Annals of Operations Research (2022) 312:1119–1141
https://doi.org/10.1007/s10479-021-04515-0

ORIGINAL RESEARCH

Scheduling problem in *seru* production system considering DeJong's learning effect and job splitting

Zhe Zhang[1] · Xiaoling Song[1] · Huijun Huang[1] · Yong Yin[2] · Benjamin Lev[3]

Accepted: 23 December 2021 / Published online: 9 January 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Seru is a relatively new type of Japanese production mode originated from the electronic assembly industry. In practice, *seru* production has been proven to be efficient, flexible, response quickly, and can cope with the fluctuating production demands in a current volatile market. This paper focuses on scheduling problems in *seru* production system. Motivated by the realty of labor-intensive assembly industry, we consider learning effect of workers and job splitting with the objective of minimizing the total completion time. A nonlinear integer programming model for the *seru* scheduling problem is provided, and it is proved to be polynomial solvable. Therefore, a branch and bound algorithm is designed for small sized *seru* scheduling problems, while a local search-based hybrid genetic algorithm employing shortest processing time rule is provided for large sized problems. Finally, computational experiments are conducted, and the results demonstrate the practicability of the proposed *seru* scheduling model and the efficiency of our solution methods.

Keywords *Seru* scheduling · Learning effect · Job splitting · Branch and bound · Genetic algorithm

✉ Zhe Zhang
zhangzhe@njust.edu.cn

Xiaoling Song
songxiaoling@njust.edu.cn

Huijun Huang
njustwww@163.com

Yong Yin
yyin@mail.doshisha.ac.jp

Benjamin Lev
bl355@drexel.edu

1 School Economics and Management, Nanjing University of Science and Technology, Nanjing 210094, People's Republic of China

2 Graduate School of Business, Doshisha University, Karasuma-Imadegawa, Kamigyo-ku, Kyoto 602-8580, Japan

3 Drexel University, Philadelphia, PA 19104, USA

1 Introduction

In Industrial Revolution 4.0, fast response is widely recognized as another dimension of demand in the manufacturing industry in addition to product volume and product variety (Yin et al., 2018). Hence, more and more companies must reconfigure their production system since the traditional assembly line could not response to current volatile market quickly enough (Nikkei-Business, 2016; Wang et al., 2019). In this situation, *seru* production system (SPS), which was first implemented at a factory producing video cameras for Sony company named Sony Minokamo in 1992 (Liu et al., 2014a), is used to cope with the effect of fluctuating demand, and it could achieve efficiency, flexibility and fast response simultaneously. In fact, due to the autonomy in SPS, *seru* becomes a key factor in the smart manufacturing Industry 4.0 (Yin et al., 2018). Roth et al. (2016) reviewed the research of operations management during the last 25 years and pointed out that *seru* production was one of eight promising future research directions because *seru* could response quickly to customer requirements with high efficiency and flexibility. Therefore, *seru* production has been regarded as a new "beyond lean" production mode for Industry 4.0 (Yin et al., 2017). *Seru* is the Japanese pronunciation of cell, and the SPS is a work-cell-based assembly systems decomposed from the traditional assembly line (Lian et al., 2018; Zhang et al., 2021). Figure 1 is an example for an assembly line converting into SPS. The underlying philosophy of SPS is to benefit from both the high-efficiency advantage of the assembly line and the flexibility advantage of cellular manufacturing systems (CMSs) (Yu and Tang, 2019). In fact, according to references (Kimura and Yoshita, 2004; Kono, 2004; Noguchi, 2003), the benefits are not just efficiency and flexibility but also the reduction of throughput time, setup time, work-in-process (WIP) inventory, labor hours, shop floor space, and finished product inventory. For example, Sony Minokamo reduced 10,000 square meters of floor space and 170 workers by SPS just after one year (Yamada, 2009). Another example was in Canon company, the average working

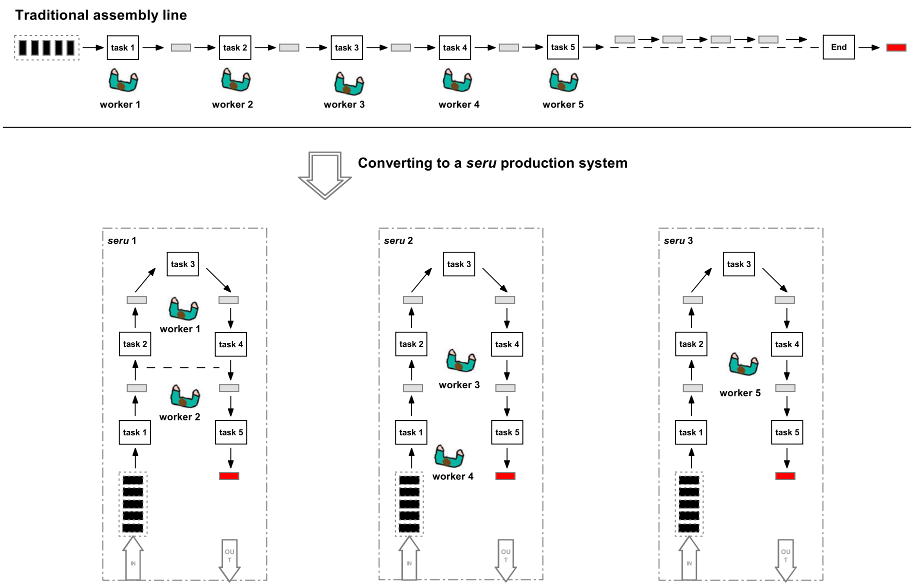

Fig. 1 An assembly line is decomposed into a *seru* production system

time of WIP was shortened from three days to six hours, and 720,000 square meters of workshop space in 54 factories were reduced after implementing SPS (Nikkei Mechanical, 2003; Hisashi, 2006). Some researchers also show that *seru* is more adaptive and competitive in an unpredictable environment with multiple product models, fluctuating volumes, and short product life cycles (Kaku, 2017; Zhang et al., 2017). Hence, SPS has been considered as a potential production system for Industry 4.0 (Treville et al., 2017; Yin et al., 2018). Unfortunately, although so many remarkable benefits have been proven in production practice, the research on SPS is still minimal due to its short history. However, SPS still has recently attracted attention from some leading scholars and practitioners throughout the world (Roth et al., 2016; Treville et al., 2017; Yin et al., 2018). In this paper, for the first time, we will focus on *seru* scheduling problems considering DeJong's learning effect and job splitting, and hopefully it could improve the theory of *seru* scheduling problems, along with providing the professional guidance to practical *seru* production.

In practice, as an extension or upgrade to the Toyota's traditional JIT material system (JIT-MS), the key to obtain the SPS's high performance is JIT organization system (JIT-OS) (Stecke et al., 2012). The difference between JIT-MS and JIT-OS is that JIT-MS focuses on materials while JIT-OS on organizations, i.e., *serus*. The mechanism of JIT-OS is the correct *serus*, in the right place, at the appropriate time, in the exact amount (Ayough et al., 2020; Sun et al., 2019; Yin et al., 2018). It contains three-stage decisions in JIT-OS, i.e., *seru* formation, *seru* loading and *seru* scheduling (Sun et al., 2020; Yu and Tang, 2019). First, a *seru* production system is configured with the appropriate *serus* amounts and types by *seru* formation. Then, by *seru* loading, the products ordered are allocated to each *seru* appropriately. Finally, considering the due date and schedule rule, the implement production plans are obtained by *seru* scheduling. Previous research mainly focused on *seru* formation and *seru* loading. Kaku et al. (2009) studied the insight of line-*seru* conversion problems by simulation experiments and pointed out the number of *serus* should be formatted in different cases. 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 bi-objective mathematical model to minimize the total training cost and to balance the total processing times among multi-skilled workers in each *seru*. To evaluate the performance of after converting the assembly line to SPS, Yu et al. (2012, 2013, 2014, 2017) constructed a series of mathematical models to investigate the line-*seru* conversion performances including the total throughput time and the total labor hours, and the mathematical characteristics such as solution space, combinatorial complexity and non-convex properties, were also analyzed. Shao et al. (2016) considered a line-*seru* conversion problem with stochastic orders based on queuing theory and developed a non-linear combination optimization model to confirm the *seru* formation. Luo et al. (2016) proposed a combinatorial optimization model for *seru* loading problem considering worker-operation assignment in single period and studied uncertain *seru* loading problems by a bi-objective model to minimize the makespan and the total tardiness penalty cost (Luo et al. 2017). Then, they designed a simulated annealing and genetic algorithm for a bi-level *seru* loading problem in SPS (Luo et al., 2021). Lian et al. (2018) developed a mathematical model to improve the inter-*seru* and inter-worker workload balance to solve worker grouping, *seru* loading and task assignment concurrently, and a meta-heuristic algorithm based on NSGA-II was designed to solve the proposed model. Jiang et al. (2021) transformed the *seru* scheduling problem into the assignment problem and proved they could be solved in polynomial time. Yılmaz (2020a) provided workforce related operational strategies of SPS for the workforce scheduling and focused on a bi-objective *seru* workforce scheduling problem considering the inter worker transfer (Yılmaz, 2020b). Zhang et al. (2022) proposed a logic-based Benders decomposition method for the *seru* scheduling

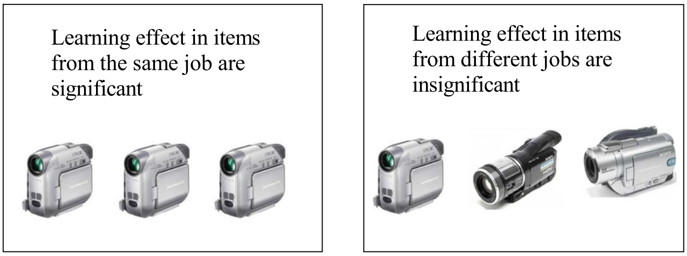

Fig. 2 Learning effects from jobs

problem. However, the research on scheduling problem in SPS is still very rare due to its complexity even though it is one of the most important critical factors of JIT-OS. This paper will focus on the *seru* scheduling problems considering DeJong's learning effect and job splitting and provide efficient solution methods for it.

Moreover, it can be noticed that the learning effect will occur and the processing time will be reduced when the job come from the same batch consecutively (Janiak et al., 2013; Sun et al., 2013). The later a given job is scheduled in the sequence repeatedly, the shorter its processing time (Mosheiov, 2001; Pei et al., 2019). Taking camera assembly manufactory shown in Fig. 2 as an example, it is intuitive that learning effect for the same product style is significant, while insignificant for the different product styles. Many researchers concerned on the learning effect in production scheduling. Biskup (1999) analyzed learning effects in production scheduling problems and pointed out that the well-known learning effect had never been considered in connection with scheduling problems. (Mosheiov and Sidney, 2003) studied the scheduling problem of makespan and total flow-time minimization, a due-date assignment problem and total flow-time minimization with the job-dependent learning curves, where the learning in the production process of some jobs to be faster than that of others. Rostami et al. (2020) investigated an integrated scheduling of production and distribution activities considering deterioration and learning effect, and proved that the integrated decision can reduce costs significantly. Wang et al. (2020) constructed a joint decision model to solve cell formation and product scheduling problems together in cellular manufacturing systems considering the learning and forgetting effect, and designed an improved bacterial foraging algorithm to minimize the makespan. Biskup (2008) proposed state-of-the-art review on production scheduling problems with learning effect.

In practice, an order is normally composed by several identical jobs, and a job batch can be split into several sub-jobs to improve the delivery time (Huang, 2010; Kim, 2018). Job spitting is always a hot issue in production scheduling. Nessah and Chu (2010) proposed a new lower bound of total weighted completion time for infinite split scheduling with job release dates and unavailability periods. Huang and Yu (2017) discussed the theoretical applications of subjects about multi-objective optimization, lot-splitting, and ant colony optimization. Liu et al. (2014b) proposed a lower bound of the production scheduling problem and a job-splitting algorithm corresponding to the lower bound, while a branch-and-bound algorithm and a hybrid differential evolution algorithm were also designed. Kim and Kim (2020) formulated the production scheduling problem with job splitting while the flexible idle time considering job splitting was inserted into initial schedule. To minimize both the makespan and electric power consumption, Chen et al. (2020) proposed a multi-objective mixed-integer programming for energy-efficient hybrid flow shop scheduling with job spitting. Figure 3 shows three different schedules for *seru* scheduling considering learning effect and job splitting, and it

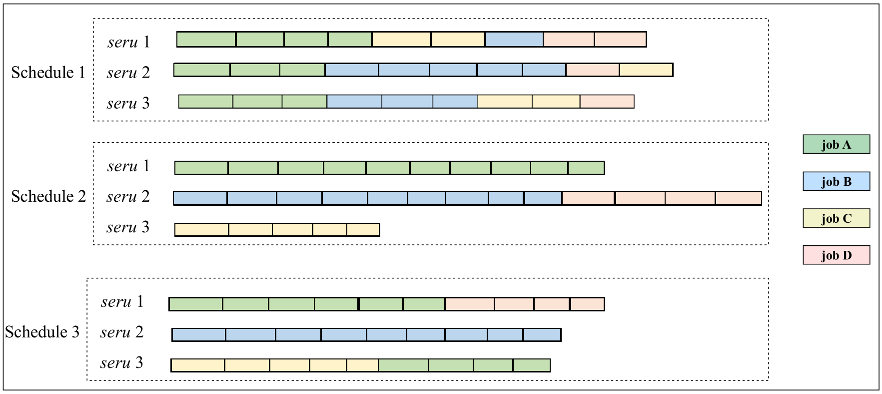

Fig. 3 A Gantt chart for a *seru* scheduling considering learning effect and job splitting

is interesting in production practice. For example, there are 10 jobs A, 9 jobs B, 5 jobs C, and 4 jobs D in a batch need to be scheduled in SPS. Schedule 1 is to finish the current job as soon as possible, thus, it just schedules the job to the current least heavily loaded *seru* one by one until all jobs are assigned in this SPS. In schedule 2, all jobs processed by one *seru*, i.e., jobs cannot be split. Total 10 jobs A are assigned to *seru* 1, 9 jobs B and 4 jobs D are assigned to *seru* 2, 5 jobs C are assigned to *seru* 3, respectively. The completion timed maybe longer due to the unbalanced job assignment. Schedule 3 splits the jobs in an appropriate way considering the learning effect at the same time, which is also the problem studied in this paper. As it can be seen, in Schedule 3, 10 jobs A are split into two parts and assigned to *seru* 1 (6 jobs A) and *seru* 3 (4 jobs A). For job B, C and D, they are allocated to one *seru*, i.e., *seru* 2, *seru* 3 and *seru* 1, respectively. Hence, comparing schedule 3 with schedule 1, the total processing time of schedule 3 is shorter than that of schedule 1 which is affected by the learning effect. Similarly, comparing schedule 3 with schedule 2, the total completion time of schedule 3 is shorter. By above comparisons, it can be seen that schedule 3 is the best schedule which considers both the total processing time and the total completion time. Therefore, this paper will also concentrate on *seru* scheduling problem considering job splitting and learning effect simultaneously to minimize the total completion time in SPS.

The remainder of this paper is organized as follows: Sect. 2 describes the *seru* scheduling problem and presents the analytical property. A non-linear integer programming model formulation is presented in Sect. 3, and the model analysis is also provided. Then, in Sect. 4, B&B algorithm, and a local search-based hybrid genetic algorithm (LS-hGA) employing shortest processing time (SPT) rule are designed for the small and large sized problems, respectively. Finally, Sect. 5 reports the computational results and conducts comparison analysis. Concluding remarks are made in Sect. 6, along with the discussion about further research.

2 Problem description

SPS has three types of *seru*, i.e., divisional *seru*, rotating *seru* and *yatai* (Liu et al., 2014a), see Fig. 4. Yu and Tang (2019) provided a detailed description about three *seru* types according to the evolution of SPS. First, if the workers in an assembly line can handle more than one task, i.e., partially cross-trained workers, then, an assembly line could be decomposed several short lines and divisional *serus* are configured. In other words, a divisional *seru* is a short line staffed with partially cross-trained workers, where each worker will handle more tasks

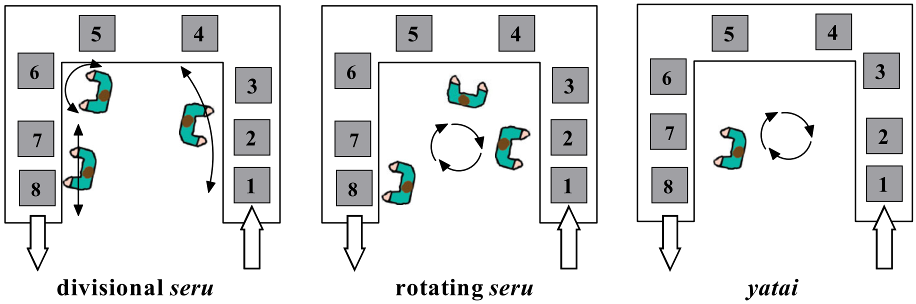

Fig. 4 Three types *seru*

compared to the original assembly line (Yin et al., 2018). In practice, the tasks in a divisional *seru* are divided into different sections in charge by the partially cross-trained workers (Stecke et al., 2012). Subsequently, some workers are completely cross-trained along with the worker training on *seru* implementation. It means that these workers could assemble all tasks of a job from start to finish, thus, the rotating *serus* can be constructed (Stecke et al., 2012). Now, the equipment is shared by completely cross-trained workers, and they move in rotating *seru* with one worker following another (Yin et al., 2017). In addition, the worker returns to the first workstation and start a new round when a job is completed (Liu et al., 2014a). In the end, some rotating *serus* could evolve into *yatais* when the waste of workers' talents is eliminated (Stecke et al., 2012). *Yatai* is an ideal production mode and only contains one completely cross-trained worker handles all tasks of a job, and it is a small single-person production unit with highly autonomous (Yu and Tang, 2019). *Seru* 1 in Fig. 1 is divisional *seru*, *seru* 2 is rotating *seru* and *seru* 3 is *yatai*, respectively. The *seru* type discussed in this paper is *yatai* because it is the most sensitive type for learning effect, and the divisional *seru* and rotating *seru* are left for the future research.

Following the first study on the learning effect in aircraft industry manufacturing (Wright, 1936), many researchers proposed a large variety of position-based learning effect models. In Wright's model, the learning effect is described as the cost: $C_x = C_1 x^b$, where x is the cumulative production count, C_x is the cumulative average cost for producing x units, C_1 is the cost for producing the first unit, and b is the learning curve exponent (i.e., learning index, $b \leq 0$). Obviously, C_x will decrease when x is increasing. Biskup (1999) constructed a production scheduling model with the learning effect: $p_{jr} = \bar{p}_j r^a$, where $\bar{p}_j$ is the processing time for producing job j for the first time, p_{jr} is the actual processing time of job j in position r of a schedule (i.e., the rth repetition), and $a \leq 0$ is also the learning index. Mosheiov and Sidney (2003) proposed a job-dependent learning index a_j and the learning effect is $p_{jr} = \bar{p}_j r^{a_j}$, while Bachman and Janiak (2004) provided a linearized learning effect model as $p_{jr} = \bar{p}_j - r v_j$, where v_j is a given coefficient. Similarly, Cheng et al. (2013) applied $p_{jr} = \bar{p}_j (1 + \sum_{k=1}^{r-1} \beta_k ln \bar{p}_{[k]})^a r^b$ for production scheduling with a position-weighted learning effect. Unfortunately, all these models suffer a common drawback: if the job is processed late among many jobs, then p_{jr} is close to zero, which is not going to occur in practice. In this situation, DeJong proposed a new learning effect model to cope with this defect as

$$T_s = T_1 \left(M + \frac{(1 - M)}{s^m} \right) \tag{1}$$

where s stands for the sth cycle, T_1 is the processing time required for the first cycle of a batch, T_s is the processing time required for the sth cycle of the batch, M is the factor of incompressibility in production practice ($0 \leq M \leq 1$), and m is positive number and represents the exponent of reduction and is ($0 < m < 1$). According to Eq. (1), the processing time for the sth cycle product will fall when s increases, but it will approach a certain limit $T_1 M$. Thus, DeJong's learning effect model overcomes the drawback. In addition, when $M = 0$, Eq. (1) is $T_s = T_1 \frac{1}{s^m} = T_1 s^{-m}$, $0 < m < 1$, which means that Wright's log-linear learning effect model is a special case of DeJong's model. In this paper, we will use DeJong' model to describe the learning effect in *seru* scheduling problems.

In the *seru* scheduling problem, there are $j \in J \equiv \{1, 2, \ldots, n_J\}$ job need to be processed on $i \in I \equiv \{1, 2, \ldots, n_I\}$ *serus*. For each job j, it has N_j identical quantities which can be split into several sub-parts and assigned in parallel *serus*. Meanwhile, the processing time of job j is denoted as p_j. In the assembly process, learning effect will occur the same job is processed consecutively. Let $t(\tau, p_j)$ be the processing time of job j in τth repetition, and τ a non-negative integer. $t(\tau, p_j)$ satisfies

$$t(\tau, p_j) \geq t(\tau + 1, p_j)$$

where $t(0, p_j) = 0$ and $t(1, p_j) = p_j$. Define $f(\tau, p_j)$ as the total processing time of consecutively processing τ items, and

$$f(\tau, p_j) = \Sigma_{\upsilon=1}^{\tau} t(\upsilon, p_j)$$

Then, the following theorems hold.

Theorem 1 *Assume that there are Q quantities in job j, and two sub-jobs are split with quantity τ_1 and τ_2 with $\tau_1 + \tau_2 = Q$. Without loss of generality, let $\tau_1 \geq \tau_2$, then:*
(1) $f(\tau_1, p_j) + f(\tau_2, p_j) \geq f(\tau_1 + 1, p_j) + f(\tau_2 - 1, p_j)$;
(2) $\min\left(f(\tau_1, p_j) + f(\tau_2, p_j)\right) = f(Q, p_j)$.

Proof (1) Following the property of function $t(\tau, p_j)$, we know that $t(\tau_1, p_j) \geq t(\tau_1 + 1, p_j)$. Thus,

$$t(\tau_1 + 1, p_j) \leq t(\tau_1, p_j) \leq t(\tau_2, p_j)$$

due to $\tau_1 \geq \tau_2$. Then,

$$t(\tau_1 + 1, p_j) \leq t(\tau_2, p_j)$$

i.e.,

$$\begin{aligned} & f(\tau_1 + 1, p_j) - f(\tau_1, p_j) \leq f(\tau_2, p_j) - f(\tau_2 - 1, p_j) \\ & \quad \Rightarrow f(\tau_1, p_j) + f(\tau_2, p_j) \geq f(\tau_1 + 1, p_j) + f(\tau_2 - 1, p_j). \end{aligned}$$

(2) Since $f(\tau_1, p_j) + f(\tau_2, p_j) \geq f(\tau_1 + 1, p_j) + f(\tau_2 - 1, p_j)$, then

$$\min\left(f(\tau_1, p_j) + f(\tau_2, p_j)\right) = f(\tau_1 + 1, p_j) + f(\tau_2 - 1, p_j)$$

moreover,

$$f(\tau_1 + 1, p_j) + f(\tau_2 - 1, p_j) = f(\tau_1 + \tau_2, p_j) = f(Q, p_j)$$

hence,

$$\min\left(f(\tau_1, p_j) + f(\tau_2, p_j)\right) = f(Q, p_j)$$

□

From Theorem 1, we know that job splitting will increase the total processing time when considering learning effect.

Theorem 2 *Let π be an optimal schedule of SPS, then all sub-jobs coming from the same lot and scheduled on the same seru i should be processed as a single sub-job in π.*

Proof Assume that job j has sub-jobs 1 with τ_1 quantities and sub-jobs 2 with τ_2 assigned on the same *seru* i, and sub-jobs 1 is assigned before sub-jobs 2 while sub-jobs 1 and 2 are not allocated continuously. Now, let sub-jobs 1 move to the front of sub-jobs 2, and job j's total processing time will decrease as $f(\tau_1 + \tau_2, p_j)$.

According to Theorem 1 (2), we know that $f(\tau_1 + \tau_2, p_j) \leq f(\tau_1, p_j) + f(\tau_2, p_j)$. Therefore, all sub-jobs coming from the same lot and scheduled on the same *seru* i should be processed as a single sub-job in the optimal schedule π. □

Theorem 3 *For each single seru i in SPS, there exists an optimal schedule π that assigns sub-jobs follow the same lot.*

Proof . Assume that π is an optimal schedule, and the last completed sub-job of job j is completed in *seru* i. Subsequently, check other sub-jobs of job j assigned in other *serus* in SPS. If these sub-jobs are moved to the last position in its original *seru*, then job j's completion time will remain the same. According to Theorem 2, the completion time of all other jobs will maintain stable or decrease. Hence, by removing job j, an optimal schedule which is no worse than the original one will be obtained. Therefore, all the assigned sub-jobs from job j could follow the same lot. □

3 Model formulation

Based on Theorems 1–3, the non-linear integer programming model of *seru* scheduling problems considering DeJong's learning effect and job splitting is constructed in this section.

3.1 Notation

i	*seru* index, $i \in I \equiv \{1, 2, \ldots, n_I\}$
j	job index, $j \in J \equiv \{1, 2, \ldots, n_J\}$
r	position index, $r \in \{1, 2, \ldots, n_J\}$
p_j	processing time of job j
p_{rj}	processing time of job j in the rth repetition
N_j	number of job j
a	a constant, such that $a \geq \max_j N_j$
q_{ij}	quantities of job j assigned to *seru* i
c_{ij}	completion time of job j at *seru* i
CT_j	completion time of job j
M	incompressible factor, $0 \leq M \leq 1$
b	learning index, $-1 \leq b \leq 0$
T	total completion time of all jobs

$$x_{ijk} = \begin{cases} 1, & \text{if } q_{ij} > k; \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{jr} = \begin{cases} 1, & \text{if } j \text{ is assigned in the position } r; \\ 0, & \text{otherwise.} \end{cases}$$

$$z_{ij} = \begin{cases} 1, \text{ if } q_{ij} > 0; \\ 0, \text{ otherwise.} \end{cases}$$

3.2 Modeling

The objective of *seru* scheduling problem considering learning effect and job splitting in this paper is to minimize the total completion time, thus:

$$\min T = \sum_{j=1}^{n_J} CT_j \tag{2}$$

Since each job in SPS has one position in the sequence, so

$$\begin{array}{l} \sum_{j=1}^{n_J} y_{jr} \leq 1, \quad r = 1, 2, \ldots, n_J \\ \sum_{r=1}^{n_J} y_{jr} \leq 1, \quad j = 1, 2, \ldots, n_J \end{array} \tag{3}$$

Further, the sum of items quantity from job j assigned to *seru* i is equal to the total items number of jobs j, here

$$\sum_{i=1}^{n_I} q_{ij} = N_j, j = 1, 2, \ldots, n_J \tag{4}$$

Also, the items quantity of job j assigned to *seru* i can be interpreted by the binary variable x_{ijk} as

$$q_{ij} = \sum_{k=1}^{N_j} x_{ijk}, i = 1, 2, \ldots, n_I, j = 1, 2, \ldots, n_J \tag{5}$$

Moreover, if there are two quantity constants k and $k^{'}$, and $k^{'} > k$, then

$$x_{ijk} \geq x_{ijk^{'}}, \forall k^{'} > k \tag{6}$$

Because the items quantity of job j assigned to *seru* i is non-negative integer, and z_{ij} is a binary variable, hence

$$q_{ij} \geq z_{ij} \tag{7}$$

Similarly,

$$az_{ij} \geq q_{ij} \tag{8}$$

Considering the DeJong's learning effect, the processing time of job j in the rth repetition is

$$p_{jr} = p_j(M + (1 - M)r^b) \tag{9}$$

where $-1 \leq b \leq 0, 0 \leq M \leq 1$. $M = 0$ imply a completely manual operation as Wright's model, and $M = 1$ represents $p_{jr} = p_j$, respectively. Hence, the completion time of job j at *seru* i can be denoted as

$$\begin{array}{l} c_{ij} = z_{ij} \sum_{r=1}^{n_J} y_{jr} \left(\sum_{m=1}^{r} \sum_{n=1}^{n_J} y_{nm} p_n \sum_{k=1}^{N_n} x_{ink}(M + (1 - M)r^b) \right) \\ i = 1, 2, \ldots, n_I; j = 1, 2, \ldots, n_J \end{array} \tag{10}$$

where Eq. (10) is a non-linear constraint which contains the multiplication of x_{ijk}, y_{jr} and z_{ij}. In addition, the completion time job j at *seru* i should be less than or equal to the completion time of job j, thus

$$c_{ij} \leq CT_j \tag{11}$$

Finally, there are also have some logical constraints as

$$\begin{aligned} & x_{ijk}, y_{jr}, z_{ij} \in \{0, 1\}, \forall i, j, k, r \\ & n_I, n_J, N_j, q_{ij} \in Z \\ & T, CT_j \in R, \forall i, j, k, r \\ & -1 \leq b \leq 0, 0 \leq M \leq 1 \end{aligned} \tag{12}$$

Hence, the non-linear integer programming model of *seru* scheduling problem considering DeJong's learning effect and job splitting could be constructed as follows:

$$\begin{cases} \min T = \sum_{j=1}^{n_J} CT_j \\ s.t. \begin{cases} \sum_{j=1}^{n_J} y_{jr} = 1 \\ \sum_{r=1}^{n_J} y_{jr} = 1 \\ c_{ij} \leq CT_j \\ \sum_{i=1}^{n_I} q_{ij} = N_j \\ q_{ij} = \sum_{k=1}^{N_j} x_{ijk} \\ c_{ij} = z_{ij} \sum_{r=1}^{n_J} y_{jr} \left(\sum_{m=1}^{r} \sum_{n=1}^{n_J} y_{nm} p_n \sum_{k=1}^{N_n} x_{ink} (M + (1 - M) r^b) \right) \\ x_{ijk} \geq x_{ijk'}, \forall k' > k \\ q_{ij} \geq z_{ij} \\ a z_{ij} \geq q_{ij} \\ x_{ijk}, y_{jr}, z_{ij} \in \{0, 1\}, \forall i, j, k, r \\ n_I, n_J, N_j, q_{ij} \in Z \\ T, CT_j \in R, \forall i, j, k, r \\ -1 \leq b \leq 0; 0 \leq M \leq 1 \\ r = 1, 2, \ldots, n_J; i = 1, 2, \ldots, n_I; j = 1, 2, \ldots, n_J \end{cases} \end{cases} \tag{13}$$

3.3 Model analysis

Assume that now there are n_I parallel *serus* in SPS, and the quantity of jobs assigned to *seru* i is $N_i = \sum_{j=1}^{n_J} q_{ij}, i = 1, 2, \ldots, n_I$. Thus, the allocation of n_J jobs to n_I *serus* can be expressed as

$$A(n_J, n_I) = \left(\sum_{j=1}^{n_J} q_{1j}, \sum_{j=1}^{n_J} q_{2j}, \ldots, \sum_{j=1}^{n_J} q_{n_I j} \right) \tag{14}$$

with $\sum_{i=1}^{n_I} N_i = n_J$. Therefore, $\sum_{j=1}^{n_J} CT_j$ can be rewritten as

$$\sum_{j=1}^{n_J} CT_j = \sum_{i=1}^{n_I} \sum_{r=1}^{N_i} (N_i - r + 1)[p_j(M + (1 - M)r^b)] \tag{15}$$

Since each item of job can be only processed in one position of *seru* and each position of *seru* also can only process one item of job, so if the vector $A(n_J, n_I)$ is given, then the *seru* scheduling problem reduced to the matching problem with the objective:

$$\min T = \sum_{j=1}^{n_J} \sum_{r=1}^{N_i} p_j \theta_{jr} \tag{16}$$

where $\theta_{jr} = (N_i - r + 1)[(M + (1 - M)r^b)], i = 1, 2, \ldots, n_I, r = 1, 2, \ldots, N_i$. For provide the complexity result on *seru* scheduling problem, the following lemma should be stated first.

Lemma 1 *Let α_n and β_n be two sequences of non-negative numbers, and the sum of products $\sum_{n=1}^{N} \alpha_n \beta_n$ is the smallest if the sequences are monotonic in the opposite way, while the largest if the sequences are monotonic in the same way.*

Proof See Hardy et al. (1967) on Page 261. □

According to Lemma 1, for Eq. (16), the largest processing time should be matched to the smallest θ_{jr}, the second largest processing time with the second smallest θ_{jr}, and so on. Hence, the minimum total completion time of *seru* scheduling problem is obtained.

Theorem 4 *For the seru scheduling problem of SPS considered in this paper, it is polynomial solvable in $O(n_J^{n_I} \log^{n_J})$ time given the number of serus n_I.*

Proof For the allocation of n_J jobs to n_I *serus* $A(n_J, n_I)$, $N_i = \sum_{j=1}^{n_J} q_{ij}$ may be $0, 1, \ldots, n_J$ for $i = 1, 2, \ldots, n_I$. Thus, if the number of jobs on the first $n_I - 1$ *serus* is known, the number of jobs processed on the last n_I *seru* could be determined uniquely because $\sum_{i=1}^{n_I} N_i = n_J$. Therefore, an upper bound on the number of allocations $A(n_J, n_I)$ is $(n_J + 1)^{(n_I - 1)}$. Moreover, as a matching problem, $\min \sum_{j=1}^{n_J} p_j \theta_j$ requires $O(n \log^n)$ time to solve (Ji and Cheng, 2010). In this case, the *seru* scheduling problem of SPS in this paper $\min \sum_{i=1}^{n_I} \sum_{r=1}^{N_i} p_j \theta_{jr}$ is solvable in $O(n_J^{n_I} \log^{n_J})$ time. □

4 Solution methods

4.1 Branch and bound algorithm (B&B)

According to Theorem 4, the proposed *seru* scheduling problem model is polynomial solvable. Hence, branch and bound algorithm (B&B) is designed to solve the small-sized *seru* scheduling problem. In B&B, a node at the lth level in B&B tree represents a partial schedule where l jobs are scheduled. Let ${}_{j_{[l]}i}$ be the distributing array of job $j_{[l]}$, and

$$\begin{array}{l} {}_{j_{[l]}i} = \{\delta_{j_{[l]}1}, \delta_{j_{[l]}2}, \ldots, \delta_{j_{[l]}n_I}\} \\ \sum_{i=1}^{n_I} \delta_{j_{[l]}i} = N_{[j]} \end{array} \tag{17}$$

where $j_{[l]}$ is lth assigned job index. Thus, the node at lth level ξ_l can be defined as:

$$\xi_l = \left\{(j_{[1]}, {}_{j_{[l]}i}), (j_{[2]}, {}_{j_{[2]}i}), \ldots, (j_{[l]}, {}_{j_{[l]}i})\right\} \tag{18}$$

For example, assume that there is a SPS with two *serus* and three jobs need to be scheduled. Node {(1, (1, 1)), (3, (2, 2)), (2, (3, 2))} indicates that job 1 is split into two sub-jobs, and one is assigned to *seru* 1, while the other assigned to *seru* 2. Similarly, job 3 is also split into two sub-jobs, and two items are assigned to *seru* 1, while other two items are assigned to *seru* 2. Job 2 is split into two sub-jobs, and three items are assigned to *seru* 1, while other two items are assigned to *seru* 2. Further, the schedule sequence is job 1 → job 3 → job 2. For branching in B&B algorithm, the depth-first strategy with complete node enumeration is employed (Clausen and Perregaard, 1999), and the following dominance rule (DR) will be used in B&B algorithm: assume that there are two partial solutions π_1 and π_2 for the *seru* scheduling problem of SPS, and they are both assigned the same jobs to the *serus*. Without loss of generality if workloads of all *serus* in π_1 are no larger than that in π_2, and current π_1's total completion time is no larger than π_2, then π_1 dominates π_2 , and the partial solution π_2 should be deleted from B&B process.

In addition, the lower bound (LB) of given node will be determined as follows. Let π be a given node, and J_A be the set of assigned jobs, while J_{NA} be the set of unassigned jobs. Without splitting, the unassigned jobs in J_{NA} are re-indexed according to the total processing time by the ascending order as $1, 2, \ldots, |J_{NA}|$. Let $\mathbf{w}_{n_I} = (w_1, w_2, \ldots, w_{n_I})$, $w_i < w_{i+1}, i = 1, 2, \ldots, n_I - 1$ be the vector of current *serus* workload for the node π, then LB of π can be obtained by the following theorem.

Theorem 5 *Define a function as*

$$g_{\mathbf{w}}(x) = \sum_{i=1}^{n_I} \max\{x - w_i, 0\}, x \in (w_i, +\infty) \tag{19}$$

where $g_{\mathbf{w}}^{-1}$ is its inverse function. c_j is the completion time of job j, and AW_j is the possible additional workload from unassigned job. Then, the lower bound (LB) of the node π is the completion time of assigned job j plus the additional workload from unassigned job, i.e.,

$$LB = \sum_{j \in J_A} c_j + \sum_{j=1}^{|J_{NA}|} g_{\mathbf{w}}^{-1}\left(\sum_{\iota=1}^{j} AW_\iota\right) \tag{20}$$

Proof If $x \in [w_i, w_{i+1}]$, $w_i < w_{i+1}, i = 1, 2, \ldots, n_I - 1$, then

$$g_{\mathbf{w}}(x) = \sum_{i=1}^{n_I} \max\{x - w_i, 0\} = \sum_{\kappa=1}^{i} (x - w_\kappa) \tag{21}$$

Evidently, $g_{\mathbf{w}}(x)$ increases monotonously with x in interval $[w_i, w_{i+1}]$. If $x \geq w_{n_I}$, then

$$g_{\mathbf{w}}(x) = \sum_{i=1}^{n_I} \max\{x - w_i, 0\} = \sum_{i=1}^{n_I} (x - w_i) \tag{22}$$

and $g_{\mathbf{w}}(x)$ still increases monotonously with x in $[w_{n_I}, +\infty]$. Since the function $g_{\mathbf{w}}(x)$ is continuous, thus, in the whole definition domain, its monotonicity is preserved. In other words, $g_{\mathbf{w}}(x)$ increases monotonously with $x \in (w_1, +\infty)$.

Moreover, the optimal schedule assigns jobs in a fixed sequence according to Theorems 2 and 3. Assume that the unassigned jobs in J_{NA} are allocated in the sequence $[1], [2], \ldots, [|J_{NA}|]$ and the completion time is $c_j \leq c_{j+1}$. Now, proving

$$g_{\mathbf{w}}^{-1}\left(\sum_{\iota=1}^{j} AW_\iota\right) \leq c_{[j]} \tag{23}$$

equals to

$$g_{\mathbf{w}}(c_{[j]}) = \sum_{i=1}^{n_I} \max\{c_{[j]} - w_i, 0\} \geq \sum_{\iota=1}^{j} AW_\iota \tag{24}$$

according to $g_{\mathbf{w}}(x)$'s monotonically increasing. For the left side, $\sum_{i=1}^{n_J} \max\{c_{[j]} - w_i, 0\}$ is greater than or equal the additional workload of [1], [2], ..., [j] jobs assigned to the *seru*, while for the right side, $\sum_{\iota=1}^{j} AW_\iota$ is the minimum additional workload of [1], [2], ..., [j]. In addition, $\sum_{j \in J_A} c_j$ is a fixed value in node π, thus, LB for nodes π in the B&B process is obtained and the Theorem 5 is proved. □

Based on the discussion above, the detailed B&B algorithm procedure is shown as Algorithm 1.

Algorithm 1: Procedure of B&B algorithm for *seru* scheduling problem

Step 1. Define the node set $s = \emptyset$, and no job assigned in node π_0. Let $s = s \cup \pi_0$, and the upper bound UB$= +\infty$.
Step 2. Select a node π_l with the deepest level from s. Then, for the nodes with the same level l, the node with the minimum lower bound (LB) is also obtained by Eq. (20).
Step 3. Denote J_A as the assigned job set of node π, and the job set which is not assigned yet is $J_{NA} = n_J - J_A$.
Step 4.
while $J_{NA} \neq \emptyset$ **do**
    select a job $j \in J_A$, and let H be the set of all possible distributing arrays satisfying Eq. (17);
    while $H \neq \emptyset$ **do**
        select $h \in H$, then let $\pi_{l+1} = \pi_l \cup (j, h)$ and $LB(\pi_{l+1})$ be the lower bound of π_{l+1};
        if $l + 1 < n_J$ **then**
            if $LB(\pi_{l+1}) < UB$ *and* π_l *is not dominated by the existing nodes using DR* **then**
                $s = s \cup \{\pi_{l+1}\}$, $H = H - h$;
            end
        end
        else Calculate the total completion time T of π_{n_J};
        if $T < UB$ **then**
            $T =$ UB, and $\pi_{n_J} = \pi_{\min}$;
        end
    end
    $J_{NA} = J_{NA} - j$;
end
Step 5. Repeat step 2 until $s = \emptyset$. **Step 6.** Output the optimal schedule $\pi_{\min}$.

4.2 Local search-based hybrid genetic algorithm (LS-hGA)

Although the B&B algorithm proposed in the last subsection is useful at solving small-sized *seru* scheduling problem, its computational time is still exponentially growing as the scheduling parameters. In this case, to solve the large-sized *seru* scheduling problem efficiently, a local search-based hybrid genetic algorithm (LS-hGA) employing shortest processing time (SPT) rule will be design in this subsection. In fact, hybrid genetic algorithm has been proved to be effective for the production scheduling problems already (Al-Hakim, 2001; Defersha and Rooyani, 2020; Li and Gao, 2016; Zhang et al., 2009).

4.2.1 Individual representation

In this paper, the optimization *seru* scheduling problem concerns two points: one is determining job's sequence in SPS, and the other is job's splitting. According to the structure of proposed model and the analysis mentioned above, the encoding approach in this paper is based on two dimensions chromosome. Each chromosome in LS-hGA includes job's index order

$$OI = \{j_{[1]}, j_{[2]}, \ldots, j_{[n_J]}\} \tag{25}$$

and job's splitting ratio in n_I *serus*

$$SR = \{sr_{ij}\}, i = 1, 2, \ldots, n_I, j = 1, 2, \ldots, n_J \tag{26}$$

satisfying $\sum_{i=1}^{n_I} sr_{ij} = 1, sr_{ij} \in [0, 1]$. Therefore, the complete chromosome could be denoted as $Ch = \left\{OI, \{sr_{ij}\}\right\}$.

In LS-hGA, the splitting ratio sr_{ij} can be decoded by

$$\begin{aligned} q_{ij} &= \lfloor sr_{ij} \times N_j \rfloor, \forall i < n_I \\ q_{n_I j} &= N_j - \textstyle\sum_{i=1}^{n_I - 1} \lceil sr_{ij} \times N_j \rceil \end{aligned} \tag{27}$$

Hence, chromosome's total completion time can be gained if q_{ij} and the scheduling sequence are obtained.

4.2.2 Procedure of genetic algorithm

Denote the hth individual of LS-hGA in the tth generation as

$$Ch^{h,t} = \left\{OI^{h,t}, SR^{h,t}\right\} = \left\{\{j_{[1]}^{h,t}, j_{[2]}^{h,t}, \ldots, j_{[n_J]}^{h,t}\}, \{sr_{ij}\}^{h,t}\right\} \tag{28}$$

where $h \in H = 1, 2, \ldots, n_{pop-size}$. For each generation t, the population will be evolved according until the maximum iteration number $t_{\max}$ is reached, and the best individual will be selected to perform the local search. The detailed procedure of LS-hGA is presented in Algorithm 2 and 3.

Algorithm 2: Procedure of local research

1 **Step 1.** In SPS, find the *seru* i' having the maximal load. Then, denote the last assigned job index as j', and the sub-job size as $Q_{i'j'}$. Remove $Q_{i'j'}$ items sub-job from *seru* i'.

2 **Step 2.** Confirm the *seru* i'' having the minimum load. If $Q_{i''j'}$ is the sub-job size of job j' assigned to *seru* i'', then let $Q_{i''j'} = Q_{i''j'} + 1$, while $Q_{i'j'} = Q_{i'j'} - 1$.

3 **Step 3.** If $Q_{i'j'} = 0$, stop. If the total completion time is smaller than the original one, then the original chromosome is replaced. Otherwise, go to the step 2.

5 Computation results and analysis

To test the performance of B&B algorithm for the small sized *seru* scheduling problems and LS-hGA for the large sized ones, two sets of numerical experiments are conducted and performances are analyzed.

Algorithm 3: Procedure of hGA

Step 1. Set $h = 1$.
Step 2. From the parent chromosomes, select the chromosome according to the minimum total completion time c_{ij}. Then, put this chromosome into the children population, and $h = h + 1$.
Step 3. Following the roulette wheel rules, select two parents with index h_1 and h_2. Generate an integer in $[0, n_J]$ randomly, and denote it as IN_1.
Step 4. To generate child's job sequence, let the first IN_1 job index equal to the first IN_1 jobs' indices from $OI^{h_2,t}$. Subsequently, select the left $n_J - IN_1$ jobs' indices from $OI^{h_2,t}$. In this process, by crossover, the fixed scheduling sequence, i.e., shortest processing time (SPT) rule is used to generate the children sequence.
Step 5. In order to generate the job splitting of child, set the first job distributing array IN_1 and $n_J - IN_1$ job distributing array from $SR^{h_1,t}$, respectively.
Step 6. Randomly generate an integer in $[0, n_J]$ again and denote it as IN_2, and generate a real number RN randomly in $[0, 1]$ if $RN < P_{mutation}$. The new generated array will replace IN_2 job distributing array.
Step 7. Set $h = h + 1$.
Step 8. If $h < n_{pop-size}$, go back to the step 3.

5.1 Experiments settings

The following procedure will be used to randomly generate two sets of numerical experiments:

(1) Data setting for small sized *seru* scheduling problems: $n_I = \{2, 3, 4, 5\}$, $n_J = \{6, 7, 8, 9, 10\}$, and items quantity of per job is an integer randomly generated from uniform distribution within $[1, 10]$.
(2) Data setting for large sized *seru* scheduling problems: $n_I = \{5, 10, 15, 20\}$, $n_J = \{20, 40, 60, 80, 100\}$, and items quantity of per job is an integer randomly generated from uniform distribution within $[1, 100]$.

In addition, the processing time of a single item for a job j is also an integer generated from the discrete uniform distribution within $[10, 100]$. For each combination of n_I and n_J, 100 numerical examples are generated randomly. Further, set learning index $b = -0.8$, $M = 0.5$, $n_{pop-size} = 100$, $t_{\max} = 500$, $P_{crossover} = 0.7$ and $P_{mutation} = 0.3$. All experiments are conducted on Windows 10 with an Intel 7, 8 GB RAM system, and the algorithms are developed with MATLAB R2019a.

5.2 Results and analysis

5.2.1 Computational results of the B&B algorithm

The computational results of the B&B algorithm for the small sized *seru* scheduling problems are shown in Table 1. From Table 1, we determine that the B&B algorithm can solve the small sized problems efficiently when $n_I \leq 3$, $n_J \leq 8$, and the item of per job is smaller than 10. However, its computational time will grow exponentially as the scheduling parameters increase, and uncompleted test times will also grow vigorously when $n_I \geq 4$, $n_J \geq 9$. In this situation, we can conclude that B&B algorithm cannot cope with the large sized *seru* scheduling problems considered in this paper, and the requirement of LS-hGA is justify manifestly.

Table 1 Computational results of B&B algorithm for small sized problems

n_I	n_J	Uncompleted test times	CPU time (millisecond)
2	6	0	1.62
	7	0	2.97
	8	0	10.40
	9	0	55.39
	10	0	173.26
3	6	0	3.87
	7	0	29.61
	8	0	2647.66
	9	0	142691.35
	10	18	–
4	6	0	67.82
	7	0	1524.33
	8	0	643212.08
	9	33	–
	10	41	–
5	6	0	188.92
	7	0	3567.28
	8	38	135887.67
	9	44	–
	10	50	–

5.2.2 Computational results of LS-hGA

Similarly, the computational results of the LS-hGA for small sized *seru* scheduling problems are presented in Table 2, where RD^1 is the relative deviation of the optimal schedule obtained by LS-hGA from the result of instance obtained by B&B algorithm, and RD^1 can be calculated by

$$\begin{aligned} RD^1 &= \frac{\sum_{\gamma=1}^{100} RD_\gamma^1}{100} \times 100\% \\ RD_\gamma^1 &= \frac{T_{LS-hGA} - T_{B\&B}}{T_{B\&B}} \times 100\% \end{aligned} \tag{29}$$

After 50 runs of LS-hGA, T_{LS-hGA} is the average total completion time, and $T_{B\&B}$ is the optimal solution of instance gained from by B&B algorithm. Table 2 shows that LS-hGA has a relatively steady calculation CPU time compared with B&B algorithm, and the average CPU time of LS-hGA is just 146.72 milliseconds. The larger of n_I and n_J, the more obvious computational time advantage of LS-hGA. Moreover, when n_I is fixed, RD^1 will increase following the n_J; while the RD^1 will increase following the n_I even though n_J/n_I decreases.

Further, the computational results of the LS-hGA for large sized *seru* scheduling problems are provided in Table 3, where RD^2 is the relative deviation of the optimal schedule provided by LS-hGA from the best solution $T_{\min}$. Similarly, RD^2 is obtained by

$$\begin{aligned} RD^2 &= \frac{\sum_{\gamma=1}^{100} RD_\gamma^2}{100} \times 100\% \\ RD_\gamma^2 &= \frac{T_{LS-hGA} - T_{\min}}{T_{\min}} \times 100\% \end{aligned} \tag{30}$$

Table 2 Computational results of LS-hGA for small sized problems

n_I	n_J	CPU time (millisecond)	n_J/n_I	RD^1(%)
2	6	116.28	3.00	1.19
	7	122.91	3.50	2.31
	8	119.37	4.00	3.72
	9	107.43	4.50	4.01
	10	131.72	5.00	5.29
3	6	146.72	2.00	4.11
	7	181.48	2.33	6.69
	8	147.66	2.66	7.33
	9	176.35	3.00	10.26
	10	188.27	3.33	–
4	6	138.61	1.50	8.17
	7	184.22	1.75	10.23
	8	149.42	2.00	14.92
	9	163.24	2.25	–
	10	171.49	2.50	–
5	6	162.93	1.20	10.59
	7	188.37	1.40	12.74
	8	164.26	1.60	18.37
	9	201.38	1.80	–
	10	173.57	2.00	–

Similarly, T_{LS-hGA} is the average total completion time, and $T_{\min}$ is the minimum total completion time of all 50 runs.

From Table 3, we know that LS-hGA still effective for solving the large sized *seru* scheduling problems. On the hand, CPU time is more sensitive with the quantities of *serus* n_I, for example, the CPU time runs-up sharply from $n_I = 15$ to $n_I = 20$. On the other hand, if n_I is fixed, the CPU time is almost steady. RD^2 is also sensitive with the quantities of *serus* n_I, and it also generally increases following the quantities of jobs n_J increasing. Hence, in practice, the production manager must let the appropriate quantities of *serus* be a top priority in SPS to improve the efficiency of system.

5.2.3 Comparison analysis

To scrutinize the management insights for *seru* production practice, comparison analysis for both job splitting and Dejong's learning effect are made.

For evaluating the effect of job splitting in *seru* scheduling problem, the comparison of RD^2 with un-splitting is performed by the large sized problems, and the results are shown in Table 4. It can be known that when n_J/n_I is large, the performance of un-splitting type is better, and when n_J/n_I is small, the job splitting performs well. That is because if n_J/n_I is large, earlier job splitting may have significant indirect costs to unassigned jobs in production practice, such as the extended set-up time, insufficient learning effect, and so on. If n_J/n_I is small, earlier job splitting will not have many additional costs to unassigned jobs because the balance between *serus* is one of the most important factors to minimize the total completion time for SPS.

Table 3 Computational results of LS-hGA for large sized problems

n_I	n_J	CPU time (millisecond)	n_J/n_I	RD^2 (%)
5	20	724.66	4.00	6.54
	40	768.39	8.00	10.21
	60	836.47	12.00	14.33
	80	810.14	16.00	16.57
	100	859.62	20.00	20.49
10	20	924.16	2.00	29.48
	40	897.63	4.00	33.06
	60	975.29	6.00	42.62
	80	1027.68	8.00	40.97
	100	1283.87	10.00	51.65
15	20	2479.35	1.33	56.09
	40	2184.20	2.67	67.17
	60	2639.47	4.00	69.61
	80	2968.44	5.33	72.24
	100	3047.92	6.67	78.55
20	20	10627.09	1.00	74.32
	40	13412.75	2.00	86.37
	60	12141.96	3.00	82.96
	80	16843.77	4.00	93.08
	100	18642.55	5.00	89.70

Table 4 Comparison of RD^2 for job un-splitting and splitting

n_I	n_J	n_J/n_I	Splitting	Un-splitting
5	20	4.00	6.54	9.15
	40	8.00	10.21	9.82
	60	12.00	14.33	5.33
	80	16.00	16.57	6.75
	100	20.00	20.49	6.06
10	20	2.00	29.48	40.27
	40	4.00	33.06	38.96
	60	6.00	42.62	50.09
	80	8.00	40.97	40.18
	100	10.00	51.65	39.42
15	20	1.33	56.09	90.44
	40	2.67	67.17	86.21
	60	4.00	69.61	88.14
	80	5.33	72.24	83.82
	100	6.67	78.55	87.66
20	20	1.00	74.32	98.53
	40	2.00	86.37	92.45
	60	3.00	82.96	95.09
	80	4.00	93.08	97.75
	100	5.00	89.70	95.80

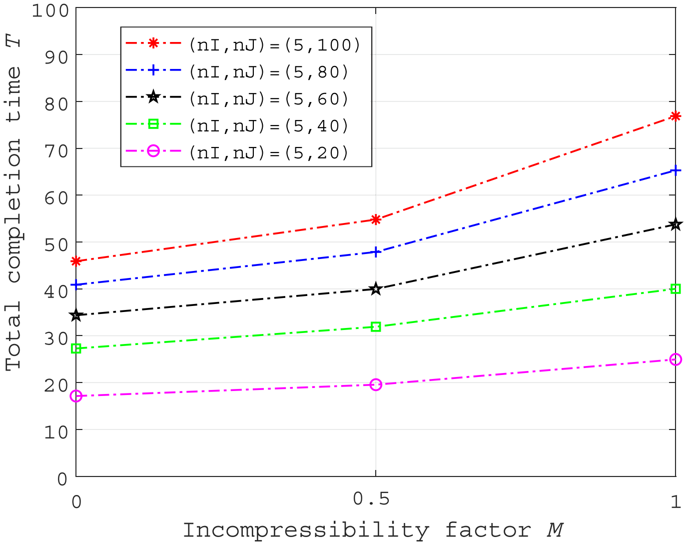

Fig. 5 Total completion time T with $M = 1$ ($n_I = 5$)

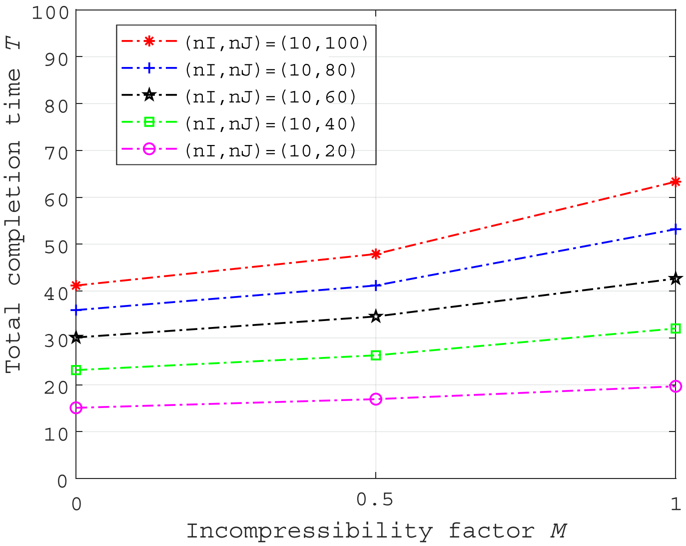

Fig. 6 Total completion time T with $M = 1$ ($n_I = 10$)

Further, for testing the Dejong's learning effect on *seru* scheduling problems, the incompressibility factor $M = 1$ is performed by the large sized problems (no learning effect). Detailed results for each case ($n_I = 5, 10, 15, 20$, $n_J = 20, 40, 60, 80, 100$) with different values of T are shown in Figs. 5, 6, 7 and 8.

Generally, for *seru* scheduling problems, the influence on the total completion time T from learning effect is significant. T usually reaches the maximum value (the worst one) in each case when $M = 1$. We also find that the learning effect becomes more evident along with more n_J jobs in the same amount of *seru* since increased range of the total completion

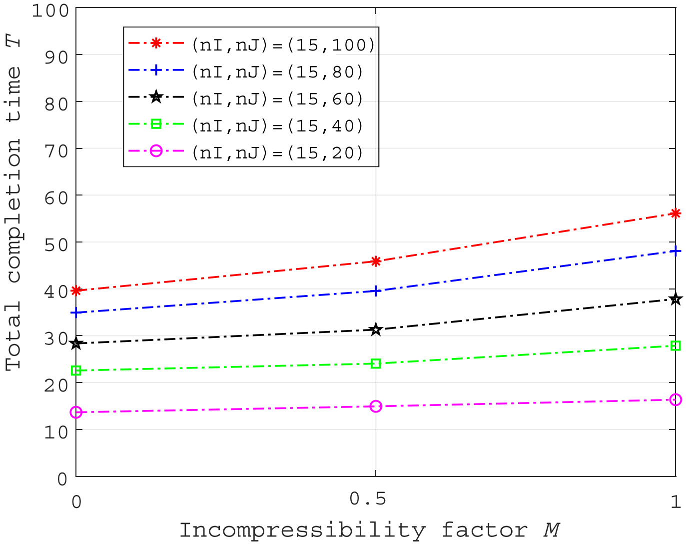

Fig. 7 Total completion time T with $M = 1$ ($n_I = 15$)

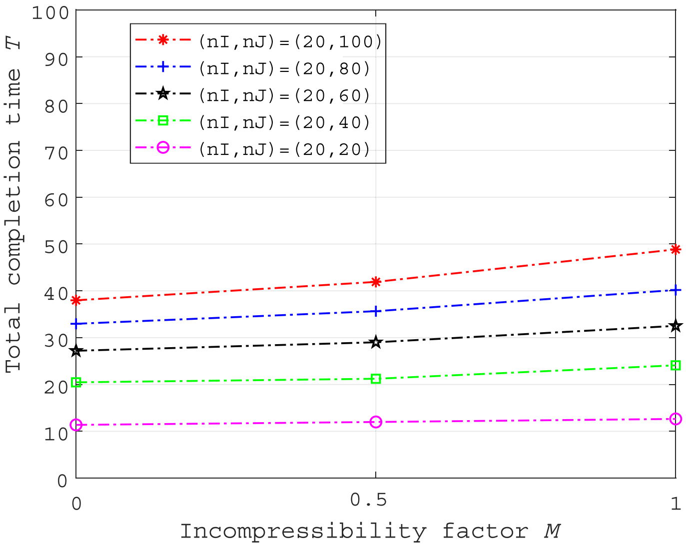

Fig. 8 Total completion time T with $M = 1$ ($n_I = 20$)

time T is descending. For example, $T(n_I = 5, n_J = 20) - T(n_I = 5, n_J = 40) > T(n_I = 5, n_J = 40) - T(n_I = 5, n_J = 60)$. This phenomenon indicates that the learning effect should be considered in practice for *seru* production managers, especially when n_J/n_I is large. Moreover, it is interesting to observe that the slope learning curve is decrease from $M = 0 \rightarrow M = 0.5$ to $M = 0.5 \rightarrow M = 1$, which indicates that the job's processing time of decrease continuously and stabilize to a fixed value even $M = 0$ (the strongest learning effect). Hence, the advantage of DeJong's learning effect is validated. Therefore, to achieve high production efficiency, flexibility, and quick response, *seru* production managers of should also give attention of DeJong's learning effect in SPS.

6 Conclusions

This paper concerns with the scheduling problem in *seru* production system considering DeJong's learning effect and job splitting to minimize the total completion time. A non-linear integer programming model is developed, and B&B algorithm is designed for the small sized problem while LS-hGA is for the large one. Computational results of experiments demonstrate the effectiveness of proposed solution methods. Managerial insights are also provided to *seru* production managers.

Future research will focus on applying the proposed model and algorithm to other *seru* types including the divisional *seru* and rotating *seru*. Also, considering the conflicts of different decision goals in the practical decision-making process, the multi-objective model should be considered in *seru* scheduling problem. Additionally, the uncertain factors, such as stochastic product processing time, uncertain worker's shortage, or redundancy, also should be concerned. Finally, software development for the practical application in SPS based on this paper is supposed to be considered in the future.

Acknowledgements This research was sponsored by National Natural Science Foundation of China (Grant Nos. 71401075, 71801129), the Natural Science Foundation of Jiangsu Province (Grant No. BK20180452), and the Fundamental Research Funds for the Central Universities (Grant No. 30920010021). We would like to give our great appreciation to all the reviewers and editors who contributed this research.

References

Al-Hakim, L. (2001). An analogue genetic algorithm for solving job shop scheduling problems. *International Journal of Production Research, 39*(7), 1537–1548.

Ayough, A., Hosseinzadeh, M., & Motameni, A. (2020). Job rotation scheduling in the seru system: Shake enforced invasive weed optimization approach. *Assembly Automation, 40*(3), 461–474.

Bachman, A., & Janiak, A. (2004). Scheduling jobs with position-dependent processing times. *Journal of the Operational Research Society, 55*, 257–264.

Biskup, D. (1999). Single-machine scheduling with learning considerations. *European Journal of Operational Research, 115*(1), 173–178.

Biskup, D. (2008). A state-of-the-art review on scheduling with effects. *European Journal of Operational Research, 188*, 315–329.

Chen, T., Cheng, C., & Chou, Y. (2020). Multi-objective genetic algorithm for energy-efficient hybrid flow shop scheduling with lot streaming. *Annals of Operations Research, 290*, 813–836.

Cheng, T., Kuo, W., & Yang, D. (2013). Scheduling with a position-weighted learning effect based on sum-of-logarithm-processing-times and job position. *Information Sciences, 221*, 490–500.

Clausen, J., & Perregaard, M. (1999). On the best search strategy in parallel branch-and-bound: Best-first search versus lazy depth-first search. *Annals of Operations Research, 90*, 1–17.

Defersha, F., & Rooyani, D. (2020). An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. *Computers & Industrial Engineering, 147*, 106605.

D & M Nikkei Mechanical. (2003). The challenge of Canon-Part 3. 588, 70–73.

Hardy, G., Littlewood, J., & Polya, G. (1967). *Inequalities*. Cambridge: Cambridge University Press.

Hisashi, S. (2006). *The change of consciousness and company by cellular manufacturing in Canon Way*. Tokyo: JMAM. (in Japanese).

Huang, R. (2010). Multi-objective job-shop scheduling with lot-splitting production. *International Journal of Production Economics, 124*(1), 206–213.

Huang, R., & Yu, T. (2017). An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting. *Applied Soft Computing, 57*, 642–656.

Janiak, A., Kovalyov, M., & Lichtenstein, M. (2013). Strong NP-hardness of scheduling problems with learning or aging effect. *Annals of Operations Research, 206*(1), 577–583.

Ji, M., & Cheng, T. (2010). Scheduling with job-dependent learning effects and multiple rate-modifying activities. *Information Processing Letters, 110*, 460–463.

Jiang, Y., Zhang, Z., Gong, X., & Yin, Y. (2021). An exact solution method for solving seru scheduling problems with past-sequence-dependent setup time and learning effect. *Computers & Industrial Engineering, 158*, 107354.

Kaku, I. (2017). Is seru a sustainable manufacturing system? *Procedia Manufacturing, 8*, 723–730.

Kaku, I., Gong, J., Tang, J., & Yin, Y. (2009). Modelling and numerical analysis of line-cell conversion problems. *International Journal of Production Research, 47*(8), 2055–2078.

Kim, H. (2018). Bounds for parallel machine scheduling with predefined parts of jobs and setup time. *Annals of Operations Research, 261*, 401–412.

Kim, Y., & Kim, R. (2020). Insertion of new idle time for unrelated parallel machine scheduling with job splitting and machine breakdowns. *Computers & Industrial Engineering, 147*, 106630.

Kimura, T., & Yoshita, M. (2004). Remaining the current situation is dangerous: Seru Seisan. *Nikkei Monozukuri, 7*, 38–61. (In Japanese).

Kono, H. (2004). The aim of the special issue on seru manufacturing. *IE Review, 45*(1), 4–5.

Li, X., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics, 174*, 93–110.

Lian, J., Liu, C., Li, W., & Yin, Y. (2018). A multi-skilled worker assignment problem in seru production systems considering the worker heterogeneity. *Computers & Industrial Engineering, 118*, 366–382.

Liu, C., Yang, N., Li, W., Lian, J., Evans, S., & Yin, Y. (2013). Training and assignment of multi-skilled workers for implementing seru production systems. *International Journal of Advanced Manufacturing Technology, 69*(5–8), 937–959.

Liu, C., Stecke, K., Lian, J., & Yin, Y. (2014a). An implementation framework for seru production. *International Transactions in Operational Research, 21*(1), 1–19.

Liu, C., Wang, C., Zhang, Z., & Zheng, L. (2014b). Scheduling with job-splitting considering learning and the vital-few law. *Computers & Operations Research, 90*, 264–274.

Luo, L., Zhang, Z., & Yin, Y. (2016). *Seru* loading with worker-operation assignment in single period. In *2016 IEEE international conference on industrial engineering and engineering management (IEEM)* (pp. 1055–1058).

Luo, L., Zhang, Z., & Yin, Y. (2017). Modelling and numerical analysis of seru loading problem under uncertainty. *European Journal of Industrial Engineering, 11*(2), 185–204.

Luo, L., Zhang, Z., & Yin, Y. (2021). Simulated annealing and genetic algorithm based method for a bi-level seru loading problem with worker assignment in seru production systems. *Journal of Industrial and Management Optimization, 17*(2), 779–803.

Mosheiov, G. (2001). Scheduling problems with a learning effect. *European Journal of Operational Research, 132*(3), 687–693.

Mosheiov, G., & Sidney, J. (2003). Scheduling with general job-dependent learning curves. *European Journal of Operational Research, 147*(3), 665–670.

Nessah, R., & Chu, C. (2010). Infinite split scheduling: a new lower bound of total weighted completion time on parallel machines with job release dates and unavailability periods. *Annals of Operations Research, 181*, 359–375.

Nikkei-Business. (2016). *How to handle Prius: Its delivery time is more than half a year*. Tokyo: Nikkei Business. (in Japanese).

Noguchi, H. (2003). Production innovation in Japan. Nikkan Kogyo Shimbun. (in Japanese).

Pei, J., Cheng, B., Liu, X., Pardalos, P., & Kong, M. (2019). Single-machine and parallel-machine serial-batching scheduling problems with position-based learning effect and linear setup time. *Annals of Operations Research, 272*, 217–241.

Rostami, M., Nikravesh, S., & Shahin, M. (2020). Minimizing total weighted completion and batch delivery times with machine deterioration and learning effect: A case study from wax production. *Operational Research, 20*(3), 1255–1287.

Roth, A., Singhal, J., Singhal, K., & Tang, C. (2016). Knowledge creation and dissemination in operations and supply chain management. *Production and Operations Management, 25*(9), 1473–1488.

Shao, L., Zhang, Z., & Yin, Y. (2016). A bi-objective combination optimisation model for line-seru conversion based on queuing theory. *International Journal of Manufacturing Research, 11*(4), 322–338.

Stecke, K., Yin, Y., Kaku, I., & Murase, Y. (2012). Seru: The organizational extension of JIT for a super-talent factory. *International Journal of Strategic Decision Sciences, 3*(1), 105–118.
Sun, L., Cui, K., Chen, J., Wang, J., & He, X. (2013). Some results of the worst-case analysis for flow shop scheduling with a learning effect. *Annals of Operations Research, 211*, 481–490.
Sun, W., Wu, Y., Lou, Q., & Yu, Y. (2019). A cooperative coevolution algorithm for the seru production with minimizing makespan. *IEEE Access, 7*, 5662–5670.
Sun, W., Yu, Y., Lou, Q., Wang, J., & Guan, Y. (2020). Reducing the total tardiness by seru production: Model, exact and cooperative coevolution solutions. *International Journal of Production Reseaech, 58*(21), 6441–6452.
Treville, S., Ketokivi, M., & Singhal, V. (2017). Competitive manufacturing in a high-cost environment: Introduction to the special issue. *Journal of Operations Management, 49–51*, 1–5.
Wang, L., Zhang, Z., & Yin, Y. (2019). Order acceptance and scheduling considering lot-spitting in *seru* production system. In *Proceeding of 2019 IEEE international conference on industrial engineering and engineering management* (pp. 1305-1309).
Wang, J., Liu, C., & Zhou, M. (2020). Improved bacterial foraging algorithm for cell formation and product scheduling considering learning and forgetting factors in cellular manufacturing systems. *IEEE Systems Journal, 14*(2), 3047–3056.
Wright, T. (1936). Factors affecting the cost of airplanes. *Journal of Aeronautical Sciences, 3*, 122–128.
Yamada, H. (2009). *Waste reduction*. Tokyo: Gentosha. (in Japanese).
Yin, Y., Stecke, K., Swink, M., & Kaku, I. (2017). Lessons from seru, production on manufacturing competitively in a high cost environment. *Journal of Operations Management, 49–51*, 67–76.
Yin, Y., Stecke, K. E., Li, D. (2018). The evolution of production systems from Industry 2.0 through Industry 4.0. *International Journal of Production Research*, 56(1&2), 848–861.
Yılmaz, Ö. (2020a). Attaining flexibility in seru production system by means of Shojinka: An optimization model and solution approaches. *Computers & Operations Research, 119*, 104917.
Yılmaz, Ö. (2020b). Operational strategies for seru production system: A bi-objective optimisation model and solution methods. *International Journal of Production Research, 58*(11), 3195–3219.
Yu, Y., & Tang, J. (2019). Review of seru production. *Frontiers of Engineering Management, 6*(2), 183–192.
Yu, Y., Gong, J., Tang, J., Yin, Y., & Kaku, I. (2012). How to carry out assembly line-cell conversion? A discussion based on factor analysis of system performance improvements. *International Journal of Production Research, 50*(18), 5259–5280.
Yu, Y., Tang, T., Yin, Y., & Kaku, I. (2013). Reducing worker(s) by converting assembly line into a pure cell system. *International Journal of Production Economics, 145*, 799–806.
Yu, Y., Tang, T., Yin, Y., & Kaku, I. (2014). Mathematical analysis and solutions for multi-objective line-cell conversion problem. *European Journal of Operational Research, 236*, 774–786.
Yu, Y., Sun, W., Tang, J., & Wang, J. (2017). Line-hybrid seru system conversion: Models, complexities, properties, solutions and insights. *Computers & Industrial Engineering, 103*, 282–299.
Zhang, Y., Li, X., & Wang, Q. (2009). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research, 193*, 869–876.
Zhang, X., Liu, C., Li, W., Evans, S., & Yin, Y. (2017). Effects of key enabling technologies for seru production on sustainable performance. *Omega, 66*, 290–307.
Zhang, Z., Wang, L., Song, X., Huang, H., & Yin, Y. (2021). Improved genetic-simulated annealing algorithm for seru loading problem with downward substitution under stochastic environment. *Journal of the Operational Research Society*. https://doi.org/10.1080/01605682.2021.1939172
Zhang, Z., Song, X., Huang, H., Zhou, X., & Yin, Y. (2022). Logic-based Benders decomposition method for the seru scheduling problem with sequence-dependent setup time and DeJong's learning effect. *European Journal of Operational Research, 297*, 866–877.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.