Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

# To reduce maximum tardiness by *Seru* Production: model, cooperative algorithm combining reinforcement learning and insights

**Guanghui Fu[a], Yang Yu[a*], Wei Sun[b] and Ikou Kaku[c]**

[a]*State Key Laboratory of Synthetic Automation for Process Industries, Department of Intelligent data and Systems Engineering, Northeastern University, Shenyang, P.R. China*
[b]*Business School, Liaoning University, Shenyang, P.R. China*
[c]*Faculty of Environmental and Information Studies, Tokyo City University, Tokyo, Japan*

**C H R O N I C L E**

*Article history*:
Received August 10 2022
Received in Revised Format August 31 2022
Accepted October 5 2022
Available online October, 11 2022

Keywords:
*Cooperative algorithm*
*Reinforcement learning*
*Maximum tardiness*
*Seru production*

**A B S T R A C T**

The maximum tardiness reflects the worst level of service associated with customer needs; thus, the principle that *seru* production reduces the maximum tardiness is investigated, and a model to minimize the maximum tardiness of the *seru* production system is established. In order to obtain the exact solution, the non-linear *seru* production model with minimizing the maximum tardiness is split into a *seru* formation model and a linear *seru* scheduling model. We propose an efficient cooperative algorithm using a genetic algorithm and an innovative reinforcement learning algorithm (CAGARL) for large-scale problems. Specifically, the GA is designed for the *seru* formation problem. Moreover, the QL-*seru* algorithm (QLSA) is designed for the *seru* scheduling problem by combining the features of meta-heuristics and reinforcement learning. In the QLSA, we design an innovative QL-*seru* table and two state trimming rules to save computational time. After extensive experiments, compared with the previous algorithm, CAGARL improved by an average of 56.6%. Finally, several managerial insights on reducing maximum tardiness are proposed.

## 1. Introduction

Nowadays, the production environment is complex and dynamic, and *seru* production emerges as the times require. A *seru* production system comprises one or more *serus*(Yu et al., 2018). By refactoring the workers, the *seru* production system can achieve better performance than assembly lines (Liu et al., 2012; Yu et al., 2018). *Seru* production has many outstanding advantages (Liu et al., 2014, 2010): high flexibility, mass production efficiency, and environmental friendliness for sustainable manufacturing. Moreover, implementing the *seru* production system can reduce the makespan, setup time, required workers, costs, and shop space (Yılmaz, 2020a; Yu et al., 2018; Zhang et al., 2022). *Seru* production has been studied by several scholars. They specialized in reducing makespan, total labor hours, training cost, and manpower (Liu et al., 2013, 2021a, 2021b; Sun et al., 2020; Ying and Tsai, 2017; Yu et al., 2012; Zhan et al., 2021). Stecke et al. (2012) described the history of *seru* and defined the various types of *seru*. *Seru* and TPS are compared. Yu et al. (2012) studied the bi-objective *seru* production model with total throughput time and labor hours. Lian et al. (2018) solved the problem of assigning multi-skilled workers. In order to minimize the total cost of training, Liu et al. (2013) established a multi-objective model. Yılmaz (2020b) investigated the workforce scheduling problem of *seru* production. Yu et al. (2017) developed several line-hybrid *seru* system models with makespan and total labor hours. A dynamic multi-objective algorithm is proposed by Liu et al. (2021a) for the rotating *seru* production problem. Liu et al. (2021b) studied makespan and workload imbalance for a hybrid *seru* production system. Fu et al. (2022) studied four dynamic *seru* production decision processes and a phased intelligent algorithm for solving

* Corresponding author
E-mail: yuyang@ise.neu.edu.cn (Y. Yu)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2023 Growing Science Ltd.
doi: 10.5267/j.ijiec.2022.10.002

the four processes. Unlike the literature (Fu et al., 2022), this paper proposes to study the problem of *seru* formation and *seru* scheduling under minimizing the maximum tardiness, which has not been studied yet. In contrast, Fu et al. (2022) studied the problem of re-optimizing a given *seru* system after a change in the product information. The two problems have different decision processes, and different algorithms are proposed for the different problems. Maximum tardiness is a crucial indicator of performance in meeting customer due dates in various manufacturing and service businesses (Allahverdi, 2004; Bai et al. 2021; Chen et al. 2021). The maximum tardiness reflects the worst level of service associated with customer needs (Aydilek et al., 2022; Pundoor and Chen, 2005). Avoiding customer dissatisfaction as much as possible is the goal of the production manager (Rostami et al. 2015). Therefore, reducing the maximum tardiness can improve the level of service.

Many studies have been done to minimize the maximum tardiness of a given production system. Guinet and Solomon (1996) investigated the minimization of maximum tardiness or maximum completion time in hybrid flow shop scheduling and used a set of list algorithms to deal with the problem. Chakravarthy and Rajendran (1999) dealt with minimizing the weighted sum of the maximum tardiness and makespan in a flow shop and proposed using heuristic algorithms using simulated annealing technology to solve it. Sbihi and Varnier (2008) studied the maximum tardiness by a B&B algorithm in single-machine scheduling with multiple maintenance periods. Ruiz and Allahverdi (2009) investigated the weighted sum of makespan and maximum tardiness of the flow shop workshop scheduling and proposed a GA to solve it. An Adaptive GA and a PSA are proposed by Assarzadegan and Rasti-Barzoki (2016) for minimization of the sum of the due date assignment costs, maximum tardiness, and distribution costs on a single machine. Chen et al. (2021) investigated the minimization of total late work and maximum tardiness in single-machine bicriteria scheduling. As a key performance indicator, maximum tardiness has not yet been investigated in *seru* production. We state that *seru* production can reduce the maximum tardiness based on extensive tests. Reinforcement learning (RL) has been extensively applied in scheduling (Ren et al. 2021). Ying-Zi and Ming-Yang (2005) proposed using the Q-learning algorithm to select composite scheduling rules, relative to single scheduling rules or random compounding to get better results. Aydin and Öztemel (2000) studied a dynamic scheduling system based on intelligent agents to select the most suitable scheduling rules in real time through the improved Q-learning algorithm. Wei and Zhao (2004) proposed using reinforcement learning for scheduling rule selection that considers machine and job selection to solve dynamic job shop problems. Li et al. (2021) combined the characteristics of GA and Q-learning to propose a GA based on Q-learning (QGA) for the problem of workshop scheduling. Chen et al.(2020) studied a self-learning GA (SLGA), which used GA as the basic optimization method and intelligently adjusted its key parameters using reinforcement learning. Wang et al. (2020) studied a dual Q-learning method to improve the adaptability of assembly plant scheduling problems to environmental changes through independent learning. Shahrabi et al. (2017) used Q-learning algorithms to adjust the parameters of variable neighborhood search algorithms in dynamic job shop scheduling. Using the dual Q-learning algorithm, Arviv et al. (2016) proposed a new reinforcement learning collaboration algorithm for complex two-robot collaborative flow workshop scheduling.

Most of the previous algorithms for the *seru* scheduling problem are meta-heuristic algorithms, which are fast(Tang et al., 2018), but their search patterns are relatively fixed and rigid(Ni et al., 2021). However, reinforcement learning provides a more purposeful search of the solution space. Therefore, we design an innovative reinforcement learning algorithm (QL-*seru* algorithm) by combining the features of the meta-heuristic algorithm and the reinforcement learning algorithm to solve the *seru* scheduling problem.

Our contributions are as follows:

- We establish a *seru* production model with minimizing the maximum tardiness.
- The non-linear *seru* production model minimizing the maximum tardiness is split into a *seru* formation model and a linear *seru* scheduling model. Then the exact solution of small-scale problems can be solved by CPLEX.
- A cooperative algorithm using a GA and an innovative QL-*seru* algorithm is proposed for larger-scale problems. The best *seru* formations obtained by GA are used as the environment in reinforcement learning.
- The QL-*seru* algorithm is proposed for the *seru* scheduling problem. Moreover, two state trimming rules are proposed.
- Several managerial insights are made on reducing the maximum tardiness.

The remainder of this research is organized as follows. Section 2 proposes the model of the *seru* production system with minimizing the maximum tardiness. Moreover, we decompose the non-linear model into a *seru* formation model and a linear *seru* scheduling model. Section 3 develops the CAGARL. We design the GA and QL-*seru* algorithm in detail. Section 4 performs extensive experiments and discusses. Section 5 gives the conclusion and further research.

## 2. Model

### *2.1. Problem description*

We consider minimizing the maximum tardiness problem of a rotating *seru* production system, as shown in Fig. 1. There are *Z* workers and *M* batches. Because of the different skill levels of different workers, a batch in different *serus* has different processing times. So, to improve the performance of the *seru* production system, better *seru* formation (i.e., the number of *serus* and worker allocation.) and better *seru* scheduling (i.e., batch scheduling) are required (Fu et al., 2022; Wu et al., 2021).

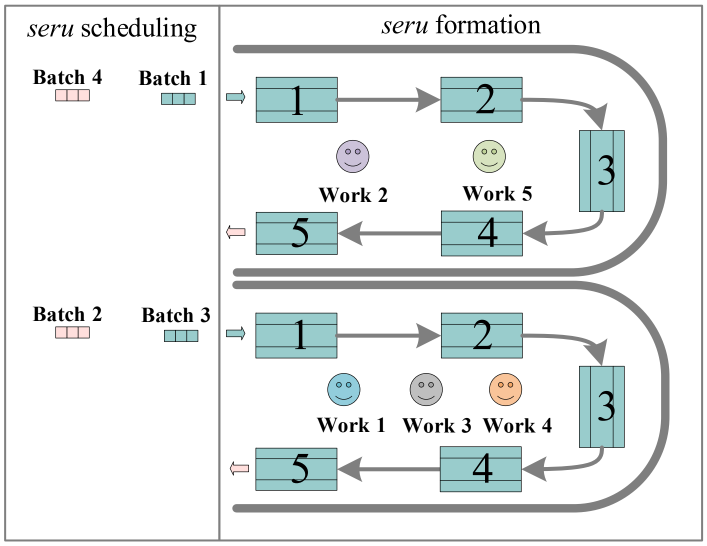


**Fig. 1.** Example of *seru* production system with 5 workers and 4 batches

The objective of the problem is to minimize the maximum tardiness. The tardiness of each batch is determined by the completion time and due date of each batch. *Seru* formation is shown in Fig. 1 as an example. The result of *seru* scheduling is shown in Fig. 2. As seen from Fig. 2, the tardiness of batch 3 is 0, and the tardiness of batches 1, 2, and 3 are 50, 170, and 20, respectively. So, the maximum tardiness of this *seru* production system is 170.

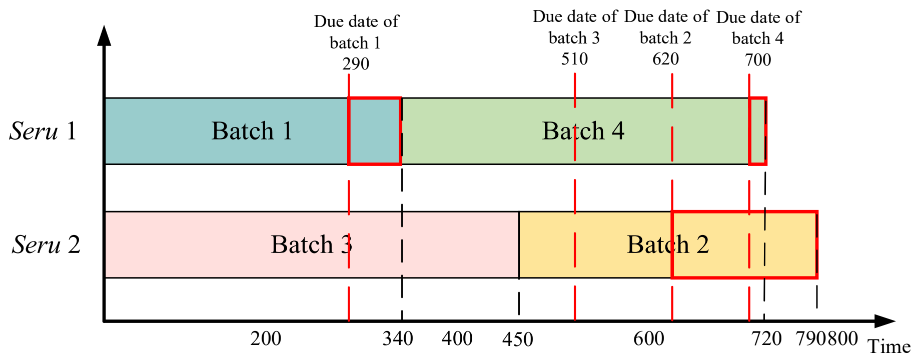


**Fig. 2.** The result of *seru* scheduling

*Seru* production problem contains two NP-hard subproblems (Sun et al., 2020; Yılmaz, 2020a). The *seru* formation is an example of an unordered set partition. Each of the *M* batches can be assigned to any of the *J serus*, so the complexity of the *seru* scheduling is $J^M$. Therefore, the complexity of the *seru* system, which contains the *seru* formation and the *seru* scheduling to minimize the maximum tardiness, is such as Eq. (1) as shown.

$$\sum_{j=1}^{J} P(Z,J)\times J^{M} \tag{1}$$

where *P (Z, J)* is the count of solutions of *Z* workers assigned to *J serus*.

## 2.2. Assumptions

We assume the following assumptions based on the literature (Kaku et al., 2009; Liu et al., 2021b; Yu et al., 2013) to model the problem explicitly.

1. The batches and types of products are given in advance.
2. The tasks required for each product are the same. Skip a task if it is not required for a product type.
3. Each worker can complete all tasks in a *seru*, different from the assembly line (each worker operates only one task).
4. The assembly tasks within each *seru* are equivalent to the ones within the assembly line. The number of tasks is *Z*.
5. A batch can only be processed in a *seru*.

*2.3. Non-linear Model*

**Object function:**

$$\min \max_{m=1}^{M}(0, FCB_m + FC_m - d_m) \tag{2}$$

subject to

$$1 \le \sum_{i=1}^{Z} L_{ij} \le Z, \forall j \tag{3}$$

$$\sum_{j=1}^{J} L_{ij} = 1, \forall i \tag{4}$$

$$\sum_{j=1}^{J}\sum_{k=1}^{M} Y_{mjk} = 1, \forall m \tag{5}$$

Eq. (2) states the objective of minimizing the maximum tardiness. Eq. (3) guarantees the number of workers in each *seru*. Eq. (4) guarantees each worker only in one *seru*. Eq. (5) guarantees that each product batch is only processed in one *seru*. Eq. (3) and Eq. (4) are the constraints related to *seru* formation, and Eq. (5) is the constraint related to *seru* scheduling. The meaning of notations for the above-stated model are present as follow. These can be found in the literatures(Fu et al., 2022; Sun et al., 2020, 2019; Yu et al., 2014).

| **Indices** | |
|---|---|
| *i* | Index of workers (*i* = 1, 2, . . ., *Z*). |
| *j* | Index of *serus* (*j* = 1, 2, . . ., *J*). |
| *n* | Index of product types (*n* = 1, 2, . . ., *N*). |
| *m* | Index of product batches (*m* = 1, 2, . . ., *M*). |
| *k* | Index of the order of product batches in a *seru* (*k* = 1, 2, . . ., *M*). |

*Decision variables*

$$L_{ij} = \begin{cases} 1, \text{if worker } i \text{ in the seru } j \\ 0, \text{otherwise} \end{cases}.$$

$$\mathrm{Y}_{mjk} = \begin{cases} 1, \text{if product batch } m \text{ is processed in seru } j \text{ in sequence } k \\ 0, \text{otherwise} \end{cases}.$$

*Variable*

*CZ$_i$* : Coefficient of variation of worker *i*'s expanded task time (Sun et al., 2020; Yu et al., 2014).

$$CZ_i = \begin{cases} 1+\varphi_i\left(Z-\eta_i\right), Z > \eta_i \\ 1, \qquad\qquad Z \le \eta_i \end{cases}, \forall i \tag{6}$$

where, $\eta_i$ is the upper limit on the number of tasks of worker *i* in a *seru*, and $\varphi_i$ is the coefficient of influencing level for worker *i* completing multiple tasks within a *seru.*

*TC$_m$*: Task time of the batch *m* per task in a *seru.*

$$TC_m = \frac{\sum_{n=1}^{N}\sum_{i=1}^{Z}\sum_{j=1}^{J}\sum_{k=1}^{M} V_{mn}T_n\beta_{ni}CZ_iL_{ij}Y_{mjk}}{\sum_{i=1}^{Z}\sum_{j=1}^{J}\sum_{k=1}^{M} L_{ij}Y_{mjk}} \tag{7}$$

where, $V_{mn}$ is 1, if the production type of batch $m$ is $n$; 0, otherwise. $T_n$ is the cycle time of product type $n$ in the original assembly line. $\beta n_i$ is the skill level of worker $i$ for each task of product type $n$.

$FC_m$: processing time of batch $m$ in a *seru.*

$$FC_m = \frac{B_m TC_m Z}{\sum_{i=1}^{Z}\sum_{j=1}^{J}\sum_{k=1}^{M} L_{ij} Y_{mjk}} \tag{8}$$

where, $B_m$ is the size of batch $m$.

$FCB_m$: starting time of product batch $m$ in a *seru.*

$$FCB_m = \sum_{s=1}^{M}\sum_{j=1}^{J}\sum_{k=1}^{M}\sum_{k'=0}^{k-1} FC_s Y_{mjk} Y_{sjk} \tag{9}$$

## *2.4. Exact solution*

As shown in Fig. 3, the nonlinear *seru* production model with minimizing the maximum tardiness is split into a *seru* formation model and a linear *seru* scheduling model.

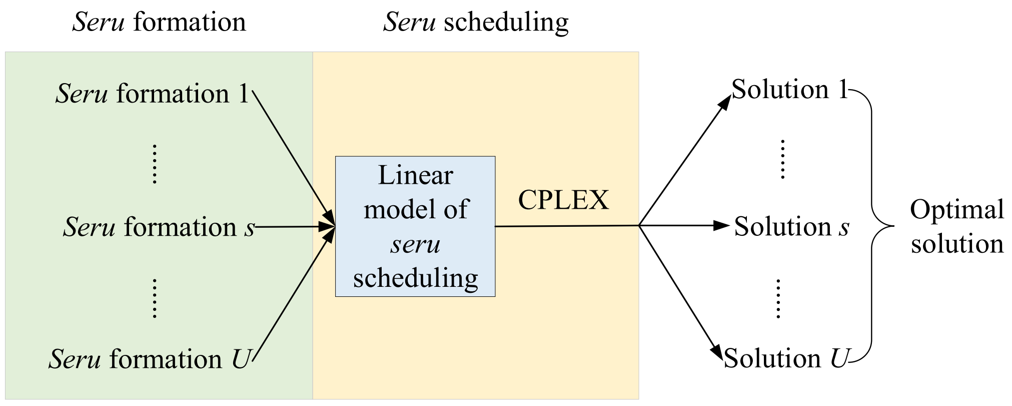


**Fig. 3.** The process of solving the exact solution

For *seru* formation, we use an unordered set partition model to exhaust each *seru* formation solution. Eq.1 shows that there are $F(Z)$ different *seru* formations. For *seru* scheduling, the linear model is proposed as follows:

*Parameter:*

$F_{mj}$: Processing time for batch $m$ in *seru* $j$.
$E$: A very large actual number

*Decision variables:*

$C_{jk}$: Completion time of the $k^{th}$ batch in *seru* $j$.
$MT$: The maximum tardiness
Therefore, the *seru* scheduling model is as follows.

*Objective function:*

$$\min MT \tag{10}$$

subject to

$$MT \geq c_{jk} - d_m - \left(1 - Y_{mjk}\right)E,\ \forall j = 1,2,...,J, m = 1,2,...,M,\ \ k = 1,...,M \tag{11}$$

$$\sum_{j=1}^{J}\sum_{k=1}^{M} Y_{mjk} = 1,\ \forall m = 1,2,...,M \tag{12}$$

$$\sum_{m=1}^{M} Y_{mjk} \leq 1, \ \forall j = 1,2,\ldots,J, k = 1,2,\ldots,M \tag{13}$$

$$c_{j1} = \sum_{m=1}^{M} Y_{mj1} F_{mj}, \ \forall j = 1,2,\ldots,J \tag{14}$$

$$c_{jk} \geq c_{j(k-1)} + \sum_{m=1}^{M} Y_{mjk} F_{mj}, \ \forall j = 1,2,\ldots,J, k = 2,\ldots,M \tag{15}$$

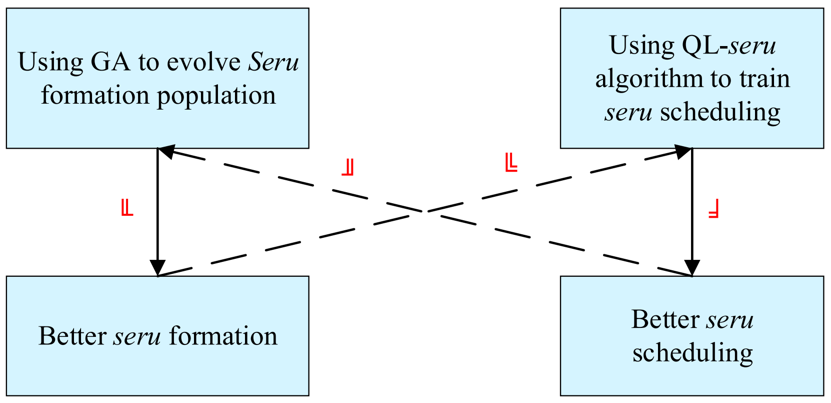
$$MT \geq 0 \tag{16}$$

Eq. (10) is the objective of minimizing the maximum tardiness. Eq. (11) indicates that the maximum tardiness cannot be less than the tardiness of each batch. Eq. (12) suggests that one batch can only be processed in one *seru*. Eq. (13) indicates that a *seru* can only process a maximum of one batch simultaneously. Eq. (14) gives the completion time of the first batch processed in each *seru*. Eq. (15) indicates that the $k^{th}$ batch cannot be processed until the $(k\text{-}1)^{th}$ batch is complete. Eq. (16) suggests that maximum tardiness is non-negative. For large-scale problems, the model cannot be solved by CPLEX. So, a cooperative algorithm using a GA and an innovative reinforcement learning algorithm (CAGARL) is proposed.

## 3. Cooperative algorithm using a GA and an innovative reinforcement learning algorithm

### *3.1. Cooperative mechanism*

It is a very effective cooperative algorithm for dealing with the enormous scope of problems with complex decisions (Ren et al., 2019; Shang et al., 2014; Sun et al., 2020; Tang, 2017). *Seru* production consists of two decision processes: *seru* formation and *seru* scheduling. Therefore, cooperative algorithms are utilized to solve the *seru* production problem. GA is used to solve the *seru* formation problem. Most of the previous algorithms for the *seru* scheduling problem are meta-heuristic algorithms, which are fast (Tang et al., 2018), but their search patterns are relatively fixed and rigid (Ni et al., 2021). However, reinforcement learning provides a more purposeful search of the solution space by learning from previous experience. Therefore, we design an innovative reinforcement learning algorithm (QL-*seru* algorithm) by combining the features of the meta-heuristic algorithm and the reinforcement learning algorithm to solve the *seru* scheduling problem. Moreover, GA and QL-*seru* algorithm collaborate. The cooperative mechanism of the QL-*seru* algorithm and GA is shown in Fig. 4.


Using GA to evolve *Seru* formation population
Using QL-*seru* algorithm to train *seru* scheduling
╜
╚
╙
╛
Better *seru* formation
Better *seru* scheduling

╙ : Obtain better *seru* formation;
╚ : Provide current better *seru* formation as the environment in QL-*seru* algorithm;
╛ : Obtain better *seru* scheduling;
╜ : Provide current better *seru s*cheduling to assist GA to evolve *seru* formation population;

**Fig. 4.** Cooperative mechanism of QL-*seru* algorithm and GA

### *3.2. GA for seru formation*

The *seru* formation is solved by the GA combining local search to obtain a better solution(Berahhou et al., 2022).

#### *3.2.1. Solution expression of seru formation*

In order to represent the *seru* formation, the sequence encoding means proposed by Yu et al. (2012) are used. Suppose there are *Z* workers, the solution can be shown by a vector that contains *Z* workers and *Z*-1 separators, and elements with numbers greater than *Z* indicate separators. Therefore, *Z*-1 separators can split up to *Z serus* at most.

### *3.2.2. Selection, crossover, mutation and neighbor strategy*

Selection strategy: adopts the binary tournament selection (Beyer and Deb, 2001).
Mutation operation: two gene interchanges.
Crossover operation: the order crossover (Davis, 1985).
Neighbor strategy: exchange two unique elements (Sun et al., 2019).

## *3.3. QL-seru algorithm for seru scheduling*

The *seru* scheduling is solved by the QL-*seru* algorithm combining local search.

### *3.3.1. QL-seru algorithm*

We propose an innovative QL-*seru* algorithm (QLSA) for the *seru* scheduling problem by combining the meta-heuristic and reinforcement learning algorithm features. We set the states in reinforcement learning in the way encoded in the meta-heuristic algorithm, and each state represents a *seru* scheduling. The new states are generated using the exploration and development approach in reinforcement learning. Given that the generated *seru* formation is frequently the same, especially in the convergence phase or when the number of iterations is relatively large (the number of repetitions is shown in Table 8), repeated training of the same formation is meaningless. It consumes a significant amount of computational time. Consequently, we store historical data (QL-*seru* table) to avoid repeated training, i.e., the stored states will not be trained. In addition, for a new state, if the state is worse than the stored average objective, then the state will be discarded. In the QLSA, at each time step $t$, $a_t$ is the current action, and the $s_t$ is the current state. $r_{t+1}$ is the reward. The q-value update function is expressed as Eq. (17).

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a' \in A} Q(s', a')) \tag{17}$$

In Eq. (17), $\alpha$ is the learning rate, $\gamma$ is the discount rate.

### *3.3.2. State definition*

In order to combine the features of the reinforcement learning algorithm and the meta-heuristic algorithm, the state is designed as the code of *seru* scheduling under the current best *seru* formation. We set the state to a vector of $M$+$Z$-1 dimension, one vector for each state. In the vectors, numbers less than or equal to $M$ represent batches, and numbers greater than $M$ represent separators. The state is shown in Fig. 5.

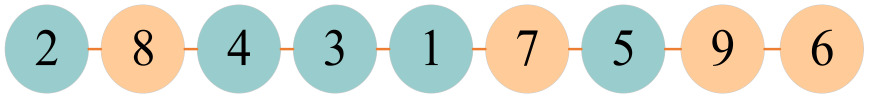


**Fig. 5.** An example of a state

As shown in Fig. 5, this state represents a *seru* system with 4 workers and 5 batches. This state indicates that 5 batches are divided into 3 groups. Then assign each batch group to each *seru* in turn, and if the number of *seru* is less than the number of batch groups, process the remaining batch groups on the finished *serus*. The initial state $s_0$ is the state encoding for the best scheduling of the previous round of cooperatives.

### *3.3.3. Action definition*

Since *seru* scheduling is NP-hard, to reduce the action space, we set the action to move the elements in the scheduling encoding left and right.

There are 2*($M$+$Z$-1)-2 actions in the action space. Select an element from the state-coded $M$-$Z$-1 element and move left or right. For the first element, you can only move to the right. For the last element, you can only move to the left. So, there are a total of 2*($M$-$Z$-1)-2.

For example, if the current state is Fig. 5, we take action '2-1', i.e., the element moves to the right. When one of these actions is taken, a new state code is obtained, and the new state is reached. After taking action '2-1', get the state encoding as shown in Fig. 6. This state indicates that 5 batches are divided into 3 groups. Group 1 includes batch 2 and batch 4. Group 2 includes batch 3 and batch 1. Group 3 includes batch 5.

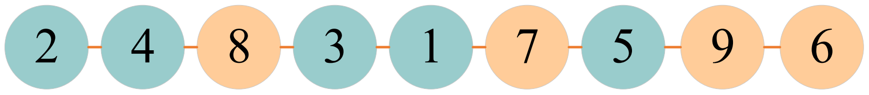


**Fig. 6.** New state after taking action '2-1'

### *3.3.4. Reward method*

To obtain solutions of *seru* production more intuitively. We set the maximum tardiness that is calculated by the current state (*seru* scheduling) and the current environment (*seru* formation) as the reward, as shown in Eq. (18). The agent is not directed on what to perform. Instead, it tries to find out which actions will produce higher returns, which can clearly yield positive returns (Chen et al., 2020). The best reward corresponds to a better solution for *seru* scheduling.

$$r_t = -f(s_t) \tag{18}$$

where $f(s_t)$is the maximum tardiness obtained by the current state.

### *3.3.5. Action selection strategy*

In order to balance development and exploration, the $\varepsilon$-greedy strategy is used to select the action. $g_0$ is a random number between 0 and 1, and $\varepsilon$ is the greedy rate. When $\varepsilon > g_0$, the action with the maximum q-value is taken, on the contrary, randomly, as shown in Eq. (19)(Chen et al., 2020; Fu et al., 2022).

$$\pi(s_t, a_t) = \begin{cases} \max\limits_{a} Q(s_t, a)\ , & \varepsilon > g_0 \\ a_{rand}\ , & \varepsilon \le g_0 \end{cases} \tag{19}$$

where $\pi(s_t, a_t)$ is a select policy for the action at state $s_t$.

### *3.3.6. QL-seru table*

After obtaining the current better *seru* formation by GA, we use the current better *seru* formation as the environment for the reinforcement learning algorithm when solving the *seru* scheduling. Based on the feature that the *seru* formation will be repeatedly generated, an innovative QL-*seru* table for saving computational time using historical data is proposed.

In the QL-*seru* table, we store the last *p seru* formations, the trained states of the *p* formations, and the average objective of the trained states of each *p* formations. If the newly produced *seru* formation (*nsf*) is in the stored *p* formations, we use the following two methods to avoid repeated or unnecessary training. The two state trimming rules are as follows:

**State trimming rules 1 (Repeat state skipping method)**: The states will not be trained for the stored states of *nsf*.
**State trimming rules 2 (Poor state rejection method)**: For the newly obtained states of *nsf*, if the objective corresponding to the current state is worse than the stored average objective, the state will not be trained.

After training the current *seru* formation as the environment, if the current *seru* formation is in the QL-*seru* table, we will update the trained state of the current *seru* formation as the environment. If not, we will add the *seru* formation, the trained states of the formation, and the corresponding average objective into the QL-*seru* table.

### *3.3.7. Procedure of the QLSA*

The flowchart of the QL-*seru* algorithm is shown in Fig. 7. Situation 1 represents that the current *seru* formation does not exist in the QL-*seru* table, and situation 2 represents that the current *seru* formation exists in the QL-*seru* table.

We give an example of situation 2 in the QL-*seru* algorithm.
Suppose that there are 3 workers, 2 batches, and the current best formation is {{1,3},{2}}.
Step 1. Current state $s_t$ = {3,2,4,1} (Batch 2 is processed in *seru* 1, and batch 1 is assigned to *seru* 2.).
Step 2. Choose the action at='3-1' (The third element of the state code is shifted right.).
Step 3. Obtain the next state $s_{t+1}$= {3,2,1,4} (Batches 2 and 1 are processed in *seru* 1 in order) and the reward (maximum tardiness).
Step 4. Determines if state $s_{t+1}$ meets the state trimming rules. If yes, apply the trimming rules and proceed to the next round of training. Otherwise, update the Q table and current better *seru* scheduling, then proceed to the next training round.

The execution flow of the entire QLSA is described in Algorithm 1.

**Algorithm 1: QLSA**

```
Input: Max_Epochs(maximum number of iterations), BF (current best formation), QL-seru table.
Output: BS (best scheduling).
(1) Initialize.
   q-table, state, action, nest state, nest action (correspond to Q-table, current state, current action, next state and next
action in reinforcement learning, respectively).
  q-table, state_0
  Q_marker (a flag to determine whether the current formation is stored in the QL-seru table)
  AO (average objective) ← 0
  best_reward (The best reward under the current formation) ←The initial value is calculated by the initial state and the
current formation.
(2) if (BF in QL-seru table) then
      Read the QL-seru table
      Q_marker ← 1
      AO ← Read the average objective of the current best formation, which is stored in the QL-seru table
  end if
(3) while (i< Max_Epochs) do
  (3-1) Choose the action ($a_t$).
  (3-2) Obtain the next state ($s_{t+1}$).
  (3-3) if (Q_marker==1) then
          if ($s_{t+1}$ in QL-seru table) then
            continue
          end if
     end if
  (3-4) Get the reward ← the current maximum tardiness (seru formation: BF; seru scheduling: $s_{t+1}$).
  (3-5) if (Q_marker==1) then
          if (reward >AO) then
            continue
          end if
     end if
  (3-4) Find the maximum q-value for $a_t$ taken in the $s_{t+1}$.
  (3-5) Update the q-table by the bellman equation.
  (3-6) if (reward< best_reward) then
     best_reward ← reward
     best_scheduling ← $s_{t+1}$
     end if
  (3-7) $s_t$ ← $s_{t+1}$
  end while
(4) Update the QL-seru table.
(5) Local search to find out if there is better scheduling.
Output BS.
```

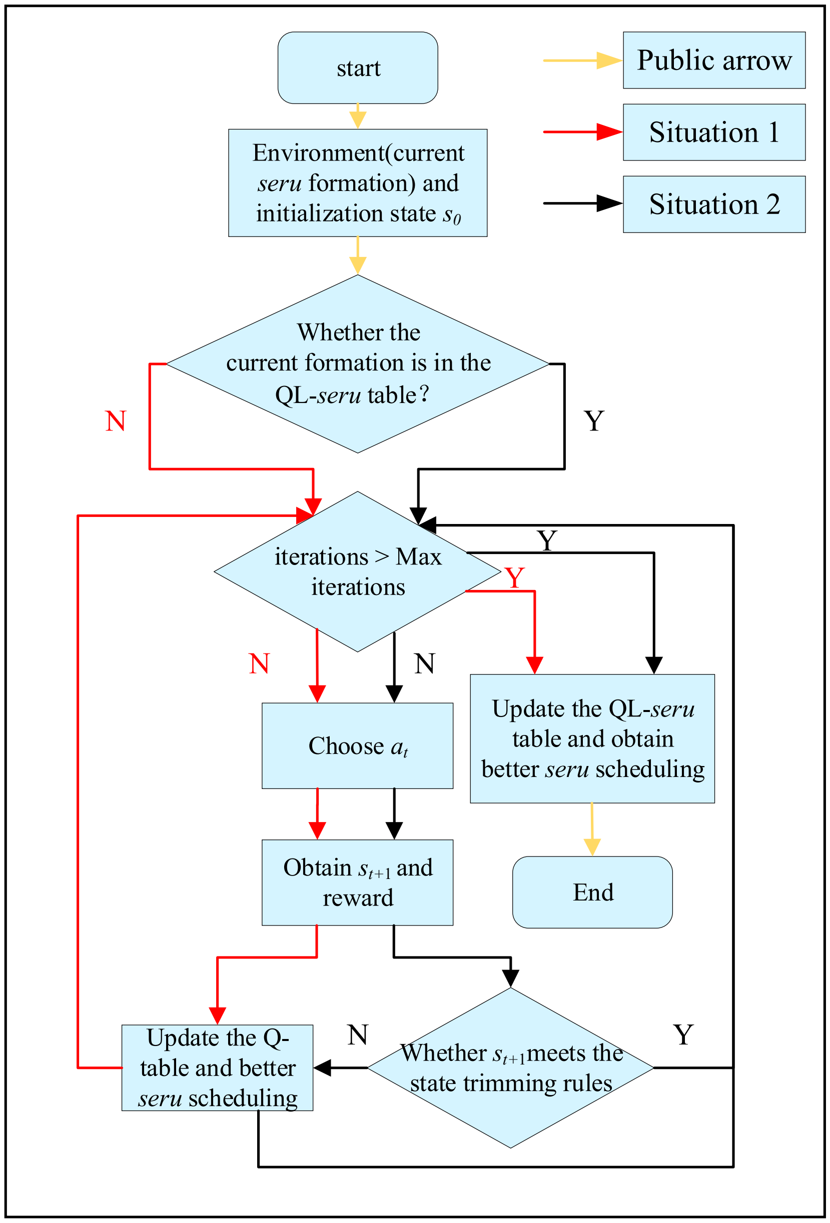

start
Public arrow
Situation 1
Situation 2
Environment(current seru formation) and initialization state $s_0$
Whether the current formation is in the QL-seru table?
N
Y
iterations > Max iterations
Y
Y
N
N
Choose $a_t$
Update the QL-seru table and obtain better seru scheduling
Obtain $s_{t+1}$ and reward
End
Update the Q-table and better seru scheduling
N
Whether $s_{t+1}$ meets the state trimming rules
Y

**Fig. 7.** Flowchart of QL-*seru* algorithm

## 4. Experimental results

All experiments were implemented on a personal computer (Intel Core (TM) i7-8700 processor at 3.20 GHz, Windows 10, and 8.0 GB of RAM). CAGARL was written in *C*#.

### 4.1. Test data

There are five different product types. $T_n$=1.8, $\eta_i$=20. The data for $\varepsilon_i$, $\beta_{ni}$, and batches are as follow.

**Table 1**
Coefficient of influencing level for worker *i* completing multiple tasks within a *seru* ($\varphi_i$)

| worker | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\varepsilon_i$ | 0.18 | 0.19 | 0.2 | 0.21 | 0.2 | 0.2 | 0.2 | 0.22 | 0.19 | 0.19 |
| worker | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| $\varepsilon_i$ | 0.18 | 0.23 | 0.24 | 0.22 | 0.16 | 0.24 | 0.18 | 0.18 | 0.21 | 0.18 |

**Table 2**
The data of worker's level of skill ($\beta_{ni}$)

| Product/Worker | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.92 | 0.95 | 0.99 | 1.13 | 0.96 | 1.21 | 1.04 | 0.98 | 0.97 | 0.92 |
| 2 | 0.96 | 0.97 | 1.01 | 1.27 | 1.22 | 1.1 | 1.07 | 1.02 | 1.03 | 0.96 |
| 3 | 1.24 | 1.09 | 1.15 | 0.92 | 0.91 | 1.01 | 1.24 | 1.1 | 1.12 | 1.24 |
| 4 | 1.09 | 1.12 | 1.09 | 1.12 | 1.1 | 1.15 | 1.07 | 1.11 | 1.19 | 1.09 |
| 5 | 1.2 | 1.18 | 1.21 | 1.25 | 1.18 | 1.23 | 1.14 | 1.2 | 1.26 | 1.2 |
| Product/Worker | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 1 | 0.95 | 0.98 | 0.99 | 1.01 | 1.04 | 0.99 | 1.04 | 0.93 | 0.96 | 1.08 |
| 2 | 1.04 | 1.07 | 0.95 | 1.1 | 1.1 | 0.97 | 1.01 | 1.06 | 0.98 | 1.04 |
| 3 | 1.03 | 1.07 | 1.11 | 1.05 | 1.05 | 1.08 | 1.11 | 1.07 | 1.12 | 1.09 |
| 4 | 1.14 | 1.15 | 1.17 | 1.13 | 1.15 | 1.11 | 1.15 | 1.13 | 1.14 | 1.11 |
| 5 | 1.19 | 1.15 | 1.1 | 1.18 | 1.11 | 1.22 | 1.24 | 1.14 | 1.21 | 1.13 |

**Table 3**
The data of Batches

| *Batch number* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| *Product type* | 3 | 5 | 3 | 4 | 1 | 4 | 1 | 2 | 2 |
| *Batch size* ($B_m$) | 55 | 53 | 54 | 49 | 49 | 55 | 54 | 48 | 48 |
| *Due date* | 184 | 228 | 366 | 422 | 588 | 551 | 807 | 780 | 973 |
| *Batch number* | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| *Product type* | 3 | 2 | 4 | 3 | 4 | 5 | 5 | 1 | 4 |
| *Batch size* ($B_m$) | 48 | 46 | 58 | 48 | 52 | 48 | 51 | 54 | 57 |
| *Due date* | 1,078 | 1,100 | 1,294 | 1,392 | 1,400 | 1,599 | 1,601 | 1,710 | 1,802 |
| *Batch number* | 19 | 20 | 21 | 22 | 23 | 24 | 25 | | |
| *Product type* | 2 | 5 | 1 | 3 | 4 | 5 | 2 | | |
| *Batch size* ($B_m$) | 54 | 49 | 53 | 46 | 45 | 46 | 45 | | |
| *Due date* | 1,998 | 2,107 | 2,199 | 2,209 | 2,311 | 2,410 | 2,510 | | |

### *4.2. Parameter settings*

The number of iterations of the cooperative algorithm is 100. For GA, the population size is 200, the crossover rate is 0.9, and the mutation rate is 0.1, respectively. And for the QLSA, $\alpha$=0.01, $\gamma$=0.9, ε=0.1, Max_Epochs=35000. Moreover, the number of search neighbors in the local search is 75. For the comparison algorithm (Sun et al. (2019)), we compare the results at the same running time.

### *4.3. Performances of the CAGARL*

A more significant number of experiments have been conducted to evaluate CAGARL. In addition, *ASMT* is defined as evaluating how much the maximum tardiness has been reduced by the *seru* production in comparison to the assembly line, as shown in Eq. (20). To evaluate the algorithm, we use the cooperative coevolution algorithm proposed by Sun et al. (2019) to compare. The gap between the CAGARL and the exact or comparative algorithm is referred to as Eq. (21) and Eq. (22).

$$ASMT=\frac{MT\_Line - MT\_CAGARL}{MT\_Line} \tag{20}$$

$$GAP1=\frac{MT\_CAGARL - MT\_Exact}{MT\_Exact} \tag{21}$$

$$GAP2=\frac{MT\_Sun - MT\_CAGARL}{MT\_CAGARL} \tag{22}$$

*MT_Line*: the maximum tardiness of the assembly line. *MT_CAGARL*: the maximum of the *seru* production system solved by CAGARL. *MT_Sun*: the maximum of the *seru* production system solved by the algorithm proposed by Sun. *MT_Exact*: the maximum of the *seru* production system solved by exact solution.

To ensure the fairness of the experiments, we use the same mathematical model and parameters in the comparison experiments. CAGARL and the comparison algorithm use the same running time to compare. The solution of the maximum tardiness for the assembly line (The formula for calculating the maximum tardiness of the assembly line is shown in Eqs. (23-26). To minimize the maximum tardiness of the assembly line, EDD rule is used to process the batches.), the solution of minimizing the maximum tardiness of the *seru* system solved by the exact algorithm, CAGARL and sun's algorithm are shown in Table 4. Table 5 shows the values of *GAP*1, *GAP*2, and *ASMT*. Table 6 shows the running time of the exact algorithm and CAGARL (sun's algorithm with the same time as CAGARL).

$$TL_m=\max_{1\le i\le Z}(V_{mn}T_n\beta_{ni}),\forall i \tag{23}$$

$$FL_m = \sum_{n=1}^{N}\sum_{i=1}^{Z} V_{mn} T_n \beta_{ni} + (B_m - 1) TL_m \tag{24}$$

$$f_m = \sum_{i=1}^{m} FL_i \tag{25}$$

$$MT\_Line = \max_{1 \le m \le M} \{ f_m - d_m, 0 \} \tag{26}$$

$TL_m$: the task time of batch $m$; $FL_m$: the processing time of batch $m$; $f_m$: the complete time of batch $m$;

**Table 4**
Solution of the assembly line, exact algorithm, comparison algorithm, and CAGARL

| | Worker/Batch | 5 | 6 | 7 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|---|---|---|
| 5 | *MT_Line* | 70 | 141 | 141 | 156 | 260 | 337 | 424 |
| | *MT_Exact* | 0 | 25 | 25 | 25 | - | - | - |
| | *MT_Sun* | 0 | 25 | 36 | 155 | 489 | 744 | 1092 |
| | *MT_CAGARL* | 0 | 25 | 25 | 25 | 25 | 55 | 96 |
| 6 | *MT_Line* | 81 | 163 | 163 | 189 | 310 | 407 | 519 |
| | *MT_Exact* | 0 | 8 | 8 | 8 | - | - | - |
| | *MT_Sun* | 0 | 27 | 89 | 214 | 443 | 729 | 819 |
| | *MT_CAGARL* | 0 | 8 | 8 | 8 | 27 | 27 | 121 |
| 8 | *MT_Line* | 98 | 187 | 187 | 221 | 366 | 479 | 618 |
| | *MT_Exact* | 2 | 29 | - | - | | - | - |
| | *MT_Sun* | 2 | 34 | 58 | 189 | 428 | 705 | 1099 |
| | *MT_CAGARL* | 2 | 29 | 29 | 29 | 29 | 34 | 209 |
| 10 | *MT_Line* | 120 | 217 | 217 | 258 | 435 | 574 | 750 |
| | *MT_Exact* | - | - | - | - | - | - | - |
| | *MT_Sun* | 3 | 33 | 59 | 149 | 325 | 731 | 1011 |
| | *MT_CAGARL* | 3 | 22 | 33 | 33 | 33 | 33 | 211 |
| 15 | *MT_Line* | 160 | 275 | 275 | 335 | 570 | 749 | 994 |
| | *MT_Exact* | - | - | - | - | - | - | - |
| | *MT_Sun* | 1 | 30 | 30 | 221 | 376 | 700 | 1081 |
| | *MT_CAGARL* | 1 | 30 | 30 | 30 | 30 | 108 | 67 |
| 20 | *MT_Line* | 200 | 334 | 334 | 431 | 706 | 933 | 1239 |
| | *MT_Exact* | - | - | - | - | - | - | - |
| | *MT_Sun* | 2 | 36 | 58 | 183 | 424 | 1128 | 1496 |
| | *MT_CAGARL* | 2 | 31 | 58 | 126 | 197 | 208 | 385 |

**Table 5**
Performance comparison

| | Worker/Batch | 5 | 6 | 7 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|---|---|---|
| 5 | *GAP*1 | 0 | 0 | 0 | - | - | - | - |
| | *GAP*2 | 0 | 0 | 0.31 | 0.84 | 0.95 | 0.93 | 0.91 |
| | *ASMT* | 1.00 | 0.82 | 0.82 | 0.84 | 0.90 | 0.84 | 0.77 |
| 6 | *GAP*1 | 0 | 0 | 0 | - | - | - | - |
| | *GAP*2 | 0 | 0.70 | 0.91 | 0.96 | 0.94 | 0.96 | 0.85 |
| | *ASMT* | 1.00 | 0.95 | 0.95 | 0.96 | 0.91 | 0.93 | 0.77 |
| 8 | *GAP*1 | 0 | 0 | 0 | - | - | - | - |
| | *GAP*2 | 0 | 0.15 | 0.50 | 0.85 | 0.93 | 0.95 | 0.81 |
| | *ASMT* | 0.98 | 0.84 | 0.84 | 0.87 | 0.92 | 0.93 | 0.66 |
| 10 | *GAP*1 | 0 | 0 | 0 | - | - | - | - |
| | *GAP*2 | 0 | 0.33 | 0.44 | 0.78 | 0.90 | 0.95 | 0.79 |
| | *ASMT* | 0.98 | 0.90 | 0.85 | 0.87 | 0.92 | 0.94 | 0.72 |
| 15 | *GAP*1 | - | - | - | - | - | - | - |
| | *GAP*2 | 0 | 0 | 0 | 0.86 | 0.92 | 0.85 | 0.94 |
| | *ASMT* | 0.99 | 0.89 | 0.89 | 0.91 | 0.95 | 0.86 | 0.93 |
| 20 | *GAP*1 | - | - | - | - | - | - | - |
| | *GAP*2 | 0 | 0.14 | 0 | 0.31 | 0.54 | 0.82 | 0.74 |
| | *ASMT* | 0.99 | 0.91 | 0.83 | 0.71 | 0.72 | 0.78 | 0.69 |

As seen from Table 4 and Table 5, implementing the *seru* production system can significantly reduce the maximum tardiness. Compared to the exact algorithm, CAGARL can solve the problem on a larger scale. CAGARL can obtain the same solution as the exact algorithm. For large-scale problems, CAGARL can obtain better solutions than SUN'S.

**Table 6**
Running time of the exact algorithm and CAGARL

| Worker/Batch | | 5 | 6 | 7 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|---|---|---|
| 5 | EXACT | 13 | 16 | 20 | 321 | - | - | - |
| | CAGARL | 28 | 40 | 45 | 61 | 88 | 116 | 143 |
| 6 | EXACT | 47 | 58 | 74 | 497 | - | - | - |
| | CAGARL | 20 | 40 | 46 | 61 | 91 | 126 | 156 |
| 8 | EXACT | 1033 | 1435 | - | - | - | - | - |
| | CAGARL | 41 | 46 | 52 | 66 | 95 | 144 | 177 |
| 10 | EXACT | - | - | - | - | - | - | - |
| | CAGARL | 48 | 56 | 64 | 87 | 108 | 164 | 207 |
| 15 | EXACT | - | - | - | - | - | - | - |
| | CAGARL | 36 | 71 | 72 | 109 | 158 | 203 | 245 |
| 20 | EXACT | - | - | - | - | - | - | - |
| | CAGARL | 81 | 89 | 83 | 122 | 147 | 195 | 204 |

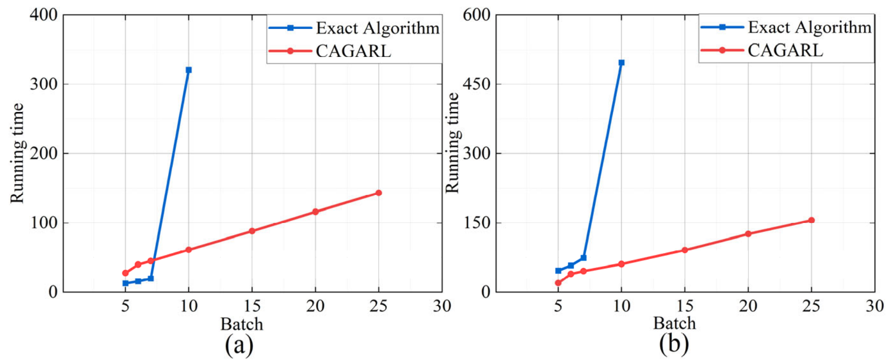


**Fig. 8.** The running time of the exact algorithm and CAGARL at instances with 5 workers (a) and 6 workers (b)

As can be seen from Table 6 and Fig. 8, the exact algorithm only can solve small-scale problems. Compared to the exact algorithm, as the size becomes more extensive, the runtime variation of CAGARL is relatively smooth. The calculation time of CAGARL can meet the actual needs of production.

### *4.4. Performances of QL-seru*

Combining the characteristics of *seru* production, the QL-*seru* table is proposed to save computation time innovatively. With all parameters being the same, Table 7 shows the running time of CAGARL and the algorithm without the QL-*seru* table. In Table 7, T1 is the running time of CAGARL (with the QL-*seru* table), and T2 is the running time of the algorithm without the QL-*seru* table (Q-learning). *MRT* is the gap between T1 and T2. *MRT* is calculated as shown in Eq. (27).

$$MRT = \frac{\text{T2 - T1}}{\text{T2}} \tag{27}$$

**Table 7**
T1 and T2

| Worker/Batch | | 5 | 6 | 7 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|---|---|---|
| 5 | T1 | 28 | 40 | 45 | 61 | 88 | 116 | 143 |
| | T2 | 61 | 59 | 76 | 120 | 159 | 177 | 207 |
| | *MRT* | 0.54 | 0.32 | 0.41 | 0.49 | 0.45 | 0.34 | 0.31 |
| 6 | T1 | 20 | 40 | 46 | 61 | 91 | 126 | 156 |
| | T2 | 40 | 49 | 82 | 127 | 153 | 180 | 240 |
| | *MRT* | 0.50 | 0.18 | 0.44 | 0.52 | 0.41 | 0.30 | 0.35 |
| 8 | T1 | 41 | 46 | 52 | 66 | 95 | 144 | 177 |
| | T2 | 39 | 55 | 87 | 130 | 184 | 192 | 228 |
| | *MRT* | -0.05 | 0.16 | 0.40 | 0.49 | 0.48 | 0.25 | 0.22 |
| 10 | T1 | 48 | 56 | 64 | 87 | 108 | 164 | 207 |
| | T2 | 89 | 78 | 104 | 135 | 194 | 206 | 253 |
| | *MRT* | 0.46 | 0.28 | 0.38 | 0.36 | 0.44 | 0.20 | 0.18 |
| 15 | T1 | 36 | 71 | 72 | 109 | 158 | 203 | 245 |
| | T2 | 59 | 86 | 114 | 145 | 174 | 213 | 239 |
| | *MRT* | 0.39 | 0.17 | 0.37 | 0.25 | 0.09 | 0.05 | -0.03 |
| 20 | T1 | 81 | 89 | 83 | 122 | 147 | 195 | 204 |
| | T2 | 88 | 112 | 134 | 153 | 185 | 217 | 247 |
| | *MRT* | 0.08 | 0.21 | 0.38 | 0.20 | 0.21 | 0.10 | 0.17 |

As shown in Table 7, T1 is almost always smaller than T2, indicating that our innovative QLSA can significantly reduce the computation time. Table 8 shows the number of generated duplicate environments when the cooperative algorithm is iterated 100 times, i.e., when the QLSA algorithm is used to update the *seru* scheduling 50 times.

**Table 8**
The number of repetitive environments generated

| Worker/Batch | 5 | 6 | 7 | 10 | 15 | 20 | 25 |
|---|---|---|---|---|---|---|---|
| 5 | 49 | 50 | 50 | 50 | 50 | 50 | 50 |
| 6 | 48 | 49 | 47 | 47 | 41 | 49 | 48 |
| 8 | 48 | 47 | 45 | 48 | 45 | 49 | 49 |
| 10 | 45 | 46 | 48 | 48 | 46 | 47 | 47 |
| 15 | 44 | 37 | 46 | 43 | 40 | 32 | 39 |
| 20 | 34 | 26 | 27 | 24 | 22 | 17 | 22 |

Table 8 shows that when using QLSA update *seru* scheduling 50 times, the environment repeat generation may even reach 50 times when the number of workers is small.

*4.5. Benefits of minimizing the maximum tardiness*

The lower bound of delay and the worst-case satisfaction can be improved by minimizing the maximum tardiness. We compare optimizing the maximum tardiness with the result of optimizing the makespan (Using the EDD rule again in the same *seru* to reduce the maximum tardiness without changing the makespan). Then the total tardiness, the number of tardy batches, makespan, and maximum tardiness under two optimization objectives are calculated and compared, shown in Table 9 and Fig. 9.

**Table 9**
Performance comparisons of the *seru* system when maximum tardiness and makespan are the objectives for the instances with 6 product batches

| *M* | *Z* | *O_MT* | | | | *O_C*$_{max}$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | *TT* | *NDB* | $C_{max}$ | *MT* | *TT* | *NDB* | $C_{max}$ | *MT* |
| | 5 | 25 | 1 | 613 | 25 | 365 | 2 | 609 | 344 |
| | 6 | 19 | 4 | 591 | 8 | 557 | 5 | 588 | 165 |
| | 8 | 64 | 4 | 592 | 29 | 79 | 4 | 590 | 38 |
| 6 | 10 | 59 | 6 | 598 | 22 | 55 | 4 | 592 | 25 |
| | 15 | 31 | 2 | 617 | 30 | 453 | 5 | 598 | 175 |
| | 20 | 33 | 2 | 619 | 31 | 686 | 5 | 618 | 34 |

*O_MT*: optimize the maximum tardiness; *O_C*$_{max}$: optimize the makespan; *MT*: maximum tardiness; $C_{max}$: makespan; *TT*: total tardiness; NDB: the number of tardy batches.

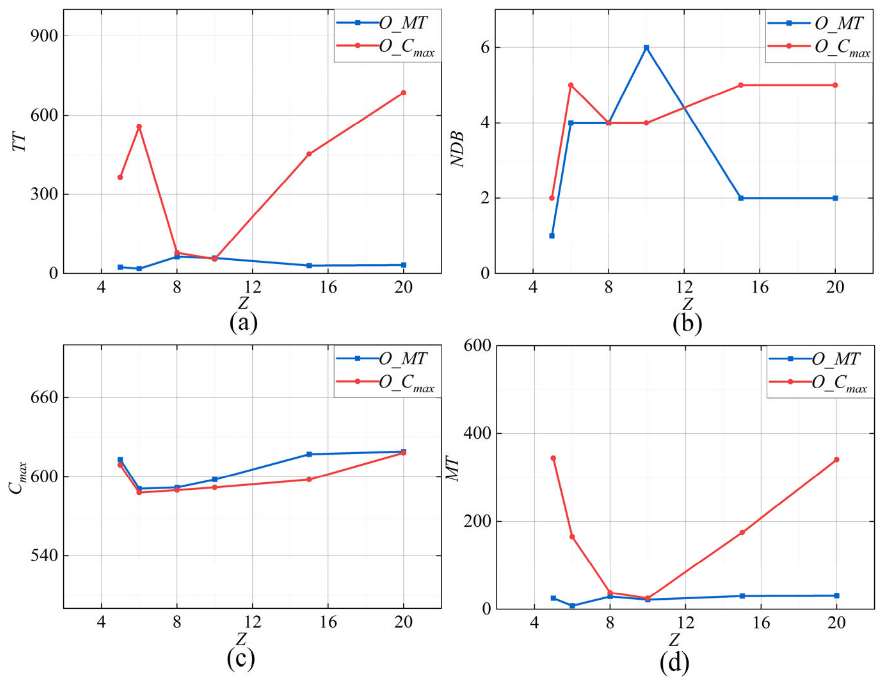


**Fig. 1.** Performance comparisons of the seru system when maximum tardiness and makespan are the objectives for the instances with 6 product batches

From Table 9 and Fig. 9. compared with the makespan, using maximum tardiness as the optimization objective for *seru* systems can significantly reduce the maximum tardiness, total tardiness, and the number of tardy batches for *seru* systems. Moreover, it is clear from Fig. 9(c) that the makespan is not much worse when optimizing the maximum tardiness than the makespan.

## *4.6. Discussion and insights*

### *4.6.1 Discussion*

(1) *Seru* production can reduce maximum tardiness by an average of 87% compared to the assembly line. As a result, *seru* production can better meet the needs of customers.

(2) CAGARL can obtain better maximum tardiness than SUN'S. The algorithm can solve large-scale problems and achieve the same optimal solution as the exact algorithm in small-scale problems.

(3) Compared to optimizing the makespan, optimizing the maximum tardiness can obtain better total tardiness, number of tardy batches, and maximum tardiness, while the makespan does not increase much.

### *4.6.2 Insights*

**Insight 1.** We should construct a system with one or two *seru* to minimize the maximum tardiness of the *seru* system.

**Table 10**
The best solution

| *M* | *Z* | Solution |
|---|---|---|
| 6 | 5 | [(1-5)]-'1-4 6 5' |
| | 6 | [(1,2,3),(6,4,5)]-'2 4 5/1 3 6' |
| | 8 | [(4,6,5,2),(3,8,7,1)]-'1 3 6/2 4 5' |
| | 10 | [(10,2,4-6),(1,3,7-9)]-'1 3 6/2 4 5' |
| | 15 | [(1-15)]-'1-4 6 5' |
| | 20 | [(1-20)]-'1-4 6 5' |
| 10 | 5 | [(1-5)]-'1-4 6 5 8 7 9 10' |
| | 6 | [(6,4,5),(3,2,1)]-'1 3 6 7 10/2 4 5 8 9' |
| | 8 | [(1,3,7,8),(2,4,5,6)]-'2 4 5 7 9/1 3 6 8 10' |
| | 10 | [(1-10)]-'1-4 6 5 8 7 10 9' |
| | 15 | [(1-15)]-'1-4 6 5 7-10' |
| | 20 | [(13,16,19,6,8),(10,11,12,14,15,18,4,5),(1,17,2,20,3,7,9)-'4 8 10/1 3 5 9/2 6 7'] |

[(1,2,3),(6,4,5)] denotes *seru* formation. '2 4 5/1 3 6' denotes *seru* scheduling.

As can be seen from Table 10, the best solutions are mostly for one or two *seru* cases. Therefore, to make the maximum tardiness smaller, we can construct one or two *seru*.

**Insight 2.** To minimize the maximum tardiness of the *seru* system, $\frac{\sum_{i=1}^{Z} x_{i1}}{\sum_{m=1}^{M}\sum_{k=1}^{M} y_{m1k}} \approx \frac{\sum_{i=1}^{Z} x_{i2}}{\sum_{m=1}^{M}\sum_{k=1}^{M} y_{m2k}} \approx \ldots \approx \frac{\sum_{i=1}^{Z} x_{iJ}}{\sum_{m=1}^{M}\sum_{k=1}^{M} y_{mJk}}$.

As shown in Table 10, for the 6 workers and 6 batches, in the solutions obtained, there are 3 workers and 3 batches in each *seru*, and the ratio of the number of workers to the number of batches in each *seru* is the same as 1. Moreover, for 8 workers and 6 batches, each *seru* has 4 workers and 3 batches in the solutions obtained. Even many cases only form a *seru*.

**Insight 3.** The maximum tardiness may remain unchanged when batch size increases.

The maximum tardiness may remain unchanged when the batch size increases. As shown in Fig. 10 and Table 4, when the number of workers is 6, the maximum tardiness for the instances with 6 product batches, 7 product batches, and 10 product batches is all obtained in batch 6. So, we can consider producing more batches, and the maximum tardiness will not change.

## 5. Conclusions

This article investigates how to minimize maximum tardiness as much as possible. The paper's contribution is as follows. Firstly, we propose a *seru* production model that minimizes the maximum tardiness. Secondly, in this paper, the non-linear *seru* production model with minimizing the maximum tardiness is split into a *seru* formation model and a linear *seru* scheduling model. Moreover, the optimal solution is obtained using an exact algorithm. Thirdly, a cooperative algorithm is proposed to deal with larger-scale problems. The GA is designed for the *seru* formation problem in the cooperative algorithm, and the QL-*seru* algorithm is used to deal with the *seru* scheduling problem. We have innovatively designed the Q-*seru* table and proposed two state trimming rules to save computational time. Finally, we conducted many experiments to demonstrate the advantages of CAGARL and discussed significantly reducing the maximum tardiness by implementing the *seru* system.

There are still some issues that need to be investigated. Firstly, consider the setup time for the batch. Secondly, consider reducing delay costs by implementing the *seru* production system in the future. Finally, the application of deep reinforcement learning in the *seru* system deserves further study.

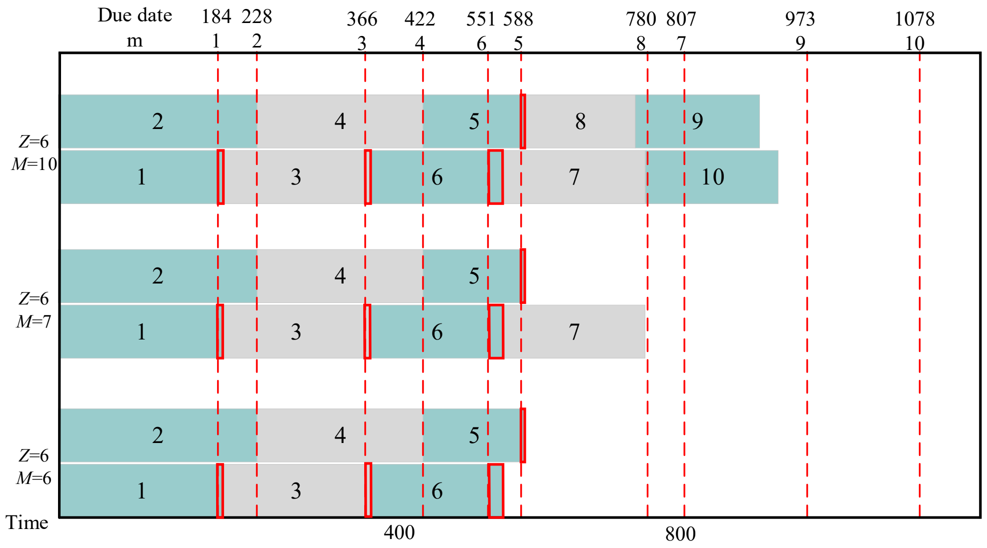


**Fig. 2.** Maximum tardiness for different batch sizes

## Acknowledgments

This research is supported by the National Natural Science Foundation of China (72171043, 71831006, and 71620107003).

## References

Allahverdi, A. (2004). A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. *Computers & Operations Research*, *31*(2), 157-180.

Arviv, K., Stern, H., & Edan, Y. (2016). Collaborative reinforcement learning for a two-robot job transfer flow-shop scheduling problem. *International Journal of Production Research*, *54*(4), 1196-1209.

Assarzadegan, P., & Rasti-Barzoki, M. (2016). Minimizing sum of the due date assignment costs, maximum tardiness and distribution costs in a supply chain scheduling problem. *Applied Soft Computing*, *47*, 343-356.

Aydilek, H., Aydilek, A., Allahverdi, M., & Allahverdi, A. (2022). More effective heuristics for a two-machine no-wait flowshop to minimize maximum lateness. *International Journal of Industrial Engineering Computations*, *13*(4), 543-556.

Aydin, M. E., & Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, *33*(2-3), 169-178.

Bai, D., Bai, X., Yang, J., Zhang, X., Ren, T., Xie, C., & Liu, B. (2021). Minimization of maximum lateness in a flowshop learning effect scheduling with release dates. *Computers & Industrial Engineering*, *158*, 107309.

Berahhou, A., Benadada, Y., & Bouanane, K. (2022). Memetic algorithm for the dynamic vehicle routing problem with simultaneous delivery and pickup. *International Journal of Industrial Engineering Computations*, *13*(4), 587-600.

Beyer, H. G., & Deb, K. (2001). On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on evolutionary computation*, *5*(3), 250-270.

Chakravarthy, K., & Rajendran, C. (1999). A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning & Control*, *10*(7), 707-714.

Chen, R., Yang, B., Li, S., & Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, *149*, 106778.

Chen, R., Yuan, J., Ng, C. T., & Cheng, T. C. E. (2021). Bicriteria scheduling to minimize total late work and maximum tardiness with preemption. *Computers & Industrial Engineering*, *159*, 107525.

Davis, L. (1985, August). Applying adaptive algorithms to epistatic domains. In *IJCAI* (Vol. 85, pp. 162-164).

Fu, G., Han, C., Yu, Y., Sun, W., & Kaku, I. (2022). A phased intelligent algorithm for dynamic seru production considering seru formation changes. *Applied Intelligence*, 1-22. https://doi.org/10.1007/s10489-022-03579-0

Guinet, A. G. P., & Solomon, M. M. (1996). Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. *International journal of production research*, *34*(6), 1643-1654.

Kaku, I., Gong, J., Tang, J., & Yin, Y. (2009). Modeling and numerical analysis of line-cell conversion problems. *International Journal of Production Research*, *47*(8), 2055-2078. https://doi.org/10.1080/00207540802275889

Li, Z., Wei, X., Jiang, X., & Pang, Y. (2021). A kind of reinforcement learning to improve genetic algorithm for multiagent task scheduling. *Mathematical Problems in Engineering*, *2021*. https://doi.org/10.1155/2021/1796296

Lian, J., Liu, C., Li, W., & Yin, Y. (2018). A multi-skilled worker assignment problem in seru production systems considering the worker heterogeneity. *Computers & Industrial Engineering*, *118*, 366-382. https://doi.org/10.1016/j.cie.2018.02.035

Liu, C., Li, W., Lian, J., & Yin, Y. (2012). Reconfiguration of assembly systems: From conveyor assembly line to serus. *Journal of Manufacturing Systems*, *31*(3), 312-325. https://doi.org/10.1016/j.jmsy.2012.02.003

Liu, C., Lian, J., Yin, Y., & Li, W. (2010). Seru Seisan-an innovation of the production management Mode in Japan. *Asian Journal of Technology Innovation*, *18*(2), 89-113. https://doi.org/10.1080/19761597.2010.9668694

Liu, C., Stecke, K. E., Lian, J., & Yin, Y. (2014). An implementation framework for seru production. *International Transactions in Operational Research*, *21*(1), 1-19. https://doi.org/10.1111/itor.12014

Liu, C., Yang, N., Li, W., Lian, J., Evans, S., & Yin, Y. (2013). Training and assignment of multi-skilled workers for implementing seru production systems. *The International Journal of Advanced Manufacturing Technology*, *69*(5), 937-959.

Liu, F., Fang, K., Tang, J., & Yin, Y. (2022). Solving the rotating seru production problem with dynamic multi-objective evolutionary algorithms. *Journal of Management Science and Engineering*, *7*(1), 48-66.

Liu, F., Niu, B., Xing, M., Wu, L., & Feng, Y. (2021). Optimal cross-trained worker assignment for a hybrid seru production system to minimize makespan and workload imbalance. *Computers & Industrial Engineering*, *160*, 107552.

Ni, F., Hao, J., Lu, J., Tong, X., Yuan, M., Duan, J., ... & He, K. (2021, August). A Multi-Graph Attributed Reinforcement Learning Based Optimization Algorithm for Large-Scale Hybrid Flow Shop Scheduling Problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (pp. 3441-3451).

Pundoor, G., & Chen, Z. L. (2005). Scheduling a production–distribution system to optimize the tradeoff between delivery tardiness and distribution cost. *Naval Research Logistics (NRL)*, *52*(6), 571-589.

Ren, J., Ye, C., & Yang, F. (2021). Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network. *Alexandria Engineering Journal*, *60*(3), 2787-2800.

Ren, Z., Pang, B., Wang, M., Feng, Z., Liang, Y., Chen, A., & Zhang, Y. (2019). Surrogate model assisted cooperative coevolution for large scale optimization. *Applied Intelligence*, *49*(2), 513-531.

Rostami, M., Kheirandish, O., & Ansari, N. (2015). Minimizing maximum tardiness and delivery costs with batch delivery and job release times. *Applied Mathematical Modelling*, *39*(16), 4909-4927. https://doi.org/10.1016/j.apm.2015.03.052

Ruiz, R., & Allahverdi, A. (2009). Minimizing the bicriteria of makespan and maximum tardiness with an upper bound on maximum tardiness. *Computers & Operations Research*, *36*(4), 1268-1283.

Sbihi, M., & Varnier, C. (2008). Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness. *Computers & Industrial Engineering*, *55*(4), 830-840. https://doi.org/10.1016/j.cie.2008.03.005

Shahrabi, J., Adibi, M. A., & Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering*, *110*, 75-82. https://doi.org/10.1016/j.cie.2017.05.026

Shang, R., Wang, Y., Wang, J., Jiao, L., Wang, S., & Qi, L. (2014). A multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem. *Information Sciences*, *277*, 609-642.

Stecke, K. E., Yin, Y., Kaku, I., & Murase, Y. (2012). Seru: the organizational extension of JIT for a super-talent factory. *International Journal of Strategic Decision Sciences (IJSDS)*, *3*(1), 106-119. https://doi.org/10.4018/jsds.2012010104

Sun, W., Wu, Y., Lou, Q., & Yu, Y. (2019). A cooperative coevolution algorithm for the seru production with minimizing makespan. *IEEE Access*, *7*, 5662-5670.

Sun, W., Yu, Y., Lou, Q., Wang, J., & Guan, Y. (2020). Reducing the total tardiness by Seru production: model, exact and cooperative coevolution solutions. *International Journal of Production Research*, *58*(21), 6441-6452.

Tang, J., Yang, Y., & Qi, Y. (2018). A hybrid algorithm for urban transit schedule optimization. *Physica A: Statistical Mechanics and its Applications*, *512*, 745-755.

Tang, R. (2017). Decentralizing and coevolving differential evolution for large-scale global optimization problems. *Applied Intelligence*, *47*(4), 1208-1223. https://doi.org/10.1007/s10489-017-0953-9

Wang, H., Sarker, B. R., Li, J., & Li, J. (2021). Adaptive scheduling for assembly job shop with uncertain assembly times based on dual Q-learning. *International Journal of Production Research*, *59*(19), 5867-5883.

Wei, Y., & Zhao, M. (2004). Composite rules selection using reinforcement learning for dynamic job-shop scheduling, in: IEEE Conference on Robotics, Automation and Mechatronics, 2004. IEEE, pp. 1083–1088.

Wu, Y., Wang, L., & Chen, J. F. (2021). A cooperative coevolution algorithm for complex hybrid seru-system scheduling optimization. *Complex & Intelligent Systems*, *7*(5), 2559-2576.

Ying, K. C., & Tsai, Y. J. (2017). Minimising total cost for training and assigning multiskilled workers in seru production systems. *International Journal of Production Research*, *55*(10), 2978-2989.

Ke, F., Zhao, D., Sun, G., & Feng, W. (2019, June). Precise Evaluation for Continuous Action Control in Reinforcement Learning. In *Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference* (pp. 67-70).

Yılmaz, Ö. F. (2020a). Operational strategies for seru production system: a bi-objective optimisation model and solution methods. *International Journal of Production Research*, *58*(11), 3195-3219.

Yılmaz, Ö. F. (2020). Attaining flexibility in seru production system by means of Shojinka: An optimization model and solution approaches. *Computers & Operations Research*, *119*, 104917.

Yu, Y., Gong, J., Tang, J., Yin, Y., & Kaku, I. (2012). How to carry out assembly line–cell conversion? A discussion based on factor analysis of system performance improvements. *International Journal of Production Research*, *50*(18), 5259-5280.

Yu, Y., Sun, W., Tang, J., & Wang, J. (2017). Line-hybrid seru system conversion: Models, complexities, properties, solutions and insights. *Computers & Industrial Engineering*, *103*, 282-299.

Yu, Y., Tang, J., Gong, J., Yin, Y., & Kaku, I. (2014). Mathematical analysis and solutions for multi-objective line-cell conversion problem. *European Journal of Operational Research*, *236*(2), 774-786.

Yu, Y., Tang, J., Sun, W., Yin, Y., & Kaku, I. (2013). Reducing worker (s) by converting assembly line into a pure cell system. *International Journal of Production Economics*, *145*(2), 799-806.

Yu, Y., Wang, J., Ma, K., & Sun, W. (2018). Seru system balancing: Definition, formulation, and exact solution. *Computers & Industrial Engineering*, *122*, 318-325.

Zhan, R., Zhang, J., Cui, Z., Peng, J., & Li, D. (2021). An Automatic Heuristic Design Approach for Seru Scheduling Problem with Resource Conflicts. *Discrete Dynamics in Nature and Society*, *2021*. https://doi.org/10.1155/2021/8166343

Zhang, Z., Song, X., Huang, H., Zhou, X., & Yin, Y. (2022). Logic-based Benders decomposition method for the seru scheduling problem with sequence-dependent setup time and DeJong's learning effect. *European Journal of Operational Research*, *297*(3), 866-877. https://doi.org/10.1016/j.ejor.2021.06.017