have dropped into a local optimal.

The specific total tardiness cost and CPU time under instance characteristics for all algorithms are listed in Table 3. Table 3 shows that the A2C-based method

Table 3. Average total tardiness cost and CPU times on test instances for all compared algorithms.

		Total tardiness cost					CPU times (s)				
		50 iterations			300 iterations		50 iterations			300 iterations	
		A2C	IG	GA	IG	GA	A2C	IG	GA	IG	GA
<i>n</i>	50	68.24	83.57	297.31	85.60	203.10	0.07	36.94	36.55	234.51	186.81
	80	84.26	148.65	709.36	156.57	517.14	0.12	162.41	76.17	902.59	396.99
	100	93.76	236.51	1172.17	223.99	888.87	0.14	409.45	121.62	2346.21	626.68
	120	144.06	392.37	1694.67	384.06	1388.16	0.17	603.09	185.61	3476.31	987.24
	150	119.87	470.96	2590.34	467.30	1977.28	0.23	1801.89	272.40	11083.99	1426.86
<i>m</i>	3	53.36	205.73	817.41	208.92	593.57	0.14	530.53	155.48	3513.9	808.90
	4	57.38	154.31	657.95	163.87	495.84	0.11	429.34	98.42	2638.28	512.86
	6	151.24	340.82	1811.87	333.82	1428.78	0.15	671.28	127.68	3750.61	677.98
	8	113.59	135.69	735.13	128.20	515.58	0.11	95.17	49.92	461.48	251.67
<i>X</i>	3	57.56	196.26	1094.80	202.99	746.25	0.15	830.63	144.82	5089.9	756.15
	4	84.50	171.27	764.26	164.72	580.34	0.12	240.39	87.73	1354.31	448.32
	6	84.79	169.75	637.25	168.28	481.51	0.10	206.80	88.68	1240.70	455.99
	8	189.82	405.95	1928.04	408.83	1639.33	0.15	468.19	125.04	2783.75	684.07
Ave		93.12	218.46	1047.24	218.76	795.67	0.13	469.51	113.14	2818.81	591.50

obtains the best results in all test instances both in solution quality and CPU time. Specifically, the A2C outperforms IG and GA by 57.37% and 91.11%, using only 2.77‰ and 1.15‰ CPU time of IG and GA, when 50 optimising iterations are set for meta-heuristics. When the iterations increased to 300, A2C outperforms IG and GA by 57.43% and 88.30%, using only 0.46‰ and 2.20‰ CPU time of IG and GA. Note that, in Algorithm 3, the rescheduling is activated at decision points having newly arrived jobs. This kind of rescheduling activation mechanism spends less CPU time than that of activation for each job (Liu, Jin, and Price 2017, 2018) and more CPU time than that of activation for several jobs (Valledor et al. 2018). If the rescheduling is only activated five times as did in (Valledor et al. 2018), the CPU time of IG and GA will reduce at most around 18 times on test instances. Because each test instance has 90 new jobs on average, activating at most 90 rescheduling events. Even though the CPU times of IG and GA are divided by 18, the average CPU time of A2C is approximately only 0.50% and 2.07% of those of IG and GA with 50 iterations. The A2C still has a big advantage on CPU time and will have a larger advantage on objective value since the solution quality of meta-heuristics will get worse if less optimisation is performed.

The mechanism of solving the dynamic scheduling and reconfiguration between the DRL and meta-heuristics are different. The DRL chooses the best action at each rescheduling point based on current state features. Rather than using action spaces, the IG and GA find the optimal job sequence through searching the solution spaces. The current solution π_T is optimised to obtain the best job sequence and production mode sequence. The optimal solution may be hard to find when dynamic reconfiguration is considered. However, the DRL method establishes two separate intelligent systems for dynamic

scheduling and reconfiguration and trains the two intelligent systems/agents using extensive production data. For the DRL-based approach, its significant advantages may result from the properly designed DRL system of scheduling and reconfiguration based on expert knowledge, the generalisation capability of the neural network, and the efficient usage of historical data.

Table 3 also shows that the average CPU times of A2C for calculating a test instance is 0.13 s. With the number of jobs in those test instances considered, the decision time for a single job is only 1.47 ms, which is almost instantaneous and can be used for real-time scheduling and reconfiguration.

In Table 3, when the number of jobs n increases, the disparity between A2C and IG and GA also increases. When n is 50, A2C outperforms IG and GA (under 300 iterations) by 20.28% and 66.40%. However, when n is 150, A2C outperforms them by 74.35% and 93.94%. This shows A2C-based approach has more advantages in large instances. It may be because the decision quality of DRL agents is irrelevant to the number of jobs. Nevertheless, for larger instances with more jobs, it is hard to search an optimal solution by meta-heuristics.

Compared between the two meta-heuristics, IG outperforms GA by 79.14% and 72.51% at 50 and 300 iterations, respectively. However, the CPU times of IG are 3.15 and 3.77 times larger than these of GA. When the iteration increase from 50 to 300, GA improved by 24.02%, indicating that GA requires more optimising iterations to find the optimal solution. However, IG shows a little decrease when the iteration increases. This may be because IG has dropped into a local optimal.

All the instances, training and test results, and video of solving the test instances using the A2C-based trained model are available at https://osf.io/qfvu9/?view_only=e10ee7f017f3449887ea8d39c83339a4.

6.3. Comparison with single dispatching rule

To further verify whether the trained agent has learned to choose appropriate actions for different situations or not, we compare the trained agent with each efficient single dispatching rule (SDR), as performed in (Lin et al. 2019a; Luo 2020; Yang, Xu, and Wang 2021). The SDR in this study is an action pair with a scheduling action and a reconfiguration action. Under a specific SDR, the scheduling and reconfiguration actions are fixed and used at all decision points. Since no dispatching rule performs best for all situations (Luo 2020), the trained agent selecting appropriate actions for different situations should outperform the best SDR using the fixed action for all situations, even though the fixed action is the best single action.

To find several efficient SDRs, we use the trained A2C agent to calculate the whole set of test instances 10 times. The scheduling and reconfiguration actions selected at each decision step are recorded. The proportions of selected scheduling and reconfiguration actions by the trained agent at all decision steps on all test instances are shown in Figure 8. Figure 8(a) shows that the frequently selected scheduling actions are the a_4 and a_2 , indicating that selecting jobs with the minimum due date and the maximum unit tardiness cost have larger advantages in most situations. Figure 8(b) shows that the A_1 and A_3 are the frequently selected reconfiguration actions, indicating that selecting the production mode with the maximum total current unit tardiness cost and the total number of jobs tend to be better choices for most situations.

For scheduling actions, the top two frequently selected actions, a_4 and a_2 , are used. Also, the two famous dispatching rules, FIFO (a_5) and SPT (a_6), are considered. For reconfiguration actions, the top two actions, A_1 and A_3 , are used. By pairing the $[a_4, a_2]$ and $[A_1, A_3]$ in order, eight SDRs are obtained. The $\{a_4, A_1\}$, $\{a_4, A_3\}$, and $\{a_2, A_1\}$ are the SDR1, SDR2, and SDR3, respectively. In addition, the random policy, denoted as Rand, is used

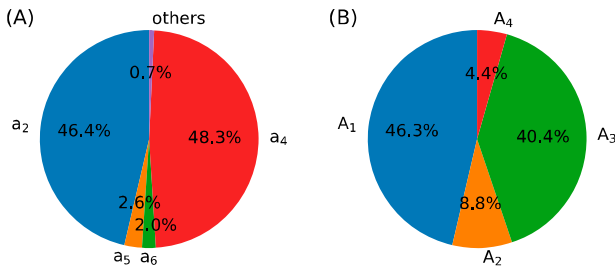


Figure 8. The proportions of selected actions by the trained agent at all decision steps on all test instances. (A) The scheduling action. (B) The reconfiguration action.

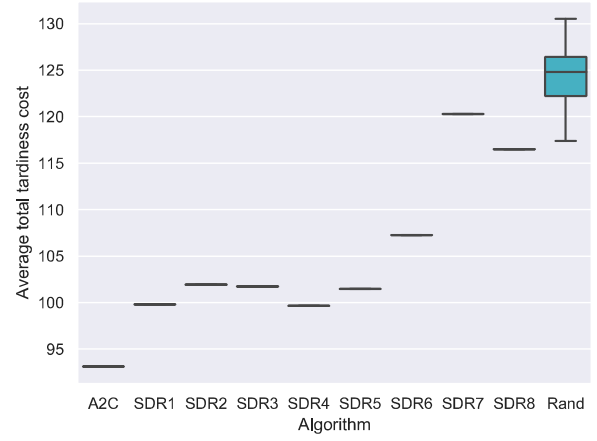


Figure 9. Comparison between A2C and single dispatching rules on test instances.

as a baseline. The random policy selects scheduling and reconfiguration actions randomly at each decision point.

The average total tardiness cost of the trained A2C, SDRs, and random policy on the test instances are provided in Figure 9. The comparison experiment is repeated 10 times. Figure 9 shows that the A2C outperforms the best SDRs, indicating that choosing appropriate actions for different situations is better than simply choosing the fixed best single action for all situations. This also indicates that the trained agent has learned to choose appropriate actions among those efficient actions for different situations.

The efficiency/fitness of the frequently selected actions can be evaluated from Figure 9. The SDR1-SDR4 obtain better results than those of SDR6-SDR8 and the random policy, indicating the $[a_4, a_2]$ and $[A_1, A_3]$ are efficient scheduling and reconfiguration actions. The SDR6-SDR8 obtain worse results due to the inefficient scheduling actions, a_5 and a_6 , since the reconfiguration actions in SDR6-SDR8 are also the A_1 and A_3 . Recall that, in Figure 8, the $[a_4, a_2]$ and $[A_1, A_3]$ have larger proportions, and the a_5 and a_6 have small proportions. This indicates that the trained agent has learned to distinguish and choose more efficient actions. The inefficient actions can be removed, and more efficient actions should be designed to further improve the results in future studies. When the action space changes, the DRL agent should be retrained.

7. Conclusions

This paper studied the intelligent scheduling and reconfiguration for an RFL with dynamic job arrival using DRL in smart manufacturing to provide an intelligent decision-making for dynamic scheduling and reconfiguration. To the best of our knowledge, this is the first

attempt to solve the intelligent scheduling and reconfiguration using DRL. To solve the problem, a system architecture of intelligent scheduling and reconfiguration in smart manufacturing is proposed, and a mathematical model is established to minimise total tardiness cost of all jobs. Also, a DRL system containing scheduling and reconfiguration agents is proposed. The state features, actions, and rewards are designed for the two agents. Moreover, the procedures for using both DRL and meta-heuristics to solve dynamic scheduling and reconfiguration with new job arrival are proposed. Finally, a large range of instances is generated to train and test the A2C-based agents.

The training results show that the average total tardiness cost decreases significantly during training epochs, indicating the A2C-based agents have effectively learned to generate better solutions for unseen instances. The average episode rewards of the two agents show a marked increase during the training epochs, indicating the two agents indeed have learned to choose good actions at decision points, verifying the correctness of our system modelling for scheduling and reconfiguration. After training, the test results show that our A2C-based approach outperforms two traditional meta-heuristics, IG and GA, by a large margin. Specifically, the A2C-based approach outperforms IG and GA under 300 iterations by 57.43% and 88.30%, respectively, using only 0.46‰ and 2.20‰ CPU times of IG and GA. The huge advantages verify the great benefits of the DRL-based approach in solving dynamic scheduling and reconfiguration. Most notably, our A2C-based trained model can generate a scheduling or reconfiguration action within 1.47 ms when a new job comes, which is almost instantaneously. The DRL-based approach can be used for real-time optimisation and online decision-making in an actual reconfigurable workshop with dynamic events.

Compared with meta-heuristics, the DRL-based approach can provide better solutions based on the properly designed DRL system, generate a decision instantly after trained, and make full use of production data to realise online and offline learning. These benefits show a promising prospect of using DRL to solve intelligent scheduling and reconfiguration in a real production environment. Our study can help to build a real-time optimisation, self-organising, and self-learning RMS in smart manufacturing.

Future research could further consider the specific production configurations in an RMS, such as the number of workstations, vehicles, and workers. Besides, system modelling should be further studied, such as designing more efficient states and actions or automatically generating system features and actions through learning.

Moreover, other DRL algorithms should be proposed to improve the convergence and test results of this study.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This research was funded by the National Natural Science Foundation of China (61803367) and the Natural Science Foundation of Liaoning Province (2019-MS-346).

Notes on contributors



Shenglou Yang received the B.S. degree from Guangxi University, China, in 2015. He is currently working toward the Ph.D. degree at Shenyang Institute of Automation, Chinese Academy of Sciences. His research interests include production scheduling, deep reinforcement learning, and process modelling and optimisation.



Zhigang Xu received the Ph.D. degree from Shenyang Institute of Automation, Chinese Academy of Sciences. He is a senior visiting scholar from the University of Michigan. He is currently a member of the State Key Laboratory of Robotics, Chinese Academy of Sciences, where he is also the Head of the Intelligent Equipment and Production Line Laboratory, Shenyang Institute of Automation. He has published over 100 academic papers and 200 Chinese patents in these areas. He has presided over many major research projects, such as the National 863 programme, national major science and technology projects, and space station projects. His research interests include intelligent manufacturing, intelligent factory planning, robotic mechanisms, and ground simulation of spacecraft.

ORCID

Shenglou Yang  <http://orcid.org/0000-0002-6353-2560>

References

- Barenji, Ali Vatankhah, Reza Vatankhah Barenji, Danial Roudi, and Majid Hashemipour. 2016. "A Dynamic Multi-Agent-Based Scheduling Approach for SMEs." *The International Journal of Advanced Manufacturing Technology* 89 (9-12): 3123–3137. doi:10.1007/s00170-016-9299-4.
- Bortolini, Marco, Francesco Gabriele Galizia, and Cristina Mora. 2018. "Reconfigurable Manufacturing Systems: Literature Review and Research Trend." *Journal of Manufacturing Systems* 49: 93–106.
- Campos Sabioni, Rachel, Joanna Daaboul, and Julien Le Duigou. 2021. "Concurrent Optimisation of Modular Product and Reconfigurable Manufacturing System Configuration: A Customer-Oriented Offer for Mass Customisation." *International Journal of Production Research*, 1–17. doi:10.1080/00207543.2021.1886369.

- Dou, J. P., J. Li, and C. Su. 2016a. "Bi-objective Optimization of Integrating Configuration Generation and Scheduling for Reconfigurable Flow Lines Using NSGA-II." *International Journal of Advanced Manufacturing Technology* 86 (5-8): 1945–1962. doi:10.1007/s00170-015-8291-8.
- Dou, Jianping, Jun Li, and Chun Su. 2016b. "Bi-objective Optimization of Integrating Configuration Generation and Scheduling for Reconfigurable Flow Lines Using NSGA-II." *The International Journal of Advanced Manufacturing Technology* 86 (5-8): 1945–1962.
- Dou, J. P., J. Li, D. Xia, and X. Zhao. 2020. "A Multi-Objective Particle Swarm Optimisation for Integrated Configuration Design and Scheduling in Reconfigurable Manufacturing System." *International Journal of Production Research* 1–21. doi:10.1080/00207543.2020.1756507.
- Dou, Jianping, Jun Li, Dan Xia, and Xia Zhao. 2021. "A Multi-Objective Particle Swarm Optimisation for Integrated Configuration Design and Scheduling in Reconfigurable Manufacturing System." *International Journal of Production Research* 59 (13): 1–21.
- Fang, Y., C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan. 2019. "Digital-Twin-Based Job Shop Scheduling Toward Smart Manufacturing." *IEEE Transactions on Industrial Informatics* 15 (12): 6425–6435. doi:10.1109/TII.2019.2938572.
- Fernandez-Viagas, V., J. M. Molina-Pariente, and J. M. Framinan. 2020. "Generalised Accelerations for Insertion-Based Heuristics in Permutation Flowshop Scheduling." *European Journal of Operational Research* 282 (3): 858–872. doi:10.1016/j.ejor.2019.10.017.
- Fernandez-Viagas, Victor, Rubén Ruiz, and Jose M. Framinan. 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. doi:10.1016/j.ejor.2016.09.055.
- Fernandez-Viagas, V., J. M. S. Valente, and J. M. Framinan. 2018. "Iterated-Greedy-Based Algorithms with Beam Search Initialization for the Permutation Flowshop to Minimise Total Tardiness." *Expert Systems with Applications* 94: 58–69. doi:10.1016/j.eswa.2017.10.050.
- Haddou Benderbal, Hichem, Mohammed Dahane, and Lyes Benyoucef. 2017. "Modularity Assessment in Reconfigurable Manufacturing System (RMS) Design: An Archived Multi-Objective Simulated Annealing-Based Approach." *The International Journal of Advanced Manufacturing Technology* 94 (1-4): 729–749. doi:10.1007/s00170-017-0803-2.
- Hatami, Sara, Rubén Ruiz, and Carlos Andrés-Romano. 2015. "Heuristics and Metaheuristics for the Distributed Assembly Permutation Flowshop Scheduling Problem with Sequence Dependent Setup Times." *International Journal of Production Economics* 169: 76–88. doi:10.1016/j.ijpe.2015.07.027.
- Ivanov, D., B. Sokolov, W. W. Chen, A. Dolgui, F. Werner, and S. Potryasaev. 2021. "A Control Approach to Scheduling Flexibly Configurable Jobs with Dynamic Structural-Logical Constraints." *Iise Transactions* 53 (1): 21–38. doi:10.1080/24725854.2020.1739787.
- Karabulut, K. 2016. "A Hybrid Iterated Greedy Algorithm for Total Tardiness Minimization in Permutation Flowshops." *Computers & Industrial Engineering* 98: 300–307. doi:10.1016/j.cie.2016.06.012.
- Kazemi, Hamed, Mohammad Mahdavi Mazdeh, and Mohammad Rostami. 2017. "The Two Stage Assembly Flow-Shop Scheduling Problem with Batching and Delivery." *Engineering Applications of Artificial Intelligence* 63: 98–107. doi:10.1016/j.engappai.2017.05.004.
- Liao, Wenzhu, and Yanxiang Fu. 2019. "Min–Max Regret Criterion-Based Robust Model for the Permutation Flow-Shop Scheduling Problem." *Engineering Optimization* 52 (4): 687–700. doi:10.1080/0305215x.2019.1607848.
- Lin, C. C., D. J. Deng, Y. L. Chih, and H. T. Chiu. 2019a. "Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network." *IEEE Transactions on Industrial Informatics* 15 (7): 4276–4284. doi:10.1109/TII.2019.2908210.
- Lin, Chun-Cheng, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. 2019b. "Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network." *IEEE Transactions on Industrial Informatics* 15 (7): 4276–4284.
- Liu, Chien-Liang, Chuan-Chin Chang, and Chun-Jan Tseng. 2020. "Actor-Critic Deep Reinforcement Learning for Solving Job Shop Scheduling Problems." *IEEE Access* 8: 71752–71762.
- Liu, Weibo, Yan Jin, and Mark Price. 2017. "New Scheduling Algorithms and Digital Tool for Dynamic Permutation Flowshop with Newly Arrived Order." *International Journal of Production Research* 55 (11): 3234–3248. doi:10.1080/00207543.2017.1285077.
- Liu, Weibo, Yan Jin, and Mark Price. 2018. "New Meta-Heuristic for Dynamic Scheduling in Permutation Flowshop with New Order Arrival." *The International Journal of Advanced Manufacturing Technology* 98 (5-8): 1817–1830.
- Luo, Shu. 2020. "Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning." *Applied Soft Computing* 91: 106208. doi:10.1016/j.asoc.2020.106208.
- Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. "Asynchronous Methods for Deep Reinforcement Learning." Paper presented at the International Conference on Machine Learning.
- Pagnozzi, Federico, and Thomas Stützel. 2016. "Speeding up Local Search for the Insert Neighborhood in the Weighted Tardiness Permutation Flowshop Problem." *Optimization Letters* 11 (7): 1283–1292. doi:10.1007/s11590-016-1086-5.
- Pan, Quan-Ke, Liang Gao, Li Xin-Yu, and Framinan M. Jose. 2019. "Effective Constructive Heuristics and Meta-Heuristics for the Distributed Assembly Permutation Flowshop Scheduling Problem." *Applied Soft Computing* 81: 105492. doi:10.1016/j.asoc.2019.105492.
- Pan, Quan-Ke, Rubén Ruiz, and Pedro Alfaro-Fernández. 2017. "Iterated Search Methods for Earliness and Tardiness Minimization in Hybrid Flowshops with Due Windows." *Computers & Operations Research* 80: 50–60. doi:10.1016/j.cor.2016.11.022.
- Rahman, Humyun Fuad, Mukund Nilakantan Janardhanan, and Izabela Ewa Nielsen. 2019. "Real-Time Order Acceptance and Scheduling Problems in a Flow Shop Environment Using Hybrid GA-PSO Algorithm." *IEEE Access* 7: 112742–112755.
- Rahman, Humyun Fuad, Ruhul Sarker, and Daryl Essam. 2016. "A Genetic Algorithm for Permutation Flowshop Scheduling Under Practical Make-to-Order Production System." *Artificial Intelligence for Engineering Design, Analysis and*

- Manufacturing* 31 (1): 87–103. doi:10.1017/s0890060416000196.
- Ribas, Imma, Ramon Companys, and Xavier Tort-Martorell. 2019. “An Iterated Greedy Algorithm for Solving the Total Tardiness Parallel Blocking Flow Shop Scheduling Problem.” *Expert Systems with Applications* 121: 347–361. doi:10.1016/j.eswa.2018.12.039.
- Ruiz, Rubén, Quan-Ke Pan, and Bahman Naderi. 2019. “Iterated Greedy Methods for the Distributed Permutation Flowshop Scheduling Problem.” *Omega* 83: 213–222. doi:10.1016/j.omega.2018.03.004.
- Ruiz, Ruben, and Thomas Stutzle. 2007. “A Simple and Effective Iterated Greedy Algorithm for the Permutation Flowshop Scheduling Problem.” *European Journal of Operational Research* 177 (3): 2033–2049. doi:10.1016/j.ejor.2005.12.009.
- Schaller, Jeffrey, and Jorge M. S. Valente. 2019. “Heuristics for Scheduling Jobs in a Permutation Flow Shop to Minimize Total Earliness and Tardiness with Unforced Idle Time Allowed.” *Expert Systems with Applications* 119: 376–386. doi:10.1016/j.eswa.2018.11.007.
- Shi, Daming, Wenhui Fan, Yingying Xiao, Tingyu Lin, and Chi Xing. 2020. “Intelligent Scheduling of Discrete Automated Production Line via Deep Reinforcement Learning.” *International Journal of Production Research* 58 (11): 3362–3380.
- Ta, Quang Chieu, Jean-Charles Billaut, and Jean-Louis Bouquard. 2015. “Matheuristic Algorithms for Minimizing Total Tardiness in the m-Machine Flow-Shop Scheduling Problem.” *Journal of Intelligent Manufacturing* 29 (3): 617–628. doi:10.1007/s10845-015-1046-4.
- Valledor, Pablo, Alberto Gomez, Paolo Priore, and Javier Puente. 2018. “Solving Multi-Objective Rescheduling Problems in Dynamic Permutation Flow Shop Environments with Disruptions.” *International Journal of Production Research* 56 (19): 6363–6377. doi:10.1080/00207543.2018.1468095.
- Wang, Kai, Hao Luo, Feng Liu, and Xiaohang Yue. 2018. “Permutation Flow Shop Scheduling With Batch Delivery to Multiple Customers in Supply Chains.” *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48 (10): 1826–1837. doi:10.1109/tsmc.2017.2720178.
- Wang, Haoxiang, Bhaba R. Sarker, Jing Li, and Jian Li. 2020. “Adaptive Scheduling for Assembly Job Shop with Uncertain Assembly Times Based on Dual Q-Learning.” *International Journal of Production Research*, 1–17.
- Waschneck, Bernd, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. 2018. “Optimization of Global Production Scheduling with Deep Reinforcement Learning.” *Procedia CIRP* 72 (1): 1264–1269.
- Wu, Cheng-Hung, Fang-Yi Zhou, Chi-Kang Tsai, Cheng-Juei Yu, and Stéphane Dauzère-Pérès. 2020. “A Deep Learning Approach for the Dynamic Dispatching of Unreliable Machines in Re-entrant Production Systems.” *International Journal of Production Research* 58 (9): 2822–2840.
- Xu, Jianyou, Win-Chin Lin, Junjie Wu, Shuenn-Ren Cheng, Zi-Ling Wang, and Chin-Chia Wu. 2016. “Heuristic Based Genetic Algorithms for the Re-entrant Total Completion Time Flowshop Scheduling with Learning Consideration.” *International Journal of Computational Intelligence Systems* 9 (6): 1082–1100. doi:10.1080/18756891.2016.1256572.
- Yang, Shengluo, and Zhigang Xu. 2020. “The Distributed Assembly Permutation Flowshop Scheduling Problem with Flexible Assembly and Batch Delivery.” *International Journal of Production Research*, 1–19. doi:10.1080/00207543.2020.1757174.
- Yang, S., Z. Xu, and J. Wang. 2021. “Intelligent Decision-Making of Scheduling for Dynamic Permutation Flowshop via Deep Reinforcement Learning.” *Sensors (Basel)* 21 (3), doi:10.3390/s21031019.
- Yelles-Chaouche, Abdelkrim R., Evgeny Gurevsky, Nadjib Brahimi, and Alexandre Dolgui. 2020. “Reconfigurable Manufacturing Systems from an Optimisation Perspective: A Focused Review of Literature.” *International Journal of Production Research*, 1–19. doi:10.1080/00207543.2020.1813913.
- Yuan, Minghai, Kun Deng, W. A. Chaovalitwongse, and Shuo Cheng. 2016. “Multi-Objective Optimal Scheduling of Reconfigurable Assembly Line for Cloud Manufacturing.” *Optimization Methods and Software* 32 (3): 581–593. doi:10.1080/10556788.2016.1230210.
- Zhang, Cong, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. 2020. “Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning.” *arXiv preprint arXiv:2010.12367*.
- Zhang, Zhicong, Weiping Wang, Shouyan Zhong, and Kaishun Hu. 2013. “Flow Shop Scheduling with Reinforcement Learning.” *Asia-Pacific Journal of Operational Research* 30 (5): 1350014.