A two-stage heuristic algorithm for dynamic *seru* scheduling problems with resource constraints

Yiran Xiang[1], Zhe Zhang[1]*, Xiaoling Song[1], Xue Gong[1], Yong Yin[2]
[1]School of Economics and Management, Nanjing University of Science and Technology, Nanjing, 210094, P. R. China
[2]Graduate School of Business, Doshisha University, Karasuma-Imadegawa, Kamigyo-ku, Kyoto, 602-8580, Japan
*Corresponding author: email: zhangzhe@njust.edu.cn (Zhe Zhang); tel:+86-13451933538

Abstract. Originating from the Japanese electronics assembly industry, the *seru* production mode offers high efficiency, flexibility, and rapid responsiveness in manufacturing. This paper addresses the unspecified dynamic *seru* scheduling problem with resource constraints (UDSS-R), where resource usage must not exceed the available total at any given time. The UDSS-R problem is formulated as a mixed-integer linear programming (MILP) model aimed at minimizing the makespan. A two-stage heuristic algorithm is proposed subsequently: the first stage addresses the regular *seru* scheduling problem (without resource constraints) by assigning jobs to *serus*, and the second stage uses a dynamic programming algorithm based on the 0-1 knapsack problem to finalize the schedule. Computational experiments demonstrate the practicality and effectiveness of the proposed MILP model and the two-stage heuristic algorithm in solving the UDSS-R problem.

Keywords: scheduling; *seru* production; dynamic; resource constraint; heuristic algorithm

1. Introduction

With the widespread application of emerging information technology and artificial intelligence, manufacturing systems need to be highly flexible and efficient to respond quickly to volatile market changes (Lin et al., 2022 [1]). It has brought significant challenges to the manufacturing industry, particularly high-tech sectors such as electronics (Frank et al., 2019 [2]; Senkul & Toroslu, 2005 [3]). In this case, Japanese companies like Sony and Canon have developed a Japanese-style cellular manufacturing, known as *seru* (the Japanese pronunciation of cell) production, distinguishing it from traditional cellular manufacturing (Liu et al., 2010 [4]).

Seru is an assembly unit consisting of one or several workers, movable workstations, and cheap, light equipment (Kaku et al., 2009 [5]). *Seru* production system (SPS) comprises several *serus*, and each *seru* within SPS can be quickly constructed, modified, disassembled and rebuilt, allowing the SPS to respond rapidly to unexpected changes (Jiang et al., 2022 [6]; Ren et al., 2024 [7]). SPS includes three *seru* types: divisional *seru*, rotating *seru*, and *yatai* (Luo et al., 2017 [8]; Jiang et al., 2021 [9]). When the conveyor assembly line is reconfigured, the first *seru* type formed is the divisional *seru*, where production tasks are distributed among several workers in the unit, with each worker performing a certain number of tasks. In the rotating *seru*, although there are several workers in the unit, each worker operates all the production tasks from start to finish, rotating within the *seru*. In y*atai,* one worker handles all the production tasks entirely from start to finish (Ying & Tsai, 2017 [10]; Yu & Tang, 2018 [11]), respectively, see Fig. 1.

In practice, SPS can quickly produce, assemble, and transport small to medium batches of customized products, ensuring efficiency, flexibility, and responsiveness. Many enterprises have achieved significant benefits after implementing SPS. Yin et al. (2017) [12] provided detailed data on the benefits of implementing SPS in Canon and Sony, and conducted an empirical analysis of the flexibility and quality of the SPS. SPS has also been adopted in Europe, South Korea, China and other countries (Liu et al., 2022 [13]). Other benefits of SPS include reduced production time, setup time, labor hours, work-in-process inventory and finished product inventory (Sengupta & Jacobs, 2004 [14]). Although *seru* production has achieved great success in practice, research on *seru* production still has a long way to go due to its relatively short history. Nevertheless, SPS has garnered increasing attention from researchers and practitioners because of its high flexibility and efficiency. *Seru* is gradually becoming an alternative to lean systems approaches and has been regarded as the "next generation of lean" in recent years (Liu et al., 2015 [15]). It attracted several well-known scholars to conduct a series of valuable studies (Roth et al., 2016 [16]; Treville et al., 2017 [17]). In SPS, the just-in-time organization system (JIT-OS) is key to achieving high performance (Stecke et al., 2012 [18]; Yin et al., 2018 [19]). JIT-OS involves three decision-making stages: *seru* formation, *seru* loading and *seru* scheduling. The first two stages focus on preparing materials and equipment for production, while *seru* scheduling involves detailed job processing plans (such as job sequencing, worker assignment, resource allocation, etc.) in each *seru* (Luo et al., 2021 [20]).

This paper will investigate the *seru* scheduling problem. Specifically, we consider the scheduling problem in SPS (SPS-s) with an additional consideration: each job in a *seru* requires a specific resource, such as workers, tools, or fixtures. We refer to this problem as the unspecified dynamic *seru* production system scheduling problem with resource constraints (UDSS-R). This study focuses on UDSS-R with worker resource constraints, assuming all assignable workers are multi-skilled and capable of handling all types of operations. The term "unspecified" indicates there is no pre-fixed job-*seru* assignment, while "dynamic" means the worker-to-*seru* assignments can change over time. UDSS-R involves scheduling a set of jobs to multiple *serus*, where each job requires a certain number of workers, and the total number of workers in the SPS is fixed. The problem constraints require that the number of workers utilized at any time does not exceed the total available.

Among the various objectives in scheduling problems, the minimization of the makespan (denoted by C_{max}), which is the maximum job completion time, is the most commonly studied. This paper aims to minimize the makespan for UDSS-R by designing a two-stage heuristic algorithm. A dynamic programming algorithm based on the 0-1 knapsack problem is proposed to solve the UDSS-R problem of worker assignment. Assuming that processing a job in a *seru* requires a certain number of multi-skilled workers, the number of required workers varies depending on the job and the *seru* assigned to it. This approach makes the problem more realistic and better suited to guide actual production practices.

The rest of the paper is structured as follows: Section 2 outlines the literature related to the problem studied. Section 3 formally introduces the issues discussed in this paper. Section 4 introduces the complete mathematical model of UDSS-R. In Section 5, a two-stage heuristic algorithm is designed. Section 6 performs computational tests on the proposed algorithm. Finally, in Section 7, the research conclusions and future research directions are given.

2. Literature review

The existing research on SPS mainly focuses on *seru* formation, *seru* loading, and *seru* scheduling (Zhang et al., 2022a [21]). Through the formation of *seru*, the appropriate SPS is configured, and then, through the loading and scheduling of the *seru*, the job will be assigned to *seru* to execute the production plan (Zhang et al., 2024 [22]). For *seru* formation, Liu et al. (2014) [23] provided practitioners with a general framework and several basic principles that should be followed when implementing *seru* production from a practical perspective. Yu et al. (2014) [24] revealed mathematical characteristics of *seru*, such as solution space, combinatorial complexity and non-convex properties. Scholars have conducted research on line-*seru* conversion such as Shao et al., 2016 [25]; Yu et al., 2017 [26]. For *seru* loading, Wang et al. (2020) [27] considered the *seru* loading problem of order acceptance and designed a genetic algorithm with matrix crossover. Zhang et al. (2022a) [21] solved the *seru* loading problem with downward substitution and random product demand and yields. For *seru* scheduling, Wang et al. (2022) [28] focused on the order acceptance and scheduling problem, considering lot-spitting with outsourcing decisions simultaneously in SPS. Zhang et al. (2022b) [29] designed a column generation-based exact solution method for *seru* scheduling problems, and Zhang et al. (2022c) [30] provided a logic-based Benders decomposition method for the *seru* scheduling problem with sequence-dependent setup time and DeJong's learning effect. Considering DeJong's learning effect and job splitting, Zhang et al. (2022d) [31] constructed a nonlinear integer programming model for the *seru* scheduling problem and designed a branch-and-bound algorithm for small-sized problems while a local search-based hybrid genetic algorithm for large-sized problems. Li et al. (2024) [32] investigated the application of dynamic NSGA-II and multi-population search to handle alterations in production schedules caused by various dynamic events. Based on the literature, many factors affect the optimal results of SPS-s, such as setup time, arrival date, batch segmentation, assignment of multi-skilled workers, and the learning effect (Zhang et al., 2023 [33]; Li et al., 2023 [34]). However, most studies on SPS-s in the scientific literature have not considered *seru* production system scheduling with resource constraints. In a practical manufacturing environment, the resources in SPS are limited, highlighting the gap between academic research and the actual needs of the production sector. In this paper, we will study the *seru* scheduling problem considering resource constraints.

Generally, there are two types scheduling problems under resource constraints: static and dynamic (Abbaszadeh et al. (2021) [35]). In the static type, resource allocation is fixed throughout the scheduling process (Daniels et al., 1996 [36], Hasannia-Kolagar et al. (2023) [37]), while in the dynamic type, resources can be assigned and reassigned according to job allocation (Edis & Oguz, 2012 [38]). In the existing literature, the resource allocation of SPS is typically static, meaning that resource allocation to *seru* is given and fixed for the entire time range, and few studies on *seru* scheduling have considered dynamic scheduling in SPS under resource constraints. Fortunately, since SPS is a typical parallel production system, unrelated parallel machine scheduling (uPMS) problems provide us inspirations. Fanjul-Peyro et al. (2017) [39] proposed mathematical models and meta-heuristics for uPMS problems with additional resource constraints. Yunusoglu and Topaloglu (2022) [40] proposed two branching strategy constraint programming (CP) models to reduce computation time. Villa et al. (2018) [41] studied uPMS problems with one scarce additional resource and proposed two heuristic strategies. Fleszar and Hindi (2018) [42] proposed a two-stage heuristic algorithm for uPMS problems with a renewable resource constraint, where the job is first assigned to the machine and then the CP model is used to schedule the jobs on the machine. Yepes-Borrero et al. (2020) [43] designed three meta-heuristic algorithms to solve uPMS problems with setup time and additional resources. Yepes-Borrero et al. (2021) [44] proposed a Pareto frontier search algorithm based on iterative greedy method to solve the bi-objective uPMS problems with setup time and additional resources.

In this paper, we will study the UDSS-R problem, which has been largely overlooked in existing literature. We introduce dynamic scheduling considerations in *seru* production systems under resource constraints. The research aims to bridge the gap between academic studies and practical needs in SPS environments. By exploring dynamic scheduling, this study seeks to enhance current methodologies of *seru* scheduling problems and provide guidance for practical production in SPS.

3. Problem description

In this paper, the SPS-s problem involves a set of *n* jobs, where each job can be processed by *m* *serus* starting at time 0, without preemption. The processing time p_{ij} of job *j* depends on the *seru* *i* that schedule it. Let x_{ij} be a binary variable, indicating whether job *j* is assigned to *seru i*, and C_{max} be the makespan. The following linear programming model can be formulated to solve the SPS-s problem.

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

$$\sum_i x_{ij} = 1, \forall j \tag{2}$$

$$\sum_j p_{ij} x_{ij} \le \mathrm{C}_{max}, \forall i \tag{3}$$

$$x_{ij} \in \{0,1\} \tag{4}$$

where constraints (2) ensure that each job is assigned to only one *seru*, constraints (3) ensure that the makespan is at least as large as the total time occupied by each *seru*, and constraints (4) ensure that the assigned variable is binary. To illustrate what UDSS-R is and the necessity of considering resource constraints, a lower bound

$$\mathrm{LB} = \frac{1}{W_{max}} \sum_i \sum_j w_{ij} p_{ij} x_{ij} \le C_{max} \tag{5}$$

is added to the SPS-s model (Grigoriev et al., 2005 [45], Nasiri and Hamid, 2020 [46]), which can be strengthened by adding an aggregate resource constraint, based on calculating the minimum total resource requirement and dividing it by the resource availability. Specifically, in this paper, w represents worker resources, and assume that all assignable workers are multi-skilled and capable of handling all types of operations. The following example demonstrates the differences between SPS-s and UDSS-R.

Example 1. Consider an instance of UDSS-R with a worker resource constraint, where *m*=3 *serus*, *n*=6 jobs, and a maximum of $W_{max}=5$ workers can be allocated. The specific processing data is shown in Table 1.

Use the established SPS-s model to assign jobs to *serus*, arbitrarily sort the jobs in each *seru*, and get the solution of $C_{max}=11$ in Fig. 2a. However, this solution is infeasible: between time 0 and time 1, and between time 4 and time 7, the total number of workers used is too high, exceeding the worker resource constraints. By keeping the job-*seru* allocation unchanged, the feasible solution shown in Fig. 2b can be obtained by introducing idle time, with $C_{max}=13$, which satisfies the worker resource constraints in SPS, but the makespan is increased to 13 units, and two *serus* have idle time. The optimal UDSS-R solution is shown in Fig. 2c, with $C_{max}=12$, it can be observed that worker resources are better utilized and idle time is reduced.

Through the analysis of Example 1, it is evident that the SPS-s problem is different from the UDSS-R problem. The UDSS-R problem is much more complicated because the job-*seru* assignment and the start and completion time of each job must be determined. Its solution not only needs to assign jobs to *serus* but also needs to ensure that worker resources are not over-utilized at any time when scheduling jobs in the *seru*. Additionally, sometimes due to a shortage of workers, a *seru* may not be able to process the next job, resulting in idle time.

In the UDSS-R, the resources could be workers, tools, fixtures etc. In fact, worker resource constraints are a very important issue in the production process (Nembhard & Bentefouet, 2014 [47]; Su et al., 2021 [48]). Yılmaz (2020) [49] pointed out that the worker resource in an SPS is critical for adapting to changes in demand. Without loss of generality, we consider the UDSS-R with worker resource constraints, and assume that all assignable workers are completely multi-skilled, meaning all workers can handle all types of operations. In this paper, the *seru* in SPS is regarded as a black box, and the proposed model and methods are applicable to all *seru* types.

The UDSS-R problem in this paper consists of finding the best allocation of *n* jobs to *m serus*, while determining the best order for each *seru* to satisfy the worker resource constraints at any time, the objective is to minimize the makespan.

4. Mathematical model

In this section, a mixed-integer linear programming (MILP) model for solving the UDSS-R problem is constructed, where processing each job in each *seru* requires a certain number of workers, and the total number of worker resources in the entire SPS is limited. Assume that all jobs can be processed in all *serus*, and a job cannot be interrupted once it is being processed in a *seru*. The processing time and the number of workers required for each job in each *seru* are known and are related to both the job and the *seru*. The solution to UDSS-R includes the set of jobs that should be processed in each *seru*, the order of the jobs representing the processing sequence in the *seru*, and the start and end times of each job. Due to worker resource constraints, idle time may be necessary to obtain a feasible solution.

4.1 Notations

For convenience, following notations are introduced.

Indices

$i = 1, 2, ..., m$	Index for *serus*
$j = 1, 2, ..., n$	Index for jobs
$t = 1, 2, ..., T_{max}$	Index for time

Parameters

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

w_{ij} number of workers required to process job *j* in *seru i*

W_{max} total number of worker resources in SPS

Decision Variables

C_{max} makespan

X_{ijt} binary variable takes value 1 if job *j* is assigned to *seru i* and completes its processing at time *t*, and zero otherwise. Note that this variable only exists for $t \geq p_{ij}$.

4.2 Mathematical formulation

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

$$min \;\; C_{max} \tag{6}$$

Determine the makespan:

$$\sum_{i} \sum_{t \geq p_{ij}} t X_{ijt} \leq C_{max}, \forall j \tag{7}$$

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

$$\sum_{i} \sum_{t \geq p_{ij}} X_{ijt} = 1, \forall j \tag{8}$$

Ensure that each *seru* does not process more than one job at any time, thus:

$$\sum_{j} \sum_{s \in \{max\{t, p_{ij}\}, \ldots, t+p_{ij}-1\}} X_{ijs} \le 1, \forall i, t \tag{9}$$

Ensure that the workers used at any time do not exceed the total number of worker resource W_{max} in SPS, so:

$$\sum_{i} \sum_{j} \sum_{s \in \{max\{t, p_{ij}\}, \ldots, t+p_{ij}-1\}} w_{ij} X_{ijs} \le \mathrm{W}_{max}, \forall t \tag{10}$$

where

$$\mathrm{X}_{ijt} \in \{0,1\}, \forall i, j, t \tag{11}$$

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

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

$$s.t. \begin{cases} \sum_{i} \sum_{t \ge p_{ij}} t X_{ijt} \le C_{max}, \forall j \\ \sum_{i} \sum_{t \ge p_{ij}} X_{ijt} = 1, \forall j \\ \sum_{j} \sum_{s \in \{max\{t, p_{ij}\}, \ldots, t+p_{ij}-1\}} X_{ijs} \le 1, \forall i, t \\ \sum_{i} \sum_{j} \sum_{s \in \{max\{t, p_{ij}\}, \ldots, t+p_{ij}-1\}} w_{ij} X_{ijs} \le \mathrm{W}_{max}, \forall t \\ \mathrm{X}_{ijt} \in \{0,1\}, \forall i, j, t \end{cases} \tag{12}$$

5. Two-stage heuristic algorithm for UDSS-R

5.1 Solution procedure

Solving the UDSS-R problem requires determining the assignment of jobs under worker resource constraints, scheduling the sequence of jobs in each *seru* according to these constraints, and obtaining the start and end times of each job. Due to the strong NP-hard nature of the *seru* scheduling problem (Yu & Tang, 2019 [50]), the proposed MILP model cannot solve large-scale problems. To obtain a feasible solution within an acceptable time frame, a two-stage heuristic algorithm integrating dynamic programming (DP) technology is designed. The two-stage heuristic algorithm proposed in this paper first uses the MILP model to solve the SPS-s problem for assigning jobs to *serus*, and then performs scheduling using the DP algorithm based on the 0-1 knapsack problem. The two-stage heuristic algorithm for UDSS-R is shown as follows. The flow chart of the whole algorithm is shown in Fig. 3.

Step 1: Use the MILP model used to solve the SPS-s problem for assigning jobs to *serus*.

Step 2: Execute scheduling using the DP algorithm based on the 0-1 knapsack problem.

5.2 The first stage

In the first stage, job-*seru* assignment can be completed very rapidly using the CPLEX solver. Through the CPLEX solver, we obtain the optimal solution to the SPS-s problem, denoted as S^*. This solution assigns jobs to *serus*, and S^* at this point is a 0-1 matrix representing the job-*seru* assignment relationship. If job *j* is allocated to *seru i*, it is 1; otherwise, it is 0. After that, the final allocation matrix of job-*seru* can be obtained, which is composed of the job matrix, the processing time matrix, and the needed workers matrix, as shown in matrix form in Fig. 4. Each column of the matrix represents a time stage, and each row represents the job sequence number assigned to a *seru*, the corresponding processing time, and the number of workers required. Its pseudo code is shown in **Algorithm 1**, where *j_original* is the job matrix, *p_original* and *w_original* are the corresponding processing time and needed workers matrix.

Algorithm 1: Stage 1. Job-*seru* assignment

Input: *m*, *n*, W_{max}, Matrix: $j\_original$, $p\_original$, $w\_original$

Output: Matrix: j_start, p_start, w_start, $distr$

Initialization $S^* \leftarrow [\ ]$, $j\_o \leftarrow [\ ]$, $p\_o \leftarrow [\ ]$, $w\_o \leftarrow [\ ]$, $jobcount \leftarrow [\ ]$

S^* ← Use the CPLEX solver to solve the SPS-s problem

$j\_o = S^* .* j\_original$

$p\_o = S^* .* p\_original$

$w\_o = S^* .* w\_original$

for *i*←1 **to** *m* **do** // Calculate the maximum number of jobs in *seru* in SPS

    $jobcount(i,:) = sum\left(S^* * (i,:)\right)$

end

$jobcountmax = max(jobcount)$

$j\_start = zeros(m, jobcountmax)$

$p\_start = zeros(m, jobcountmax)$

$w\_start = zeros(m, jobcountmax)$

for *i*←1 **to** *m* **do**

    j_start ← Put the jobs assigned to *seru i* into the j_start matrix in order

    p_start ← The processing time matrix corresponds to the j_start matrix

    w_start ← The needed worker matrix corresponds to the j_start matrix

end

$distr = [j\_start; p\_start; w\_start]$; Generate a matrix for storing job-*seru* assignment result.

Taking Example 1 as an example, *j_original*, *p_original*, and *w_original* are as follows:

$$j\_original = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix}$$

$$p\_original = \begin{pmatrix} 8 & 8 & 5 & 7 & 8 & 7 \\ 8 & 4 & 9 & 7 & 3 & 7 \\ 1 & 9 & 10 & 9 & 6 & 2 \end{pmatrix}$$

$$w\_original = \begin{pmatrix} 1 & 3 & 2 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 \\ 3 & 3 & 1 & 2 & 2 & 1 \end{pmatrix}$$

5.3 The second stage

In the second stage, the jobs in each *seru* are scheduled according to the constraints on the total number of worker resources in SPS, following these steps.

Step 1: Fix job_{11} in *seru* 1 at Time stage 1, set the processing time p_{11} of job_{11} to the value T, and the jobs in the rest of the *serus* move right one time stage. At this time, consider which jobs in $seru\ i(i \neq 1)$ should be scheduled to Time stage 1. Think of Time stage 1 as a knapsack and use the **0-1 knapsack algorithm** to complete the scheduling of Time stage 1, the jobs scheduled into Time stage 1 are recorded as 1; otherwise, they are recorded as 0. The scheduling of Time stage 1 is then completed. The application of the **0-1 knapsack algorithm** is shown in Fig. 5.

Knapsack parameter setting: The knapsack load is the number of workers that can be allocated, calculated as $W_{last} = W_{max} - w_{11}$ at this time stage. The goods are the jobs with the longest processing time in each *seru* which meet the processing time constraint $p_{ij'} \leq T$ and the worker resource constraint $w_{ij'} \leq W_{last}$. The weight is the number of workers required for each job, and the value is the processing time of each job. The goal is to maximize the total value of the jobs put in the knapsack. The pseudo code of the **0-1 knapsack algorithm** is shown in **Algorithm 2**.

Algorithm 2: Define function $knapsack\ (P^*, W^*, W_{last})$

Input: P^*, W^*, W_{last}

Output: job_putin

1 Initialization $f \leftarrow [\ ]$, $job\_putin \leftarrow [\ ]$

2 $n = length(W^*)$

3 **for** $j \leftarrow 1$ **to** W_{last} **do**

if $W^*(n) \le j$, **then**
$f(n, j) \leftarrow P^*(n)$
else
$f(n, j) \leftarrow 0$
end
end
for $i \leftarrow n-1$ **to** 1 **do**
for $j \leftarrow 1$ to W_{last} **do**
if $j \le W^*(i)$, **then**
$f(i, j) \leftarrow f(i+1, j)$
else
$f(i, j) \leftarrow max\Big(f\left(i+1, j\right), f\left(i+1, j-W^*\left(i\right)\right)+P^*\left(i\right)\Big)$
end
end
end
$j \leftarrow W_{last}$
for $i \leftarrow 1$ **to** $n-1$ **do**
if $f(i, j) == f(i+1, j)$, **then**
$job\_putin(i) = 0$
else
$job\_putin(i) = 1$
$j = j - W^*(i)$
end
end
if $f(n, j) == 0$, **then**
$job\_putin(n) = 0$
else
$job\_putin(n) = 1$
end
end

Step 2: Remove the scheduled Time stage 1 from the distr matrix and record it in the fea matrix.

Step 3: Repeat Steps 1 and 2 until all jobs in *seru* 1 are scheduled. If all jobs in SPS are scheduled at this time, go to Step 7; otherwise, go to Step 4.

Step 4: Set the processing time of the job with the longest processing time in Time stage 1 of the distr matrix to the value T, and move the jobs in the rest of the *serus* one time stage to the right. Treat Time stage 1 as a knapsack, and use the ***0-1 knapsack algorithm*** to complete the scheduling of Time stage 1, using the same method as in Step 1.

Step 5: Remove the scheduled Time stage from the distr matrix and record it in the fea matrix.

Step 6: Repeat Steps 4 and 5 until all jobs are scheduled.

Step 7: Improve the fea matrix. If there is an idle *seru* in a certain time stage and the remaining worker resources of this time stage $W_{last} > 0$, then filter the jobs behind the Time stage in this idle *seru*. Find the job with the largest number of workers that satisfies the time constraint $p_{ij'} \le T$ and the worker resource constraint $w_{ij'} \le W_{last}$, and schedule this job into this Time stage. Note that the remaining worker resources W_{last} should be updated after the job is placed. Repeat the above process until all the time stages are improved.

Step 8: Remove the columns of the Time stages without job scheduling from the fea matrix to obtain the final result matrix.

The pseudo code for the key steps is shown in **Algorithm 3**.

Algorithm 3: Stage 2. *Seru* scheduling

Input: *m*, *n*, W_{max}, Matrix: $distr$

Output: Matrix: fea

Step 1-3

Initialization $fea \leftarrow [\,]$

nn ←Number of jobs in *seru* 1

for $j \leftarrow 1$ **to** nn **do**

$j\_start = distr(1:m,:)$

$p\_start = distr(m+1:2*m,:)$

$w\_start = distr(2*m+1:3*m,:)$

Fix job_{11} in *seru* 1 at Time stage 1, and the jobs in the rest of the *serus* move right one time stage.

T ← set the processing time p_{11} of job_{11} to the value T,

$W_{last} \leftarrow W_{max} - w_{11}$

J^* ← Select at most one job j' from each *seru* i $(i \ne 1)$, which satisfies $p_{ij'} \le T$, $w_{ij'} \le W_{last}$, and the processing time $p_{ij'}$ is the longest

$P^* \leftarrow J^*$ corresponding processing time set

$W^* \leftarrow J^*$ corresponding needed workers set
if the number of J^* is 1, **then**
$fea \leftarrow$ schedule this job directly into Time stage 1 corresponding to the *seru*
if J^* is empty set, **then**
$fea \leftarrow fea$
else
$[job\_putin] \leftarrow knapsack(P^*, W^*, W_{last})$ // 0-1 knapsack algorithm
$fea \leftarrow$ Schedule the job in job_putin into Time stage 1 corresponding to the *seru*
end
Remove the scheduled Time stage 1 from the $distr$ matrix
end

Step 4-6

while $mm \neq 0$ **do**
$mm \leftarrow$ The number of unscheduled products in SPS
$T \leftarrow$ The maximum processing time $p_{i'j'}$ in the first column of the $distr$ matrix corresponds to the job j'
The jobs in the rest of the *serus* move right one time stage
$W_{last} \leftarrow W_{max} - w_{i'j'}$
$J^* \leftarrow$ Select at most one job j'' from each *seru* i $(i \neq i')$, which satisfies $p_{ij''} \leq T$, $w_{ij''} \leq W_{last}$, and the processing time $p_{ij''}$ is the longest
$P^* \leftarrow J^*$ corresponding processing time set
$W^* \leftarrow J^*$ corresponding needed workers set
if the number of J^* is 1, **then**
$fea \leftarrow$ schedule this job directly into Time stage 1 corresponding to the *seru*
if J^* is empty set, **then**
$fea \leftarrow fea$
else
$[job\_putin] \leftarrow knapsack(P^*, W^*, W_{last})$; //0-1 knapsack algorithm

$fea \leftarrow$ Schedule the job in job_putin into Time stage 1 corresponding to the *seru*

end

Remove the scheduled Time stage 1 from the $distr$ matrix

end

The result matrix, *fea* matrix, is the final solution to the UDSS-R problem. It contains three sub-matrices: the job matrix, the processing time matrix, and the needed workers matrix. The job matrix represents the job-*seru* assignment and the job sequence within each *seru*, the processing time matrix indicates that the start time of each job *j* is the start time TIME of the Time stage where the job is located, and the end time is $TIME + p_{ij}$.

6. Computational experiments

In this section, we conduct computational experiments and test different-sized instances to evaluate the performance of the proposed MILP model and the two-stage heuristic algorithm in solving the UDSS-R problem. All instances are randomly generated, and the results are discussed. The SPS-s model is solved by CPLEX to obtain the optimal solution. The generation of instances and the algorithm for solving the UDSS-R problem proposed in this paper are implemented in MATLAB. Both CPLEX and MATLAB software run on a personal computer equipped with an Inter (R) Core (TM) i7-10710U CPU at 1.10GHz and 16 GB of main memory.

6.1 Data setting

For the instances of the UDSS-R problem, we choose the combination of the total number of *serus* (*m*) and the number of jobs (*n*) to reflect the scale of the experiment. Since the distribution of the number of workers required is $U(1,9)$, with $w_{min} = 1$ and $w_{max} = 9$, we calculate the total number of workers $W_{max} = m \times (\mathrm{w}_{max} + w_{min}) / 2 = 5 \times m$ in SPS. The other parameters of the UDSS-R problem are completely random within a given range. The parameters in Table 2 are used to generate the test instances set. $U(a,b)$ is the uniform distribution of random integers between *a* and *b* (including both extremes), which is the most commonly used distribution for generating scheduling problem instances. There are a total of 4 test instances, denoted by $n \times m$, 15×1000 is considered small-size, 30×2000 and 60×5000 are considered medium-size, 100×10000 is deemed large-size. All instances are repeated 50 times, so the total number of instances to be tested is 200.

6.2 Experimental results and analysis

In order to evaluate the performance of the proposed model and the two-stage heuristic algorithm, we use CPU time and relative percentage deviation (RPD) as numerical indicators. Appendix Table 1 records the total CPU time of all instances, while Appendix Table 2 records the CPU time used by the dynamic programming algorithm based on the 0-1 knapsack problem (CPU_KG) in the second stage. Appendix Table 3 records the RPD of all instances. The point-line diagram for the RPD of all instances is shown in Fig. 6. The RPD of each tested instance is measured as follow:

$$RPD = \frac{Method_{sol} - LB}{LB} \times 100\%$$

Where $Method_{sol}$ is the solution obtained by the tested algorithm, and LB is a hypothetical lower bound, which is the optimal solution *makespan* of the SPS-s problem. In our instances, this may be an unattainable value.

Since 50 tests are performed on instances of different sizes, Table 3 summarizes the test results for different instance sizes.

It can be seen from Fig. 6 and Table 3 that for different-sized instances, the solution obtained using the two-stage heuristic algorithm is stable, and the RPD is stable within an acceptable range. The total CPU time for all instances is within a satisfactory range, even for large-size instances of 100×10000, the longest total CPU time is only 540.4948s. Additionally, the comparison between the total CPU time and the CPU_KG time shows that most of the CPU time is consumed in the CPLEX solver of the first stage, while the second stage *seru* scheduling can almost be ignored, with the longest time being only 16.4867s. Therefore, it can be concluded that the solution obtained using the two-stage heuristic algorithm is always satisfactory. In the actual production scheduling process, the two-stage heuristic algorithm we propose is a good choice for managers, because it can obtain a feasible solution to the UDSS-R problem within an acceptable time, and the solution is satisfactory. If we consider both the quality and efficiency of the solution, the two-stage heuristic algorithm proposed in this paper can be regarded as a good algorithm with good performance.

7. Conclusion

With the objective of minimizing the makespan, this paper studies the unspecified dynamic *seru* production system scheduling problem with resource constraints (UDSS-R), which requires that the number of workers used at any time does not exceed the total number of workers in the SPS. The models for the scheduling problem in SPS (SPS-s) and UDSS-R are presented in turn, followed by a two-stage heuristic algorithm. Computational experiments are conducted on different-sized instances, and the results show that for different-sized UDSS-R problems, the proposed two-stage heuristic algorithm performs well. It can find a good solution for all-sized instances of the proposed problem in a short CPU time.

Future research will focus on applying the proposed models and the two-stage heuristic algorithm to solve more *seru* scheduling problems. We will also attempt to use the exact methods of meta-heuristics to solve small-scale *seru* scheduling problems. Additionally, other practical factors apart from resource constraints, such as setup time, job-splitting, and learning effects, should also be considered.

Compliance with Ethical Standards

Funding: This research was supported by the Natural Science Foundation of China under the Grant No. 72472073. We would like to give our great appreciation to all the reviewers and editors who contributed this research.

Conflict of Interest: All authors declare that they have no conflict of interest.

Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] Lin, Q., Zhao, Q. and Lev, B. "Influenza vaccine supply chain coordination under uncertain supply and demand", *European Journal of Operational Research*, 297(3), pp. 930-948 (2022). https://doi.org/10.1016/j.ejor.2021.05.025.

[2] Frank, A., Dalenogare, L. and Ayala, N. "Industry 4.0 technologies: Implementation patterns in manufacturing companies", *International Journal of Production Economics*, 210, pp. 15-26 (2019). https://doi.org/10.1016/j.ijpe.2019.01.004.

[3] Senkul, P. and Toroslu, I. "An architecture for workflow scheduling under resource allocation constraints", *Information Systems*, 30(5), pp. 399-422 (2005).

[4] Liu, C., Lian, J., Yin, Y., et al. "*Seru* Seisan-an innovation of the production management Mode in Japan", *Asian Journal of Technology Innovation*, 18(2), pp. 89-113 (2010). https://doi.org/10.1080/19761597.2010.9668694.

[5] Kaku, I., Gong, J., Tang, J., et al. "Modeling and numerical analysis of line-cell conversion problems", *International Journal of Production Research*, 47(8), pp. 2055-2078 (2009). https://doi.org/10.1080/00207540802275889.

[6] Jiang, Y., Zhang, Z., Song, X., et al. "Scheduling controllable processing time jobs in *seru* production system with resource allocation", *Journal of the Operational Research Society*, 73(11), pp. 2551-2571 (2022). https://doi.org/10.1080/01605682.2021. 1999182.

[7] Ren, Y., Tang, J., Yu, Y., et al. "A two-stage stochastic programming model and parallel Master-Slave adaptive GA for flexible *seru* system formation", *International Journal of Production Research*, 62(4), pp. 1144-1161 (2024). https://doi.org/ 10.1080/00207543.2023. 2177087.

[8] Luo, L., Zhang, Z. and Yin, Y. "Modelling and numerical analysis of *seru* loading problem under uncertainty", *European Journal of Industrial Engineering*, 11(2), pp. 185-204 (2017). https://doi.org/10.1504/EJIE.2017.083255.

[9] Jiang, Y., Zhang, Z., Gong, X., et al. "An exact solution method for solving *seru* scheduling problems with past-sequence-dependent setup time and learning effect", *Computers & Industrial Engineering*, 158, pp. 107354 (2021). https://doi.org/10.1016/ j.cie.2021.107 354.

[10] Ying, K. and Tsai, Y. "Minimising total cost for training and assigning multiskilled workers in *seru* production systems", *International Journal of Production Research*, 55(10), pp. 2978-2989 (2017). https://doi.org/10.1080/00207543.2016.1277594.
[11] Yu, Y. and Tang, J. "*Seru* production mode", Science Press. (2018). (in Chinese)
[12] Yin, Y., Stecke, K., Swink, M., et al. "Lessons from *seru* production on manufacturing competitively in a high cost environment", *Journal of Operations Management*, 49, pp. 67-76 (2017). https://doi.org/10.1016/j.jom.2017.01.003.
[13] Liu, C., Li, Z., Tang, J., et al. "How *SERU* production system improves manufacturing flexibility and firm performance: an empirical study in China", *Annals of Operations Research*, 316, pp. 529-554 (2022). https://doi.org/10.1007/s10479-020-03 850-y.
[14] Sengupta, K. and Jacobs, F. "Impact of work teams: a comparison study of assembly cells and assembly line for a variety of operating environments", *International Journal of Production Research*, 42(19), pp. 4173-4193 (2004). https://doi.org/10.1080/0020754041000 1720421.
[15] Liu, C., Dang, F., Li, W., et al. "Production planning of multi-stage multi-option *seru* production systems with sustainable measures", *Journal of Cleaner Production*, 105, pp. 285-299 (2015). https://doi.org/10.1016/j.jclepro.2014.03.033.
[16] Roth, A., Singhal, J., Singhal, K., et al. "Knowledge creation and dissemination in operations and supply chain management", *Production and Operations Management*, 25(9), pp. 1473-1488 (2016). https://doi.org/10.1111/poms.12590.
[17] Treville, S., Ketokivi, M. and Singhal, V. "Competitive manufacturing in a high-cost environment: Introduction to the special issue", *Journal of Operations Management*, 49(1), pp. 1-5 (2017). https://doi.org/10.1016/10.1016/j.jom.2017.02.001.
[18] Stecke, K., Yin, Y., Kaku, I., et al. "*Seru*: the organizational extension of JIT for a super-talent factory", *International Journal of Strategic Decision Sciences*, 3(1), pp. 106-119 (2012). http://doi.org/10.4018/jsds.2012010104.
[19] Yin, Y., Stecke, K. and Li, D. "The evolution of production systems from Industry 2.0 through Industry 4.0", *International Journal of Production Research*, 56(1-2), pp. 848-861 (2018). https://doi.org/10.1080/00207543.2017.1403664.
[20] Luo, L., Zhang, Z. and Yin, Y. "Simulated annealing and genetic algorithm based method for a bi-level *seru* loading problem with worker assignment in *seru* production systems", *Journal of Industrial & Management Optimization*, 17(2), pp. 779-803 (2021). https://doi.org/779.10.3934/jimo.2019134.
[21] Zhang, Z., Wang, L., Song, X., et al. "Improved genetic-simulated annealing algorithm for *seru* loading problem with downward substitution under stochastic environment", *Journal of the Operational Research Society*, 73(8), pp. 1800-1811 (2022). https://doi.org/10.1080/01605682.2021.1939172.
[22] Zhang, Z., Song, X., Gong, X., et al. "Coordinated *seru* scheduling and distribution operation problems with DeJong's learning effects", *European Journal of Operational Research*, 313(3), pp. 452-464 (2024). https://doi.org/10.1016/j.ejor.2023. 08.022.
[23] Liu, C., Stecke, K., Lian, J., et al. "An implementation framework for *seru* production", *International Transactions in Operational Research*, 21(1), pp. 1-19 (2014). https://doi.org/10.1111/itor.12014.
[24] Yu, Y., Tang, J., Gong, J., et al. "Mathematical analysis and solutions for multi-objective

line-cell conversion problem", *European Journal of Operational Research*, 236(2), pp. 774-786 (2014). https://doi.org/10.1016/j.ejor.2014.01.029.

[25] Shao, L., Zhang, Z. and Yin, Y. "A bi-objective combination optimisation model for line-*seru* conversion based on queuing theory", *International Journal of Manufacturing Research*, 11(4), pp. 322-338 (2016). https://doi.org/10.1504/IJMR.2016.082821.

[26] Yu, Y., Sun, W., Tang, J., et al. "Line-*seru* conversion towards reducing worker (s) without increasing makespan: models, exact and meta-heuristic solutions", *International Journal of Production Research*, 55(10), pp. 2990-3007 (2017). https://doi.org/10.1080/00207543.2017.1284359.

[27] Wang, Y., Zhang, Z. and Yin, Y. "Workload-based order acceptance in *seru* production system", *International Journal of Manufacturing Research*, 15(3), pp. 234-251 (2020). https://doi.org/10.1504/IJMR.2020.108197.

[28] Wang, L., Zhang, Z. and Yin, Y. "Order acceptance and scheduling problem with outsourcing in *seru* production system considering lot-spitting", *European Journal of Industrial Engineering*, 16(1), pp. 91-116 (2022). https://doi.org/10.1504/EJIE.2022.119371.

[29] Zhang, Z., Gong, X., Song, X., et al. "A column generation-based exact solution method for *seru* scheduling problems", *Omega*, 108, pp. 102581 (2022). https://doi.org/10.1016/j.omega.2021.102581.

[30] Zhang, Z., Song, X., Huang, H., et al. "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, pp. 866-877 (2022). https://doi.org/10.1016/j.ejor.2021.06.017.

[31] Zhang, Z., Song, X., Huang, H., et al. "Scheduling problem in *seru* production system considering DeJong's learning effect and job splitting", *Annals of Operations Research*, 312, pp. 1119-1141 (2022). https://doi.org/10.1007/s10479-021- 04515-0.

[32] Li, X., Zhang, Z., Sun, W., et al. "Parallel dynamic NSGA-II with multi-population search for rescheduling of *seru* production considering schedule changes under different dynamic events", *Expert Systems with Applications*, 238, pp. 121993 (2024). https://doi.org/10.1016/j.eswa.2023.121993.

[33] Zhang, Z., Song, X., Gong, X., et al. "An effective heuristic based on 3-opt strategy for *seru* scheduling problems with learning effect", *International Journal of Production Research*, 61(6), pp. 1938-1954 (2023). https://doi.org/10.1080/00207543. 2022. 2054744.

[34] Li, X., Yu, Y., Sun, W., et al. "Reducing tardy batches by *seru* production: Model, exact solution, cooperative coevolution solution, and insights", *Computers & Operations Research*, 160, pp. 106048 (2023). https://doi.org/ 10.1016/j.cor.2022.106048.

[35] Abbaszadeh, N., Asadi-Gangraj, E. and Emami, S. "Flexible flow shop scheduling problem to minimize makespan with renewable resources", *Scientia Iranica*, 28(3), pp. 1853-1870 (2021). https://doi.org/10.24200/SCI.2019.53600.3325.

[36] Daniels, R., Hoopes, B. and Mazzola, J. "Scheduling parallel manufacturing cells with resource flexibility", *Management Science*, 42(9), pp. 1260-1276 (1996). https://doi.org/10.1287/ mnsc.42.9.1260.

[37] Hasannia-Kolagar, S., Asadi-Gangraj, E., Paydar, M., et al. "Robust bi-objective operating rooms scheduling problem regarding the shared resources", *Scientia Iranica*, 30(6), pp.

2203-2221 (2023). https://doi.org/10.24200/SCI.2021.57603.5321.

[38] Edis, E. and Oguz, C. "Parallel machine scheduling with flexible resources", *Computers & Industrial Engineering*, 63(2), pp. 433-447 (2012). https://doi.org/10.1016/j.cie.2012.03.018.

[39] Fanjul-Peyro, L., Perea, F. and Ruiz, R. "Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources", *European Journal of Operational Research*, 260(2), pp. 482-493 (2017). https://doi.org/10.1016/j.ejor.2017.01.002.

[40] Yunusoglu, P. and Topaloglu, S. "Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times", *International Journal of Production Research*, 60(7), pp. 2212-2229 (2022). https://doi.org/10.1080/00207543.2021.1885068.

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

[42] Fleszar, K. and Hindi, K. "Algorithms for the unrelated parallel machine scheduling problem with a resource constraint", *European Journal of Operational Research*, 271(3), pp. 839-848 (2018). https://doi.org/10.1016/j.ejor.2018.05.056.

[43] Yepes-Borrero, J., Villa, F., Perea, F., et al. "GRASP algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources", *Expert Systems with Applications*, 141, pp. 112959 (2020). https://doi.org/ 10.1016/j.eswa.2019.112959.

[44] Yepes-Borrero, J., Perea, F., Ruiz, R., et al. "Bi-objective parallel machine scheduling with additional resources during setups", *European Journal of Operational Research*, 292(2), pp. 443-455 (2021). https://doi.org/10.1016/j.ejor.2020.10.052.

[45] Grigoriev, A., Sviridenko, M. and Uetz, M. "Integer Programming and Combinatorial Optimization (IPCO 2005)", In M. Jünger, & V. Kaibel (Eds.), Lecture Notes in Computer Science 3509, pp. 182-195 (2005).

[46] Nasiri, M. and Hamid, M. "The stage shop scheduling problem: lower bound and metaheuristic", *Scientia Iranica*, 27(2), pp. 862-879 (2020). https://doi.org/10.24200/SCI.2018.5199. 1146.

[47] Nembhard, D. and Bentefouet, F. "Selection policies for a multifunctional workforce", *International Journal of Production Research*, 52(16), pp. 4785-4802 (2014). https://doi.org/10.1080/ 00207543.2014.887231.

[48] Su, B., Xie, N. and Yang, Y. "Hybrid genetic algorithm based on bin packing strategy for the unrelated parallel workgroup scheduling problem", *Journal of Intelligent Manufacturing*, 32, pp. 957-969 (2021). https://doi.org/10.1007/s10845-020-01597-8.

[49] Yılmaz, Ö. F. "Operational strategies for *seru* production system: a bi-objective optimisation model and solution methods", *International Journal of Production Research*, 58(11), pp. 3195-3219 (2020). https://doi.org/10.1080/00207543.2019.1669841.

[50] Yu, Y. and Tang, J. "Review of *seru* production", *Frontiers of Engineering Management*, 6(2), 183-192 (2019). https://doi.org/10.1007/s42524-019-0028-1.

Fig. 1 Three *seru* types

Fig. 2 Solutions of the instance from Example 1 obtained using three different methods

Fig. 3 Flowchart of the heuristic algorithm
Fig. 4 Job-*seru* assignment result matrix
Fig. 5 The application of the 0-1 knapsack algorithm
Fig. 6 Point-line diagram for the RPD of all instances (Note: RPD: relative percentage deviation)
Table 1 Specific processing data of Example 1
Table 2 Parameter settings
Table 3 Results for the two-stage heuristic algorithm

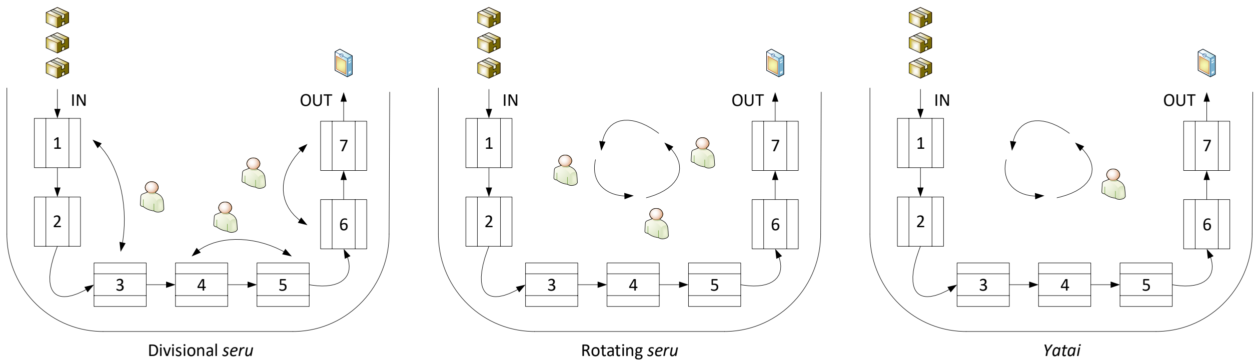

Fig. 1 Three *seru* types

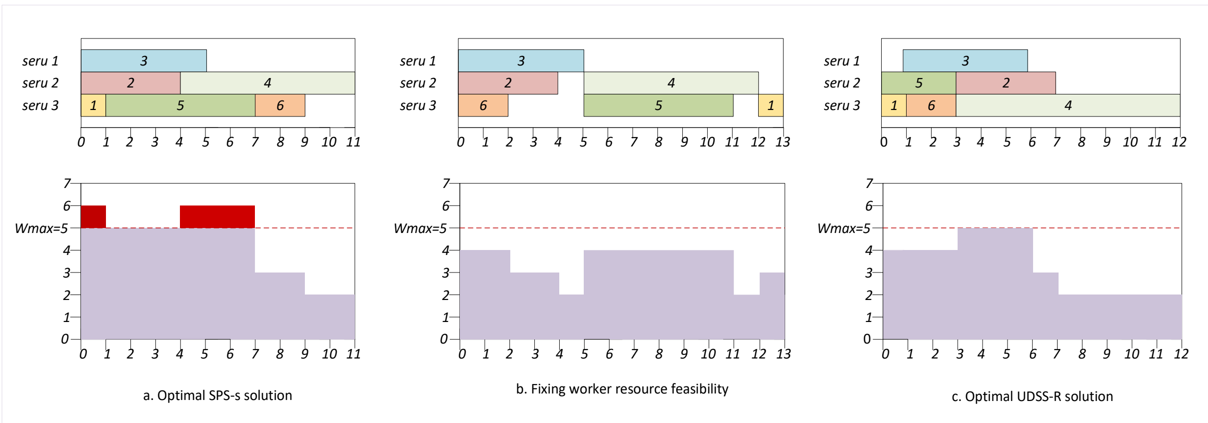

Fig. 2 Solutions of the instance from Example 1 obtained using three different methods

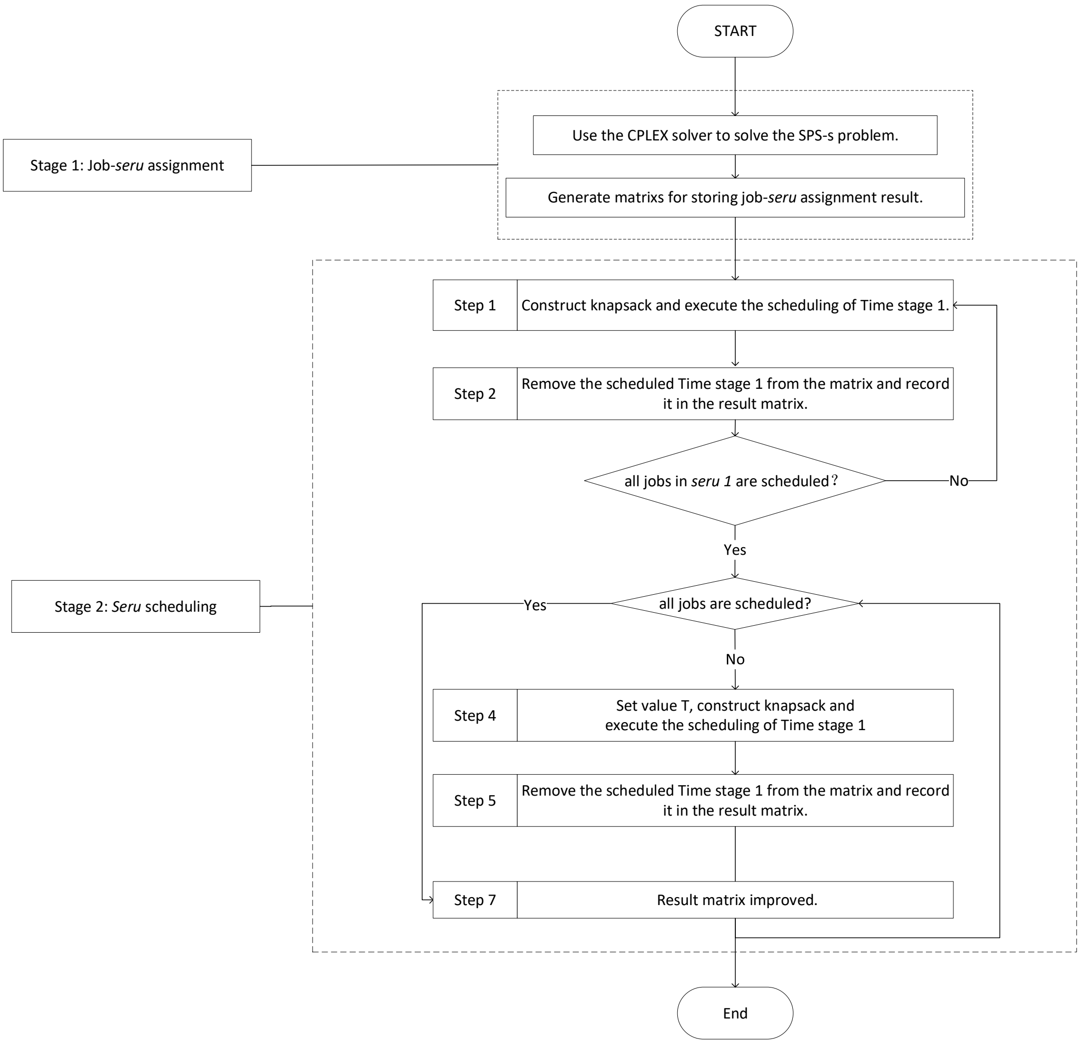

Fig. 3 Flowchart of the heuristic algorithm

Job matrix

j_start =

Time stage	1	2	3	4	5	6	7	8	...	...
seru 1	job_{11}	job_{12}	job_{13}	job_{14}	job_{15}	job_{16}	job_{17}	job_{18}	......	......
seru 2	job_{21}	job_{22}	job_{23}	job_{24}	job_{25}	job_{26}	job_{27}	job_{28}	......	......
seru i	......	......	......	......	......	......	......	......	......	......
seru m	job_{m1}	job_{m2}	job_{m3}	job_{m4}	job_{m5}	job_{m6}	job_{m7}	job_{m8}	......	......

Processing time matrix

p_start =

Time stage	1	2	3	4	5	6	7	8	...	...
seru 1	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}	p_{18}	......	......
seru 2	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{28}	......	......
seru i	......	......	......	......	......	......	......	......	......	......
seru m	p_{m1}	p_{m2}	p_{m3}	p_{m4}	p_{m5}	p_{m6}	p_{m7}	p_{m8}	......	......

Needed workers matrix

w_start =

Time stage	1	2	3	4	5	6	7	8	...	...
seru 1	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	......	......
seru 2	w_{21}	w_{22}	w_{23}	w_{24}	w_{25}	w_{26}	w_{27}	w_{28}	......	......
seru i	......	......	......	......	......	......	......	......	......	......
seru m	w_{m1}	w_{m2}	w_{m3}	w_{m4}	w_{m5}	w_{m6}	w_{m7}	w_{m8}	......	......

Fig. 4 Job-*seru* assignment result matrix

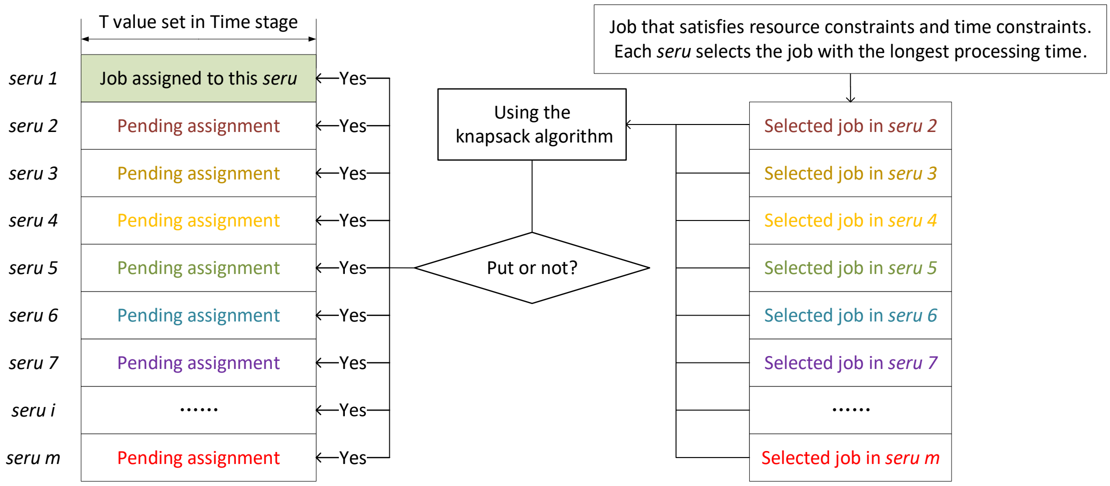

Fig. 5 The application of the 0-1 knapsack algorithm

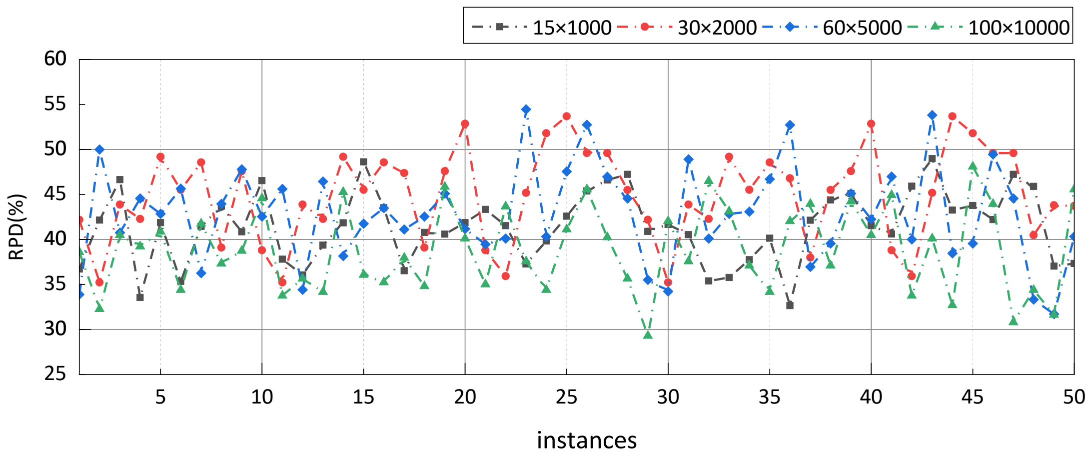

Fig. 6 Point-line diagram for the RPD of all instances (Note: RPD: relative percentage deviation)

Table 1 Specific processing data of Example 1

	Processing time (p_{ij})						Needed workers (w_{ij})					
	job1	job2	job3	job4	job5	job6	job1	job2	job3	job4	job5	job6
Seru 1	8	8	5	7	8	7	1	3	2	2	2	1
Seru 2	8	4	9	7	3	7	1	1	2	2	1	1
Seru 3	1	9	10	9	6	2	3	3	1	2	2	1

Table 2 Parameter settings

Parameters	Value
Instance size $\{n \times m\}$	$\{15 \times 1000, 30 \times 2000, 60 \times 5000, 100 \times 10000\}$
Total number of worker resource in SPS W_{max}	$5 \times m$

The number of needed workers w_{ij}	$U(1,9)$
Processing time p_{ij}	$U(1,100)$

Table 3 Results for the two-stage heuristic algorithm

	Total CPU(s)		CPU_KG(s)		%RPD	
	Avg.	Max.	Avg.	Max.	%Avg.	%Max.
15×1000	73.1189	305.1329	0.8978	1.2123	41.3959	48.9796
30×2000	52.4430	248.1956	1.0645	2.5690	44.8839	53.6885
60×5000	56.9517	127.6715	3.3342	6.6038	42.8100	54.4444
100×10000	244.3461	540.4948	12.3972	16.4867	38.8863	48.1013

Appendix

To evaluate the performance of the proposed model and the two-stage heuristic algorithm, we use CPU time and RPD as numerical indicators. Appendix Table 1 records the total CPU time of all instances, while Appendix Table 2 records the CPU time used by the dynamic programming algorithm based on the 0-1 knapsack problem (CPU_KG) in the second stage. Appendix Table 3 records the RPD of all instances.

Appendix Table 1 The total CPU time of all instances

The total CPU(s)	15×1000	30×2000	60×5000	100×10000
1	13.8417	248.1956	65.0038	320.2405
2	131.9131	188.1499	88.4853	198.2675
3	83.5990	6.7252	127.6715	294.3986
4	3.3038	68.9444	94.6163	269.3894
5	48.8603	15.0665	56.4863	250.9665
6	6.8248	7.1968	39.0973	173.8681
7	86.4453	21.5987	48.7597	380.8670
8	182.6751	6.7854	101.3479	179.3365
9	207.9544	145.6615	37.4541	172.7532
10	62.2109	9.6210	47.4538	166.4192
11	80.5950	190.2099	40.0118	292.1425
12	1.7444	7.0956	54.5607	173.4890
13	43.0704	69.1671	54.4708	168.4344
14	188.4842	13.0067	42.0859	183.4697
15	186.1418	7.1294	80.9124	465.6384
16	73.7543	21.9069	41.7465	163.6788

17	3.1982	65.4172	59.9047	183.5723
18	10.5422	8.2745	85.1915	278.2500
19	84.5589	138.8955	41.6287	174.7109
20	305.1329	12.6111	39.7185	245.1181
21	2.2152	11.4844	38.6620	168.1707
22	2.8629	6.0919	49.3647	173.1665
23	1.9980	5.8741	66.7426	255.1541
24	4.5529	5.4729	39.2524	168.7271
25	42.9606	6.1469	63.0676	255.7643
26	178.8912	7.0275	66.9156	486.4341
27	1.4672	145.9318	52.4537	151.1583
28	38.4945	5.7967	42.6740	233.5171
29	6.1048	210.4090	74.3087	356.3842
30	7.0156	205.4529	53.5894	156.2845
31	151.4237	7.9146	49.7545	370.2229
32	26.6623	70.4493	38.5681	238.8048
33	7.7575	11.5855	43.4678	161.1122
34	6.7048	7.4945	42.5528	165.3905
35	238.4856	22.0248	87.8063	202.1167
36	235.0655	68.3435	42.2681	176.3662
37	97.0622	8.0177	40.9624	237.1723
38	2.1596	5.7967	42.2991	175.8877
39	2.2516	123.3746	47.0583	540.4948
40	108.4808	12.1214	52.6542	182.7442
41	176.3049	9.0556	38.9373	277.3285
42	8.9388	7.1615	39.3885	186.3085
43	58.4912	6.4742	40.4500	297.3759
44	8.7079	7.1568	48.3425	181.8830
45	48.2535	6.7661	74.0564	222.1465
46	43.6586	9.5055	40.7142	157.6290
47	1.9544	156.2902	39.3746	156.3467
48	32.8256	9.1016	41.3883	177.2117
49	2.3530	4.5520	88.9406	170.7412
50	74.9771	11.8619	44.2453	504.1030

Appendix Table 2 The CPU_KG time of all instances

CPU_KG(s)	15×1000	30×2000	60×5000	100×10000
1	0.9817	0.9756	6.6038	14.1105
2	0.8931	0.4199	4.7053	12.3475
3	0.7390	0.9452	4.0115	12.3686
4	0.6638	1.0544	4.2363	12.2994
5	0.7134	1.2065	4.2063	13.1565
6	0.7948	1.0868	2.8773	12.4481
7	0.6753	0.9887	5.2297	12.6970

8	0.7551	1.1454	2.7679	12.8665
9	0.7344	0.9915	2.7341	12.9832
10	0.6809	0.9910	2.7838	11.6392
11	0.7050	1.8799	2.8118	12.1125
12	0.6844	0.9256	2.7807	13.1590
13	0.8004	0.9771	2.7708	12.2344
14	0.7942	0.9467	2.7559	13.5497
15	0.7818	0.9094	2.7724	11.9784
16	1.0843	0.8569	2.8565	12.7888
17	0.7472	1.1072	2.8447	13.1223
18	0.8822	0.9445	2.8015	12.4200
19	0.7789	0.9155	2.8287	13.1609
20	0.8529	0.9511	2.8085	12.2281
21	0.7452	0.9244	2.8620	12.3107
22	0.7829	1.0419	2.8447	13.1365
23	0.8080	0.9541	2.8026	11.7941
24	0.9229	0.9719	3.0324	12.4471
25	0.9006	1.0669	3.5176	12.3843
26	0.8312	1.2575	3.2556	5.0241
27	0.8072	1.2418	2.8437	9.3883
28	0.9645	1.2467	2.9740	11.4571
29	0.9348	2.5690	2.8287	12.3342
30	0.9356	1.0029	4.4594	13.2645
31	0.9737	1.0046	3.4145	12.3029
32	1.2123	0.8893	2.7881	12.2548
33	0.9775	1.0355	3.8278	12.3122
34	0.8948	1.2345	3.1128	13.2305
35	1.1456	1.1948	3.6963	16.4867
36	1.1255	0.8435	3.6881	14.0862
37	1.0322	0.9977	3.5824	13.5623
38	1.0296	1.2467	3.5391	10.9677
39	1.0516	0.9046	3.5583	12.2248
40	1.0108	1.2114	2.8442	13.3642
41	1.0849	0.9456	2.9373	13.3785
42	0.6131	1.0015	3.0585	14.8685
43	1.0412	1.1342	2.8600	14.0959
44	1.0379	1.1068	2.7925	13.0230
45	1.0535	0.9961	3.1664	10.8765
46	1.0986	1.0155	3.3742	10.6290
47	1.0344	1.0702	3.5746	11.9367
48	1.0156	0.9716	3.0283	15.3817
49	1.0430	0.8920	3.9306	9.4012
50	1.0371	1.0319	4.3253	8.2630

Appendix Table 3 RPD of all instances

RPD (%)	15×1000	30×2000	60×5000	100×10000
1	36.7117	42.1875	33.8889	38.6076
2	42.1429	35.2227	50.0000	32.2785
3	46.6520	43.8735	40.7609	40.5063
4	33.5614	42.2925	44.5652	39.2405
5	41.8764	49.1936	42.8571	40.8805
6	35.3430	45.5253	45.6522	34.3949
7	41.4631	48.5714	36.2637	41.7722
8	43.6957	39.1129	43.9560	37.3418
9	40.8602	47.5807	47.8022	38.7500
10	46.5368	38.8000	42.5414	44.5860
11	37.8022	35.2227	45.6044	33.7580
12	36.0544	43.8735	34.4263	35.6688
13	39.3750	42.2925	46.4481	34.1772
14	41.8502	49.1936	38.1720	45.2830
15	48.6141	45.5253	41.7582	36.0760
16	43.5165	48.5714	43.4783	35.2564
17	36.5297	47.3896	41.1111	37.9747
18	40.7725	39.1129	42.5414	34.8101
19	40.5896	47.5807	45.1087	45.8599
20	41.8655	52.8455	41.1765	40.1274
21	43.3333	38.8000	39.4595	35.0319
22	41.5254	35.9184	40.1099	43.6709
23	37.2844	45.1738	54.4444	37.5796
24	39.8305	51.7928	40.3226	34.3949
25	42.5764	53.6885	47.5410	41.1392
26	45.3917	49.6000	52.7473	45.5696
27	46.6063	49.6000	46.9613	40.2516
28	47.2222	45.4918	44.5652	35.6688
29	40.8898	42.1875	35.5191	29.2994
30	41.6667	35.2227	34.2391	42.0382
31	40.5286	43.8735	48.9130	37.5796
32	35.3982	42.2925	40.1099	46.4968
33	35.7759	49.1936	42.8571	43.1250
34	37.7528	45.5253	43.0939	37.1069
35	40.1345	48.5714	46.7033	34.1772
36	32.6622	46.8000	52.7174	42.0382
37	42.1296	38.0000	36.9565	43.9490
38	44.3966	45.4918	39.5604	37.1069
39	45.0980	47.5807	45.1087	44.2308
40	41.5350	52.8455	42.2460	40.5063
41	40.6114	38.8000	46.9945	44.9367
42	45.9161	35.9184	40.0000	33.7580

43	48.9796	45.1738	53.8044	40.1274
44	43.2671	53.6885	38.4615	32.7044
45	43.7637	51.7928	39.5604	48.1013
46	42.2078	49.6000	49.4506	43.9490
47	47.2350	49.6000	44.5652	30.8176
48	45.9016	40.4762	33.3333	34.3949
49	37.0288	43.8017	31.7204	31.6456
50	37.3333	43.7247	40.3226	45.5696

Biography of author:

Yiran Xiang: 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*: 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.

Xiaoling Song: Xiaoling Song received her PhD in Management Science and Engineering, in 2016, and BS in Management Science, in 2012, from Sichuan University, Sichuan, China. She is currently an Associate Professor with the Department of Management Science and Engineering, School of Economics and Management in Nanjing University of Science and Technology, Nanjing China. Her research focuses on decision making, and multi-objective optimization.

Xue Gong: 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: Yong Yin has graduated at Tohoku University in Japan. He is a Professor of Graduate School of Business in Doshisha University. His current research interests are in the areas of beyond lean, *seru* production systems, sustainable development, production and operations management, and so on.