DBS-21-02

「Logic-based Benders decomposition method for the *seru* scheduling problem with sequence-dependent setup time and DeJong's learning effecgt」

Zhe Zhang[a], Xiaoling Song[a], Huijung Huang[a], Xiaoyang Zhou[b], Yong Yin[c]

[a]School Economics & Management, Nanjing University of Science and Technology, Nanjing 210094, P. R. CHINA
[b]School of Management, Xi'an Jiaotong University, Xi'an 710049, P. R. CHINA
[c]Graduate School of Business, Doshisha University, Karasuma-Imadegawa, Kamigyo-ku, Kyoto, 602-8580, Japan

June, 2021

Abstract

This paper concentrates on the scheduling problem in *seru* production system (SPS), where *seru* is a successful new-type production mode arising from the Japanese labor-intensive electronic assembly industry. Motivated by the practical situations, the sequence-dependent setup time and DeJong's learning effect are considered in *seru* scheduling problems, and the objective is to minimize the make span. The *seru* scheduling problem is formulated as a mixed integer programming (MIP), and then reformulated to a set partitioning master problem and some independent sub-problems by employing the logic-based Benders decomposition (LBBD) method. Subsequently, the set partitioning master problem is used to assign jobs to *serus* of SPS, and the subproblems are applied to find the optimal schedules in each *seru* given the assignment of the master problem. Finally, computational studies are made, and results indicate that the LBBD method is able to return high-quality schedules for solving *seru* scheduling problems.

Keywords: scheduling, *seru* production system, decomposition, sequence-dependent setup time, learning effect

Manuscript

Click here to view linked References

Logic-based Benders decomposition method for the *seru* scheduling problem with sequence-dependent setup time and DeJong's learning effect

Zhe Zhang[a], Xiaoling Song[a], Huijung Huang[a], Xiaoyang Zhou[b], Yong Yin[c]

[a] *School Economics & Management, Nanjing University of Science and Technology, Nanjing 210094, P. R. CHINA*
[b] *School of Management, Xi'an Jiaotong University, Xi'an 710049, P. R. CHINA*
[c] *Graduate School of Business, Doshisha University, Karasuma-Imadegawa, Kamigyo-ku, Kyoto, 602-8580, Japan*

Abstract

This paper concentrates on the scheduling problem in *seru* production system (SPS), where *seru* is a successful new-type production mode arising from the Japanese labor-intensive electronic assembly industry. Motivated by the practical situations, the sequence-dependent setup time and DeJong's learning effect are considered in *seru* scheduling problems, and the objective is to minimize the makespan. The *seru* scheduling problem is formulated as a mixed-integer programming (MIP), and then reformulated to a set partitioning master problem and some independent subproblems by employing the logic-based Benders decomposition (LBBD) method. Subsequently, the set partitioning master problem is used to assign jobs to *serus* of SPS, and the subproblems are applied to find the optimal schedules in each *seru* given the assignment of the master problem. Finally, computational studies are made, and results indicate that the LBBD method is able to return high-quality schedules for solving *seru* scheduling problems.

Keywords: scheduling, *seru* production system, decomposition, sequence-dependent setup time, learning effect

1. Introduction

Along with the high-speed development of information technology, product life cycles decrease and production demands are fluctuating. More manufacturing companies recognize that the fast response is another dimension of demand in production practice apart from the product volume and product variety (Yin et al., 2018 [63]). In this situation, traditional assembly lines, including Toyota production system (TPS) and lean, can not cope with the volatile market with short product life cycles, uncertain product types, and fluctuating production volumes because they are fit for a stable market and can not make speedy and timely adjustments. Accordingly, *seru* seisan ("*seru*" means cell, and "seisan" means production in Japanese), which is a new production organization deriving from the Japanese electronic industry practice, is applied to offset the influence of fluctuant demands, and it could achieve efficiency, flexibility and fast response, simultaneously (Stecke et al., 2012 [50]). *Seru* production system (SPS) is reconfigured from the traditional assembly line, and it is composed of one or more workers and some simple and cheap equipment to assemble products. Fig. 1 is an example of converting an assembly line into SPS contains three parallel *serus*. In *seru* 1, a partially cross-trained worker 1 handles tasks 1 to 3, and a partially cross-trained worker 2 handles tasks 4 to 5; in *seru* 2, completely cross-trained workers 3 and 4 handle all tasks from 1 to 5 without disruption, and they move repeatedly from the entrance to the exit of *seru* 2; in *seru* 3, a single completely cross-trained worker 5 handles all tasks from 1 to 5, and this worker is equipped with not only technical but also managerial skills. Compared to the traditional production mode, *serus* in SPS can be constructed, modified, dismantled, and reconstructed frequently in a short time to accommodate the volatile market requirement, so SPS is very flexible. The comparison between SPS and other production systems, such as Ford, TPS, and cellular manufacturing (CM), are presented in Liu et al., (2014) [32]), Yin et al. (2018) [63], and Yu and Tang (2019) [69].

In fact, since SPS combines advantages of Toyota's lean philosophy and Sony's one person production organization, it has brought tremendous benefits to its users and is denoted as "the next generation of lean" in Japanese

Email address: zhangzhe@njust.edu.cn. (Zhe Zhang)

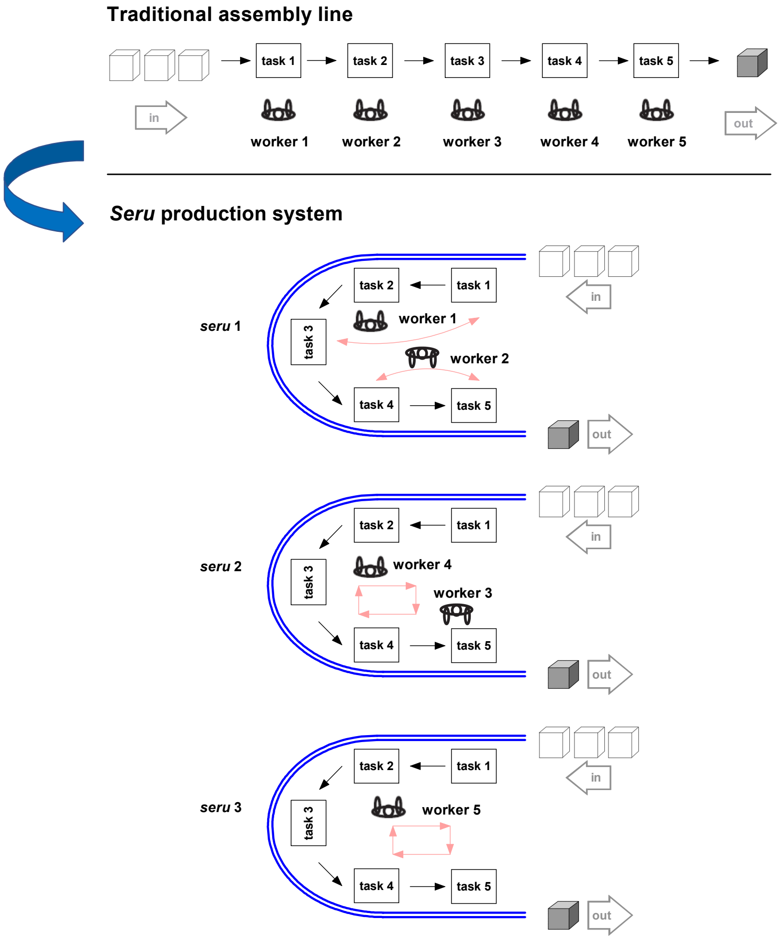

Figure 1: An example of an assembly line to be decomposed into SPS

production practice (Shinobu, 2003 [49]; Yin et al., 2007 [62]). Many global electronics giants, such as Sony, Canon, Panasonic, Samsung, and LG, have adopted the *seru* production mode already (Yin et al., 2018 [63]). As it turns out, SPS can reduce space requirements, workforce quantity, lead time, setup time, work-in-process (WIP) inventory, finished product inventory, and cost (Stecke et al., 2012 [50]). Although so many benefits have been obtained, SPS is still largely unknown outside Japan and the research on *seru* is few due to its short history. Fortunately, because SPS can achieve high efficiency, high flexibility and fast response simultaneously in practice, it has attracted vast attention from some leading scholars and practitioners in operations management (OM) recently. Hopp and Spearman (2020) [24] provided a useful construct for lean training and implementation by describing four "Lenses of Lean", and showed that SPS can elevate both efficiency and responsiveness. Lewis (2019) [27] summarized an overview of the current and classic OM research and indicated that *seru* production mode was one feasible solution to cope with market requirements for smaller volumes and higher variety. Yin et al. (2018) [63] analyzed the demand drivers for production system evolution from Industry 2.0 through Industry 4.0 by employing supply-demand relationships, and pointed out that SPS may be the potential smart manufacturing system in the Internet of Things (IoT) age. Treville et al. (2017) mentioned that some Japanese electronics enterprises could achieve a fast response to market demands using SPS in [53]. Roth et al. (2016) [45] listed eight possible future research directions in OM after summarizing the development process of OM over the past 25 years, and pointed out that *seru* was one of the new research fields worthy of our attention. In this paper, we will study the *seru* scheduling problem considering both the sequence-dependent setup time and DeJong's learning effect for the first time, and hope that this research could improve the theoretical research in SPS and provide professional guidance to *seru* production managers.

According to the evolution of SPS, there are three types of *seru* in production practice, including divisional *seru*, rotating *seru* and *yatai* (Akino, 1997 [5]). At the beginning, if the worker in an assembly line is partially cross-trained, which means that he/she can take charge of more than one task, then this assembly line can be converted into a number of short lines. For example, divisional *serus* are constructed (example of divisional *seru* is *seru* 1 in Fig. 1). Subsequently, with more worker training and skills improving, some workers in divisional *serus* are completely cross-trained, and they can handle all tasks of a job. The equipments are shared in the *serus*, and these completely cross-trained workers move one after the other until a job is finished. The worker will return to the first workstation and start a new round in this *seru* when the job is finished. Hence, the rotating *seru* formation is accomplished (example of rotating *seru* is *seru* 2 in Fig. 1). Finally, as the technical and managerial skill of completely cross-trained workers improving further, some rotating *serus* could evolve into *yatais*, which contain only one completely cross-trained worker who takes charge of all tasks from start to finish. *Yatai* is a small but highly autonomous single-person production unit, and it is the highest evolution form of *seru* production mode implementation (example of *yatai* is *seru* 3 in Fig. 1). The detailed description about these three types of *seru* can be found in Stecke et al. (2012) [50], Liu et al. (2014) [32], Yu and Tang (2019) [69]. In this paper, the *seru* type is *yatai* since it is sensitive to the learning effect. The divisional *seru* and rotating *seru* are left for future research.

In production practice, a new planning system, named just-in-time organization system (JIT-OS), is used to manage and control SPS, and practical industrial cases of JIT-OS are shown in Yin et al. (2008) [61] and Stecke et al. (2012) [50]. As an extension of the Toyota's traditional JIT material system (JIT-MS), the implementation mechanism of JIT-OS is similar: the correct *serus*, in the right place, at the appropriate time, in the exact amount (Stecke et al., 2012 [50]). The difference between JIT-OS and JIT-MS is that JIT-OS focuses on organizations (i.e., *serus*) while JIT-MS on materials (Liu et al., 2014 [32]). There are three decisions in JIT-OS, including *seru* formation, *seru* loading and *seru* scheduling. First, a SPS with an appropriate number of *serus* is configured by *seru* formation. Kaku et al. (2009) [25] studied the *seru* formation problem by computational experiments, and figured out that the appropriate number of *serus* when an assembly line was converted into SPS and the appropriate number of workers allocated to each *seru* in different cases. Liu et al. (2013) [31] constructed a bi-objective mathematical model for *seru* formation, and investigated the training and assignment problem of workers when an assembly line is reconfigured into SPS. Yu et al. (2012, 2013, 2014) [65, 66, 67] constructed a series of mathematical models to evaluate the performance of converting an assembly line to SPS, and the mathematical characteristics were also analyzed. Shao et al. (2016) [48] developed a multi-objective combinational optimization model based on queuing theory for *seru* formation problems with stochastic customer's orders. Yu et al. (2017) [68] studied *seru* formation problem from different aspects, including mathematical models, complexities, properties, solutions and insights. Ren and Wang (2019) [44] studied the effect of *seru* formation problem on the waiting time from the customer perspective, and investigated the average waiting queue length changed by the line-*seru* conversion. They pointed out that this conversion can reduce the average waiting queue length in multi-variety and small-batch production. Zhang et al. (2020) [70] designed a PSO-based algorithm for the *seru* formation problem in an unbalanced SPS by considering the lot splitting and setup time. After the SPS is configured, the customer-ordered products are allocated to *serus* properly by *seru* loading. For *seru* loading problem, Lian et al. (2012) [29] dealt with *seru* loading problem to minimize the variable production cost of all *serus* in SPS, and designed a heuristic algorithm according to the earliest due date (EDD) principle. Luo et al. (2016) [35] proposed a combinatorial optimization model for *seru* loading problems in divisional *seru* in a single period, and took the worker-operation assignment into account. To minimize the makespan and the total tardiness penalty cost, Luo et al. [36] studied the *seru* loading problem under uncertainty, where the proportional coefficient, setup time, tardiness penalty coefficient were fuzzy random variables. Then, Luo et al. (2019) [37] constructed a bi-level programming model to address the *seru* loading problem with worker assignment, and designed a simulated annealing and genetic algorithm-based method as the solution method. With the objective of minimizing the makespan, Sun et al. (2019) [51] developed a cooperative co-evolution algorithm which combined genetic algorithm and ant colony optimization algorithm for solving both *seru* formulation and *seru* loading problems at the same time. At last, the production plan will be obtained within the due date by *seru* scheduling. Unfortunately, due to the complexity, the studies on *seru* scheduling problem in SPS are still very rare.

In this situation, the methodology of parallel machine scheduling (PMS) problem inspires us to solve *seru* scheduling problems because SPS is a typical parallel production system. Generally speaking, there are two main categories of problems in PMS, including identical parallel machine scheduling and unrelated parallel machine scheduling problems. Due to the differences of both worker skill levels in SPS and production efficiency of each *seru*, the unrelated PMS problem is more likely to be similar to a *seru* scheduling problem. However, the *seru* scheduling problem is much more complicated than the unrelated PMS problem. For example, even in the simplest *seru* type of *yatai*, where only one completely cross-trained worker takes charge of all tasks from start to finish, it is still an unrelated PMS problem with worker element consideration, including worker heterogeneity with different skill levels, learning effects, and so on. In divisional *seru*, apart from these worker elements, matching between workers and tasks, as well as worker assignment are needed to be considered. While in rotating *seru*, apart from allocating workers to the appropriate *seru*, the production efficiency of each *seru* will be determined by the slowest worker since the workers move one after the other in the *seru*. Hence, the unrelated PMS problem is a special case of *seru* scheduling problems, and its related effective theory and methods could be tested and applied in the new-type SPS. In fact, for unrelated PMS problems, many effective models and algorithms have been proposed. Fanjul-Peyro and Ruiz (2010) [16] proposed a set of simple iterated greedy local search-based metaheuristics for unrelated PMS problems, and the generated solutions presented a very good quality in a very short amount of time. Vallad and Ruiz (2011) [55] provided a genetic algorithm and included a fast local search and a local search enhanced crossover operator for the unrelated parallel machine scheduling problem, in which machine and job sequence-dependent setup times were considered. Lin et al. (2013) [30] proposed two heuristics and a genetic algorithm (GA) to obtain non-dominated solutions to multiple-objective unrelated PMS problems, and the computational results showed that the proposed heuristics were computationally efficient and provided solutions of reasonable quality. Fanjul-Peyro et al. (2019) [17] proposed new mixed integer linear programs and a decomposition algorithm for unrelated PMS problems, and obtained optimal solutions for extremely large instances of up to 1000 jobs and 8 machines. Liu and Lei (2020) [33] designed an artificial bee colony algorithm for distributed unrelated PMS problems with preventive maintenance to minimize makespan, and the whole swarm was divided into one employed bee colony and three onlooker bee colonies. Ewees et al. (2021) [15] modified a salp swarm algorithm based on the firefly algorithm to enhance the solution quality of unrelated PMS problems, and carried out an extensive comparison to several existing metaheuristic methods. Cheng and Sin (1990) [13], Mokotoff (2001) [38], Edis et al. (2013) [14] proposed a survey of PMS problem research. In this paper, we first employ the methodology of unrelated PMS problems for *seru* scheduling problems, which is a new idea in SPS. Further, it will provide a broader application area for PMS theory and methods.

Moreover, in *seru* production scheduling problems, the consideration of setup time between jobs is an essential issue because a minimum time must elapse between consecutive jobs executed in the same *seru*. Setup time is the time for preparing necessary resources, such as workers or tools, to perform a task, i.e., operation or job (Salvendy, 2001 [47]), and it has been proved to be important in some industrial applications. For example, the reactors must be cleaned in chemical plants when changing from processing one mixture to another; also, in printed circuit board assembly, it was reported that from 20% to 50% loss of available capacity may arise from setup activities (Trovinger and Bohn, 2005 [54]). Here are two types of setup time in scheduling problems, including sequence-independent and sequence-dependent, respectively. If the setup time depends only on the job to be processed, then it is sequence-independent; otherwise, if the setup time depends on both the job to be processed and its immediately preceding job, it is sequence-dependent (Wilbrecht and Prescott, 1969 [58]; Lee et al., 1997 [26]). In this paper, we will consider sequence-dependent setup time because it is sensitive and significant in SPS. Actually, the necessity of considering sequence-dependent setup time in production scheduling problems has been recognized widely in several studies. Ruiz and Maroto (2006) [46] designed a genetic algorithm which incorporated new characteristics and four new crossover operators for a complex generalized flowshop scheduling problem with sequence-dependent setup time. Pearn et al. (2008) [42] addressed the multi-stage wafer probing scheduling problem with reentry and sequence-dependent setup time, and proposed two strategies to solve this problem for minimizing the total workload. Alfieri (2009) [1] studied a practical multi-objective flowshop scheduling problem with sequence-dependent setup time in a cardboard company, and presented a simulation-based environment where the production sequence can be found by a tabu-search based heuristic algorithm interactively. Nishi and Hiranaka (2013) [39] applied the lagrangian relaxation and cut generation technique to solve sequence-dependent setup time flowshop scheduling problems, and the proposed problem with additional setup time constraints was solved by a novel dynamic programming effectively. Pan et al. (2017) [40] proposed a total of nine algorithms for the hybrid flowshop scheduling problem with sequence-dependent setup times, and conducted a set of computational experiments to demonstrate the effectiveness of algorithms. Li et al. (2020) [28] designed a machine position-based mathematical model and proposed an improved artificial bee colony algorithm for the distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup time. Allahverdi et al. (1999, 2008, 2015) reviewed the scheduling problems with setup time in [2, 3, 4], including the sequence-dependent setup time.

Also, for a given job, it takes less processing time when scheduled later than an earlier time of the whole product life cycle. In other words, the learning effect occurs (Biskup, 2008 [9]). The consideration of learning effects in SPS is also necessary because the average product life cycle, such as for producing electronics products, is more than six months (Yokoi, 2014) [64]. Under these circumstances, the learning effect of both partially and completely cross-trained workers in *serus* is inevitable over a large time span. For example, given the assumption of assigning 10 *yatais* to assemble product A in *seru* production system (SPS), both the processing time and production efficiency of SPS at the last day will differ from the first day due to worker learning effects during the product life cycle (i.e., 180 days). Moreover, the production efficiency of each *yatai* varies due to the different improvements from workers' learning effects, and the processing time required will be 60% or 80% of the original processing time. Hence, these worker element considerations, including the worker learning effects, are essential differences of *seru* scheduling problems compared to unrelated PMS problems, and considering learning effects in SPS is of great significance in practical productions. In fact, by following the first research on the learning effect from Wright (1936) [59] in aircraft industry manufacturing, many scholars considered the learning effect in production scheduling and proposed a large variety of position-based learning effect models. At the beginning, the learning effect in Wright's model is depicted as a log-linear cost model: $C_x = C_1 x^b$, where C_1 is the cost for producing the first unit product, C_x is the cumulative average cost for producing x units, and $b \leq 0$ is the learning index. In this learning effect model, C_x will decrease when x increases evidently. In Biskup (1999) [8], the learning effect in a production scheduling problem is: $p_{jr} = \bar{p}_j r^a$, where $\bar{p}_j$ is the original processing time of job j, p_{jr} is the actual processing time of job j in the rth repetition (i.e., the position r of a schedule), and $a \leq 0$ is the learning index. Similarly, Low and Lin (2011) [34] used $p_{jr} = \bar{p}_j(\sum_{j=r}^{n} p_{[j]}/\sum_{j=1}^{n} p_{[j]})^a b^{r-1}$ to describe the position-weighted learning effect in a production scheduling problem, and $p_{jr} = \bar{p}_j(1 + \sum_{k=1}^{r-1} \beta_k ln \bar{p}_{[k]})^a r^b$ in Cheng et al. (2013) [12]. Many other extensions of the position-based learning effect model have been proposed, such as Wang and Wang (2013) [56], Wu et al. (2016) [60], Cheng et al. (2019) [11]. Unfortunately, all of these learning effect models mentioned above expose a common drawback: if there are a large number of jobs, then p_{jr} is close to zero if this job is processed in a later sequence. Obviously, that is not going to happen in production practice. In this case, described by DeJong's learning curve, a new learning effect model was constructed as

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

to cope with this defect (Badiru, 1992 [6]). In Eq (1), T_1 is the processing time for the first cycle of a batch, T_s is the processing time for the sth cycle, $0 \leq M \leq 1$ is the incompressibility factor, and $0 < m < 1$ represents the reduction exponent. When $M = 0$, Eq. (1) is transformed into Wright's log-linear learning effect model to imply a completely manual operation, and $M = 1$ represents a completely machine-dominated operation $p_{jr} = p_j$, respectively. Obviously, in Eq. (1), the processing time of the sth cycle will fall according to the increasing s, but it will be convergent to a certain limit $T_1 M$. Therefore, the drawback of other learning effect models are overcome by DeJong's learning curve. In this paper, DeJong' model will be used to depict the learning effect in *seru* scheduling problem in SPS.

The remainder of this paper is organized as follows: the mixed-integer programming (MIP) model is formulated in section 2, including a detailed description for the scheduling problem in *seru* production systems by considering sequence-dependent setup time and DeJong's learning effect. Then, the *seru* scheduling MIP model is decomposed by the logic-based Benders decomposition (LBBD) method in section 3, and the solution methodology is proposed to include Benders cuts and find sub-optimal solutions in section 4. In section 5, computational experiments are conducted and the results are reported and analyzed. The conclusions and further research are made in section 6.

2. Model formulation

The mixed-integer programming (MIP) model of the *seru* scheduling problem with the sequence-dependent setup time and DeJong's learning effect will be formulated in this section.

2.1. Problem description

In the *seru* scheduling problem of this paper, a set of jobs $j \in J \equiv \{1, 2, \cdots, n_J\}$ will be scheduled on a set of parallel *serus* $i \in I \equiv \{1, 2, \cdots, n_I\}$ to minimize the makespan of SPS. Each *seru* starts from time zero onward and handles no more than one job at a time. Besides, preemption of jobs is not allowed in SPS. The jobs are processed contiguously from time zero onward, and no *seru* is idle before all jobs are started. Each job has a processing time required to be processed, and the processing time of job j in the rth repetition on *seru* i considering DeJong's learning effect is $p^i_{jr} = p^i_j(M + (1 - M)r^b)$, where p^i_{jr} represents the processing time of job j in the rth repetition on *seru* i, p^i_j is the single processing time of job j on *seru* i, M is the incompressible factor $0 \leq M \leq 1$, and b is the learning index $-1 \leq b \leq 0$. The setup time $s^i_{jj'}$ considered in this paper are both sequence and *seru* dependent, i.e., the setup time on *seru* i between job j and j' is different from that on the same *seru* i between job j' and j. Moreover, the setup time between job j and j' on *seru* i is different from that between job j and j' on *seru* i'. Generally, the setup time in SPS also comply with the triangle inequality $s^i_{jj'} \leq s^i_{jj''} + s^i_{j''j'}$.

Now, define a partial schedule on a *seru* to be a schedule which is formed by a subset of J jobs on this *seru*. Thus, a schedule for a *seru* scheduling problem consists of n_I partial schedules, i.e., one for each *seru*, where n_I is the quantities of *serus* in SPS. Further, for a given *seru* scheduling problem, there is a predetermined job ordering restriction in SPS: for each job j, a set of jobs in J must be scheduled before or after job j. Hence, we can define a feasible partial schedule on a *seru* as a partial schedule on this *seru* and this partial schedule satisfies the given job ordering restriction. Let

$$\begin{aligned} A^i_j &= \{j' \in J \mid \text{job } j' \text{ can succeed job } j \text{ in a feasible partial schedule on } seru\ i\} \\ B^i_j &= \{j' \in J \mid \text{job } j' \text{ can precede job } j \text{ in a feasible partial schedule on } seru\ i\} \end{aligned} \tag{2}$$

and

$$\begin{aligned} y^i_{0j} &= \begin{cases} 1, & \text{if job } j \text{ is processed first on } seru\ i; \\ 0, & \text{otherwise.} \end{cases} \\ y^i_{j,n_J+1} &= \begin{cases} 1, & \text{if job } j \text{ is processed last on } seru\ i; \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{3}$$

where n_J is the quantity of jobs needing to be scheduled in SPS. Therefore, the *seru* scheduling optimization problem in this paper concerns two parts: (1) determine how to assign the jobs to *serus*, (2) determine the job sequence processed on each *seru*, where the sequence-dependent setup time and DeJong's learning effect are considered to minimize the makespan.

2.2. Notation

(1) Indices

i	*seru* index, $i \in I \equiv \{1, 2, \cdots, n_I\}$
j	job index, $j \in J \equiv \{1, 2, \cdots, n_J\}$
r	position index, $r \in \{1, 2, \cdots, n_J\}$

(2) Parameters

p^i_j	normal processing time of job j in *seru* i
p^i_{jr}	actual processing time of job j at the rth position in *seru* i considering the learning effect
M	incompressible factor, $0 \leq M \leq 1$
b	learning index, $-1 \leq b \leq 0$
c_j	completion time of job j
LCT_i	latest completion time of jobs in *seru* i
$s^i_{jj'}$	setup time from job j to j' on *seru* i
st_j	setup time factor of job j, $st_j \geq 0$
$C_{\max}$	maximum completion time of the whole SPS (makespan)
V	a large positive number
J_0	set of jobs to be scheduled with an additional dummy node which is indexed by 0

(3) Decision variables

$$x_j^i = \begin{cases} 1, & \text{if job } j \text{ is assigned to } seru\ i; \\ 0, & \text{otherwise.} \end{cases}$$

$$y_{jj'}^i = \begin{cases} 1, & \text{if job } j' \text{ is processed immediately after job } j \text{ in } seru\ i; \\ 0, & \text{otherwise.} \end{cases}$$

$$z_{jr}^i = \begin{cases} 1, & \text{if job } j \text{ is assigned in position } r \text{ in } seru\ i; \\ 0, & \text{otherwise.} \end{cases}$$

2.3. Modeling

The objective of the *seru* scheduling problem considered in this paper is to minimize the makespan, which is usually used in the parallel production system because the schedules with low makespan tend to balance the workload in the whole system (Pinedo, 1995 [43]). Since the makespan is denoted as the maximal completion time of jobs in all *serus*, i.e., $C_{\max} = \max_{i \in I}\{LCT_i\}$, hence:

$$\min C_{\max} \tag{4}$$

where

$$LCT_i = \textstyle\sum_{r=1}^{n_J} \sum_{i \in I} \sum_{j \in J} \left(y_{jj'}^i s_{jj'}^i + z_{jr}^i p_{jr}^i \right), \forall i \in I \tag{5}$$

$$C_{\max} \geq LCT_i, \forall i \in I \tag{6}$$

$$p_{jr}^i = p_j^i (M + (1 - M) r^b), i \in I, j \in J, r \in \{1, 2, ..., n_J\} \tag{7}$$

$$s_{jj'}^i = st_j \sum_{\upsilon=1}^{r-1} p_{j\upsilon}^i, i \in I, j \in J, j' \in J \backslash j,\ r \in \{2, 3, \cdots, n_J\} \tag{8}$$

In Eq. (5), for each *seru* i, the latest completion time is equal to the sum of processing time and setup time for all jobs in *seru* i. For Eq. (6), the makespan $C_{\max}$ is the maximal completion time of all *serus*, hence, $C_{\max}$ is greater than or equal to LCT_i.

Further, to ensure that each job is assigned to only one *seru*, the set of constraints are employed.

$$\sum_{i=1}^{I} x_j^i = 1, \forall j \in J \tag{9}$$

where $x_0^i = 1$ places the dummy job in a *seru*, i.e., signifying the start and end of a sequence of jobs. Also, for *seru* i in SPS, each position r can be only occupied by one job j and each job j can only appear in one position r, so

$$\begin{aligned} &\textstyle\sum_{r=1}^{n_J} z_{jr}^i = 1, \forall i \in I, j \in J \\ &\textstyle\sum_{j=1}^{n_J} z_{jr}^i = 1, \forall i \in I, r \in \{1, 2, \cdots, n_J\} \end{aligned} \tag{10}$$

Moreover, since each job j has only a single predecessor and successor in *seru* i, thus

$$\sum_{j' \in A_j^i \cup \{0\}} y_{jj'}^i + \sum_{j' \in B_j^i \cup \{0\}} y_{jj'}^i = 1, i \in I, j \in J, j' \in J \backslash j \tag{11}$$

where if $\sum_{j' \in A_j^i \cup \{0\}} y_{jj'}^i = 1$, then job j is scheduled early; and if $\sum_{j' \in B_j^i \cup \{0\}} y_{jj'}^i = 1$, then job j is scheduled tardily. In addition, the condition that only one job can be scheduled first in each *seru* can be guaranteed by

$$\sum_{j \in J} y_{0j}^i \leq 1, \forall i \in I, j \in J \tag{12}$$

A job j can only have a predecessor in a *seru* i if this job also has a successor in the same *seru*,

$$\sum_{j' \in B^i_j \cup \{0\}} y^i_{jj'} = \sum_{j' \in A^i_j \cup \{n_J+1\}} y^i_{j'j}, \forall i \in I, j \in J, j' \in J \backslash j \tag{13}$$

And, if job j precedes job j' in *seru* i, then the earliest completion time of job j' must be greater than or equal to the sum of the completion time of job j, the setup time from job j to job j' and the processing time of job j', hence,

$$c_{j'} + V(1 - y^i_{jj'}) \geq c_j + s^i_{jj'} + z^i_{j'r} \times p^i_{j'r}, i \in I, j \in J, j' \in J \backslash j, r \in \{1, 2, \cdots, n_J\} \tag{14}$$

$$c_0 = 0 \tag{15}$$

From Eq. (14), we know that if job j' is processed immediately after job j in *seru* i, then $y^i_{jj'} = 1$, $1 - y^i_{jj'} = 0$, and this constraint is simplified as $c_{j'} \geq c_j + s^i_{jj'} + z^i_{j'r} \times p^i_{j'r}$. On the contrary, if job j' is not processed immediately after job j in *seru* i, then $y^i_{jj'} = 0$, and the large positive number V makes this constraint non-binding. Therefore, Eq. (14) ensures that a valid job sequence will be scheduled in each *seru*, and the processing time overlap can be avoided.

Based on the discussions above, the mixed-integer programming (MIP) model for the *seru* scheduling problem considering sequence-dependent setup time and DeJong's learning effect can be constructed as:

$$\begin{cases} \min C_{\max} \\ s.t. \begin{cases} LCT_i = \sum_{r=1}^{n_J} \sum_{i \in I} \sum_{j \in J} \left(y^i_{jj'} s^i_{jj'} + z^i_{jr} p^i_{jr} \right), \forall i \in I \\ C_{\max} \geq LCT_i, \forall i \in I \\ p^i_{jr} = p^i_j (M + (1 - M) r^b), i \in I, j \in J, r \in \{1, 2, ..., n_J\} \\ s^i_{jj'} = st_j \sum_{v=1}^{r-1} p^i_{jv}, i \in I, j \in J, j' \in J \backslash j, \ r \in \{2, 3, \cdots, n_J\} \\ \sum_{i=1}^{I} x^i_j = 1, \forall j \in J \\ \sum_{r=1}^{n_J} z^i_{jr} = 1, \forall i \in I, j \in J \\ \sum_{j=1}^{n_J} z^i_{jr} = 1, \forall i \in I, r \in \{1, 2, \cdots, n_J\} \\ \sum_{j' \in A^i_j \cup \{0\}} y^i_{jj'} + \sum_{j' \in B^i_j \cup \{0\}} y^i_{jj'} = 1, i \in I, j \in J, j' \in J \backslash j \\ \sum_{j \in J} y^i_{0j} \leq 1, \forall i \in I, j \in J \\ \sum_{j' \in B^i_j \cup \{0\}} y^i_{jj'} = \sum_{j' \in A^i_j \cup \{n_J+1\}} y^i_{j'j}, \forall i \in I, j \in J, j' \in J \backslash j \\ c_{j'} + V(1 - y^i_{jj'}) \geq c_j + s^i_{jj'} + z^i_{j'r} \times p^i_{j'r}, i \in I, j \in J, j' \in J \backslash j, r \in \{1, 2, \cdots, n_J\} \\ c_0 = 0 \\ x^i_j \in \{0, 1\}, y^i_{jj'} \in \{0, 1\}, z^i_{jr} \in \{0, 1\}, i \in I, j \in J, j' \in J \backslash j, r \in \{1, 2, \cdots, n_J\} \end{cases} \end{cases} \tag{16}$$

The proposed MIP model (16) has four decision variables, and they represent the decision making associated with a job: x^i_j, $y^i_{jj'}$, z^i_{jr} and c_j, and define the *seru* that a job is processed on, the sequence of processing, the position of job in the *seru*, and the completion time, respectively.

3. Logic-based Benders decomposition (LBBD) method

As a generalization of Benders decomposition (Benders, 1962 [7]), LBBD was introduced by Hooker (2000) [21] and refined by Hooker and Ottosson (2003) [22] for solving highly combinatorial problems, such as planing and scheduling (Hooker, 2007 [23]). In fact, LBBD has been applied successfully to a wide range of combinatorial optimization problems, including bin-packing (Pisinger and Sigurd, 2007 [41]), location-allocation (Fazel-Zarandi and Beck, 2012 [18]), inventory-location (Wheatley et al., 2015 [57]), scheduling problem (Hooker, 2007 [23]; Sun et al., 2019 [52]), home health care delivery (Heching et al., 2019 [19]), etc. In this section, the LBBD method is also used to decompose the *seru* scheduling MIP model into a master problem and a set of independent single *seru* scheduling subproblems.

Generally, to decompose a problem by the LBBD, the first step is to partition the decision variables into two vectors x and y, and then the problem can be viewed as

$$\min f(x, y) \tag{17}$$

$$s.t. \quad (x, y) \in C \tag{18}$$

$$x \in D_x, y \in D_y \tag{19}$$

where f is a real-valued objective function, C is the feasible set defined by the collection of constraints containing variables x, y, D_x and D_y are the domains of x and y, respectively. Fix x to be a given value $x^h \in D_x$, and the following subproblem is obtained:

$$\min f(x^h, y) \tag{20}$$

$$s.t. \quad (x^h, y) \in \overline{C} \tag{21}$$

$$y \in D_y \tag{22}$$

The feasible set C is relaxed as $\overline{C}$, which is the constraint that results from fixing $x = x^h$ in D_x. The inference dual of the subproblem is the problem of inferring the tightest possible lower bound on $f(x^h, y)$ from $\overline{C}$. Different from the classical Benders decomposition, there are no structural restrictions in LBBD, such as linearity, on the different components of the decomposition. Formally, in iteration h, the master problem can be redefined as (Hooker, 2007 [23]):

$$\min z \tag{23}$$

$$s.t. \; x \in \overline{C} \tag{24}$$

$$z \geq B_{x^h}(x), \quad h = 1, 2, \cdots H - 1 \tag{25}$$

$$z \in R, \mathbf{x} \in D_x \tag{26}$$

where z is a real-valued decision variable, $B_{x^h}(x)$ is a Benders cut on the objective function f in iteration h, $x^1, x^2, \cdots x^{H-1}$ are solutions of the previous $H - 1$ master problems, and constraints (25) are derived from solving the subproblem.

The process of solving a LBBD model is as follows: in iteration h, the solution x^h is produced by solving the master problem to optimality. Then, x^h is used to formulate the subproblems, and each subproblem is solved by producing bounding functions, i.e., Benders cuts. Let y^h be the subproblem solution, and if the h-th master problem solution satisfies all the Benders cuts gained from iteration 1 to h, then the process will converge to a globally optimal solution (x^h, y^h). Otherwise, solve the master problem again and $h := h + 1$. Repeat the process iteratively until the master problem and the subproblems are convergent.

3.1. Master problem

In the master problem, all jobs are assigned to *serus* in SPS by the decision variable x^i_j. Different from the *seru* scheduling MIP (16), the master problem is a relaxation of (16), which means that when assigning all jobs to *serus*, multiple disjoint sequences are allowed instead of a single determined sequence of jobs in each *seru*. Essentially, the decision variable c_j, and constraints (14) and (15) are removed from the proposed *seru* scheduling MIP model in the master problem. Moreover, the decision variables $y^i_{jj'}$ and z^i_{jr} are relaxed to be any real valued number between 0 and 1. Thus, the master problem is:

$$\min C_{\max} \tag{27}$$

$$s.t. \text{ constraints } (5) - (13)$$

$$Benders \;\; cut \tag{28}$$

$$x^i_j \in \{0, 1\} \tag{29}$$

$$0 \leq y^i_{jj'} \leq 1 \tag{30}$$

$$0 \leq z^i_{jr} \leq 1 \tag{31}$$

Constraint (28) is the Benders cut, which will be defined in subsection 4.1. When solving the problem, if the makespan of any subproblem is larger than that of its master problem, the Benders cut will be added.

3.2. Sequencing subproblems

According to the assignment from the master problem, the subproblems will find the optimal schedules in each *seru*. Let x_j^{ih*}, $y_{jj'}^{ih*}$, z_{jr}^{ih*}, and $C_{\max}^{h*}$ be the solution obtained from the master problem in iteration h, where x_j^{ih*} provides an assignment of each job to one of the n_I *serus*. Hence, n_I separate subproblems will be created, and each subproblem represents one *seru* and contains only these assigned jobs, for example, for *seru* i, only jobs j where $x_j^{ih*} = 1$ is contained. Further, given a fixed assignment x_j^{ih*}, the sequence of jobs in a *seru* does not affect any other *seru* in SPS, therefore, n_I subproblems could be solved independently.

Since a subproblem of the *seru* scheduling problem needs the sequence of all the assigned jobs to minimize the makespan, it is similar to find a Hamiltonian cycle, i.e., a tour that passes through all the nodes, with the minimal sum of edge weights. Let the complete graph $G = (V, E, W)$ denote the subproblem, where V is the set of job nodes, $E = (j, j')$ is the set of edges, and W is the edge weight which is equal to $p_{jr}^i + s_{jj'}^i$. If j is the first job to be processed in *seru* i, then the edge weight from the dummy node 0 to job j is equal to the setup time of job j, i.e., s_{0j}^i; if j is the last job to be processed in *seru* i, then the edge weight from job j to the dummy node 0 is equal to the processing time of job j, i.e, p_{jr}^i. Due to the different W between any two jobs, the subproblem can be finally modelled as an asymmetric travelling salesman problem (ATSP) similarly. The ATSP representation example for a subproblem of the *seru* scheduling problem containing four jobs is shown in Fig. 2.

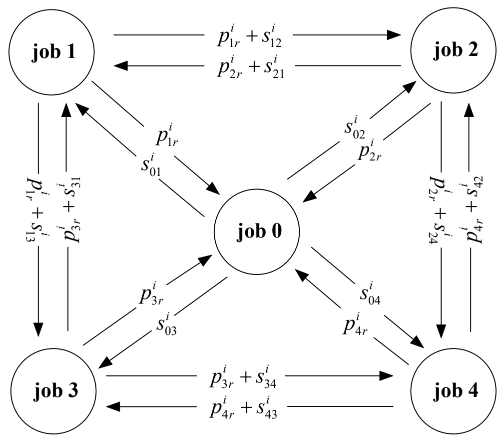

Figure 2: ATSP representation for a subproblem

From Fig. 2, it can be seen clearly that if the order of jobs in *seru* i to be processed is the sequence $1 \rightarrow 3 \rightarrow 4 \rightarrow 2$, then the distance travelled would be

$$s_{01}^i + p_{11}^i + s_{13}^i + p_{32}^i + s_{34}^i + p_{43}^i + s_{42}^i$$

In other words, the tour distance is equal to the makespan of processing four jobs in this order. It is also observed that a solution to the subproblem corresponds to a minimal length sequence of jobs.

4. Solution methodology

In this section, the solution methodology will be provided for the LBBD model, including developing the Benders cut and finding sub-optimal solutions.

4.1. Benders cut

When solving n_I subproblems, if the makespan of a *seru* i is less than or equal to the makespan $C_{\max}$ of the master problem, then the solution is feasible regarding the master problem and no cut needs to be added. Otherwise, the Benders cut is created and the master problem is updated.

Given a set of jobs that are assigned to the same *seru* i from iteration h of the master problem, and it is denoted as $J^{ih} = \{j : x_j^{ih*} = 1\}$. In order to define the cut, the maximal setup time (i.e., $\max ST_j^h$) for job j is introduced first, where job j directly succeeds another job that is assigned in the master problem to the same *seru* i in iteration h , i,e,

$$\max ST_j^h = \max_{j' \in J^{hi}; j' \neq j} \left(s_{j'j}^i \right) \tag{32}$$

Let TAT_j^{ih} be the total assembly time of job j in *seru* i, and

$$TAT_j^{ih} = \max ST_j^h + p_{jr}^i \tag{33}$$

Then, the cut used in this paper is

$$C_{\max} \geq C_{\max}^{ih*} - \sum_{j \in J^{ih}} \left(1 - x_j^i\right) TAT_j^{ih} \tag{34}$$

where $C_{\max}^{ih*}$ is the makespan found in iteration h for *seru* i.

Thus, depending on the jobs that are assigned, this cut places a lower bound (LB) on the makespan in next iterations. It means that if the same assignment is given to the subproblem, then $x_j^i = 1$, and $\sum_{j \in J^{ih}} \left(1 - x_j^i\right) TAT_j^{ih} = 0$ in Eq. (34). In this case, the makespan of subproblem $C_{\max}^{ih*}$ is a new LB on the makespan of master problem $C_{\max}$. Otherwise, a different assignment is made to the subproblems, and at least one of the $x_j^i = 0$. Thus, the makespan in the subsequent iteration is bounded by the makespan found in the subproblem minus the corresponding TAT_j^{ih} value(s), i.e., $C_{\max}^{ih*} - \sum_{j \in J^{ih}} \left(1 - x_j^i\right) TAT_j^{ih}$. The cut employed in this paper presents the two following properties shown in Theorem 4.1 and 4.2.

Theorem 4.1. *The cut proposed in Eq. (34) must remove the current solution from the master problem space.*

Proof. Suppose that in the subsequent iteration, there is an exactly same set of jobs being assigned to *seru* i. Then, the master problem must increase the value of the makespan. Otherwise, for *seru* i, a change must be made to the job assignment. Therefore, the current solution in iteration h is removed from the master problem space in either case. □

Theorem 4.2. *The cut proposed in Eq. (34) does not remove any globally optimal solution of the seru scheduling problem.*

Proof. In order to prove Theorem 4.2, an assumption that there is a globally optimal schedule violating the cut in Eq. (34) is made first, and then the contradiction holds.

Let J^i be a set of jobs assigned to *seru* i in the current iteration, and C_{J^i} be the optimal makespan of *seru* i. Assume that there is a globally optimal schedule which violates the cut generated from Eq. (34) for *seru* i in iteration h. Now, let J^{i*} be the set of jobs assigned to *seru* i corresponding to this globally optimal schedule, and $C_{J^{i*}}$ be the makespan. Since there are $\bar{J}^i := J^i - J^{i*}$ jobs not being assigned from J^i to *seru* i, so for $j \in \bar{J}^i$, $x_j^i = 0$. Hence we have the following property due to the violation:

$$C_{J^{i*}} < C_{J^i} - \sum_{j \in \bar{J}^i} TAT_j^{ih} \tag{35}$$

Given the schedule corresponding to $C_{J^{i*}}$, define a reduced schedule that contains $\hat{J}^i := J^{i*} \cap J^i$ jobs, which are assigned in the same order as the globally optimal schedule. Let $C_{\hat{J}^i}$ be the makespan of the reduced schedule in *seru* i, because the setup time satisfies $s_{jj'}^i \leq s_{jj''}^i + s_{j''j'}^i$, so $C_{\hat{J}^i} \leq C_{J^{i*}}$. Hence, the reduced schedule also violates the cut Eq. (34), i.e.,

$$C_{\hat{J}^i} < C_{J^i} - \sum_{j \in \bar{J}^i} TAT_j^{ih} \tag{36}$$

At the end of the reduced schedule, each job is placed one by one in $\bar{J}^i$, and the reduced schedule will be extended to a schedule containing all J^i jobs now. Because the $\max ST_j^h$ is the maximal setup time of job j, so the makespan C'_{J^i}

satisfies:

$$\begin{aligned} C'_{J^i} &< C_{\hat{J}^i} + \sum_{j\in\bar{J}^i}\left(p^i_{jr} + \max ST^h_j\right) \\ &= C_{\hat{J}^i} + \sum_{j\in\bar{J}^i} TAT^{ih}_j \end{aligned} \tag{37}$$

So we have

$$C_{\hat{J}^i} \geq C'_{J^i} - \sum_{j\in\bar{J}^i} TAT^{ih}_j \tag{38}$$

Further, C_{J^i} is the optimal, i.e., minimal, makespan of *seru* i, hence, $C_{J^i} \leq C'_{J^i}$. Then,

$$C_{\hat{J}^i} \geq C_{J^i} - \sum_{j\in\bar{J}^i} TAT^{ih}_j \tag{39}$$

Therefore, the contradiction occurs between Eq. (36) and Eq. (39), and we can conclude that the cut from Eq. (34) will not remove any globally optimal solution. □

Based on Theorems 4.1 and 4.2, we know that the cut from Eq. (34) used in this paper is a valid cut.

4.2. Sub-optimal solutions

In this subsection, a method is provided for finding sub-optimal solutions of LBBD to obtain the globally feasible solutions. This method is about maintaining the best solution found up to now. When solving the master problem, a feasible solution will assign job j to *seru* i and n_I subproblems are constructed. Ignore the $C_{\max}$ value in the master problem, and the maximal makespan over all n_I subproblems is globally feasible. Thus, the globally feasible solution for each feasible master solution can be found, and the best schedule found so far can be tracked. Following this way, a feasible schedule for the global *seru* scheduling problem exists as long as the first feasible master solution is obtained and n_I subproblems are solved.

Also, this method offers another stopping criterion of LBBD except for the terminate condition mentioned in section 3, i.e., when an optimal master solution is found and its makespan is equal to the best global solution found so far, LBBD can also be stopped.

5. Computational experiments

To test the performance of the LBBD method compared to solving the MIP model in Eq. (16) directly for the *seru* scheduling problem, computational experiments are made and the performances are analyzed. Both the LBBD method and MIP model are coded in MATLAB R2019a by combining with Concorde TSP, and they are tested on 10th GEN Intel Core i7-10510U CPU (in 32-bit mode), 16 GB main memory, 1TB SSD, running on Windows 10.

5.1. Experiment settings

The parameters of test problems for LBBD are generated in Table 1.

Table 1: Parameters used in the experiments

Parameters	Value
n_I	{2,5,8}
n_J	$[10, 50]$, by the increment of 10 jobs
p^i_{jr}	uniform distribution $U[5, 100]$
learning index	$b = -0.7$, $M = 0.5$
setup time factor	$st_j = 0.5$
V	10^6

In addition, to obtain the sequence-dependent setup time by satisfying the triangular inequality, the Manhattan distance is used in this paper. Supposed that for each *seru* i in SPS, each job j in *seru*-job pair (i, j) is given two different sets of coordinates (x^i_{j1}, y^i_{j1}), (x^i_{j2}, y^i_{j2}) on the Cartesian plane according to the triangular inequality assumption, where the coordinates along the x and y axis are generated by the uniform distribution $U[0, 50]$. The asymmetric setup time of job j to j' are the Manhattan distance from the coordinates of j to j'. Let l and u be the lower and upper bound of the setup time, respectively, then the Manhattan distance is used to provide the setup time by linearly scaling a distance of 0 to l of the setup time distribution and 100 to u. In this paper, set $l = 25$ and $u = 50$, respectively.

5.2. Results

The scatter plots of the pairwise comparisons between the LBBD and MIP model are shown in Fig. 3, and the time limit is set to be 4 hours (14400 seconds) for each instance. In Fig. 3, each point corresponds to an instance, and the time is recorded at which an optimal solution are found. The format of the graphs means that points below the $y = x$ line demonstrate a superior performance. The points at the right side ($x = 14400$) in Fig. 3 represent the instances that the MIP model can not find optimal solutions within 4 hours. Obviously, the LBBD method presents a greater improvement than the MIP model. Except for two instances which are solved in less than 10 seconds, the LBBD method outperforms MIP model.

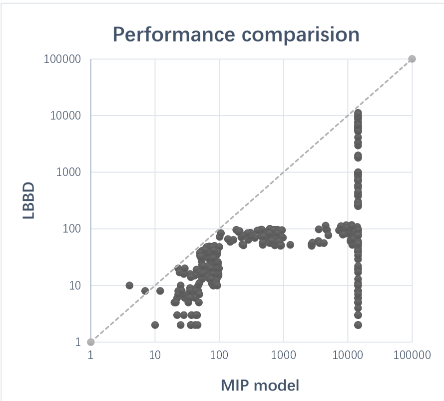

Figure 3: Pairwise runtime comparison between LBBD and MIP model

Table 2 shows more detailed results. It can be seen that the CPU runtime of both the LBBD and MIP model increases dramatically along with the growing quantity of jobs n_J. The LBBD method provides a vast improvement over the MIP model, which can only solve instances with up to $n_J = 20$. Even with only 20 jobs, and 5 or more *serus*, the MIP model can not gain the optimal solutions for all instances. On the contrary, the LBBD method can solve all instances of the *seru* scheduling problem within an acceptable time. Hence, it can be concluded that the LBBD has a faster calculation speed than the MIP for exact solutions.

5.3. Test on large instances

In order to show the superiority of LBBD, large instances with $n_I = \{10, 15, 20\}$ and $n_J = \{200, 600, 100\}$ are tested, and other parameters are set as the same in Table 1. Meanwhile, a metaheuristic algorithm, i.e. adaptive genetic algorithm (A-GA), is designed for a comparative analysis. Because the crossover probability p_c and mutation probability p_m will affect the convergence directly, self-learning p_c and p_m are adopted to calibrate A-GA parameters so as to prevent GA from falling into local optimal solutions (Ho et al., 2007 [20]). The adaptive adjustments of p_c and p_m are as follows:

$$p_c = \begin{cases} p_{c_{\max}} - \frac{(p_{c_{\max}} - p_{c_{\min}})(Fit_a - Fit_s)}{Fit_a - Fit_{\min}} & , \quad Fit_a > Fit_s \\ p_{c_{\max}} & , \quad Fit_a \leq Fit_s \end{cases} \tag{40}$$

Table 2: CPU runtime and unsolved instances comparisons between LBBD and MIP model

n_J	n_I	LBBD average runtime (seconds)	LBBD number of unsolved instances	MIP average runtime (seconds)	MIP number of unsolved instances
10	2	0.57	0	1.22	0
	5	2.36	0	14.74	0
	8	5.13	0	30.85	0
20	2	3.91	0	1010.42	0
	5	88.67	0	7984.37	6
	8	302.50	0	10246.81	13
30	2	7.93	0	6927.35	8
	5	553.22	0	14400.00	22
	8	1562.19	0	14400.00	22
40	2	14.96	0	14400.00	22
	5	1026.10	0	14400.00	22
	8	3685.46	0	14400.00	22
50	2	59.37	0	14400.00	22
	5	2089.61	0	14400.00	22
	8	5211.98	0	14400.00	22

$$p_m = \begin{cases} p_{m_{\max}} - \frac{(p_{m_{\max}} - p_{m_{\min}})(Fit_a - Fit_{can})}{Fit_a - Fit_{\min}} & , \quad Fit_a > Fit_s \\ p_{m_{\max}} & , \quad Fit_a \le Fit_s \end{cases} \tag{41}$$

where $p_{c_{\max}}$, $p_{m_{\max}}$ and $p_{c_{\min}}$, $p_{m_{\min}}$ are the upper and lower bounds of p_c and p_m, and equal to 0.9, 0.5, 0.6, 0.1, respectively (Chen et al., 2020 [10]). Fit_a is the average fitness of the population, Fit_s is the smaller fitness value of any two crossover chromosomes, $Fit_{\min}$ is the best fitness of the current population, and Fit_{can} in Eq. (41) is the fitness value of the candidate mutation individual. Further, the deviation (RD) indicator of makespan obtained by the LBBD and A-GA is employed, and

$$RD = \left| \frac{C_{\max}^{A-GA} - C_{\max}^{LBBD}}{C_{\max}^{LBBD}} \right| \times 100\%$$

After 600 runs of the A-GA with $pop\_size = 300, GEN = 500$, the average runtime of both the LBBD and A-GA, as well as RD are reported in Table 3.

Table 3: Results of large instances for LBBD and A-GA

n_J	n_I	Average runtime (seconds) LBBD	Average runtime (seconds) A-GA	RD (%)
200	10	9124	4678	8.65
	15	11026	5071	9.33
	20	13579	5935	18.61
600	10	9684	7939	3.39
	15	12697	8447	16.02
	20	13970	9241	10.68
1000	10	12166	9626	20.45
	15	14002	10177	4.39
	20	14400	11595	–

The results from Table 3 show that LBBD is still effective and able to handle larger instances for *seru* scheduling problems (except for the largest case with $n_J = 1000$ and $n_I = 20$, and it is also solvable but the runtime time exceeds 14400 seconds). It is interesting to observe that the runtime required by the LBBD grows significantly as the number of n_I increases, but it is more stable in A-GA. Fig. 4 presents the scatter diagram of RD. It can be seen that the solution returned by A-GA is not always satisfying compared to the optimal solution obtained by the LBBD method, and the unstable *RD* indicates that it does not have any specific rule to follow. Overall speaking, the average runtime of the A-GA is less than that of the LBBD method, but the quality of solution returned by the A-GA is worse. Thus, in real production practice, the LBBD method is indeed a very good choice for managers because apart from the scalability for large instances, it can obtain exact solutions of *seru* scheduling problems with an acceptable time compared to other exact methods (such as MIP model), and can get higher-quality solutions compared to metaheuristic algorithms (such as A-GA).

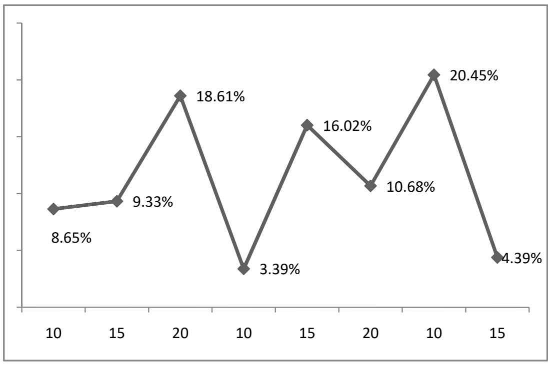

Figure 4: Scatter diagram of the relative deviation (*RD*)

5.4. *Sensitivity analysis*

In order to scrutinise the management insights to production practice, sensitivity analysis for both the sequence-dependent setup time and Dejong's learning effect are conducted in this subsection.

5.4.1. *Change of sequence-dependent setup time*

To verify the effect of the sequence-dependent setup time on *seru* scheduling problems, a sensitive analysis of setup time factor st_j is made. Without loss of generality, $st_j = \{0.2, 0.4, 0.6, 0.8\}$ are selected to compare with original $st_j = 0.5$. Four combinations of *serus* and jobs $n_J = 10, n_I = 2, 5, 8$; $n_J = 50, n_I = 2, 5, 8$; $n_J = 200, n_I = 10, 15, 20$; $n_J = 1000, n_I = 10, 15, 20$ are tested. The results are shown in Table 4, and the mean of absolute deviation is equal to

$$\frac{\left|C_{\max}^{st_j=0.5} - C_{\max}^{st_j=0.2}\right| + \left|C_{\max}^{st_j=0.5} - C_{\max}^{st_j=0.4}\right| + \left|C_{\max}^{st_j=0.5} - C_{\max}^{st_j=0.6}\right| + \left|C_{\max}^{st_j=0.5} - C_{\max}^{st_j=0.8}\right|}{4 \times C_{\max}^{st_j=0.5}}$$

The results in Table 4 demonstrate that the sequence-dependent setup time has a significant impact on the system robustness performance of SPS. Along with the change of setup time factor st_j from 0.2 to 0.8, the mean of absolute deviation of the makespan $C_{\max}$ is changed only from 0.0066 to 0.0497. Meanwhile, when the job number becomes larger, the deviation is more stable. Overall speaking, although the proportion of the sequence-dependent setup time to processing time will lead to the longer flow time, the robustness of SPS is guaranteed. Hence, it can be concluded that the sequence-dependent setup time should be given as an explicit consideration in *seru* scheduling problems.

5.4.2. *Change of Dejong's learning effect*

In order to test the Dejong's learning effect on *seru* scheduling problems explicitly, a sensitive analysis of the incompressibility factor M ($0 \leq M \leq 1$) is made. When $M = 0$, the learning effect is the strongest, and Dejong's learning effect is transformed into Wright's log-linear learning effect to imply a completely manual operation. On the contrary, when $M = 1$ there is no learning effect, and it represents a completely machine-dominated operation, respectively. Thus, we also select $n_J = 10, n_I = 2, 5, 8$; $n_J = 50, n_I = 2, 5, 8$; $n_J = 200, n_I = 10, 15, 20$; $n_J = 1000, n_I = 10, 15, 20$ and $M \in [0, 1]$ (with an increment of 0.1) to study the sensitivity analysis of M to the makespan

Table 4: Results summary of four combinations

n_J	n_I	Makespan $C_{\max}$					Mean of absolute deviation
		$st_j = 0.5$	$st_j = 0.2$	$st_j = 0.4$	$st_j = 0.6$	$st_j = 0.8$	
10	2	41.33	40.13	40.69	41.75	42.02	0.0178
	5	14.25	13.99	14.08	14.62	14.98	0.0268
	8	6.07	6.02	6.05	6.10	6.13	0.0066
50	2	374.31	368.24	372.18	380.82	383.26	0.0158
	5	211.97	206.38	208.70	216.77	220.84	0.0266
	8	138.02	133.94	135.79	142.28	149.61	0.0401
200	10	319.93	308.62	315.27	325.67	331.78	0.0262
	15	192.96	187.28	190.35	199.92	202.36	0.0319
	20	115.01	110.32	112.49	120.98	124.71	0.0497
1000	10	11743.21	11364.24	11597.41	11987.22	12057.39	0.0231
	15	7096.79	6885.19	6927.57	7241.36	7448.21	0.0309
	20	4947.33	4762.87	4821.58	5036.81	5247.33	0.0354

$C_{\max}$ for *seru* scheduling problems. The detailed results for each case with different values of M are shown in Fig. 5 to 8.

In general, the learning effect has a significant influence on the makespan $C_{\max}$ for *seru* scheduling problems. $C_{\max}$ usually obtains the maximum value, i.e., the worst one, when $M = 1$ (no learning effect at all) in each case. Meanwhile, $C_{\max}$ and the processing time do not decrease continuously, but stabilize to a fixed value eventually even though there are a large number of jobs such as $n_J = 1000$ in Fig. 8. Hence, the advantages of DeJong's learning effect compared to other learning effects are also verified. Moreover, compared with the slope of learning curves from $n_J = 10$ to $n_J = 1000$, it is obvious that with more n_J jobs, the learning effect becomes more evident. In addition, we also find that with more evenly number of jobs assigned to each *seru*, the makespan $C_{\max}$ is smaller. For example, $C_{\max}(n_J = 10, n_I = 2) - C_{\max}(n_J = 10, n_I = 5) > C_{\max}(n_J = 10, n_I = 5) - C_{\max}(n_J = 10, n_I = 8)$ and $C_{\max}(n_J = 1000, n_I = 10) - C_{\max}(n_J = 1000, n_I = 15) > C_{\max}(n_J = 1000, n_I = 15) - C_{\max}(n_J = 1000, n_I = 20)$. This phenomenon complies with the 'group balance principle' that suppose there are n_J jobs to be assigned to n_I *serus*, to achieve group balance of SPS, the number of jobs in each group is either $\lceil \frac{n_J}{n_I} \rceil$ or $\lceil \frac{n_J}{n_I} \rceil - 1$. Therefore, production managers of SPS should make full considerations of ratio $\frac{n_J}{n_I}$ to achieve a SPS balance and gain high production efficiency.

6. Conclusion

This paper focuses on the scheduling problem in *seru* production system considering the sequence-dependent setup time and DeJong's learning effect to minimize the makespan. A mixed-integer programming (MIP) model is developed, then logic-based Benders decomposition (LBBD) method is applied to reformulated the proposed model. Computational experiments are made, and the results indicate that the LBBD method has a good scalability and performance to generate optimal solutions for *seru* scheduling problems. Compared with MIP model in small cases, the LBBD method has a faster calculation speed for exact solutions. In addition, compared with A-GA in large cases, LBBD method can get higher-quality solutions.

Future research should concentrate on improving the computational speed of the LBBD method, especially on analyzing properties of the subproblems which can be transformed into an asymmetric travelling salesman problem. Also, applying the proposed model and LBBD method to divisional *seru* and rotating *seru* should be concerned. Both areas are important and should be studied in the future.

Acknowledgement

This research was sponsored by National Natural Science Foundation of China (Grant No. 71401075, 71801129, 71871175), the Natural Science Foundation of Jiangsu Province (Grant No. BK20180452), and the Fundamental

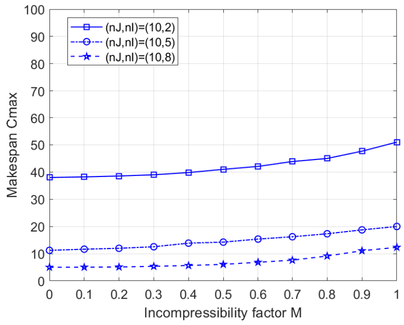

Figure 5: $C_{\max}$ with different values of M ($n_J = 10, n_I = 2, 5, 8$)

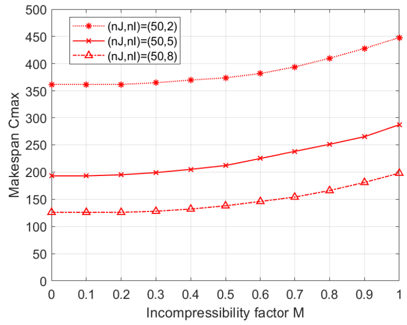

Figure 6: $C_{\max}$ with different values of M ($n_J = 50, n_I = 2, 5, 8$)

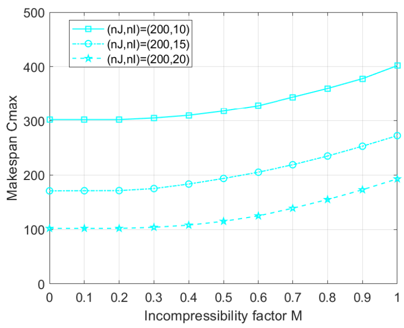

Figure 7: $C_{\max}$ with different values of M ($n_J = 200, n_I = 10, 15, 20$)

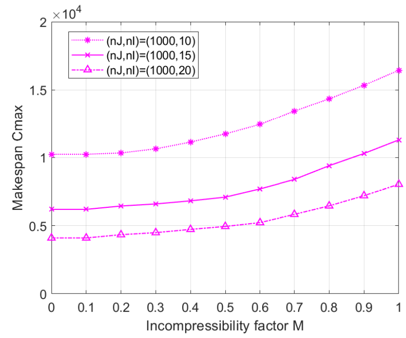

Figure 8: $C_{\max}$ with different values of M ($n_J = 1000, n_I = 10, 15, 20$)

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

[1] Alfieri, A. (2009). Workload simulation and optimisation in multi-criteria hybrid flow-shop scheduling: A case study. *International Journal of Production Research*, 47(18), 5129-5145. https://doi.org/10.1080/00207540802010823.

[2] Allahverdi, A., Gupta, J., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega-International Journal of Management Science*, 27(2), 219-239. https://doi.org/10.1016/S0305-0483(98)00042-5.

[3] Allahverdi, A., Ng, C., Cheng, T., & Kovalyov, M. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3), 985-1032. https://doi.org/10.1016/j.ejor.2006.06.060.

[4] Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2), 345-378. https://doi.org/10.1016/j.ejor.2015.04.004.

[5] Akino, S. (1997). Internationalization of Japanese company and change of production system. *Rikkyo Economic Review*, 51(1), 29-55. (in Japanese)

[6] Badiru, A. (1992). Computational survey of univariate and multivariate learning curve models. *IEEE Transactions on Engineering Management*, 39, 176-188. https://doi.org/10.1109/17.141275.

[7] Benders, J. (1962). Partitioning procedures for solving mixedvariables programming problems. *Numerische Mathematik*, 4(1), 238-252. https://doi.org/10.1007/BF01386316.

[8] Biskup, D. (1999). Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 115(1), 173-178. https://doi.org/10.1016/S0377-2217(98)00246-X.

[9] Biskup, D. (2008). A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research*, 188, 315-329. https://doi.org/10.1016/j.ejor.2007.05.040.

[10] Chen, R., Yang, B., Li, S., & Wang, S. (2020). A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. *Computers & Industrial Engineering*, 149, 106778. doi: 10.1016/j.cie.2020.106778.

[11] Cheng, B., Zhua, H., & Li, K. (2019). Optimization of batch operations with a truncated batch-position-based learning effect. *OMEGA-International Journal of Management Science*, 85, 134-143. https://doi.org/10.1016/j.omega.2018.06.006.

[12] 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. https://doi.org/10.1016/j.ins.2012.09.001.

[13] Cheng, T., & Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3), 271-292. https://doi.org/10.1016/0377-2217(90)90215-W.

[14] Edis, E., Oguz, C., & Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, 230(3), 449-463. https://doi.org/10.1016/j.ejor.2013.02.042.

[15] Ewees, A., Al-qaness, M., & Elaziz, M. (2021). Enhanced salp swarm algorithm based on firefly algorithm for unrelated parallel machine scheduling with setup times. *Applied Mathematical Modelling*, 94(24), 449-463. https://doi.org/10.1016/j.apm.2021.01.017.

[16] Fanjul-Peyro, L., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1), 55-69. https://doi.org/10.1016/j.ejor.2010.03.030.

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

[18] Fazel-Zarandi, M., & Beck, J. (2012). Using logic-based Benders decomposition to solve the capacity and distance-constrained plant location problem. *INFORMS Journal on Computing*, 24(3), 387-398. https://doi.org/10.1287/ijoc.1110.0458.

[19] Heching, A., Hooker, J., & Kimura, R. (2019). A lgic-based benders approach to home healthcare delivery. *Transportation Science*, 53(2), 510-522. https://doi.org/10.1287/trsc.2018.0830.

[20] Ho, N., Tay, J., Lai, E. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2), 316-333. https://doi.org/10.1016/j.ejor.2006.04.007.

[21] Hooker, J. (2000). Logic-based methods for optimization: combining optimization and constraint satisfaction. New York, Wiley.

[22] Hooker, J., & Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, 96(1), 33-60. https://doi.org/10.1007/s10107-003-0375-9.

[23] Hooker, J. (2007). Planning and scheduling by logic-based benders decomposition. *Operations Research*, 55(3), 588-602. https://doi.org/10.1287/opre.1060.0371.

[24] Hopp, W., & Spearman, M. (2020). The lenses of lean: Visioning the science and practice of efficiency *Journal of Operations Management*, inpress. https://doi.org/10.1002/joom.1115.

[25] 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. https://doi.org/10.1080/00207540802275889.

[26] Lee, Y., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29(1), 45-52. https://doi.org/10.1080/07408179708966311.

[27] Lewis, M. (2019). Operations Management: A Research Overview. Routledge. London.

[28] Li, Y., Li, X., Gao, L., & Meng, L. (2020). An improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times. *Computers & Industrial Engineering*, 147, Article 106638. https://doi.org/10.1016/j.cie.2020.106638.

[29] Lian, J. (2012). Study on the decision of *seru* formation and *seru* loading under *seru* seisan. Master thesis, Xi'an University of Technology. (in Chinese)

[30] Lin, Y., Fowler, J., & Pfund, M. (2013). Multiple-objective heuristics for scheduling unrelated parallel machines *European Journal of Operational Research*, 227(2), 239-253. https://doi.org/10.1016/j.ejor.2012.10.008.

[31] 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. https://doi.org/10.1007/s00170-013-5027-5.

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

[33] Liu, M., & Lei, D. (2020). An artificial bee colony with division for distributed unrelated parallel machine scheduling with preventive maintenance. *Computers & Industrial Engineering*, 141, 106320. https://doi.org/10.1016/j.cie.2020.106320.

[34] Low, C., & Lin, W. (2011). Minimizing the total completion time in a single machine scheduling problem with a learning effect. *Applied Mathematical Modelling*, 35, 1946-1951. https://doi.org/10.1016/j.apm.2010.11.006.

[35] Luo, L., Zhang, Z., & Yin, Y. (2016, December). *Seru* loading with worker-operation assignment in single period. *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pp. 1055-1058.

[36] 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. https://doi.org/10.1504/EJIE.2017.083255.

[37] Luo, L., Zhang, Z., & Yin, Y. (2019). 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*, inpress. https://doi.org/10.3934/jimo.2019134.

[38] Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research*, 18(2), 193-242.

[39] Nishi, T., & Hiranaka, Y. (2013). Lagrangian relaxation and cut generation for sequence-dependent setup time flowshop scheduling problems to minimise the total weighted tardiness. *International Journal of Production Research*, 51(16), 4778-4796. https://doi.org/10.1080/00207543.2013.774469.

[40] Pan, Q., Gao, L., Li, X., & Gao, K. (2017). Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics and Computation*, 303, 89-112. https://doi.org/10.1016/j.amc.2017.01.004.

[41] Pisinger, D., & Sigurd, M. (2007). Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS Journal on Computing*, 19(1), 36-51. https://doi.org/10.1287/ijoc.1060.0181.

[42] Pearn, W., Chung, S., Yang, M., & Shiao, K. (2008). Solution strategies for multi-stage wafer probing scheduling problem with reentry. *Journal of the Operational Research Society*, 59(5), 637-651. https://doi.org/10.1057/palgrave.jors.2602354.

[43] Pinedo, M. (1995). Scheduling: Theory, Algorithms, and Systems. Prentice-Hall, Englewood Cliffs. NJ.

[44] Ren, H., & Wang, D. (2019). Analysis of the effect of the line-*seru* conversion on the waiting time with batch arrival. *Mathematical Problems in Engineering*, Article 4036794. https://doi.org/10.1155/2019/4036794.

[45] 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. https://doi.org/10.1111/poms.12590.

[46] Ruiz, R., & Maroto, C. (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research*, 169(3), 781-800. https://doi.org/10.1016/j.ejor.2004.06.038.
[47] Salvendy, G. (2001). Handbook of Industrial Engineering: Technology and Operations Management, Third Edition. John Wiley & Sons.
[48] 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. https://doi.org/10.1504/IJMR.2016.082821.
[49] Shinobu, C. (2003). Post-lean production systems: Toward an adaptable enterprise in the age of uncertainty. Tokyo, Japan: Bunshin-Do. (in Japanese)
[50] 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): 106-119. https://doi.org/10.4018/jsds.2012010104.
[51] 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. https://doi.org/10.1109/ACCESS.2018.2889372.
[52] Sun, D., Tang, L., & Baldacci, R. (2019). A Benders decomposition-based framework for solving quay crane scheduling problems. *European Journal of Operational Research*, 273(2), 504-515. https://doi.org/10.1016/j.ejor.2018.08.009.
[53] 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. https://doi.org/10.1016/10.1016/j.jom.2017.02.001.
[54] Trovinger, S., & Bohn, R. (2005). Setup time reduction for electronics assembly: Combining simple (SMED) and IT-based methods. *Production and Operations Management*, 14, 205-217. https://doi.org/10.1111/j.1937-5956.2005.tb00019.x.
[55] Vallad, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3), 612-622. https://doi.org/10.1016/j.ejor.2011.01.011.
[56] Wang, J., & Wang, J. (2013). Scheduling jobs with a general learning effect model. *Applied Mathematical Modelling*, 37, 2364-2373. https://doi.org/10.1016/j.apm.2012.05.029.
[57] Wheatley, D., Gzara, F., & Jewkes, E. (2015). Logic-based Benders decomposition for an inventory-location problem with service constraints. *OMEGA-International Journal of Management Science*, 55, 10-23. https://doi.org/10.1016/j.omega.2015.02.001.
[58] Wilbrecht, J., & Prescott, W. (1969). The influence of setup time on job shop performance. *Management Science*, 16, 274-280. https://doi.org/10.1287/mnsc.16.4.B274.
[59] Wright, T. (1936). Factors affecting the cost of airplanes. *Journal of Aeronautical Sciences*, 3, 122-128. https://doi.org/10.2514/8.155.
[60] Wu, C., Lee, W., & Lai, P. (2016). Some single-machine scheduling problems with elapsed-time-based and position-based learning and forgetting effects. *Discrete Optimization*, 19, 1-11. https://doi.org/10.1016/j.disopt.2015.11.002.
[61] Yin, Y., Stecke, K., & Kaku, I. (2008). The evolution of *seru* production systems throughout Canon. *Operations Management Education Review*, 2, 27-40. https://doi.org/10.4135/9781526462060.
[62] 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. https://doi.org/10.1016/j.jom.2017.01.003.
[63] 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. https://doi.org/10.1080/00207543.2017.1403664.
[64] Yokoi, K. (2014). Yokoi Style of Sales. Akashi City: Pencom Publication. (in Japanese).
[65] 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. https://doi.org/10.1080/00207543.2012.693642.
[66] 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. https://doi.org/10.1016/j.ijpe.2013.06.009.
[67] 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. https://doi.org/10.1016/j.ejor.2014.01.029.
[68] 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. https://doi.org/10.1016/j.cie.2016.11.035.
[69] Yu, Y., & Tang, J. (2019). Review of seru production. *Frontiers of Engineering Management*, 6(2), 183-192. https://doi.org/10.1007/s42524-019-0028-1.
[70] Zhang, Z., Shao, L., & Yin, Y. (2020). PSO-based algorithm for solving lot splitting in unbalanced *seru* production system. *International Journal of Industrial and Systems Engineering*, 35(4), 433-450. https://doi.org/10.1504/IJISE.2020.108547.