A VNS-IG algorithm for dynamic *seru* scheduling problem with sequence-dependent setup time and resource constraints

Yiran Xiang, Zhe Zhang* and Xue Gong

School of Economics and Management,
Nanjing University of Science and Technology,
Nanjing 210094, China
Email: 2206073558@qq.com
Email: zhangzhe@njust.edu.cn
Email: xue.gong@njust.edu.cn
*Corresponding author

Yong Yin

Graduate School of Business,
Doshisha University,
Karasuma-Imadegawa, Kamigyo-ku,
Kyoto 602-8580, Japan
Email: yyin@mail.doshisha.ac.jp

Abstract: This paper is concerned with the unspecified dynamic scheduling problem by consideration of sequence-dependent setup time and resource constraints in the setups (UDSS-SR) in a new-type *seru* production system (SPS). The UDSS-SR problem is formulated as a mixed integer linear programming (MILP) model to minimise the makespan, and an iterative greedy algorithm based on variable neighbourhood search (VNS-IG) is designed subsequently to facilitate decision-making in the real environment to rationalise operations and additional resources. A set of test problems is generated, and computational experiments with different instance sizes are finally made. The results indicate that the proposed VNS-IG algorithm has good performance in solving *seru* scheduling problem in terms of solution quality and efficiency.

Keywords: scheduling; *seru* production; sequence-dependent setup time; resource constraint; hybrid intelligent algorithm.

Reference to this paper should be made as follows: Xiang, Y., Zhang, Z., Gong, X. and Yin, Y. (2024) 'A VNS-IG algorithm for dynamic *seru* scheduling problem with sequence-dependent setup time and resource constraints', *Int. J. Industrial and Systems Engineering*, Vol. 46, No. 1, pp.58–89.

Biographical notes: Yiran Xiang is a Master candidate of School of Economics and Management, Nanjing University of Science and Technology. Her research interest is *seru* scheduling problem.

Zhe Zhang received her PhD from Sichuan University in December 2011. She is an Associate Professor of Nanjing University of Science and Technology. Her current research interests are in the areas of *seru* production systems, production scheduling, advanced manufacturing and so on.

Xue Gong is an Associate Professor of Nanjing University of Science and Technology. Her current research interests are in the areas of decision-making optimization, transnational investment and so on.

Yong Yin is a Professor of Graduate School of Business in Doshisha University. His current research interests are in *seru* production systems, production and operations management and so on.

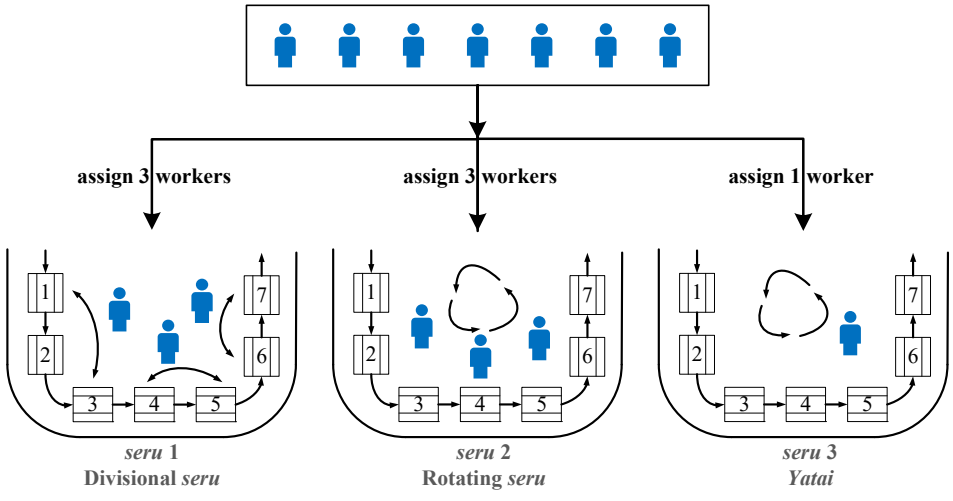
1 Introduction

In the context of Industry 4.0, the rapid development of information technology and increased consumer demand are placing production demands on manufacturers for greater flexibility, higher product quality, shorter lead times and customised production. Market demand presents the characteristics of product variety and output fluctuations, the mismatch between supply and demand in the value chain has become a problem that manufacturing enterprises continue to pay attention to (Yin et al., 2018), the flexibility and agility of the production system are becoming more and more important for manufacturers (Niakan et al., 2016), in order to achieve production flexibility and respond quickly to market fluctuations, enterprises must quickly reorganise the production system to have both efficiency and flexibility, thereby enhancing the core competitiveness (Frank et al., 2019). *Seru* production system (SPS) is an innovative production mode, developed by Japanese manufacturers in production practice, achieves efficiency, flexibility and rapid response at the same time (Wu et al., 2021). SPS is reconfigured from the traditional assembly line, which contains one or more *seru*, where *seru* is an assembly unit consisting of one or more workers and some simple equipment, *seru* has three types, including divisional *seru*, rotating *seru* and *yatai* (Shao et al., 2016; Luo et al., 2017; Yu and Tang, 2019). The SPS as shown in Figure 1 has seven workers who are cross-trained, can perform most or all tasks in the SPS. There are three workers assigned to *seru* 1, each worker completes several tasks, together to complete a job, called divisional *seru*; there are three workers assigned to *seru* 2, each worker completes tasks one by one in each workstation, each completes a job independently, after each job is completed, the worker will return to the first workstation to start a new round of job, known as rotating *seru*; one worker is assigned to *seru* 3, and the worker is responsible for all tasks in this *seru*, called *yatai*. This paper treats *seru* as a black box, and the proposed model and methods are applicable to all types *seru*.

The SPS composed of movable workstations and multi-skilled workers is reconfigurable, and the *serus* in the SPS can be frequently built, disassembled, modified and refactored in a short period of time to adapt to changing market needs, so it can be quickly reconfigured according to changes in demand, ensuring a high level of productivity and quality (Kaku, 2016; Luo et al., 2021; Wang et al., 2022). In production practice, SPS combines the advantages of other production systems and lean philosophy to bring significant benefits to users (Stecke et al., 2012), and is known as ‘the next generation of lean manufacturing’ (Yin et al., 2017). *Seru* production has been successfully implemented in the electronics industry such as Canon and Sony in Japan, and many leading Japanese companies such as Panasonic, NEC, Fujitsu, Sharp, Sanyo, etc., assembly lines have also been converted to SPS to increase productivity (Sakazume,

2005; Takeuchi, 2006; Kaku et al., 2009; Zhang et al., 2020). Liu et al. (2021a) conducted an empirical study on *seru* manufacturing flexibility in the context of Chinese enterprises, and pointed out that multi-skilled worker participation has a great impact on *seru* manufacturing flexibility. Impressively, other benefits of SPS implementation include: improved task bottlenecks (Andradóttir et al., 2013), reduced workshop space (Stecke et al., 2012), reduced completion time (Sun et al., 2019; Gai et al., 2020), reduced manpower (Yin et al., 2008), reduced total delays (Sun et al., 2020), etc. Nowadays, SPS is gaining more and more attention both in the academic and engineering field. Treville et al. (2017) pointed out that in the face of the rapid replacement of electronic products, Japanese electronics companies can use SPS to quickly respond to the market. Min et al. (2019) proposed that *seru* management and control principles also have the potential to be applied to smart manufacturing, and Yin et al. (2017) pointed out that SPS is an alternative to lean system approach that seems to offer hope for manufacturing in dynamic, high-cost market. Kaku (2017) illustrated the sustainability effects of SPS. Zhang et al. (2017) showed that the key enabling technologies for *seru* production have positive effects on sustainable performance. The SPS is known as the 'double E' (ecological and economic) production management mode (Liu et al., 2015). Roth et al. (2016) summarised the development of operations management over the past 25 years, pointing out that *seru* production is one of the new areas worth paying attention to.

Figure 1 Three *seru* types (see online version for colours)



In actual production of the SPS, the adoption of the just-in-time organisation system (JIT-OS) is key to achieving high performance and rapid response (Yu et al., 2018; Zhang et al., 2022d). The core of the JIT-OS implementation mechanism is correct *serus*, in the right place, at the appropriate time, in the exact amount (Stecke et al., 2012). In JIT-OS, there are three decision-making phases: *seru* formation, *seru* loading, and *seru* scheduling (Sun et al., 2020). By implementing *seru* formation and *seru* loading, SPS with the appropriate number of *serus*, suitable production materials and equipment are configured. Then, by implementing *seru* scheduling, consider the detailed job processing plan in each *seru* (e.g., job sequencing, labour allocation, resource allocation, etc.) (Jiang et al., 2021a). Existing research on SPS has focused on the above three areas (Zhang

et al., 2022c). For *seru* formation and *seru* loading, Liu et al. (2012) investigated the problem of how to reconfigure the conveyor assembly line to *serus*, a comprehensive mathematical model was developed to solve the problems of how many *serus* should be built and how many workers should be assigned to each *seru*. Liu et al. (2014) provided practitioners with a general framework and some basic principles that should be followed when implementing *seru* production from a practical point of view. Yu et al. (2017b) developed line-*seru* conversion to reduce workers without increasing completion time, and developed exact and meta-heuristic algorithms for examples of different sizes. Liu et al. (2013) investigated the training and assignment problem of workers when a conveyor assembly line is entirely reconfigured into several *serus*, and developed a three-stage heuristic algorithm with nine steps. Yu et al. (2017a) established several main line-hybrid *seru* system conversion models and elucidated the complexity of line-hybrid *seru* system conversion. Yu et al. (2016) selected ten scheduling rules commonly used in *seru* loading, the impact of different scheduling rules on the performance of line-*seru* conversion was studied, and the complexity of line-*seru* conversion of ten different scheduling rules was clarified from a theoretical perspective. Wang and Tang (2020) studied optimising the configurations for SPS in situations where requirements are uncertain, proposing a heuristic algorithm to solve this problem. Wang and Tang (2018) studied the formation of SPS under an uncertain demand and proposed a multi-objective optimisation model to minimise the cost of SPS and maximise service levels. Lian et al. (2018) solved the multi-skilled worker assignment problem of the SPS, which considered the differences in worker skill sets and proficiency, and developed a meta-heuristic algorithm based on NSGA-II for solving. Luo et al. (2016) considered a single period *seru* loading problem with worker-operation assignment, a mathematical model was proposed and a heuristic algorithm was designed to solve this problem. Zhang et al. (2021) solved a *seru* loading problem system with a downward substitution and random product demands and yields. Ying and Tsai (2017) studied the multi-skilled worker training and assignment problem of SPS, and designed a two-stage heuristic algorithm SAIG algorithm to effectively solve this problem. Liu et al. (2021b) investigated the issue of assigning cross-trained workers in hybrid SPSs. Jiang et al. (2021b) discussed four scheduling problems that consider discrete controllable processing times and resource allocation, and converted them into allocation problems using a general exact solution method. Yılmaz (2020b) addressed a bi-objective workforce scheduling problem by considering the inter-*seru* worker transfer in SPS, proposed a novel optimisation model to achieve two objectives, that of minimising makespan and reducing workload imbalance among workers. Yılmaz (2020a) conducted research on lab or scheduling problems in *seru* production environments, proposed a comprehensive optimisation model. Zhang et al. (2022a) investigated the scheduling problem in the SPS, which taken into account the sequence-dependent setup time and DeJong's learning effect to minimise the makespan, developed a mixed-integer programming (MIP) model, then logic-based Benders decomposition (LBBD) method was applied to reformulated the proposed model. For *seru* scheduling, when studying the scheduling optimisation problem of SPS, the influencing factors are mainly considered: setup time, configuration of multi-skilled workers, learning effect, delivery time and lot-splitting (Süer and Dagli, 2005). In this paper, we will study for the first time the *seru* scheduling problem considering setup time and resource constraints, and hope that this research can improve the theoretical research of SPS and provide professional guidance for *seru* production managers.

In the actual production process, there is a lot of additional consideration involved, setup is usually a non-productive activity between two consecutive jobs in a sequence assigned to the same *seru* for processing. Setup includes reconfiguration, cleaning, adjustment tools, colour preparation, etc. (Fanjul-Peyro et al., 2019). Most scheduling studies assume that the setups are set to be negligible or simplified as part of job processing (Allahverdi et.al., 1999; Ebrahimi et al., 2014), although this assumption simplifies the analysis and may be reasonable for some scheduling problems, other setup factors such as setup time and resource constraints in the setups must be taken into account in other production tasks that require explicit handling of setup, especially in multi-product production processes (Jiang et al., 2021a; Yepes-Borrero et al., 2020; Zhang et al., 2022b). When the setups depend on the type of job that was just completed and the job that is about to be processed, the setups depend on the order of sequence. The setup time and the resources required for setups considered in this paper depend on both the job to be processed and the job immediately preceding it, which is called sequence-dependent.

Setup time refers to the time it takes to prepare the necessary resources (such as workers or tools) to perform a task (such as operation or work) in SPS (Salvendy, 2001), the sequence-dependent setup time also depends on *seru*, the setup time between two jobs in one *seru* may be different from the setup time in other *seru*. Therefore, if you consider setup time, the order of the jobs assigned to the sequence in the *seru* is very important. In addition to the sequence-dependent setup time, we also considering the additional resources allocated to each setup. Additional resources are considered: renewable resources, which are available again after setup is complete; discrete, the amount of resources required for setups is a positive integer; and processing, resources are required only during setups. The necessity of considering sequence-dependent setup time and resource constraints in production scheduling problems, has been recognised in some research. Diana et al. (2015) proposed a clone selection algorithm to solve the problem minimising the makespan on unrelated parallel machines with sequence-dependent setup times. Ruiz and Andrés-Romano (2011) considered an unrelated parallel machine problem with machine and job sequence-dependent setup times, where the setup time depends not only on the machine and job sequence, but also on the amount of resources allocated, which can vary between minimum and maximum values. Pinheiro et al. (2020) investigated the unrelated parallel machines scheduling problem with family setups and resource constraints. In this problem, jobs were grouped into families and setup times were required between jobs belonging to different families. Rajkumar et al. (2011) aimed at the flexible workshop scheduling problem under the constraint of limited resources, proposed a GRASP algorithm. Villa et al. (2018) proposed two different approaches to the unrelated parallel machine scheduling problem with one scarce additional resource: the first method considered resource constraints throughout the process, and the second method first did not consider resource constraints to get an unfeasible solution, then repaired solutions, they developed several heuristic algorithms. In the existing literature, there is usually no limit to the resources required for simultaneous setups in SPS. In other words, at any point in time, as many setups can be made as needed in the SPS, which is not in line with the actual production environment of the SPS. In the SPS, the setups between jobs are usually done by additional resources (such as workers with a certain professional skill), the number of available resources is usually limited, so the setups that can be made at the same time in the SPS are limited. Therefore, in this paper, we will consider both the sequence-dependent setup time and the

resources required for setups, as far as we know, in the field of *seru* scheduling problem research, the unspecified dynamic scheduling problem by consideration of sequence-dependent setup time and resource constraints in the setups (UDSS-SR) is a novelty issue, and we will first study the UDSS-SR problem with the objective of minimising the makespan.

The rest of this paper is organised as follows. Section 2 gives introduction to the problem and a mathematical model. Section 3 describes the algorithm designed to solve the UDSS-SR problem. Section 4 shows the experimental activity of evaluating the proposed algorithm. Finally, some conclusions and directions for future research are given in Section 5.

2 Problem formulation

In this section, we formally introduce the mathematical model we built to solve the UDSS-SR problem. Unlike traditional *seru* scheduling, the UDSS-SR problem considers the setup time and resources required to transform production between different jobs with the goal of minimising makespan.

In the UDSS-SR problem, the SPS has been built, all *serus* are always available, each *seru* can only handle one job at a time, there is no pre-emption, all jobs can be processed in all *serus*. In addition, there is no priority limit on the sequence of the jobs, and all *serus* are available from time zero. The setup time and resources are related to *seru* and sequence, that is, the setup time and resources between job j and j' in *seru i* may be different from the setup time and resources between job j' and j in the same *seru*. In addition, the setup time and resources between jobs j and j' in *seru i* may differ from the setup time and resources between jobs j and j' in the other *seru*. The UDSS-SR problem requires a certain amount of resources before performing each job due to the limited resources in the setups, and the feasibility of the obtained solution depends on the amount of resources used at any point in time. The resource constraints may cause idle time generation in *seru*.

2.1 Notations

For convenience, following notations are introduced.

2.1.1 Indices

$i = 1, 2, \ldots, I$ index for *serus*

$j = 0, 1, 2, \ldots, J$ index for jobs

$t = 1, 2, \ldots, T_{max}$ index for time.

2.1.2 Parameters

p_{ij} The processing time of job j in *seru i*.

$s_{ijj'}$ The setup time of successive jobs j and j' in *seru i*.

$r_{ijj'}$ The resources required in the setups of successive jobs j and j' in *seru i*.

R_{max} Total number of available resources for the setups in SPS.

2.1.3 Decision variables

X_{ij} Binary variable takes value 1 if job j is assigned to *seru i.*

$Y_{ijj'}$ Binary variable takes value 1 if job j' is processed continuously after job j in *seru i*, and 0 otherwise.

$Z_{ijj't}$ Binary variable takes value 1 if job j' is processed continuously after job j in *seru i*, and job j completes its processing at time t, and 0 otherwise.

C_{max} Makespan.

Note that we have introduced a dummy job J_0, J_0 processed in each *seru* at time zero. We set $p_{i0} = 0, s_{i0j} \neq 0, r_{i0j} \neq 0, \forall i, j, j \neq 0$.

2.2 Mathematical formulation

The objective of the UDSS-SR problem considered in this paper is to minimise the makespan, we have:

$$\min \quad C_{\max} \tag{1}$$

Make sure that at most one job is assigned to the first position of the sequence of each *seru*, so:

$$\sum\nolimits_{j' \in J} Y_{i0j'} \leq 1, \forall i \tag{2}$$

Make sure that each job can only be assigned to one *seru*, so:

$$\sum\nolimits_{i \in I} X_{ij} = 1, \forall j \tag{3}$$

Make sure that job j in *seru i* is followed by only one consecutive job j':

$$X_{ij} = \sum_{j' \in J, j' \neq j} Y_{ijj'}, \forall i, j \tag{4}$$

Make sure that job j' in *seru i* is preceded by only one consecutive job j:

$$X_{ij'} = \sum\nolimits_{j \in J, j \neq j'} Y_{ijj'}, \forall i, j' \tag{5}$$

Make sure that for each pair of consecutive jobs j and j' in each *seru i*, job j must be processed before $T_{\max}$:

$$\sum_{t \leq T_{\max}} Z_{ijj't} = Y_{ijj'}, \forall i, j\ j', j \neq j' \tag{6}$$

Make sure that for each pair of consecutive jobs j and j' in each *seru i*, the processing time of job j ends at the earliest, where M is a sufficiently large value:

$$\sum_{j''\in J, j''\neq j}\sum_{t''\le T_{\max}} Z_{ij''jt''}\left(t''+s_{ij''j}+p_{ij}\right)-M\left(1-Y_{ijj'}\right)\le\sum_{t} tZ_{ijj't}, \quad \forall i,\ j,\ j',\ j\neq j' \tag{7}$$

Make sure that the total number of resources used at any one time does not exceed the total number of resources in the setups R_{max} within the SPS:

$$\sum_{i\in I, j, j'\in J, j'\neq j, t'\in\{t+p_{ij}+1,\ldots,t+p_{ij}+s_{ijj'}+1\}} r_{ijj'}Z_{ijj't'}\le R_{\max}, \quad \forall t\le T_{\max} \tag{8}$$

Make sure makespan is not less than the completion time of all *serus*:

$$C_{\max}\ge\sum_{t\le T_{\max}} tZ_{ijj't}, \forall i,\ j,\ j',\ j\neq j' \tag{9}$$

where

$$X_{ij}\ge 0, Y_{ijj'}\ge 0, Z_{ijj't}\in\{0,1\}, \forall i,\ j,\ t \tag{10}$$

To sum up, the MILP model for the UDSS-SR problem can be presented as:

(MILP) $\min\ C_{\max}$

$$\text{s.t.}\begin{cases}
\sum_{j'\in J} Y_{i0j'}\le 1, & \forall i\\
\sum_{i\in I} X_{ij}=1, & \forall j\\
X_{ij}=\sum_{j'\in J, j'\neq j} Yi_{jj'}, & \forall i,\ j\\
X_{ij'}=\sum_{j'\in J, j\neq j'} Y_{ijj'}, & \forall i,\ j'\\
\sum_{t\le T\max} Z_{ijj't}=Y_{ijj'}, & \forall i,j,\ j',\ j\neq j'\\
\sum_{j''\in J, j''\neq j}\sum_{t''\le T_{\max}} Z_{ij''j}\left(t''+s_{ij''j}+p_{ij}\right)-M\left(1-Y_{ijj'}\right)\le\sum_{t} tZ_{ijj't}, & \\
\forall i,\ j,\ j',\ j\neq j' & \\
\sum_{i\in I, j, j'\in J, j'\neq j, t'\in\{t+p_{ij}+1,\ldots,t+p_{ij}+s_{ijj'}+1\}} r_{ijj'}Z_{ijj't'}\le R_{\max}, & \forall t\le T_{\max}\\
C_{\max}\ge\sum_{t\le T_{\max}} tZ_{ijj't}, \forall i,\ j,\ j',\ j\neq j' & \\
X_{ij}\ge 0,\quad Y_{ijj'}\ge 0,\quad Z_{ijj't}\in\{0,1\}, & \forall i,\ j,\ t
\end{cases} \tag{11}$$

3 A VNS-IG algorithm for UDSS-SR

To find the solution to the UDSS-SR problem, we must solve the following three subproblems:

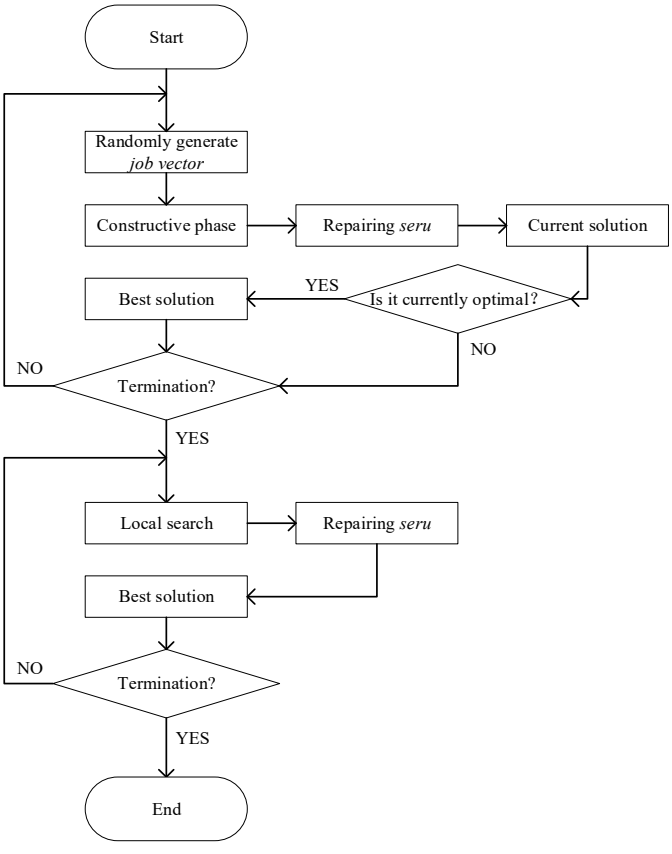
1 The job-*seru* assignment problem. That is, to get the job-*seru* assignment relationship and find a suitable *seru* for each job.

2 The job sequencing problem in *seru*. That is, to get the processing order of jobs in each *seru*.

3 The job timing problem. That is, to get the moment when each job in each *seru* starts processing and the moment when setup time starts.

To solve the UDSS-SR problem, we propose an iterated greedy algorithm based on variable neighbourhood search (VNS-IG). The proposed heuristic rule is first applied in the constructive phase to obtain a relatively better initial solution, and then a variable neighbourhood search process is applied to optimise the feasible solution obtained in the constructive phase, and the repairing algorithm is repeatedly applied in the iterative search process to ensure the feasibility of the results.

Figure 2 Flow chat of the proposed VNS-IG



In the rest of this section, we show the different phases of our proposed algorithm, mainly: the constructive phase, the repairing phase, and the variable neighbourhood search phase. In order to avoid premature convergence, we set termination conditions for the constructive phase and the variable neighbourhood search phase, and repeat the phase in which they are located until the termination conditions are satisfied. Figure 2 shows the flowchart of the whole VNS-IG algorithm.

3.1 Constructive phase

In order to generate high-quality initial solutions, we use two methods in the constructive phase:

a list scheduling (LS) heuristic algorithm (Davis and Jaffe, 1981)

b Minimum setup time and minimum resources used in the setup's priority rule (SST&MSR).

And in order to avoid local optimum, we randomise the constructive phase, generate the job vector in random order of jobs, set the maximum number of iterations, and seek the optimal solution of the constructive phase under the maximum number of iterations, and use the repairing algorithm in this phase to ensure the feasibility of the solution. The procedure of the constructive phase is summarised in Algorithm 1.

Algorithm 1 Constructive phase

```
2    Best_sol ← Inf;
3    for interation ← 1 to maximum interation number do
4        JV ← Randomly sort the jobs that will be assigned, generate a job vector.
5        Current_plan ← Apply LS heuristic
6        Current_plan ← Apply SST&MSR rule
7        Current_sol ← Apply Repairing phase
8        if Current_sol < Best_sol, then
9        Best_plan ← Current_plan
10       Best_sol ← Current_sol;
11       end
12   end
```

3.1.1 LS heuristic

LS heuristic solves the first job-*seru* assignment problem in the proposed three subproblems. By LS heuristic, the job-*seru* assignment relationship can be obtained and the appropriate *seru* is selected for each job.

LS heuristic is based on certain principles to select the appropriate *seru* for each job. Since each job *j* has different processing time in different *seru i* and the optimisation goal is to minimise makespan, we define a coefficient to reflect the processing efficiency of job *j* in *seru i*. We refer to this coefficient as e_{ij}.

This coefficient is defined as:

$$e_{ij} = \left(\min_{0 \le i' \le I} p_{i'j}\right) \Big/ p_{ij}$$

The processing efficiency e_{ij} is in the interval (0, 1). Larger e_{ij} means that job j is processed more efficiently in *seru i*, and $e_{ij} = 1$ means that job j is processed in the most efficient *seru*.

The LS heuristic process is summarised in Algorithm 2, where C_i is the completion time of *seru i*.

Step1 Construct a separate list l_i for each *seru* and calculate e_{ij} for each job in each *seru* based on the order of the jobs in the JV vector.

Step 2 Sort all unassigned jobs in non-increasing order by e_{ij}, and update the list l_i.

Step 3 If the job set $j^* \neq \varnothing$, calculate C_i of all *serus* at this time and find *seru* $i' = \arg\min_{0 \le i' \le I} C_{i'}$, assign *seru i* to process j' and get the set of jobs $\tilde{j}_{i'}$ assigned by *seru* $i' = \arg\min_{0 \le i' \le I} C_{i'}$, otherwise the LS heuristic algorithm terminates.

Step 4 $e_{i'j'} > 1\sqrt{I}$, assign *seru* i' to process job j' and get the set of jobs $\tilde{j}_{i'}$ assigned by *seru* i', otherwise $C_{i'} = C_{i'} + \infty$, and return to Step 3. Note that due to the balance of the scheduling problem, we set the set of jobs $\tilde{j}_{i'}$ to be assigned to *seru* if the $\tilde{j}_{i'}$ is greater than $\lceil J / I \rceil$, then at most the first $\lceil J / I \rceil$ jobs are assigned.

Step 5 Update the list $l_i = l_i \setminus \{\tilde{J}_{i'}\}$, and the set of jobs $J^* = J^* \setminus \{\tilde{J}_{i'}\}$, and return to Step 3.

Algorithm 2 LS heuristic

Input e_{ij}, J
Output job-*seru* assignment (*Current_plan*)
$J^* \leftarrow J$
$l_i \leftarrow$ Create a list of jobs sorted in the non-increasing order of e_{ij} for each *seru*
while $j^* \neq \varnothing$ **do**
  Find *seru* $i' = \arg\min_{0 \le i' \le I} C_{i'}$
  flag $\leftarrow 0$
  while *flag* = 0**do**
    if $e_{i'j'} > 1/\sqrt{I}$, **then**
      Assign *seru* i' to process the j'
    else
      $C_{i'} = C_{i'} + Inf$
      flag $\leftarrow 1$
    end
  end

14 $\tilde{J}_{i'} \leftarrow$ Get the set of jobs assigned to *seru i'*

15 $l_i \leftarrow l_i \setminus \{\tilde{J}_{i'}\}, J^* \leftarrow J^* \setminus \{\tilde{J}_{i'}\}$

16 **end**

3.1.2 SST&MST rule

The SST&MST rule solves the second of the three proposed subproblems of job sequencing in each *seru*, and the SST&MST rule can optimise the job ordering in the *seru* and can effectively obtain a reasonable job sequence.

The SST&MST rule in this paper considers a balanced solution of the makespan and the resources required in the setups, considering not only the setup time of different job sequences, but also measuring the resources required in the setups. Information about the resource constraints required in the setups is considered to avoid sequences with long setup times or high resource consumption in order to obtain solutions that require fewer resources (possibly allowing for increased makespan), bringing the solutions closer to feasibility, which makes the second stage, repairing phase easier.

The SST&MST rule process is summarised in Algorithm 3, where the job with the smallest coefficient λ is selected as the next job to be processed when the current job is completed until all jobs are assigned, and λ is defined as $\lambda = s_{ijj'} \times r_{ijj'}$.

Algorithm 3 SST and MST rule

Input job-*seru* assignment (*Current_plan*)

Output job scheduling results (*Current_plan*)

1 **for** $i \leftarrow 1$ to I **do**

2 $\tilde{J}_i \leftarrow$ The set of jobs assigned to *seru i*

3 **while** $\tilde{J}_{i'} \neq \varnothing$ **do**

4 Select Job j^* which λ is minimum as the next job to be processed when the current job completes

5 $\tilde{J}_i \leftarrow \tilde{J}_i \setminus \{j^*\}$

6 **end**

7 **end**

Step 1 For each *seru*, $\tilde{j}_{i'}(i = 1, 2, \ldots, I)$ is denoted as the set of jobs assigned to *seru* i processing, note that each *seru* has a dummy job J_0 at time 0, set $i = 1$.

Step 2 If $i \leq I$, go to step 3, otherwise the algorithm terminates.

Step 3 If $\tilde{J}_i \neq \varnothing$, select the job $\tilde{J}_i \neq \varnothing$, with the smallest λ as the next job to be processed when the current job completes.

Step 4 Set $\tilde{J}_i = \tilde{J}_i \setminus \{j^*\}$ and repeat Step 3 until all jobs are assigned.

Step 5 If $\tilde{J}_i = \varnothing, i = i + 1,$ return to step 2.

3.2 Repairing phase

After obtaining the job-*seru* assignment relationship by LS heuristic and optimising the job processing sequence of each *seru* according to the SST&MST rule, the solution without resource constraints will be obtained, and then the solution needs to be evaluated to verify whether the resource constraints are satisfied. The following example shows the difference between *seru* scheduling without resource constraints and *seru* scheduling with resource constraints in the setups, and how the repairing phase algorithm will be applied.

Example 1: Consider a UDSS-SR problem, with $J = 5$ jobs, $I = 3$ *serus*, and $R_{max} = 3$ resources available for setups. The specific relevant data is shown in Table 1, Table 2 and Table 3.

Table 1 p_{ij} for the example 1 with 5 jobs and 3 *serus*.

	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$	4	8	6	7	3
$i = 2$	7	7	5	6	3
$i = 3$	2	2	3	2	8

Using the LS heuristic and SSR&MST rule proposed by constructive phase to assign jobs to *serus* and optimise the ordering of jobs in each *seru*, we obtain the solution without resource constraints in the setups for Figure 3 (a). The coloured rectangles indicate the jobs being processed and the numbers in them indicate the job indexes. The white rectangles indicate the setups before each job starts in *seru*, where the setup time and resources required in the setups are shown, but this solution is infeasible, and we can see that too many resources are used for setups between time 0 to time 1 and time 6 to time 7, which exceed the resource constraints for setups, and we need to fix the solution to make it satisfy the resource feasibility.

Figure 3 Example repairing phase, (a) non-feasible solution (b) resource feasibility solution (see online version for colours)

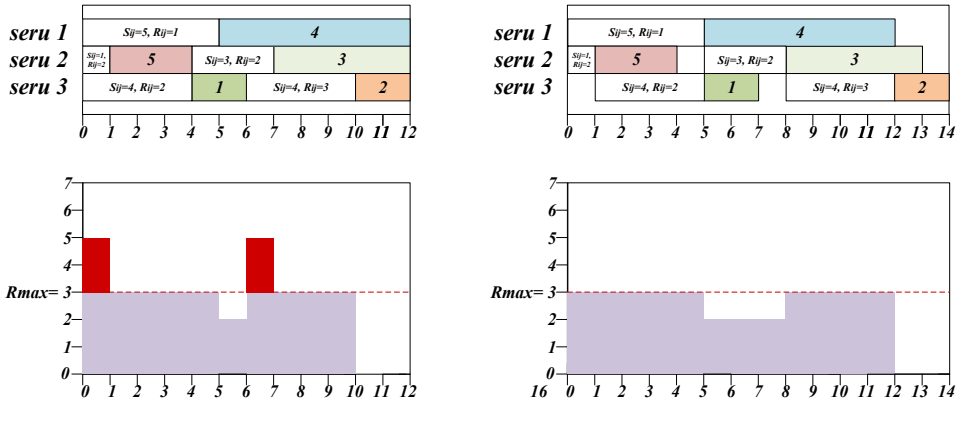


Table 2 Setup times ($s_{ijj'}$) for the example 1 with 5 jobs and 3 *serus*

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$j'=1$	4	0	7	3	9
$j'=2$	8	5	0	7	3
$j'=3$	4	9	3	0	4
$j'=4$	5	8	8	5	0
$j'=5$	3	10	2	1	3

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$j'=1$	7	0	8	6	5
$j'=2$	2	10	0	3	5
$j'=3$	3	2	4	0	1
$j'=4$	7	2	4	6	0
$j'=5$	1	3	7	1	6

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$j'=1$	5	4	0	8	7
$j'=2$	8	4	4	0	3
$j'=3$	9	9	5	4	0
$j'=4$	8	3	5	4	1
$j'=5$	6	4	0	8	7

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$j'=1$	10	3	8	3	8
$j'=2$	9	4	0	8	7
$j'=3$	3	4	4	0	3
$j'=4$	10	9	5	4	0
$j'=5$	6	3	5	4	1

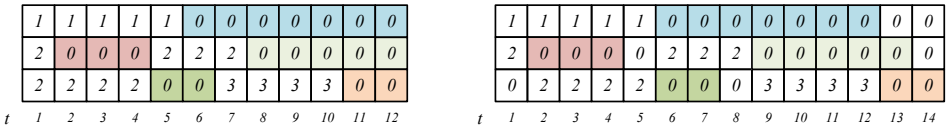
	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$j'=1$	9	4	7	3	9
$j'=2$	4	0	3	7	4
$j'=3$	3	4	3	0	3
$j'=4$	10	5	9	4	0
$j'=5$	5	3	5	4	1

Table 3 Consumption of resources $(r_{ijj'})$ for the example 1 with 5 jobs and 3 *serus*

	$r_{ijj'}(i=1)$						$r_{ijj'}(i=2)$						$r_{ijj'}(i=3)$					
	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$j'=1$	2	0	1	3	1	3	3	0	2	1	2	1	2	0	1	2	1	1
$j'=2$	3	1	0	1	3	2	1	2	0	2	2	2	3	3	0	2	2	3
$j'=3$	1	3	2	0	1	1	1	2	2	0	1	2	1	2	2	0	1	1
$j'=4$	1	3	2	2	0	2	2	1	2	1	0	1	2	2	1	3	0	1
$j'=5$	2	2	1	3	1	0	2	3	2	3	2	0	1	1	2	1	3	0

We use the matrix form to represent the job processing time, setup time, and the resources used in the setups are represented by numbers in the setup time period, and the number of matrix columns represent makespan. Figure 4 shows the matrix representation of the resources in the solution, the resources used in the solution before the repairing phase algorithm in Example 1 are represented as Figure 4 (a), and the resources used in the solution after using the repairing phase algorithm are represented as Figure 4 (b).

Figure 4 Resource matrix representation, (a) non-feasible solution resource-matrix representation (b) resource feasibility solution resource-matrix representation (see online version for colours)



(a) (b)

Specifically divides the time into time periods in units of 1. If the resources required in a certain time period exceed R_{max}, the solution must be repaired. The repairing phase algorithm judges and repairs every time period that the entire *seru* system takes to process all jobs, starting from moment 1 until makespan, calculates the total resources required for setups in each time period, and if the available resources are exceeded in a certain time period, the start moment of setups will be postponed by increasing the idle time in the *seru*. The repairing rule used in this paper is to postpone the job with latest starting setup in the *seru* (if the setting start time is the same, then select the postponement job in order) until the resource constraints are satisfied, and this process will be repeated until all moments are judged and repaired. Interestingly, if increasing the idle time to delay the setups start is not in the makespan *seru*, then deferring the setups start may not increase the makespan.

For example 1, job 3 in *seru* 2 is postponed by 1 time unit, job 1 and job 2 in *seru* 3 are delayed by 1 time unit, at which point the resource constraints in the setups on all time units are satisfied [Figure 3(b)].

The repairing process is summarised in Algorithm 4.

Algorithm 4 Repairing phase

Input *Current_plan*
Output *Current_sol*
R-m ← Represent the solution as the resource-matrix
while $t \le$ *makespan* **do**
  R_t ← Calculate consumption of resources at time period t
  if $R_t > R_{\max}$, **then**
    Postpone the job with latest starting setup in the *seru* until satisfy the resources constraint at time period t
  end
  Update *R-m* matrix
  Update *makespan*
end

3.3 Variable neighbourhood search

To further improve the algorithm, we propose three efficient job swapping heuristics in the SPS as variable neighbourhood search methods to complete the local search:

a internal swap

b external swap

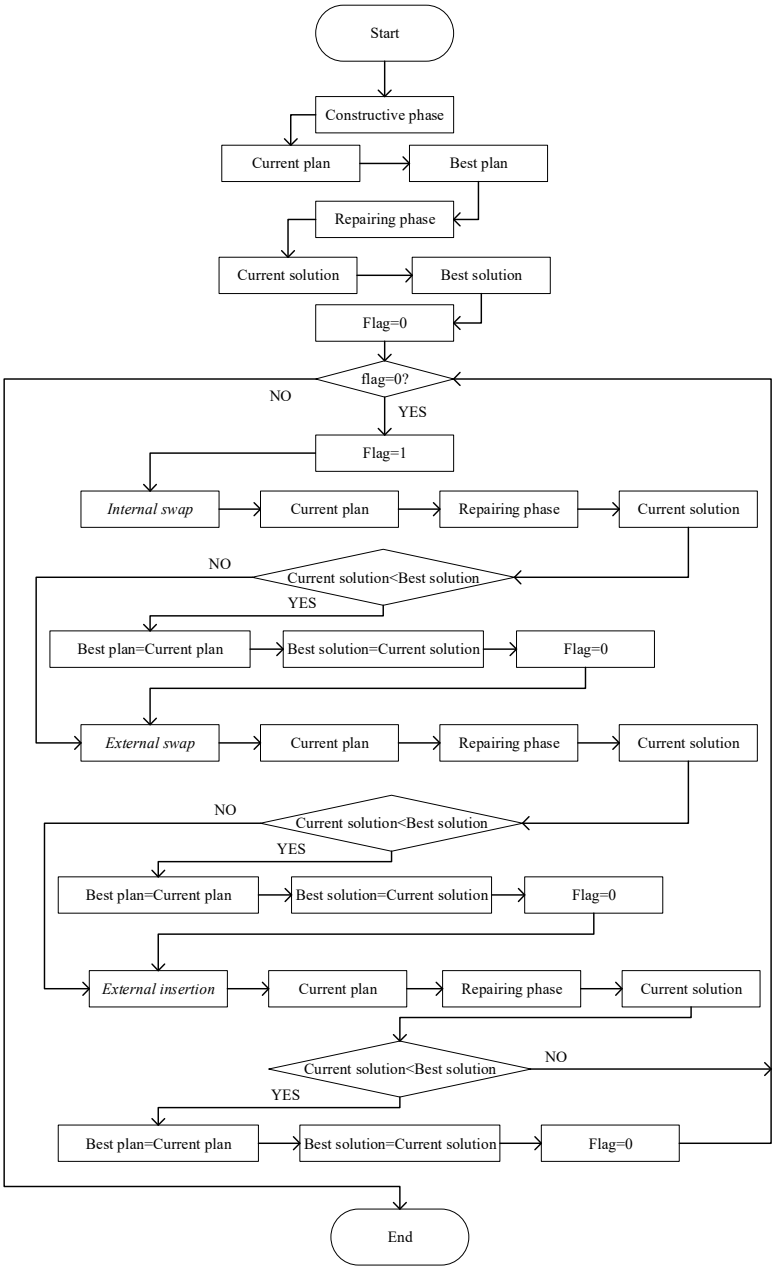
c external insertion.

Algorithm 5 Variable neighbourhood search

```
    Input Current_plan
    Output Best_plan, Best_sol
1   Current_plan ← Constructive phase
2   Current_sol ← Repairing phase
3   Best_plan ← Current_plan
4   Best_sol ← Current_sol
5   flag ← 0
6   while flag = 0 do
7       flag ← 1
8       Current_plan ← Apply Internal swap
9       Current_sol ← Apply Repairing phase
10      if current_sol < Bes_sol, then
11          Best_plan ← Current_plan
12          Best_sol ← Current_sol
13          flag ← 0
14      end
15      Current_plan ← Apply External swap
16      Current_sol ← Apply Repairing phase
17      if Current_sol < Best_sol, then
18          Best_plan ← Current_plan
19          Best_sol ← Current_sol
20          flag ← 0
21      end
22      Current_plan ← Apply External insertion
23      Current_sol ← Apply Repairing phase
24      if Current_sol < Best_sol, then
25          Best_plan ← Current_plan
26          Best_sol ← Current_sol
27          flag ← 0
28      end
29  end
```

Variable neighbourhood search will be applied after the constructive phase, the variable neighbourhood search phase process is summarised in Algorithm 5. The flowchart of the variable neighbourhood search process is shown in Figure 5, and we will repeat the variable neighbourhood search process before the termination condition. In the remainder of this section, we will describe the proposed variable neighbourhood search in detail.

Figure 5 Flow chat of variable neighbourhood search



3.3.1 Internal swap

Internal swap means that for each job *j* in each *seru i*, job *j* is swapped with any other job j' assigned to be processed in the same *seru*, the setup time and the resources required in the setups after updating the swap, and applies the repairing phase. A judgment is made for each internal swap, and after computing all possible swaps, we keep the internal swap under the solution with the minimum makespan. The internal swap process is summarised in Algorithm 6 and schematically shown in Figure 6(a).

Algorithm 6 Internal swap

```
    Input Current_plan, Current_sol
1   flag ← 0
2   while flag = 0 do
3       flag ← 1
4       for i ← 1 to I do
5           J̃ ← job in seru i
6               for J̃_j ← J̃_1 to J̃_J do
7                   for J̃_j' ← J̃_2 to J̃_J do
8                       Temporary_plan ← Swap the job J̃_j with J̃_j', update the setup time
                        and the resources used in the setups
9                       Temporary_sol ← Apply repairing phase
10                      if Temporary_sol < Best_sol, then
11                          Current_plan ← Temporary_plan
12                          Best_plan ← Current_plan
13                          Current_sol ← Temporary_sol
14                          Best_sol ← Current_sol
15                          flag ← 0
16                      end
17                  end
18              end
19          end
20  end
```

3.3.2 External swap

External swap is the job j in the *seru* where makespan is located, defines the *seru* where makespan is located as $seru_{makespan}$, that is, each job *j* in $seru_{makespan}$ is swapped with job j' of other *seru*, updating the setup time and the resources required in the setups after the swap, and performing the repairing phase. Judgment is made for each external swap, and after computing all possible swaps, we keep the external swap under the solution with the minimum makespan. The external swap process is summarised in algorithm 7 and schematically shown in Figure 6(b).

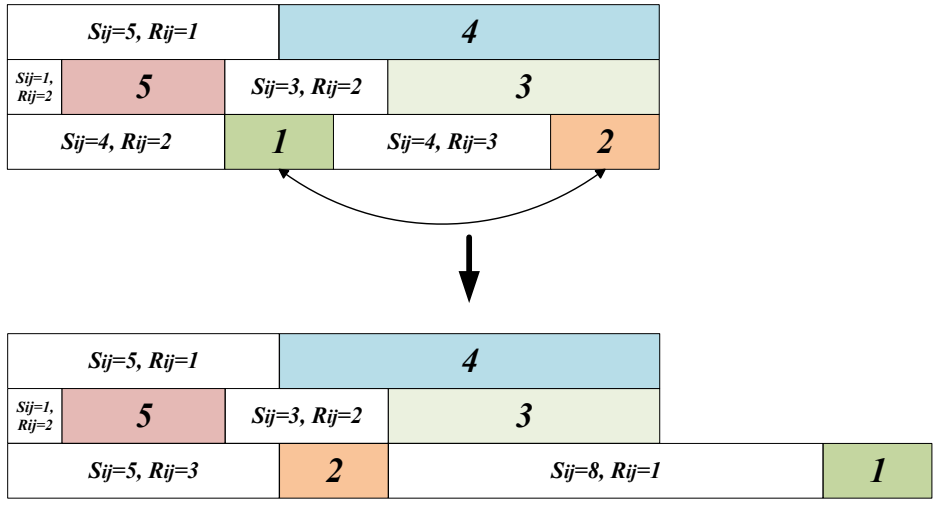
Algorithm 7 External swap

Input *Current_plan, Current_sol*
flag← 0
$seru_{makespan}$ ← *seru i* that defines the *makespan*
while flag = 0 **do**
  flag ← 1
  $\tilde{j}$ ← job in $seru_{makespan}$
  for $\tilde{j}_j \leftarrow \tilde{j}_1$ **to** $\tilde{j}_J$ **do**
    for $i' \leftarrow 1$ **to** I **do**
      if $i' \neq i$, **then**
        $\hat{j}$ ← job in *seru i'*
          for $\hat{J}_J \leftarrow \hat{J}_1$ **to** $\hat{J}_J$ **do**
          Temporary_plan← Swap the job $\tilde{J}_J$ with $\hat{J}_J$, update the setup time and the resources used in the setups
          Temporary_sol← Apply repairing phase
          if *Temporary_sol* < *Best_sol*, **then**
            Current_plan ← *Temporary_plan*
            Best_plan ← *Current_plan*
            Current_sol ← *Temporary_sol*
            Best_sol ← *Current_sol*
            flag ← 0
          end
        end
      end
    end
  end
end

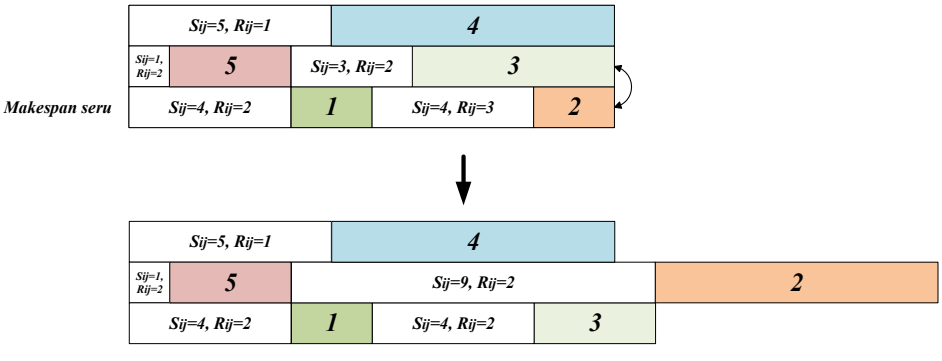
3.3.3 External insertion

External insertion is the job j in the *seru* where makespan is located, defines the *seru* where makespan is located as $seru_{makespan}$, that is, each job j in $seru_{makespan}$, inserted into any location in other *seru*, updating the setup time and the resources required in the setups after the insertion, and performing the repairing phase. Judgment is made for each external insertion, and after computing all possible insertions, we keep the external insertion under the solution with the minimum makespan. The external insertion process is summarised in Algorithm 8 and schematically shown in Figure 6(c).

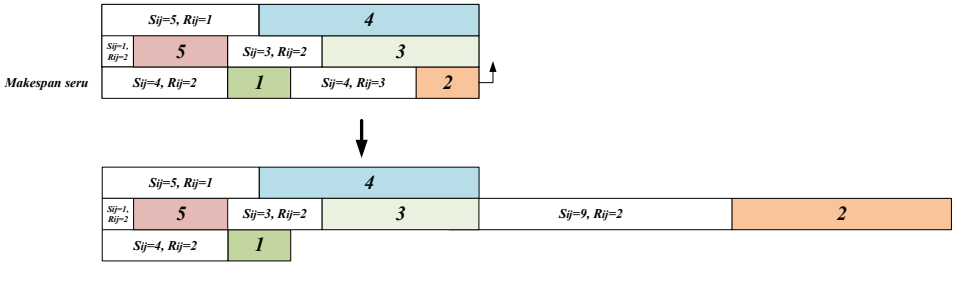
Figure 6 Three types of variable neighbourhood search (a) internal swap of *job 1* with *job 2* in *seru 3* (b) external swap of *job 3* in *seru 2* with *job 2* in *seru 3* (c) external insertion of *job 2* in *seru 3* at last position in *seru 2* (see online version for colours)



(a)



(b)



(c)

Algorithm 8 External insertion

Input *Current_plan, Current_sol*
flag ← 0
$seru_{makespan}$ ← *seru i* that defines the makespan
while flag = 0 **do**
  flag←1
  $\tilde{j}$ ← job in $seru_{makespan}$
  for $\tilde{j}_j \leftarrow \tilde{j}_1$ **to** $\tilde{j}_J$ **do**
    for $i' \leftarrow 1$ **to** I **do**
      if $i' \neq i$, **then**
        $\hat{j}$ ← job in seru i'
        for $\hat{J}_J \leftarrow \hat{J}_1$ **to** $\hat{J}_J$ **do**
          Temporary _plan ← Insertion of job $\tilde{J}_j$ in *seru i'* at any adjacent location for all jobs, update the setup times and the resources used for the sets
          Temporary_sol ← Apply repairing phase
          if *Temporary_sol* < *Best_sol*, **then**
            Current_plan ← *Temporary_plan*
            Best_plan ← *Current_plan*
            Current_sol ← *Temporary_sol*
            Best_sol ← *Current_sol*
            flag ← 0
          end
        end
      end
    end
  end
end

4 Computation experiments

In this section, we conduct computational experiments. Tests are conducted at different scale instances to evaluate the performance of the proposed VNS-IG algorithm to solve the UDSS-SR problem. We test them on randomly generated benchmarks and discuss the results. The generations of instances and the algorithm for solving the UDSS-SR problem proposed in this paper are implemented by MATLAB R2021b. MATLAB software run on a personal computer, including Inter (R) Core (TM) i7-10710U CPU with 1.10GHz speed, 16 GB main memory.

4.1 Data setting

For the instances of the UDSS-SR problem, we choose the combination of the total number of *serus* (I) and the number of jobs (J) to reflect the scale of the experiments. Since the distribution of the resources required in the setups is U = (1, 9), r_{min} = 1, r_{max} = 9, we will calculate the total resources for setups is $R_{\max} = I \times \frac{r_{\max} + r_{\min}}{2} = 5 \times I$, in SPS, I is the total number of *serus*. The other parameters of the UDSS-SR problem are completely random within a given range. The parameters in Table 4 are used to generate the test instances set. $U(a, b)$ is the uniform distribution of random integers between a and b (including the both extremes), which is the most commonly distribution used for generating scheduling problem instances. There are a total of five test instances, denoted by $I \times J$. All instances are repeated 20 times, so the total number of instances to be tested is 100.

Table 4 Parameter settings

Parameters	Value
Instance size $\{I \times J\}$	$\{10 \times 100, 15 \times 200, 20 \times 300, 25 \times 400, 30 \times 500\}$
The total resources for setups R_{max}	$5 \times I$
Setup times $s_{ijj'}$	U = (1, 20)
Resources required for setup $r_{ijj'}$	U = (1, 9)
Processing times p_{ij}	U = (1, 50)

4.2 Experimental results and analysis

To demonstrate the effectiveness of the VNS-IG algorithm, it is compared with the solution obtained from the constructive phase heuristic rule (Heuristic), and the results are presented in Table 5. The improvement ratio δ (%) and the CPU time in seconds are also presented in Table 5, where the improvement ratio δ (%) is calculated by the following equation.

$$\delta = \frac{C_{\max}^{Heuristic} - C_{\max}^{VNS-IG}}{C_{\max}^{Heuristic}} \times 100\%$$

As can be seen from Table 5, the solutions obtained by the VNS-IG algorithm are significantly better than those obtained by Heuristic, and VSN-IG has better performance with significant improvement, and the CPU time for all instances are within the satisfactory range, even for large instances of 30×500. To check whether the differences in δ are statistically significant, the one-way analysis of variance (ANOVA) method is used. We analysed different instance sizes {10 × 100, 15 × 200, 20 × 300, 25 × 400, 30 × 500} using δ as the response variable. Figure 7(a) shows a box plot of different instance sizes at 95% confidence level, Figure 7(b) shows a point line diagram for all instances. Interestingly, we observe that improvements are better in the 20 × 300 and 25×400 instances.

Table 5 Results of heuristic and the VNS-IG algorithm

Size	*Heuristic*		*VNS-IG*		$\delta(\%)$	*Avg.* $\delta(\%)$
	$C_{\max}^{Heuristic}$	*CPU time*(s)	$C_{\max}^{VNS-IG}$	*CPU time*(s)		
10×100	187	0.4311	159	15.4481	14.973	22.492
	230	0.2665	145	28.4886	36.957	
	208	0.3248	144	31.2654	30.769	
	208	0.2699	157	19.9572	24.519	
	221	0.3722	179	12.1266	19.005	
	191	0.3527	141	34.7662	26.178	
	211	0.2768	162	24.3086	23.223	
	225	0.2982	156	30.773	30.667	
	206	0.2661	192	10.3506	6.796	
	266	0.321	181	24.1542	31.955	
	189	0.2776	149	11.1146	21.164	
	221	0.287	161	22.4876	27.149	
	193	0.2594	165	11.6119	14.508	
	225	0.2693	171	14.5139	24.000	
	166	0.2902	149	11.0084	10.241	
	184	0.3943	165	6.8834	10.326	
	214	0.2895	170	7.9003	20.561	
	237	0.3397	173	22.9854	27.004	
	196	0.2821	185	13.2485	5.612	
	303	0.487	169	20.4079	44.224	
15×200	297	1.2447	220	262.641	25.926	20.133
	281	0.9772	230	147.9782	18.149	
	294	1.071	202	186.4302	31.293	
	274	0.9849	204	208.83	25.547	
	249	0.8674	205	140.7985	17.671	
	294	0.9519	205	136.2086	30.272	
	263	1.1511	216	95.5002	17.871	
	294	0.8949	238	67.5731	19.048	
	261	0.9963	213	138.8502	18.391	
	288	0.9377	243	103.1174	15.625	
	345	0.8159	222	93.6121	35.652	
	263	0.8867	223	115.3082	15.209	
	282	0.8006	229	59.1407	18.794	
	290	0.8875	258	58.6048	11.034	
	274	0.6627	239	68.9954	12.774	
	272	0.6012	203	64.4207	25.368	
	255	0.6906	228	40.4328	10.588	
	248	0.5499	202	63.3477	18.548	
	251	0.6913	215	26.4919	14.343	
	292	0.63	232	79.1137	20.548	

Table 5 Results of heuristic and the VNS-IG algorithm (continued)

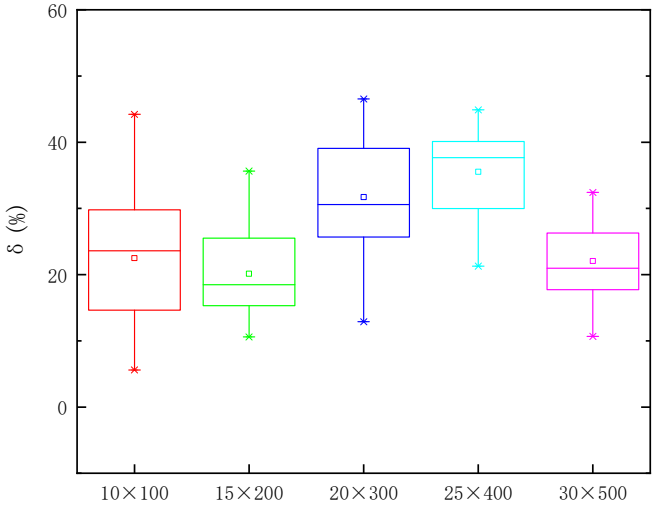
Size	*Heuristic*		*VNS-IG*		$\delta(\%)$	*Avg.* $\delta(\%)$
	$C_{\max}^{Heuristic}$	*CPU time*(s)	$C_{\max}^{VNS-IG}$	*CPU time*(s)		
20×300	432	0.9987	231	351.668	46.528	31.725
	338	1.2367	250	189.4956	26.036	
	393	0.9925	229	190.1129	41.730	
	307	0.9454	231	169.2985	24.756	
	354	1.0586	210	230.8494	40.678	
	327	1.0274	238	76.347	27.217	
	333	1.0164	261	145.5857	21.622	
	356	0.9945	222	151.953	37.640	
	304	1.2753	219	190.9431	27.961	
	418	1.0984	254	191.692	39.234	
	363	1.2609	250	210.3311	31.129	
	380	1.1603	259	104.7934	31.842	
	356	1.0649	249	106.5757	30.056	
	374	1.5002	223	268.6044	40.374	
	356	1.0539	265	76.7074	25.562	
	368	1.0778	226	490.9935	38.587	
	366	1.8422	227	395.0233	37.978	
	326	1.662	284	78.2233	12.883	
	316	2.6752	236	386.8679	25.316	
	307	2.2142	223	489.7231	27.362	
25×400	319	2.1026	239	164.0654	25.078	35.527
	437	1.6555	250	287.3242	42.792	
	373	1.6756	227	409.3324	39.142	
	319	1.7401	225	365.1709	29.467	
	414	1.6988	234	258.0641	43.478	
	343	1.6487	270	278.3417	21.283	
	421	1.6926	239	422.645	43.230	
	426	1.715	278	357.8836	34.742	
	324	1.6482	222	431.603	31.481	
	384	1.6429	273	163.9269	28.906	
	383	1.9596	251	326.898	34.465	
	450	1.6064	248	385.65	44.889	
	379	1.7074	233	368.6817	38.522	
	438	1.7278	269	435.7079	38.584	
	337	1.7548	255	191.9043	24.332	
	398	1.8541	249	531.0397	37.437	

Table 5 Results of heuristic and the VNS-IG algorithm (continued)

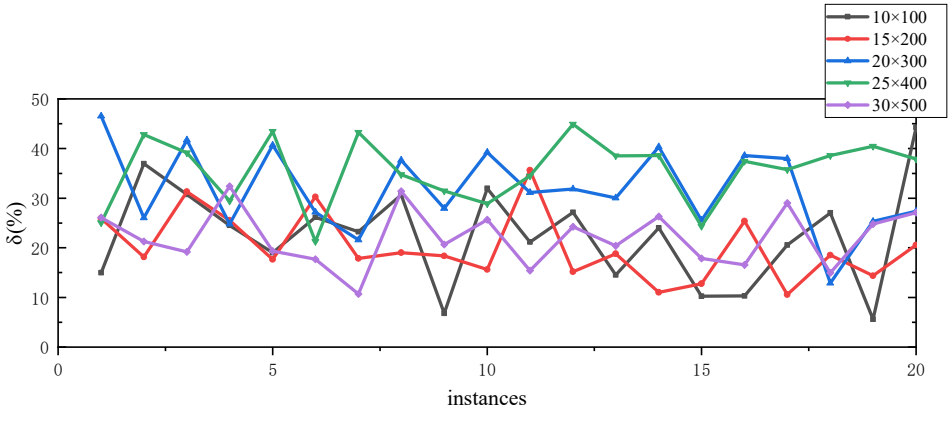
Size	*Heuristic*		*VNS-IG*		$\delta(\%)$	*Avg.* $\delta(\%)$
	$C_{\max}^{Heuristic}$	*CPU time*(s)	$C_{\max}^{VNS-IG}$	*CPU time*(s)		
25×400	386	1.9486	248	263.0265	35.751	35.527
	394	2.2335	242	393.5228	38.579	
	398	1.6529	237	586.7498	40.452	
	396	1.8509	246	359.7934	37.879	
30×500	318	4.1421	235	773.7176	26.101	22.055
	296	2.6542	233	456.1981	21.284	
	360	2.8691	291	361.1639	19.167	
	361	2.5678	244	638.7439	32.410	
	305	3.3037	246	600.3728	19.344	
	322	3.043	265	289.2475	17.702	
	281	2.5527	251	347.1778	10.676	
	366	2.6678	251	474.7473	31.421	
	329	2.6153	261	570.7909	20.669	
	316	2.5832	235	702.3107	25.633	
	318	3.046	269	444.7714	15.409	
	367	2.6672	278	464.8519	24.251	
	353	2.65	281	427.706	20.397	
	338	3.0451	249	393.2359	26.331	
	302	2.7531	248	738.4533	17.881	
	326	2.7031	272	465.4965	16.564	
	348	2.6084	247	984.9874	29.023	
	294	2.5803	250	345.3055	14.966	
	347	2.67	261	423.6897	24.784	
	358	2.5775	261	699.0969	27.095	

In summary, it can be concluded that the solutions obtained by applying the VNS-IG algorithm are always satisfactory. Therefore, in the actual production scheduling process, our proposed VNS-IG algorithm is a good choice for managers, because in addition to being able to obtain a feasible solution to the UDSS-SR problem in an acceptable time, and the feasible solution is satisfactory. Therefore, the VNS-IG algorithm proposed in this paper can be considered as a good method with good performance if we consider both the quality and efficiency of the solution.

Figure 7 Data analysis, (a) ANOVA (b) point line diagram (see online version for colours)



(a)



(b)

5 Conclusions

This paper focuses on the scheduling problem in SPS, which considers both sequence-dependent setup time and resource constraints in the setups to minimise makespan, which helps production managers to strike a balance between production efficiency and resource utilisation and helps manufacturing plants to schedule jobs and additional resources (e.g., worker resources) in a more rational manner. Therefore, we developed a mixed integer linear programming model for UDSS-SR and designed an iterative greedy algorithm based on variable neighbourhood search. Computational experiments show that the proposed solution method is effective for the UDSS-SR

problem and can find good solutions for instances of the proposed problem with a short CPU time.

Future research may add consideration of other practical factors in *seru* scheduling, such as adding additional considerations: limited resource allocation during job processing, etc. In addition, a multi-objective *seru* scheduling model can be considered considering the conflicts between different decision objectives in the actual decision process.

Acknowledgements

This research is sponsored by National Natural Science Foundation of China (Grant Nos. 71401075, 71801129), and System Science and Enterprise Development Research Center (Grant No. Xq22B06). We would like to give our great appreciation to all the reviewers and editors who contributed this research.

References

Allahverdi, A., Gupta, J.N. and Aldowaisan, T. (1999) 'A review of scheduling research involving setup considerations', *Omega*, Vol. No. 272, pp.219–239, https://doi.org/10.1016/ S0305-0483(98)00042-5.

Andradóttir, S., Ayhan, H. and Down, D.G. (2013) 'Design principles for flexible systems', *Production and Operations Management*, Vol. 22, No. 5, pp.1144–1156, https://doi.org/ 10.1111/ poms.12009.

Davis, E. and Jaffe, J.M. (1981) 'Algorithms for scheduling tasks on unrelated processors', *Journal of the ACM* (*JACM*), Vol. 28, No. 4, pp.721–736, https://doi.org/10.1145/322276.322284.

Diana, R.O.M., de França Filho, M.F., de Souza, S.R. and de Almeida Vitor, J.F. (2015) 'An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation', *Neurocomputing*, Vol. 163, pp.94–105, https://doi.org/10.1016/j.neucom.2014.06.091.

Ebrahimi, M., Ghomi, S.F. and Karimi, B. (2014) 'Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates', *Applied Mathematical Modelling*, Vol. 38, Nos. 9–10, pp.2490–2504, https://doi.org/10.1016/j.apm.2013.10.061.

Fanjul-Peyro, L., Ruiz, R. and Perea, F. (2019) 'Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times', *Computers & Operations Research*, Vol. 101, pp.173–182, https://doi.org/10.1016/j.cor.2018.07.007.

Frank, A., Dalenogare, L. and Ayala, N.F. (2019) 'Industry 4.0 technologies: implementation patterns in manufacturing companies', *International Journal of Production Economics*, Vol. 210, pp.15–26, https://doi.org/10.1016/j.ijpe.2019.01.004.

Gai, Y., Yin, Y., Tang, J. and Liu, S. (2020) 'Minimizing makespan of a production batch within concurrent systems: *seru* production perspective', *Journal of Management Science and Engineering*, https://doi.org/10.1016/j.jmse.2020.10.002.

Jiang, Y., Zhang, Z., Gong, X. and Yin, Y. (2021a) 'An exact solution method for solving *seru* scheduling problems with past-sequence-dependent setup time and learning effect', *Computers & Industrial Engineering*, Vol. 158, 107354.https://doi.org/10.1016/j.cie.2021.107354.

Jiang, Y., Zhang, Z., Song, X. and Yin, Y. (2021b) 'Scheduling controllable processing time jobs in *seru* production system with resource allocation', *Journal of the Operational Research Society*, pp.1–21, https://doi.org/10.1080/01605682.2021.1999182.

Kaku, I. (2016) 'A fundamental positive investigation into Japanese *seru* production systems', *IFAC-PapersOnLine*, Vol. 49, No. 12, pp.337–342, https://doi.org/10.1016/j.ifacol.2016.07.627.

Kaku, I. (2017) 'Is *seru* a sustainable manufacturing system?', *Procedia Manufacturing*, Vol. 8, pp.723–730, https://doi.org/10.1016/j.promfg.2017.02.093.

Kaku, I., Gong, J., Tang, J. and Yin, Y. (2009) 'Modeling and numerical analysis of line-cell conversion problems', *International Journal of Production Research*, Vol. 47, No. 8, pp.2055–2078, https://doi.org/10.1080/00207540802275889.

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

Liu, C., Dang, F., Li, W., Lian, J., Evans, S. and Yin, Y. (2015) 'Production planning of multi-stage multi-option *seru* production systems with sustainable measures', *Journal of Cleaner Production*, Vol. 105, pp.285–299, https://doi.org/10.1016/j.jclepro.2014.03.033.

Liu, C., Li, W., Lian, J. and Yin, Y. (2012) 'Reconfiguration of assembly systems: from conveyor assembly line to *serus*', *Journal of Manufacturing Systems*, Vol. 31, No. 3, pp.312–325, https://doi.org/10.1016/j.jmsy.2012.02.003.

Liu, C., Li, Z., Tang, J., Wang, X. and Yao, M.J. (2021a)', How *SERU* production system improves manufacturing flexibility and firm performance: an empirical study in China', *Annals of Operations Research*, pp.1–26, https://doi.org/10.1007/s10479-020-03850-y.

Liu, C., Stecke, K.E., Lian, J. and Yin, Y. (2014) 'An implementation framework for *seru* production', *International Transactions in Operational Research*, Vol. 21, No. 1, pp.1–19, https://doi.org/ 10.1111/itor.12014.

Liu, C., Yang, N., Li, W., Lian, J., Evans, S. and Yin, Y. (2013) 'Training and assignment of multi-skilled workers for implementing *seru* production systems', *The International Journal of Advanced Manufacturing Technology*, Vol. 69, No. 5, pp.937–959, https://doi.org/10.1007/s00170-013-5027-5.

Liu, F., Niu, B., Xing, M., Wu, L. and Feng, Y. (2021b). Optimal cross-trained worker assignment for a hybrid *seru* production system to minimize makespan and workload imbalance. Computers & Industrial Engineering, 160, 107552. https://doi.org/10.1016 /j.cie.2021.107552.

Luo, L., Zhang, Z. and Yin, Y. (2016) '*Seru* loading with worker-operation assignment in single period', in *IEEE International Conference on Industrial Engineering and Engineering Management* (*IEEE IEEM*), December, pp.1055–1058, https://doi.org/10.1109/IEEM.2016.7798039.

Luo, L., Zhang, Z. and Yin, Y. (2017) 'Modelling and numerical analysis of *seru* loading problem under uncertainty', *European Journal of Industrial Engineering*, Vol. 11, No. 2, pp.185–204, https://doi.org/10.1504/EJIE.2017.083255.

Luo, L., Zhang, Z. and Yin, Y. (2021) 'Simulated annealing and genetic algorithm based method for a bi-level loading problem with worker assignment in production systems', *Journal of Industrial & Management Optimization*, Vol. 17, No. 2, p.779, https://doi.org/779.10.3934/jimo.2019134.

Min, Q., Lu, Y., Liu, Z., Su, C. and Wang, B. (2019) 'Machine learning based digital twin framework for production optimization in petrochemical industry', *International Journal of Information Management*, Vol. 49, pp.502–519, https://doi.org/10.1016/j.ijinfomgt.2019.05.020.

Niakan, F., Baboli, A., Moyaux, T. and Botta-Genoulaz, V. (2016) 'A bi-objective model in sustainable dynamic cell formation problem with skill-based worker assignment', *Journal of Manufacturing Systems*, Vol. 38, pp.46–62, https://doi.org/10.1016/j.jmsy.2015.11.001.

Pinheiro, J.C., Arroyo, J.E. C. and Fialho, L.B. (2020) 'Scheduling unrelated parallel machines with family setups and resource constraints to minimize total tardiness', in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, July, pp.1409–1417, https://doi.org/10.1145/3377929.3398150.

Rajkumar, M., Asokan, P., Anilkumar, N. and Page, T. (2011) 'A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints', *International Journal of Production Research*, Vol. 49, No. 8, pp.2409–2423, https://doi.org/10.1080/00207541003709544.

Roth, A., Singhal, J., Singhal, K. and Tang, C. (2016) 'Knowledge creation and dissemination in operations and supply chain management', *Production and Operations Management*, Vol. 25, No. 9, pp.1473–1488, https://doi.org/ 10.1111/poms.12590.

Ruiz, R. and Andrés-Romano, C. (2011) 'Scheduling unrelated parallel machines with resource-assignable sequence-dependent setup times', *The International Journal of Advanced Manufacturing Technology*, Vol. 57, No. 5, pp.777–794, https://doi.org/10.1007/s00170-011-3318-2.

Sakazume, Y. (2005) 'Is Japanese cell manufacturing a new system? A comparative study between Japanese cell manufacturing and cellular manufacturing', *Journal of Japan Industrial Management Association*, Vol. 55, No. 6, pp.341–349, https://doi.org/10.11221/jima.55.341.

Salvendy, G. (Ed.) (2001) *Handbook of Industrial Engineering: Technology and Operations Management*, John Wiley & Sons, Inc., Hoboken, NJ, USA.

Shao, L., Zhang, Z. and Yin, Y. (2016) 'A bi-objective combination optimisation model for line-conversion based on queuing theory', *International Journal of Manufacturing Research*, Vol. 11, No. 4, pp.322–338, https://doi.org/10.1504/IJMR.2016.082821.

Stecke, K.E., Yin, Y., Kaku, I. and Murase, Y. (2012) '*Seru*: the organizational extension of JIT for a super-talent factory', *International Journal of Strategic Decision Sciences* (*IJSDS*), Vol. 3, No. 1, pp.106–119, https://doi.org/10.4018/jsds.2012010104.

Süer, G. and Dagli, C. (2005) 'Intra-cell manpower transfers and cell loading in labor-intensive manufacturing cells', *Computers & Industrial Engineering*, Vol. 48, No. 3, pp.643–655, https://doi.org/ 10.1016/j.cie.2003.03.006.

Sun, W., Wu, Y., Lou, Q. and Yu, Y. (2019) 'A cooperative coevolution algorithm for the *seru* production with minimizing makespan', *IEEE Access*, Vol. 7, pp.5662–5670, https://doi.org/10.1109/ACCESS.2018.2889372.

Sun, W., Yu, Y., Lou, Q., Wang, J. and Guan, Y. (2020) 'Reducing the total tardiness by *Seru* production: model, exact and cooperative coevolution solutions', *International Journal of Production Research*, Vol. 58, No. 21, pp.6441–6452, https://doi.org/10.1080/00207543.2019.1680898.

Takeuchi, N. (2006) *Manufacturing Methods Illustrated: Cell Production System*, JMA Management Center Inc., Tokyo, in Japanese.

Treville, S., Ketokivi, M. and Singhal, V. (2017) 'Competitive manufacturing in a high-cost environment: introduction to the special issue', *Journal of Operations Management*, Vols. 49–51, pp.1–5, https://doi.org/10.1016 /j.jom.2017.02.001

Villa, F., Vallada, E. and Fanjul-Peyro, L. (2018) 'Heuristic algorithms for the unrelated parallel machine scheduling problem with one scarce additional resource', *Expert Systems with Applications*, Vol. 93, pp.28–38, https://doi.org/10.1016/j.eswa.2017.09.054.

Wang, L., Zhang, Z. and Yin, Y. (2022) 'Order acceptance and scheduling problem with outsourcing in production system considering lot-spitting', *European Journal of Industrial Engineering*, Vol. 16, No. 1, pp.91–116, https://doi.org/10.1504/EJIE.2022.119371.

Wang, Y. and Tang, J. (2018) 'Cost and service-level-based model for a *seru* production system formation problem with uncertain demand', *Journal of Systems Science and Systems Engineering*, Vol. 27, No. 4, pp.519–537, https://doi.org/10.1007/s11518-018-5379-3.

Wang, Y. and Tang, J. (2020) 'Optimized skill configuration for the *seru* production system under an uncertain demand', *Annals of Operations Research*, https://doi.org/ 10.1007/s10479-020-03805-3.

Wu, Y., Wang, L. and Chen, J.F. (2021) 'A cooperative coevolution algorithm for complex hybrid *seru*-system scheduling optimization', *Complex & Intelligent Systems*, Vol. 7, No. 5, pp.2559–2576, https://doi.org/10.1007/s40747-021-00432-8.

Yepes-Borrero, J.C., Villa, F., Perea, F. and Caballero-Villalobos, J.P. (2020) 'GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources', *Expert Systems with Applications*, March, Vol. 141, p.112959.

Yılmaz, Ö.F. (2020a) 'Attaining flexibility in *seru* production system by means of Shojinka: an optimization model and solution approaches', *Computers & Operations Research*, Vol. 119, p.104917, https://doi.org/10.1016/j.cor.2020.104917.

Yılmaz, Ö.F. (2020b) 'Operational strategies for *seru* production system: a bi-objective optimisation model and solution methods', *International Journal of Production Research*, Vol. 58, No. 11, pp.3195–3219, https://doi.org/10.1080/00207543.2019.1669841.

Yin, Y., Kaku, I. and Stecke, K. (2008) 'The evolution of *seru* production systems throughout Canon', *Neilson Journals Publishing*, http://dx.doi.org/10.4135/9781526462060.

Yin, Y., Stecke, K.E. and Li, D. (2018) 'The evolution of production systems from Industry 2.0 through Industry 4.0', *International Journal of Production Research*, Vol. 56. Nos. 1–2, pp.848–861, https://doi.org/10.1080/00207543.2017.1403664.

Yin, Y., Stecke, K.E., Swink, M. and Kaku, I. (2017) 'Lessons from *seru* production on manufacturing competitively in a high cost environment', *Journal of Operations Management*, Vol. 49, pp.67–7, .https://doi.org/10.1016/j.jom.2017.01.003.

Ying, K.C. and Tsai, Y.J. (2017) 'Minimising total cost for training and assigning multiskilled workers in *seru* production systems', *International Journal of Production Research*, Vol. 55, No. 10, pp.2978–2989, https://doi.org/10.1080/00207543.2016.1277594.

Yu, Y. and Tang, J. (2019) 'Review of *seru* production', *Frontiers of Engineering Management*, Vol. 6, No. 2, pp.183–192, https://doi.org/10.1007/s42524-019- 0028-1.

Yu, Y., Sun, W., Tang, J. and Wang, J. (2017a) 'Line-hybrid *seru* system conversion: models, complexities, properties, solutions and insights', *Computers & Industrial Engineering*, Vol. 103, pp.282–299, https://doi.org/10.1016/j.cie.2016.11.035.

Yu, Y., Sun, W., Tang, J., Kaku, I. and Wang, J. (2017b) 'Line-*seru* conversion towards reducing worker(s) without increasing makespan: models, exact and meta-heuristic solutions', *International Journal of Production Research*, Vol. 55, No. 10, pp.2990–3007, https://doi.org/ 10.1080/00207543.2017.1284359.

Yu, Y., Wang, J., Ma, K. and Sun, W. (2018) *Seru* system balancing: definition, formulation, and exact solution', *Computers & Industrial Engineering*, Vol. 122, pp.318–325, https://doi.org/ 10.1016/j.cie.2018.05.048.

Yu, Y., Wang, S., Tang, J., Kaku, I. and Sun, W. (2016) 'Complexity of line-*seru* conversion for different scheduling rules and two improved exact algorithms for the multi-objective optimization', *SpringerPlus*, Vol. 5, No. 1, pp.1–26, https://doi.org/10.1186/s40064-016-2445-5.

Zhang, X., Liu, C., Li, W., Evans, S. and Yin, Y. (2017) 'Effects of key enabling technologies for *seru* production on sustainable performance', *Omega*, Vol. 66, pp.290–307, https://doi.org/ 10.1016/j.omega.2016.01.013.

Zhang, Z., Song, X., Huang, H., Zhou, X. and Yin, Y. (2022a) '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*, Vol. 297, No. 3, pp.866–877, https://doi.org/10.1016/j.ejor.2021.06.017.

Zhang, Z., Gong, X., Song, X., Yin, Y., Lev B. and Chen, J. (2022b) 'A column generation-based exact solution method for *seru* scheduling problems', *Omega*, Vol. 108, p.102581, https://doi.org/ 10.1016/j.omega.2021.102581.

Zhang, Z., Song, X., Huang, H., Yin, Y. and Lev, B. (2022c) 'Scheduling problem in *seru* production system considering DeJong's learning effect and job splitting', *Annals of Operations Research*, https://doi.org/ 10.1007/s10479-021-04515-0.

Zhang, Z., Song, X., Gong, X., Yin, Y., Lev, B. and Zhou, X. (2022d) 'An effective heuristic based on 3-opt strategy for *seru* scheduling problems with learning effect', *International Journal of Production Research*, https://doi.org/ 10.1080/00207543.2022.2054744.

Zhang, Z., Shao, L. and Yin, Y. (2020) 'PSO-based algorithm for solving lot splitting in unbalanced *seru* production system', *International Journal of Industrial and Systems Engineering*, Vol. 35, No. 4, pp.433–450, https://doi.org/10.1504/IJISE.2020.108547.

Zhang, Z., Wang, L., Song, X., Huang, H. and Yin, Y. (2021) 'Improved genetic-simulated annealing algorithm for *seru* loading problem with downward substitution under stochastic environment', *Journal of the Operational Research Society*, pp.1–12, https://doi.org/10.1080/01605682.2021.1939172.