Complex & Intelligent Systems
https://doi.org/10.1007/s40747-021-00432-8

ORIGINAL ARTICLE

A cooperative coevolution algorithm for complex hybrid *seru*-system scheduling optimization

Yuting Wu[1] · Ling Wang[1] · Jing-fang Chen[1]

Received: 13 April 2021 / Accepted: 9 June 2021

Abstract

Under the current volatile business environment, the requirement of flexible production is becoming increasingly urgent. As an innovative production mode, *seru*-system with reconfigurability can overcome the lack of flexibility in traditional flow lines. Compared with pure *seru*-system, the hybrid *seru*-system composed of both *seru*s and production lines is more practical for adapting to many production processes. This paper addresses a specific hybrid *seru*-system scheduling optimization problem (HSSOP), which includes three strongly coupled sub-problems, i.e., hybrid *seru* formation, *seru* scheduling and flow line scheduling. To minimize the makespan of the whole hybrid *seru*-system, we propose an efficient cooperative coevolution algorithm (CCA). To tackle three sub-problems, specific sub-algorithms are designed based on the characteristic of each sub-problem, i.e., a sub-space exploitation algorithm for hybrid *seru* formation, an estimation of distribution algorithm for *seru* scheduling, and a first-arrive-first-process heuristic for flow line scheduling. Since three sub-problems are coupled, a cooperation coevolution mechanism is proposed for the integrated algorithm by information sharing. Moreover, a batch reassign rule is designed to overcome the mismatch of partial solutions during cooperative coevolution. To enhance the exploitation ability, problem-specific local search methods are designed and embedded in the CCA. In addition to the investigation about the effect of parameter setting, extensive computational tests and comparisons are carried out which demonstrate the effectiveness and efficiency of the CCA in solving the HSSOP.

Keywords Hybrid *seru*-system · Cooperative coevolution · Estimation of distribution algorithm · Sub-space exploitation · Problem-specific local search

Introduction

Under the background of Industry 4.0, demand volatility is a growing reality in current market due to the increasing demands of consumers as well as the rapid development of manufacturing technologies. However, traditional flow lines developed for mass production cannot well adapt to the changes in global market. To achieve production flexibility, many companies have transformed the flow lines into other production systems, e.g., *seru*-system [1], which can be constructed by dismantling a flow line. A *seru*-system contains one or more *seru*s, where a *seru* is an assembly unit that consists of simple equipment and one or several multi-skilled workers to operate different types of production [2]. Such a system is more agile than the traditional production system. The agility comes from the quick reconfiguration considering the changes of demands, ensuring high levels of productivity and quality [3]. The *seru*-system can yield satisfactory performances in reducing makespan, labor time, carbon emission, required workforce, WIP inventories, and finished-product inventories [1, 4]. As an innovative production mode, the *seru*-production system has been successfully applied in many companies, e.g., Sony and Canon [5–7]. After reviewing operations management research in last 25 years of studies, Roth et al. [8] listed several promising research directions and pointed out "*seru* production systems are more flexible than Toyota's production system, and they represent the next generation of lean production that has recently been introduced to operations". Nowadays,

✉ Ling Wang
wangling@tsinghua.edu.cn

Yuting Wu
wyt20@mails.tsinghua.edu.cn

Jing-fang Chen
cjf17@mails.tsinghua.edu.cns

[1] Department of Automation, Tsinghua University, Beijing 100084, China

Published online: 27 June 2021

seru-system has gained increasing attention in both academic and engineering fields.

Generally, the *seru*-system can be classified into two types: pure *seru*-system (usually termed *seru*-system for short) composed of one or several *seru*s, and hybrid *seru*-system composed of *seru*(s) and a flow line. For pure *seru*-system, Yin et al. [7] elaborated the origin and conversion process of *seru*-system and analyzed the reasons of its successful applications. Kaku et al. [9] analyzed the human-task-related performances in line-cell conversion (LCC) including the operational tasks, skill level and the cross-training of workers. Three theoretical models including a conveyor assembly line (CAL), a cellular manufacturing (CM) and a joint type (CAL + CM) were built and a human-factor-based training method was presented. Kaku et al. [10] addressed the LCC problem by considering the total throughput time (TTPT) and total labor hour (TLH) as objectives. Yu et al. [11] focused on an LCC problem including two sub-problems: assembly cell formation and assembly cell loading. A mathematical model was developed to minimize the TTPT and the TLH, and the Pareto-optimal front of the model was investigated. Besides, several managerial insights on improving the performance of LCC were summarized to facilitate the implementation of *seru*-system. Yu et al. [12] formulated the LCC problem into a mathematical model with the objectives of reducing worker number and increasing productivity. They designed an exact algorithm to find Pareto-optimal solutions. The results showed that the performance of LCC could be improved by reducing worker number without deteriorating productivity. Yu et al. [13] developed a model to minimize the TTPT and the TLH. They analyzed the mathematical model and revealed some mathematical characteristics of the problem. A nondominated sorting genetic algorithm (NSGA-II) was employed to obtain Pareto-optimal solutions on large-sized cases. Liu et al. [14] designed an algorithm based on NSGA-II to minimize the carbon dioxide emission and makespan. Sun et al. [15] developed a cooperative coevolution algorithm by combining GA, local search and ant colony algorithm for the *seru* production with the minimization of makespan. Yılmaz [16] built a mathematical model for the workforce scheduling problem in *seru*-system and designed an NSGA-II to minimize the makespan and workload imbalance. Yılmaz [17] presented an optimization model to achieve Shojinka (refer to transferring workers in *seru*-system) and designed variants of GAs for distinct scenarios, respectively. The structural characteristics, lower bound, and upper bound were analyzed as well. Zhang et al. [18] proposed a PSO algorithm for a scheduling problem in unbalanced *seru* production system with lot splitting.

Compared to the pure *seru*-system, the hybrid *seru*-system with *seru*s and a flow line is more practical, since it can adapt to many production processes. For instance, some equipment is too expensive or bulky to copy or move to the *seru*s so that it needs to be retained in the flow line. Besides, some temporary workers must be assigned to the flow line, since they are not capable of completing all processes in a *seru*. This situation often occurs when order demand suddenly increases [21]. However, the literature about the hybrid *seru*-system is very scant. To the best of our knowledge, Kaku et al. [19] first presented the concept of hybrid *seru*-system which was formed by a CM and a CAL, and they formulated the system into a model to minimize the TTPT and TLH. Subsequently, Kaku et al. [9] constructed a theoretical model of CAL + CM with the constraints of additional operational tasks, the skill level and the cross-training of workers. Kaku et al. [10] proposed a model to describe the problem of LCC mathematically with the TTPT and the TLH as optimization objectives. Liu et al. [20] proposed a comprehensive mathematical model of hybrid *seru*-system incorporating the number of *seru*s and workers as decision variables. The model was studied on an industrial case, and the experiment results indicated that the proposed model could achieve better performances on the reconfiguration problem of flow line compared to Kaku's model [10]. Yu et al. [21] formulated several hybrid *seru*-system models under an integrated framework and analyzed the properties of hybrid *seru*-system by means of dividing sub-space. An exact algorithm and a heuristic were developed to solve hybrid *seru*-system optimization problem on different scales of instances.

From the above literature review on *seru*-system, most of the research works focus on pure *seru*-system rather than hybrid *seru*-system. Regarding the existing research on hybrid *seru*-system, the proposed algorithms cannot obtain satisfactory solutions in solving large-scale instances with limited computational time. Therefore, it is significant to develop effective algorithms for solving the large-scale hybrid *seru*-system optimization problem.

The cooperative coevolution (CC) [22] provides an effective divide-and-conquer architecture to solve large-scale optimization problems by decomposing the complex problem into several sub-problems. The effectiveness of the CC mechanism exhibits its potential to divide and conquer complex coupled optimization problems [23]. Some typical works about CC are as follows. Omidvar et al. [24] proposed a contribution based cooperative coevolution to alleviate the unbalance due to the different contribution of sub-problems to the global fitness calculation. The sub-problems were treated according to their respective contributions, which effectively saved the computational resources compared with the strategy that treats all the sub-problems equally. Omidvar et al. [25] proposed a differential grouping method that could reveal the interaction structure of the decision variables and form the least connected sub-problems. In addition, they analyzed the validity of this decomposition

strategy mathematically. Shang et al. [26] presented a multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem. The whole population was divided into several sub-populations which evolved, respectively, in each generation, and an internal elitism archive was constructed to speed up the convergence of algorithm. Trunfio et al. [27] presented a new CC algorithm based on the dimension of the sub-components and the size of population during the process of evolution, which outperformed the algorithm based on a homogeneous decomposition. Besides, they investigated the effect of the population size and dimension of sub-problems on the CC mechanism. Hu et al. [28] designed a fast interdependency identification algorithm for CC in solving large-scale global optimization problems. Specifically, the algorithm could recognize whether the variables were separable and then avoid factoring non-separable variables to achieve near-optimal decomposition for CC. Yang et al. [29] proposed a data-driven decomposition method for CC, which mined the information among variables based on historical evolutionary data and grouping the variables dynamically. Li et al. [30] presented a fuzzy decomposition algorithm based on an interaction structure matrix and designed a spectral clustering algorithm to decompose the decision variables according to the number of sub-populations. Yang et al. [31] designed an efficient recursive differential grouping method to reduce the consumption of computation on the decomposition of CC.

In this paper, a cooperative coevolution algorithm (CCA) is proposed to solve the HSSOP with the minimization of makespan criterion. We decompose a hybrid *seru*-system into three decision-making processes, i.e., hybrid *seru* formation, *seru* scheduling, and flow line scheduling. To tackle three strongly coupled sub-problems, effective sub-algorithms are designed for different sub-problems. To be specific, a sub-space exploitation algorithm (SSEA) is developed to solve the hybrid *seru* formation sub-problem by exploiting the solution space with different number of *seru*s; an estimation of distribution algorithm (EDA) is presented to solve the *seru* scheduling sub-problem by learning the distribution of high-quality batching processing sequences; and a first-arrive-first-process (FAFP) heuristic is designed to efficiently and effectively generate a flow line scheduling. To improve the performance of the integrated algorithm, a cooperation coevolution (CC) mechanism is proposed via information sharing. Moreover, a batch reassignment rule (BRR) is designed to avoid mismatch of the partial solutions. To enhance the exploitation ability of CCA, multiple local search methods are developed based on the characteristics of sub-problems. Extensive testing results and comparisons demonstrate the effectiveness of the proposed algorithm.

The remaining contents of the paper are organized as follows. Section "Formulation of HSSOP" describes the HSSOP with a computational model. In Section "CCA for HSSOP", the design of the CCA is presented in detail. Section "Computational results and comparisons" presents the investigation on parameter setting and the comparison on performances. Finally, the paper is ended with some conclusions and the future work.

Formulation of HSSOP

Problem description

The hybrid *seru*-system studied in this paper is shown in Fig. 1, which is a generalization of the model in [4]. The complicated operation of such a hybrid *seru*-system is considered as a four-stage decision process, including worker allocation, *seru* formation, *seru* scheduling and flow line scheduling. The operation process of the hybrid *seru*-system is stated as follows.

The problem to be solved in this paper concentrates on structuring the hybrid *seru*-system and scheduling batches in the system properly. There are W workers with different skill levels and N batches with various types. Different workers have different levels of proficiency for different product types. The processing time of various batches in different *seru*s is different due to the diversity of worker skill levels in each *seru*. All the batches should pass through the *seru*-system first and then through the flow line to complete all the processing tasks. A hybrid *seru*-system is constructed, where three aspects are required to be determined: the number of workers allocated to the *seru*s, the number of *seru*s to be formed and the number of workers assigned to the flow line. Step 1: determine the numbers of workers in the flow line and *seru*s. Step 2: determine the number of *seru*s to be formed and the workers allocated to each *seru* and the flow line. Then, batches should be assigned to the hybrid *seru*-system. There are N batches to be processed and each batch can be assigned to any of the *seru*s. Step 3: determine the *seru* which each batch assigns to and then determine the processing sequence of batches in each *seru*. Once the batch assignment is determined, the whole assembling process of a batch must be fulfilled within a single *seru*. Step 4: after all the batches are assembled by the *seru*-system, they are sent to the flow line for final manufacturing procedure. The sequence of batches in the flow line is determined in this step. Briefly, the above four steps are called worker allocation, *seru* formation, *seru* scheduling and flow line scheduling, respectively. The objective is to minimize the makespan of hybrid *seru*-system by reasonably arranging the four-stage decision process.

In Fig. 1, two *seru*s are constructed: *seru* 1 contains worker 1 and worker 5, while *seru* 2 contains worker 2 and worker 4. Worker 3 and worker 6 are assigned to the flow

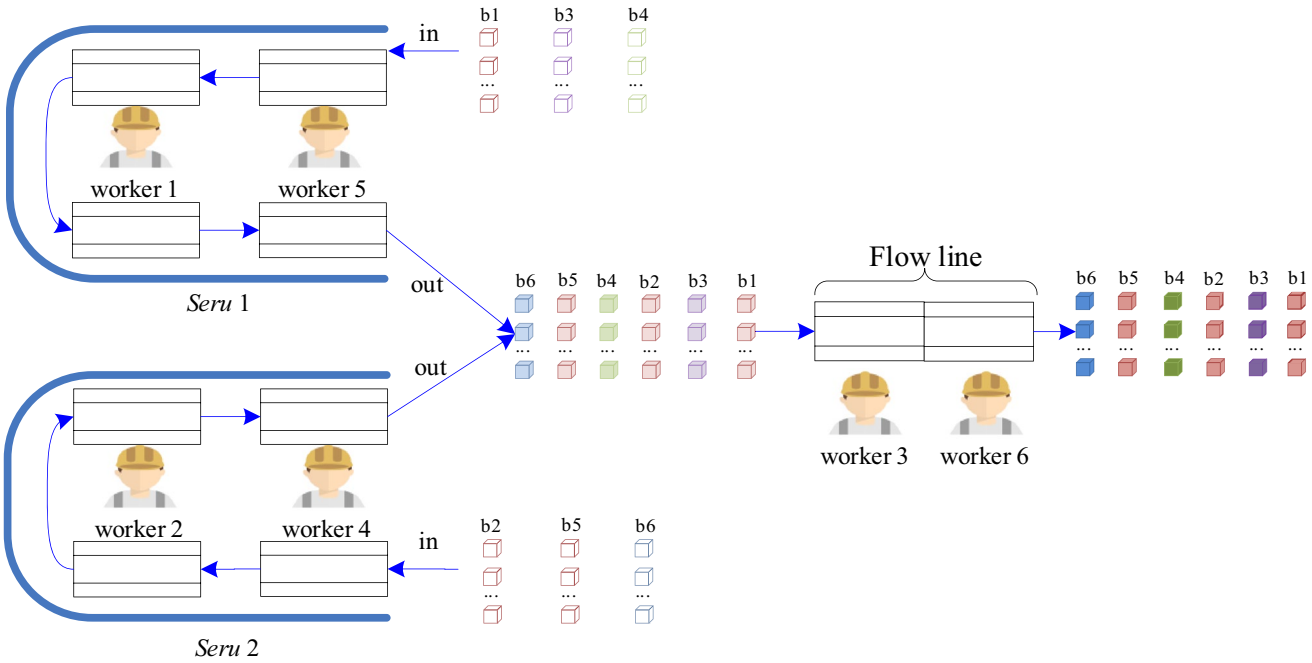

Fig. 1 An example of hybrid *seru*-system

line. There are 6 batches to be processed. Batch 1, batch 3 and batch 4 are processed in *seru* 1 in order, while batch 2, batch 5 and batch 6 are processed in *seru* 2 in sequence. Finally, six batches are sent to the flow line to finish the final process.

Computational model

The following assumptions are considered in this paper.

- The product type of each batch is known in advance. Each batch specifies a particular product type.
- The required tasks of each product are the same. If a product type does not need a task, the product skips the task.
- The tasks are the same as the ones in the original flow line. The number of tasks is W.
- In the original flow line, each worker is responsible for a task. After dismantling the flow line into a hybrid *seru*-system, workers assigned to the *seru*-system can operate all the processing tasks.
- The workers left in the flow line still process the tasks that are originally processed in the original flow line.

The notation definitions are presented as below.

Indices:
i: Index of workers ($i = 1, 2, \ldots, W$).
j: Index of *serus* ($j = 1, 2, \ldots, J$).
n: Index of product types ($n = 1, 2, \ldots, N$).
m, s: Index of product batches ($m = 1, 2, \ldots, M$).
k: Index of the sequence of product batches in a *seru* ($k = 1, 2, \ldots, M$).
r: Index of the sequence of product batches in the flow line ($r = 1, 2, \ldots, M$).
Parameters:
$V_{m,n}$: 1, if product type of batch m is n; 0, otherwise.
B_m: Size of product batch m.
T_n: Cycle time of product type n in the original flow line.
η_i: Upper bound on the number of tasks for worker i in a *seru*. If the number of tasks assigned to worker i is more than η_i, then worker i's average task time within a *seru* will be longer than her or his task time within the original flow line.
ε_i: Worker i's coefficient of influencing level of doing multiple tasks.
β_{ni}: Skill level of worker i for each task of product type n.

Binary variables:
Y_i: 1, if worker i is left in the flow line; 0, otherwise.
$X_{i,j}$: 1, if worker i is assigned to *seru* j; 0, otherwise.
$Z_{m,j,k}$: 1, if product batch m is assigned to *seru* j in sequence k; 0, otherwise.
$O_{m,r}$: 1, if product batch m is processed in flow line in sequence r; 0, otherwise.

Four binary variables correspond to the four decision processes: worker allocation, *seru* formation, *seru* scheduling and flow line scheduling. If the four binary variables are determined, the makespan of hybrid *seru*-system can be calculated as follows.

$$C_i = \begin{cases} 1 + \varepsilon_i \left(W - \sum_{i'=1}^{W} Y_{i'} - \eta_i \right), & W > \eta_i \\ 1, & W \le \eta_i \end{cases}, \quad \forall i \tag{1}$$

$$\mathrm{TC}_m = \frac{\sum_{n=1}^{N} \sum_{i=1}^{W} \sum_{j=1}^{J} \sum_{k=1}^{M} V_{m,n}\, T_n\, \beta_{n,i} C_i\, X_{i,j}\, Z_{m,j,k}}{\sum_{i=1}^{W} \sum_{j=1}^{J} \sum_{k=1}^{M} X_{i,j} Z_{m,j,k}} \tag{2}$$

$$\mathrm{FC}_m = \frac{B_m\, \mathrm{TC}_m\, W}{\sum_{i=1}^{W} \sum_{j=1}^{J} \sum_{k=1}^{M} X_{i,j}\, Z_{m,j,k}} \tag{3}$$

$$\mathrm{FCB}_m = \sum_{p=1}^{m-1} \sum_{j=1}^{J} \sum_{k=1}^{m} FC_p\, Z_{m,j,k}\, Z_{p,j,(k-1)} \tag{4}$$

$$\mathrm{TL}_m = \max_{1 \le i \le W} (V_{m,n}\, T_n\, \beta_{n,i}\, Y_i), \quad \forall i \tag{5}$$

$$\mathrm{FL}_m = \sum_{n=1}^{N} \sum_{i=1}^{W} V_{m,n}\, T_n\, \beta_{n,i}\, Y_i + \left(B_m - 1\right) TL_m \tag{6}$$

$$\mathrm{FLB}_m = \begin{cases} \mathrm{FCB}_m + \mathrm{FC}_m, & m \text{ is finished later than } s \\ \mathrm{FLB}_s + \mathrm{FL}_s, & m \text{ is finished no later than } s \end{cases}, (s | O_{m,r} = 1,\ O_{s,(r-1)} = 1,\ \forall r) \tag{7}$$

$$C_{\max} = \max_{1 \le m \le M} (\mathrm{FLB}_m + \mathrm{FL}_m) \tag{8}$$

Equation (1) is to calculate the coefficient variation of worker i's increased task time after line-*seru* conversion. If the number of worker i's tasks within a *seru* is over her/his upper bound η_i, then the worker will cost more average task time than her/his task time within the original flow line. Equation (2) is to calculate the processing time of product batch m per task in the *seru*. In a *seru*, the processing time of product type n is the average processing time of all the workers in the *seru*. Equation (3) is to calculate the processing time of product batch m in the *seru*. Equation (4) is to calculate the begin time of product batch m in the *seru*. Equation (5) is to calculate the worst task time of the workers in the flow line for product batch m. Equation (6) is to calculate the processing time of product batch m in the flow line. Equation (7) is to calculate the begin time of product batch m in the flow line. Since the hybrid system has *seru*s and a flow line, FLB_m should be the maximum between the completion time of the prior product batch s in the flow line and the completion time of product batch m in *seru*s. If the completion time of product batch m in *seru*s is later than the completion time of product batch s in the flow line, then FLB_m equals $FCB_m + FC_m$; otherwise, FLB_m equals $FLB_s + FL_s$. Equation (8) is to calculate the makespan of hybrid *seru*-system, which is the completion time of the last processing batch in the flow line.

CCA for HSSOP

The HSSOP contains multiple coupled decision-making sub-problems, which makes it difficult to solve directly. Therefore, we employ a divide-and-conquer strategy to deal with such a complex problem. According to the four-stage decision process described in 2.1, it is clear that the purpose of worker allocation and *seru* formation is to build the hybrid *seru*-system. Therefore, we combine the two decision processes as one sub-problem. Thus, the HSSOP can be divided into three sub-problems: hybrid *seru* formation (including worker allocation and *seru* formulation), *seru* scheduling and flow line scheduling.

For a non-separable problem, strong interdependency between sub-problems definitely causes the difficulty in evaluating the performance [32]. For the HSSOP, to evaluate a complete solution, it should consider three sub-problems jointly, i.e., hybrid *seru* formation (HSF), *seru* scheduling (SS), and flow line scheduling (FLS). A complete solution consists of three partial solutions: partial solution-HSF, partial solution-SS and partial solution-FLS. The description of constructing a complete solution of hybrid *seru*-system is shown in Fig. 2.

For hybrid *seru* formation, the makespan depends on both the number of *seru*s and the allocation of workers; for *seru* scheduling, the makespan depends on the batch assignment and processing sequence in *seru*-system; for flow line scheduling, the makespan depends on the batch processing sequence in flow line which is affected by the completion time of each batch in *seru*-system. The HSSOP is so complicated that it cannot be enumerated or solved by mathematical programming in limited time. Heuristic algorithms usually cannot get satisfactory solutions for large-scale instances. Comparatively, evolutionary algorithms can gain satisfactory solutions in limited time and have been successfully applied to pure *seru*-system [14–18]. Thus, it motivates us to develop an effective evolutionary algorithm to solve the HSSOP with makespan minimization criterion. Since

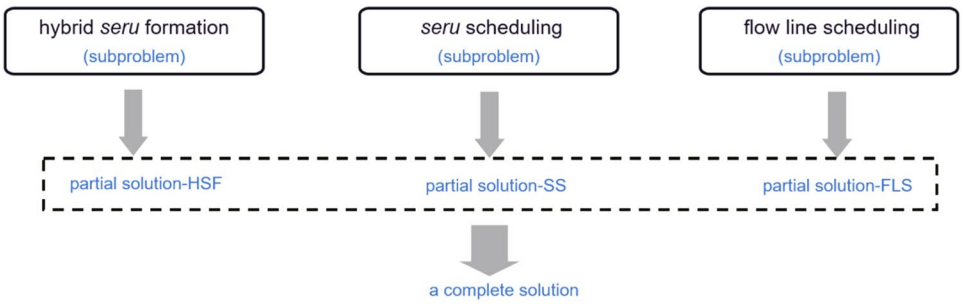

Fig. 2 A complete solution of hybrid *seru*-system

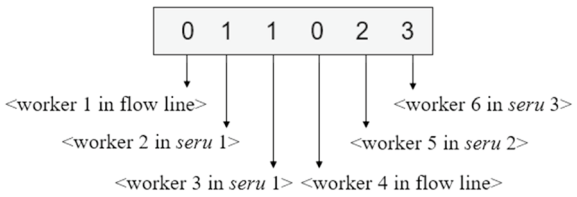

Fig. 3 The representation of a partial solution-HSF

sub-algorithms play important roles in facilitating the integrated algorithm, we design different sub-algorithms for different sub-problems according to their characteristics in this paper. So, we design a special SSEA to determine the number of *seru*s and the allocation of workers, design a special EDA to determine batch assignment and processing sequence in *seru*-system by learning the distribution of batch processing priority, and design a special FAFP heuristic to determine the batch processing sequence in flow line. Next, we introduce the SSEA, EDA and FAFP in detail.

The SSEA for hybrid seru formation

For hybrid *seru* formation, a solution should reflect the worker allocation and *seru* formation (the number of *seru*(s) to be constructed and the worker assignment in each *seru* and flow line). In this paper, solution space is partitioned into several sub-spaces according to the number of *serus*. Each sub-space contains the partial solution-HSFs with same number of *serus*. Each solution is encoded as a string with $W+1$ integer value. The i th number of the string implies whether the i th worker is allocated to the flow line or the *seru*s. Integer j ($j>0$) means that the worker is allocated to *seru* j, and 0 means that the worker is allocated to the flow line. To ensure the feasibly of the partial solution-HSF, each integer value in $[0, J]$ must appear at least once.

An example of a partial solution-HSF is shown in Fig. 3. There are 6 workers, and 3 *serus* are constructed. A feasible solution P_i can be {0 1 1 0 2 3}, which implies that worker 1 and worker 4 are assigned to the flow line, worker 2 and worker 3 are assigned to *seru* 1, worker 5 is assigned in the *seru* 2 and worker 6 is assigned in the *seru* 3.

In [21], it is shown that the minimal makespan usually exists in the sub-space-*seru* with fewer *seru*s by analyzing the features of solution space. With such prior knowledge, an SSEA is designed to evolve hybrid *seru* formation population for good partial solution-HSFs. The procedure of the SSEA is shown in Fig. 4, which includes population initialization, crossover, and local search.

Population initialization

Clearly, W workers can construct a hybrid *seru*-system with at most W-1 *serus*. Therefore, we construct W-1 sub-spaces in the initialization phase. Each sub-space contains Q partial solution-HSFs that are generated randomly. Dividing the initial population according to the number of *seru*, it is helpful to generate diverse initial partial solution-HSFs so as to enhance the capability of global exploration. An example of population initialization with 6 workers is shown in Fig. 5.

Crossover operators

To evolve the hybrid *seru* formation, two crossover operators are used for every sub-space. For each sub-space, it finds the best individual P_b and the worst individual P_w. Then, it determines the global best individual P_g among all sub-spaces. To exchange the information among different individuals, we design two crossover operators as illustrated in Fig. 6 with examples, where 10 workers, 3 *serus*, P_b= {0 3 1 0 2 1 3 2 0 1}, P_w= {2 0 2 3 1 0 2 0 3 0}, P_g= {0 3 2 0 1 3 0 1 2 2}, p 1 = 6, p 2 = 4, p 3 = 7.

Crossover operator 1: Single point crossover performed on P_b and P_w to produce a new individual P_n. To be specific, first a random position p 1 is generated, and then the elements before position p 1 in P_b and the elements after position p1 in P_w are combined to form a new individual P_n.

Crossover operator 2: Multiple-point crossover performed on P_g, P_b and P_w. To be specific, first two random positions p2 and p3 are generated; then the elements between p 2 and p 3 in P_w is reserved to the new individual P_n, and

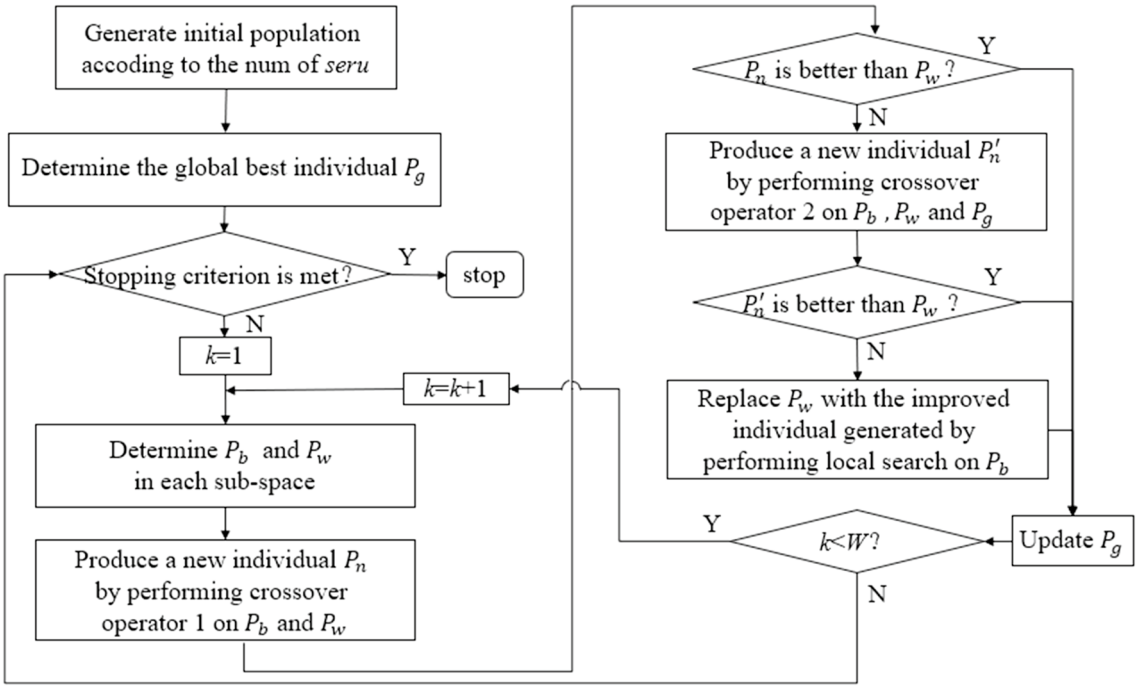

Fig. 4 The procedure of the SSEA for hybrid *seru* formation population

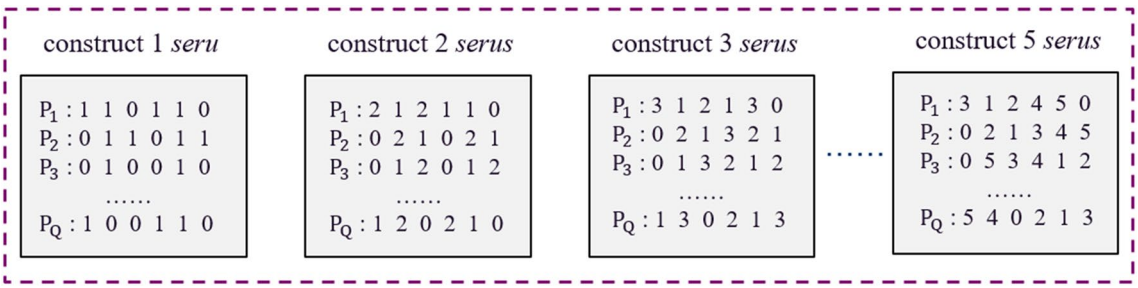

Fig. 5 An example of population initialization with 6 workers

the elements before p2 in P_g are inherited to P_n, and the elements after p3 in P_n are taken from P_g.

If P_n is better than P_w, then replace P_w with P_n and update P_g if necessary.

Local search in SSEA

In SSEA, if crossover operator 2 fails to improve the worst individual, a local search will be performed on P_b for further exploitation. In this paper, the following search operators are used in a hybrid way as Algorithm 1, where γ denotes the search depth of local search.

LS-swap: Swap the p th element and the qth element in a string, where p and q are two different integers generated randomly.

LS-insert: Insert the p th element after the q th element in a string.

LS-reverse: Reverse the sub-sequence between the p th element and the q th element in a string.

LS-exchange: Choose two different elements d and b ($0 \le d < b \le W-1$) and then exchange all d and b.

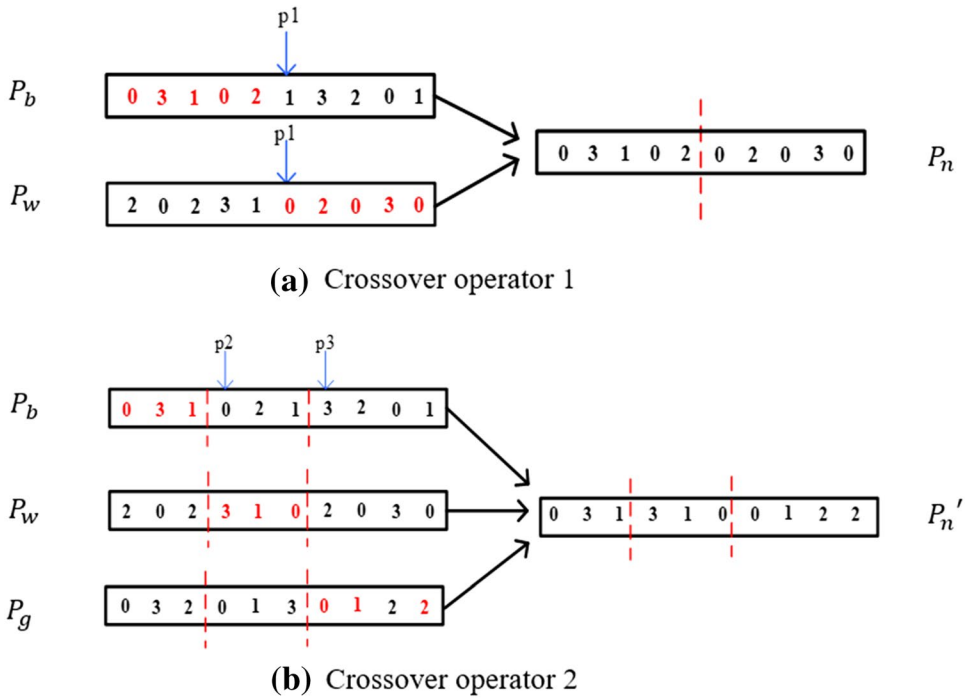

Fig. 6 Illustration of two crossover operators

Algorithm 1 Procedure of local search

Input: a solution P
1: Set l=0, k=0;
2: **While** $l<\gamma$
3: **If** k=0
4: Perform LS-swap to get P'
5: **Else if** k=1
6: Perform LS-insert to get P'
7: **Else if** k=2
8: Perform LS-reverse to get P'
9: **Else if** k=3
10: Perform LS-exchange to get P'
11: **End if**
12: **If** $P > P'$
13: P=P', k=k+1, l=l+1
14: **End if**
15: **If** k>3
16: k=0
17: **End if**
18: **End While**
Output: the improved solution

EDA for *seru* scheduling

As a population-based evolutionary algorithm, EDA [33] has been successfully applied to a variety of scheduling problems, such as flow shop scheduling [34–36], flexible job shop scheduling [37], and robotic assembly line balancing [38]. Compared with the crossover and mutation-based evolutionary algorithms such as GA, EDA employs an explicit probability model for sampling a promising area to achieve exploration.

The *seru* scheduling should solve two sub-problems, i.e., the batch assignment and the batch sequencing in each *seru*. To solve this problem, an EDA with local search (EDA-LS) is proposed by fusing the exploration ability of EDA and the exploitation ability of local search. The procedure of the EDA-LS is shown in Fig. 7. Next, we will introduce encoding and decoding, probability model and its updating mechanism, and local search in detail.

Encoding and decoding of a partial solution-SS

For *seru* scheduling, it needs to determine the batch assigned to each *seru* and the batch processing sequence in each *seru*. Let π denote an individual of *seru* scheduling (a partial solution-SS) encoded as a J-dimensional vector $[v_1\ v_2\ \ldots\ v_J]^T$, where v_j represents the batch processing sequence in *seru* j. A partial solution-SS can be represented by a batch priority string φ with a length of M. The element in the batch priority string is denoted by job number. The occurrence of a batch number in q th position of the batch priority string indicates such a batch is processed in q th order.

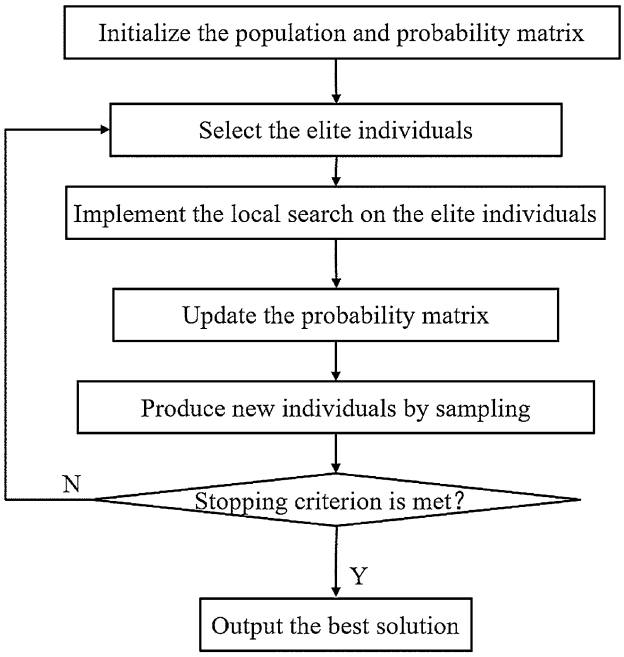

Fig. 7 Procedure of the EDA-LS for *seru* scheduling

Algorithm 2 Procedure of ECFL rule

Input: a batch priority string φ

For j=1 to J
  $N_j = 0$
End for
For q=1 to M
  Find the *seru* j that can process batch $\varphi(q)$ with the earliest completion time in the flow line.
  $\pi_j[N_j] = \varphi(q)$
  $N_j = N_j + 1$
End for

Output: a partial solution-SS π

To decode a batch priority string φ into a partial solution-SS π, an earliest completion flow line (ECFL) rule shown in Algorithm 2 is designed to assign batches to *seru* according to the batch priority string φ. Consider an example with 7 batches and 2 *seru*s, the related data are given in Table 1. An encoded partial solution-SS is illustrated in Fig. 8, while a decoded hybrid *seru* scheduling is illustrated in Fig. 9.

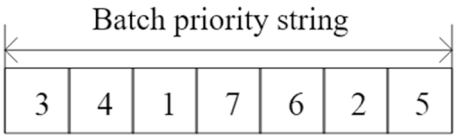

Fig. 8 Illustration of the representation of a partial solution-SS

Probability model and updating mechanism

A probability model is built to reflect the distribution of the solution space and produce new solutions. The probability model should keep updating by a reasonable mechanism to achieve exploration [39]. In this paper, the probability model and its updating mechanism are designed as follows.

According to the above encoding method, batch priority string directly affects the *seru* scheduling. Therefore, the following probability model $P(g)$ is designed to sample the batch priority string, where element $\rho_{ij}(g)$ denotes the probability that batch b_j appears in the i th position of the batch priority string at the g th generation. The elements in matrix $P(0)$ are initialized as $\rho_{ij}(0) = 1/M$ $(\forall i, j)$ for uniform sampling.

$$P(g) = \begin{bmatrix} \rho_{11}(g) & \rho_{12}(g) & \cdots & \rho_{1M}(g) \\ \rho_{21}(g) & \rho_{22}(g) & \cdots & \rho_{2M}(g) \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{M1}(g) & \rho_{M2}(g) & \cdots & \rho_{MM}(g) \end{bmatrix} \tag{9}$$

At each generation, a new batch priority string is generated by sampling the probability matrix to determine the position of each batch in the batch priority string. If the position of batch b_j has already been determined, the whole elements in j th column of probability matrix $P(g)$ are set as zero. Then, all the elements in $P(g)$ are normalized, which ensures that the sum of each row is 1. After determining the batch priority string, the processing sequence of batches in each *seru* can be obtained according to the batch priority string based on the ECFL rule. Thus, an individual is generated.

After generating EP new individuals of *seru* scheduling at each generation, EP_{size} elite individuals are selected to update the probability matrix $P(g)$, where $EP_{size} = \alpha * EP$. The probability matrix $P(g)$ is updated as follows via

Table 1 The processing time of seven batches in two *serus* and flow line

seru/flow line	Processing time of batches						
	1	2	3	4	5	6	7
seru 1	95	80	86	77	102	93	50
seru 2	101	76	94	101	96	71	43
Flow line	30	21	47	28	49	37	21

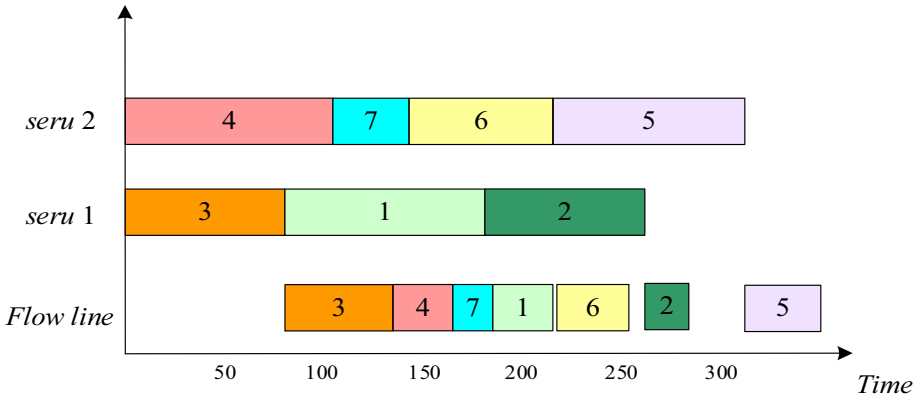

Fig. 9 An example of hybrid *seru* scheduling with seven batches and two *seru*s

incremental learning which is based on the historical information of the better individuals.

$$\rho_{ij}(g+1) = (1-\beta)\,\rho_{ij}(g) + \beta \cdot \frac{1}{EP_{size}} \cdot \sum_{h=1}^{EP_{size}} I_{ij}^{h}(g), \tag{10}$$

where $\beta \in (0,1)$ denotes the learning rate and $I_{ij}^{h}(g)$ is the indicator function corresponding to the h th elite individual.

$$I_{ij}^{h}(g) = \begin{cases} 1, & \text{if batch } j \text{ appears in the } i\text{th position in the batch priority string} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Local search in EDA

To enhance the exploitation ability of EDA, a local search is designed according to the characteristic of problem. According to the computational model, the makespan of flow line with given *seru* scheduling is $WT_1 + SPT_m + \sum_{m=2}^{M} WT_m$, where WT_m is the waiting time of the m th batch processed in flow line, and $SPT_m = \sum_{m=1}^{M}\{\sum_{n=1}^{N}\sum_{i=1}^{W} V_{mn}T_n\beta_{ni}Y_i + (B_m - 1)TL_m\}$ is the sum of processing time of batches in flow line. If the completion time of batch m in *seru*s is not later than the completion time of batch m-1 in the flow line, then the WT_m is zero; otherwise, WT_m is not zero. An example is illustrated in Fig. 10. If hybrid *seru* formation is determined, SPT_m is a constant. Therefore, the makespan of flow line varies as WT_m. We define the batches with non-zero waiting time as critical batches, e.g., batch 3, batch 6, batch 2 and batch 5 in Fig. 10. Let B_c be the set of critical batches. Clearly, the makespan of a hybrid *seru*-system can be reduced by adjusting the position of critical batches in *seru*-system.

Based on the above analysis, we design three critical batch-based local search operators to adjust the batch priority string: LS (1), LS (2) and LS (3) [40, 41]. Each local search operator incorporates a critical batch b_c $(b_c \in B_c)$ and a randomly selected batch b_r $(b_r \neq b_c)$.

LS (1): Swap the order of batch b_c and batch b_r in the batch priority string.

LS (2): Insert batch b_c to the position after batch b_r in the batch priority string.

LS (3): Invert the sequence between the position of batch b_c and batch b_r in the batch priority string.

The local search enumerates all feasible positions for better solutions. If a local search operator can generate a better

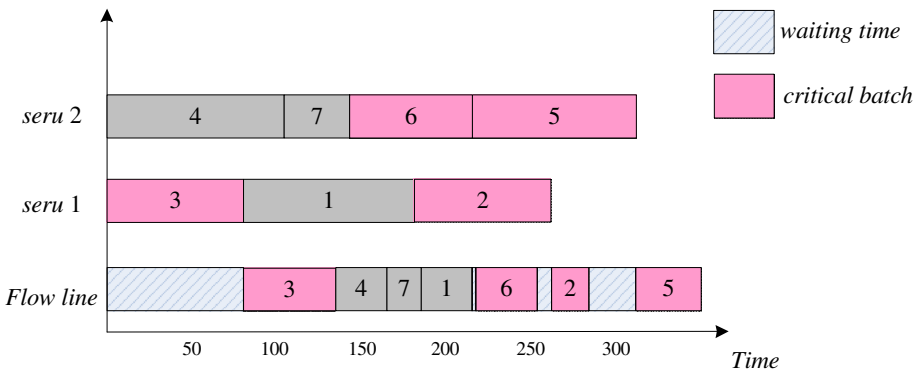

Fig. 10 Illustration of waiting time and critical batch

new individual (defined as a valid operator), then restart enumeration on the new solution; otherwise, employ the next local search operator until all the operators are used. The local search procedure is shown in Algorithm 3. Note that local search is performed only on the elite individuals.

Algorithm 3 Procedure of local search for *seru* scheduling

Input: an elite individual S
Calculate the makespan of hybrid *seru*-system according to algorithm 1, denoted as C_{max} of S.
Determine the set of critical batches B_c.
For each batch $b_c \in B_c$
  For k= 1 to 3
    Implement search operator LS(k) and gain a solution of *seru* scheduling S'. Calculate
    C_{max} of S'.
    If C_{max} of S' is improved
      $S \leftarrow S'$
      Go to line 1
    End if
  End for
End for
Output: an improved individual.

FAFP heuristic for flow line scheduling

For flow line scheduling, it needs to determine the processing order of batches in the flow line. Batches should be processed in the *seru*-system before being processed in the flow line. To reduce the waiting time of batches in the flow line, it is inferred that a better makespan can be obtained by setting higher processing priority in the flow line for the batch with earlier completion time in *seru*-system.

Based on the above analysis, first-arrive-first-process (FAFP) heuristic is designed for the flow line scheduling. To be specific, the batch that first arrives at the flow line is processed first in the flow line. Then, the makespan can be calculated according to the FAFP heuristic given a *seru* scheduling. The pseudo code is shown in Algorithm 4.

Cooperation coevolution mechanism

To evaluate the quality of a partial solution, it requires the other information of a complete solution. The quality of a complete solution depends on all the information of three parts. To form a complete solution reasonably, the optimal partial solution of each sub-problem is provided to other sub-problems. The quality of the complete solution is used to evaluate the partial solution in current sub-problem. A cooperation coevolution mechanism is employed to interact information of partial solutions generated by each sub-algorithm.

Since the FAFP heuristic is used in the flow line scheduling, the objective can be calculated when the hybrid *seru* formation and the *seru* scheduling are determined. The population of *seru* scheduling evolves by EDA. For evaluating the quality of a partial solution-SS, the best hybrid *seru* formation is used as a partial solution-HSF and the flow line scheduling determined by FAFP heuristic is used as a partial solution-SLS. The best partial solution-SS in the

Algorithm 4 The calculation of C_{max} according to FAFP heuristic

For batches in *seru* scheduling, calculate the completion time of batches in each *seru* ($FCB_{bi} + FC_{bi}$).
Sort the batches in an ascending order according to the completion time in *seru*-system $B' = \{b'_1, b'_2, \ldots, b'_i, \ldots, b'_N\}$.
Set C_{max}=$FC_{b1'}$
For i=1 **to** N
  If $FCB_{bi'} + FC_{bi'} > FLB_{bi-1'} + FL_{bi-1'}$
    C_{max}=$FCB_{bi'} + FC_{bi'} + FL_{bi'}$;
  Else
    C_{max}=C_{max}+$FL_{bi'}$;
  End if
End for

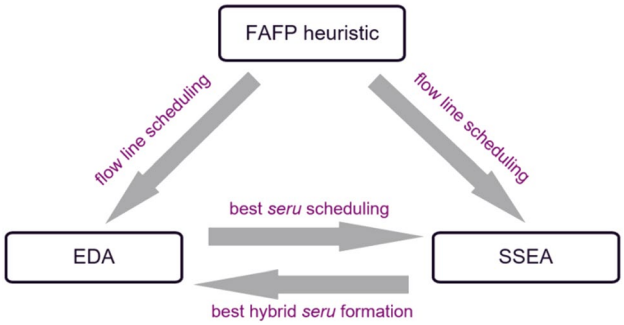

Fig. 11 The cooperation of sub-algorithms

evolution process of *seru* scheduling population is denoted as the best *seru* scheduling. Subsequently, the population of hybrid *seru* formation evolves by SSEA. For evaluating the quality of a partial solution-HSF, the best *seru* scheduling is used as a partial solution-SS and the partial solution-SLS is determined by FAFP heuristic. The best partial solution-HSF in the process of evolution is denoted as the best hybrid *seru* formation. With the mechanism of sharing best individuals shown in Fig. 11, populations coevolve so that the search capability of the integrated algorithm can be enhanced.

However, there may exist a mismatch of *seru* number in the process of evaluating partial solution-HSFs. To be specific, a partial solution-HSF and the best partial solution-SS cannot form a feasible complete solution when their *seru* numbers are different. Thus, we design a batch reassignment rule (BRR) to avoid the mismatch between partial solutions for generating legal solutions. To be specific, a batch sequence in the flow line is determined by the FAFP heuristic with the partial solution-SS. According to this batch sequence, batches are assigned to the *seru* with the earliest completion time. If there are multiple *seru*s with same completion time, select one randomly. The procedure of the BRR is shown in Algorithm 5.

Algorithm 5 Procedure of the BRR

Input: a partial solution-SS π', the *seru* num of partial solution-HSF J

For batches a partial solution-SS π', calculate the completion time in each *seru*.
Sort the batches in an ascending order according to the completion time $B = \{b_1, b_2, \ldots, b_i, \ldots, b_N\}$.
For k=1 **to** J
  $N_k = 0,\ \pi_k(0) = 0,\ R_k = 0$
End for
For i=1 **to** N
  $l = arg_k \min\{R_k\}$
  $N_l = N_l + 1$
  $\pi_l(N_l) = b_i$
  $R_l = R_l + FC_{bi}$
End for

Output: a partial solution-SS π

The procedure of CCA

With the above designs, the whole procedure of CCA is shown in Algorithm 6.

Algorithm 6 Procedure of the CCA

Input: An instance with *M* batches and *W* workers.

Initialize:
  Randomly generate a partial solution-HSF as the best partial solution-HSF P^*.
  Randomly generate a partial solution-SS as the best partial solution-SS π^*.
  Randomly generate a complete solution as the best solution S^*.
Do
  Evolve the population of hybrid *seru* formation with π^* by SSEA, BRR and FAFP, and update the best partial solution-HSF P^* and the best solution S^*.
  Evolve the population of *seru* scheduling with P^* by EDA and FAFP, and update the best partial solution-SS π^* and the best solution S^*.
Until terminal condition is met.

Output: The best solution S^*

Table 2 Data of worker's level of skill (β_{ni})

Worker/ product type	1	2	3	4	5	Worker/ product type	1	2	3	4	5
1	1.02	1.05	1.1	1.05	1.13	16	0.9	1.01	1.17	1.22	1.23
2	1.09	1.15	1.16	1.24	1.29	17	1.1	1.09	1.01	1.13	1.12
3	0.96	0.98	1.06	1.16	1.22	18	0.94	0.99	1.23	1.13	1.31
4	0.94	0.99	1.1	1.09	1.1	19	1.09	1.09	1.01	1.23	1.28
5	0.96	1.1	1.08	1.07	1.23	20	1	1.11	1.07	1.12	1.27
6	0.92	0.97	1.12	0.99	1.2	21	1.02	1.05	1.1	1.05	1.13
7	1.1	1.13	1.13	1.22	1.27	22	1.09	1.15	1.16	1.24	1.29
8	0.98	1.08	1.06	1.3	1.16	23	0.96	0.98	1.06	1.16	1.22
9	1.03	1.03	1.13	1.25	1.11	24	0.94	0.99	1.1	1.09	1.1
10	0.97	1.14	1.2	1.21	1.22	25	0.96	1.1	1.08	1.07	1.23
11	1.04	1.1	1.03	1.12	1.19	26	0.92	0.97	1.12	0.99	1.2
12	0.95	1.05	0.99	1.2	1.22	27	1.1	1.13	1.13	1.22	1.27
13	0.92	0.98	1.13	1.03	1.27	28	0.98	1.08	1.06	1.3	1.16
14	1.08	1.09	1.09	1.18	1.14	29	1.03	1.03	1.13	1.25	1.11
15	1.06	1	1.13	1.08	1.19	30	0.97	1.14	1.2	1.21	1.22

Table 3 Coefficient of influencing level of skill to multiple tasks for workers (ε_i)

Worker	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ε_i	0.18	0.19	0.2	0.21	0.2	0.2	0.2	0.22	0.19	0.19	0.18	0.23	0.24	0.22	0.16
Worker	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
ε_i	0.24	0.18	0.18	0.21	0.18	0.16	0.24	0.23	0.16	0.22	0.18	0.17	0.21	0.24	0.18

Table 4 Data of batches

Batch number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Product type	3	5	3	4	1	4	1	2	2	3	2	4	3	4	5	5	1	4
Batch size (B_m)	55	53	54	49	49	55	54	48	48	48	46	58	48	52	48	51	54	57
Batch number	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
Product type	2	5	1	3	4	5	2	3	1	4	2	3	2	1	5	4	3	2
Batch size (B_m)	54	49	53	46	45	46	45	44	53	47	53	52	50	43	42	50	46	34
Batch number	37	38	39	40	41	42	43	44	45	46	47	48	49	50				
Product type	1	3	5	2	4	5	5	1	4	2	5	1	3	4				
Batch size (B_m)	36	37	45	46	52	48	51	54	57	54	49	53	46	45				

Table 5 Parameter values

Parameters	Factor level			
	1	2	3	4
α	0.04	0.16	0.28	0.40
β	0.01	0.08	0.15	0.22
γ	5	10	15	20

In the initialization phase, a partial solution-HSF, a partial solution-SS, and a complete solution are randomly generated as the best partial solution-HSF P^*, the best partial solution-SS π^*, and the best solution S^*.

Then, the population of hybrid *seru* formation evolves by SSEA (in 3.1). When evaluating each individual in the population, the individual is combined with π^* and the partial solution-FLS generated by BRR (in 3.4) and FAFP (in 3.3) to form a complete solution. Then, the best partial solution-HSF P^* and the best solution S^* are updated if necessary.

Next, the population of *seru* scheduling evolves by EDA (in 3.2). When evaluating each individual in the population, the individual is combined with P^* and the partial solution-FLS generated by FAFP (in 3.3) to form a complete solution. Then, the best partial solution-SS π^* and the best solution S^* are updated if necessary.

Due to the information sharing in two populations, they coevolve iteratively until the termination criterion is met.

Table 6 Orthogonal array and ARVs

Experiment number	Factor level			ARV (%)
	α	β	γ	
1	1	1	1	1.43
2	1	2	2	0.69
3	1	3	3	0.51
4	1	4	4	0.36
5	2	1	2	0.36
6	2	2	1	0.3
7	2	3	4	0.2
8	2	4	3	0.4
9	3	1	3	0.41
10	3	2	4	0.37
11	3	3	1	0.34
12	3	4	2	0.51
13	4	1	4	0.64
14	4	2	3	0.22
15	4	3	2	0.46
16	4	4	1	0.62

Computational results and comparisons

Experimental settings

Since there are no existing instances on hybrid *seru*-system, a set of instances are produced by extending the existing line-hybrid *seru*-system conversion instances [21] as follows. There are five product types, and $T_n = 1.8$, $\eta_i = 10$. The data of β_{ni}, ε_i and batches are given in Tables 2, 3 and 4, respectively, which are expanded on the basis in [21]. For an instance with W workers and M batches, we use the following data from Tables 2, 3, 4, the first W rows in Table 2, the first W columns in Table 3, and the first M batches in Table 4. Thus, a total of 20 combinations are generated with $M \in \{10, 20, 30, 40, 50\}$, $W \in \{5, 10, 20, 30\}$. We code the algorithm in C# and run it on a PC with an Intel Core i5-1035G7 CPU @ 1.2-GHz under Microsoft Windows 10. Consider that different scales of instances need different computation time, we use $0.4 \times M \times W$ s CPU time as the stopping criterion for the CCA in following tests. In the CCA, EDA terminates once it reaches 20 generations and SSEA terminates once the optimal solution remains unchanged for 20 consecutive generations.

Parameter settings

Clearly, a large value of population size in EDA is helpful for sufficient exploration. However, an overlarge value will result in long convergence time and a small value may lead to insufficient evolution. Besides, an appropriate size of sub-space in SSEA is important to balance the local intensification ability and the running time. According to a large amount of preliminary experiment and investigation, we set the population size of EDA as 50 and the size of sub-space as 10. In addition to the above parameters, the CCA contains three other key parameters: the rate of elite solutions α and the learning rate β in EDA, and the search depth of local search γ in SSEA. To investigate the influence of such parameters, the Taguchi method of design-of-experiment (DOE) [42] is implemented. Four factor levels are employed for each parameter as Table 5. An orthogonal array $L_{16}(4^3)$ is chosen to generate the combinations of different values.

A total of 20 instances are used for the investigation. For each instance, the CCA with each combination t is run ten times independently to obtain the average objective value

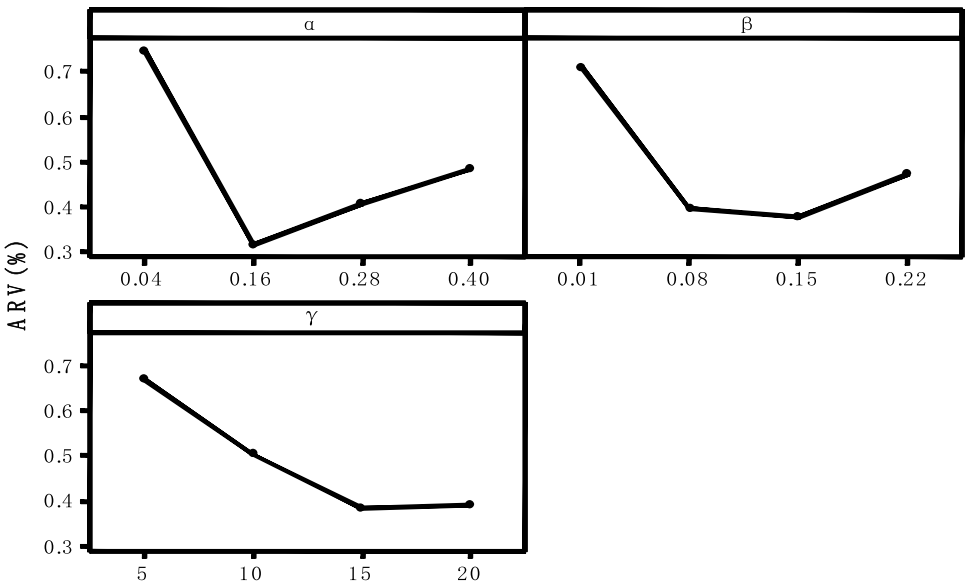

Fig. 12 Main effect of each parameter

Table 7 Comparisons of CCA with CCAnoBRR, SSEAnoLS-EDA-FAFP and SSEA-EDAnoLS-FAFP

W	N	RPD			
		CCA	CCAnoBRR	SSEAnoLS-EDA-FAFP	SSEA-EDAnoLS-FAFP
5	10	**0.00**	9.17	**0.00**	**0.00**
	20	**0.00**	5.84	0.32	0.08
	30	**0.00**	6.3	0.05	0.04
	40	0.12	6.28	**0.00**	0.09
	50	**0.00**	6.89	2.2	2.17
10	10	**0.00**	15.2	0.03	**0.00**
	20	**0.00**	13.21	0.1	0.02
	30	0.04	9.72	**0.00**	0.43
	40	**0.00**	9.41	0.72	0.54
	50	**0.00**	11.08	0.85	0.91
20	10	**0.00**	5.11	0.23	0.14
	20	**0.00**	5.03	1.28	0.97
	30	**0.00**	4.28	0.43	0.5
	40	0.07	3.53	**0.00**	0.19
	50	**0.00**	3.67	0.11	0.18
30	10	**0.00**	1.79	0.7	0.45
	20	**0.00**	1.71	0.3	0.23
	30	**0.00**	2.14	0.2	0.06
	40	**0.00**	2.84	0.63	0.46
	50	**0.00**	2.63	0.07	0.2

Note: The best value is indicated in bold

$\bar{C}_t$. The best average objective value $\bar{C}_{best}$ among 16 combinations is used as a reference value. The relative percentage deviation (RPD) of each combination t is calculated as follows:

$$\mathrm{RPD}_t = \left(\bar{C}_t - \bar{C}_{best}\right)/\bar{C}_{best} \times 100. \quad (12)$$

Thus, the average of RPD_t s on 20 instances is obtained as the average response value (ARV) for combination t. Clearly, a smaller value of ARV means a better parameter combination. The orthogonal array and the ARVs are listed in Table 6. The main effect of each parameter is shown in Fig. 12.

From the results, it can be seen that the rate of elite solutions α is the most influential parameter, while β and γ rank the second and the last. A small value of α will result in insufficient learning from elite individuals in each generation, while a larger value will lead to slow convergence. The value of learning rate β controls the learning speed of EDA. A large value gains a faster learning speed. However, an overlarge value will lead to a premature convergence. A moderate value of γ is beneficial to balance the exploitation intensification and the time consumption. According to above investigation, we choose the following setting for further tests: $\alpha = 1.06$, $\beta = 0.15$, $\gamma = 15$.

Effect of BRR and local search

To demonstrate the effectiveness of the BBR during the iteration of CC mechanism, we compare the CCA (i.e., SSEA-EDA-FAFP) with BRR to the CCA without BRR (CCAnoBRR). The process of CCAnoBRR to solve the mismatch of *seru* number between partial solution-HSFs and a partial solution-SS is as follows: allocate sth batch processing sequence ($s = 1,2,\ldots,NSS$) to jth *seru* ($j = (s + NSF - 1)\%NSF + 1$), where NSS is the *seru* number of *seru* scheduling, NSF is the *seru* number of *seru* formation, and % represents the modulus operator. To evaluate the effectiveness of local search during the evolution process of SSEA

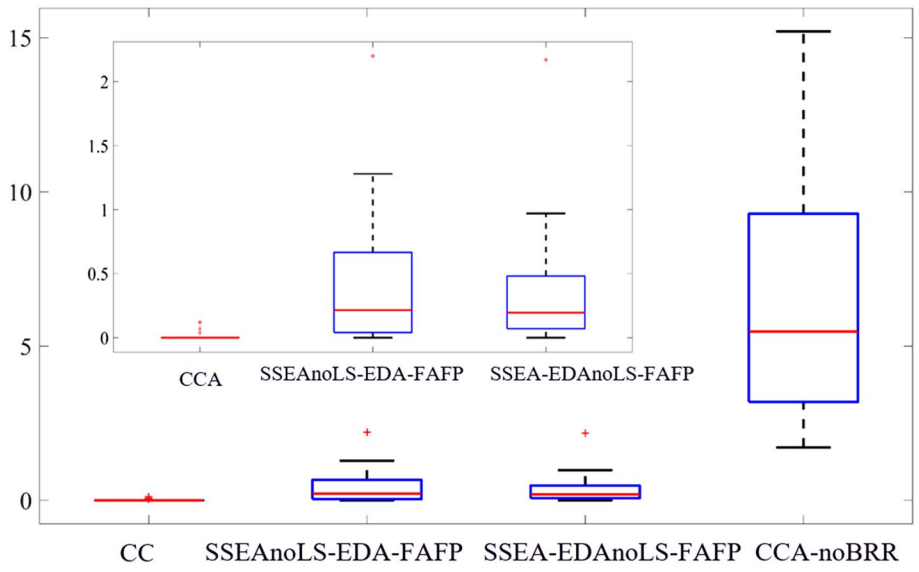

Fig. 13 Boxplot of the RPD with or without BRR and local search

Table 8 Comparisons between CCA and other algorithms

W	*N*	CCA		GA-EDA-FAFP		GA-GA-FAFP		Original flow line
		Makespan	Imp	Makespan	Imp	Makespan	Imp	Makespan
5	10	1091.1	**7.53**	1091.1	7.53	1189.6	0	1180
	20	2127.66	**11.61**	2132.7	11.4	2245.78	6.7	2407
	30	3072.92	12.82	3072.48	**12.84**	3229.8	8.37	3525
	40	3917.9	**13.61**	3917.9	**13.61**	4103.29	9.52	4535
	50	4828.74	**15.85**	4932	14.05	5160.35	10.07	5738
10	10	1101.49	**15.47**	1101.49	**15.47**	1270.92	2.46	1303
	20	2156.79	18.8	2155.03	**18.86**	2384.78	10.21	2656
	30	3118.05	**19.97**	3127.44	19.73	3385.48	13.1	3896
	40	3962.86	**21.01**	3977.91	20.71	4312.81	14.04	5017
	50	4992.57	**21.3**	4996.7	21.24	5369.06	15.37	6344
20	10	1326.59	11.62	1316.44	**12.3**	1397.87	6.87	1501
	20	2592.6	**15.63**	2607.63	15.14	2723.09	11.39	3073
	30	3768.47	**16.59**	3771.03	16.53	3907.97	13.5	4518
	40	4836.83	17.22	4813.06	**17.63**	4959.99	15.11	5843
	50	6074.17	**17.69**	6148.9	16.68	6237.37	15.48	7380
30	10	1546.85	**9.91**	1537.84	9.38	1586.01	7.09	1707
	20	3050.25	12.2	3034.01	**12.67**	3114.1	10.36	3474
	30	4412.21	**13.76**	4425.87	13.49	4516.06	11.73	5116
	40	5626.29	**15.25**	5781.92	12.91	5772.15	13.06	6639
	50	7102.39	**15.22**	7221.51	13.79	7308.78	12.75	8377

Note: The best value is indicated in bold

and EDA, we compare the CCA to CCA without local search in SSEA (SSEAnoLS-EDA-FAFP) and CCA without local search in EDA (SSEA-EDAnoLS- FAFP). In SSEAnoLS-EDA-FAFP, if crossover operator 2 fails to achieve improvement, search the next sub-space without applying local search to the best individual in current sub-space. In SSEA-EDAnoLS-FAFP, the elite individuals will be directly used to update the probability model of *seru* scheduling without further improvement by the local search.

Each instance is run ten times independently to obtain the average objective value $\bar{C}_v$. The best average objective value C_b among four algorithms is used as the reference objective value. For each run, the following relative percentage deviation (RPD) of makespan is calculated. The RPD values of different algorithms are shown in Table 7. In addition, Fig. 13 shows the boxplot of all RPD values obtained by the four algorithms.

$$\mathrm{RPD} = \left(\bar{C}_v - \bar{C}_b\right)/\bar{C}_b \times 100 \tag{13}$$

From Table 7, it can be seen that the RPDs obtained by CCA are superior to the other three algorithms on most instances. The CCA is able to obtain the best solutions consistently on 17 out of 20 instances. From Fig. 13, it can be seen that the algorithms with the BRR are much better than that without the BRR. Compared with the SSEAnoLS-EDA-FAFP and the SSEA-EDAnoLS-FAFP, the results obtained by CCA are better. Besides, the variation of RPD value obtained by CCA is smaller than that of the other algorithms. Therefore, the BRR and local search are effective in achieving better and more robust results.

Comparisons with other algorithms

According to the results in [11–13, 15–17], GA is of good performance in dealing with different scheduling problems in pure *seru*-system. To test the performance of CCA (i.e., SSEA-EDA-FAFP), the existing genetic algorithm for pure *seru*-system is employed for comparison. To be specific, the recent GA in [15] is adopted for *seru* formation. Thus, we construct two algorithms with CC mechanism, i.e., GA-EDA-FAFP and GA-GA-FAFP. To be specific, in GA-EDA-FAFP, the GA with local search in [15] is used to evolve the population of hybrid *seru* formation, and the remaining settings are the same as CCA. In GA-GA-FAFP, a GA with local search is developed to evolve the population of *seru* scheduling. The representation of *seru* scheduling and the local search are the same as those in 3.2.1, and the evolution process includes tournament selection, order crossover [43] and mutation by swapping two gene positions. The rest settings of GA-GA-FAFP are the same as GA-EDA-FAFP. Similarly, we run each algorithm ten times independently on each instance and record the best solution as the final

solution. The following *Imp* is used to measure the improvement by a certain algorithm after dismantling the original flow line into the hybrid *seru*-system.

$$Imp = \left(C_A^* - C^*\right)/C_A^* \times 100, \tag{14}$$

where C^* denotes the makespan of the hybrid *seru*-system obtained by an algorithm, and C_A^* denotes the makespan of the original flow line, which is calculated based on Eq. (6).

Clearly, a larger value of *Imp* means a better performance achieved by the algorithm. The results of different algorithms are given in Table 8. It can be seen that the CCA obtains the best solutions among three algorithms on 15 out of 20 instances, while GA-EDA-FAFP obtains best objective values only on 8 instances. Comparatively, the improvement by GA-GA-FAFP is the worst among all. Therefore, it is concluded that CCA has a better performance than other algorithms in solving the HSSOP.

Conclusion

This paper addresses a complex hybrid *seru*-system scheduling problem with the makespan minimization criterion, which contains several strongly coupled decision-making processes. The problem is decomposed in three sub-problems: hybrid *seru* formation, *seru* scheduling and flow line scheduling. For each sub-problem, suitable optimization algorithm is designed according to the characteristics of the sub-problems, and different local search procedures are designed for different sub-problems to enhance exploitation. The sub-algorithms work in an integrated framework with a cooperative coevolution mechanism to explore good solutions and a batch reassign rule to guarantee legal solutions. Extensive comparisons show that the designed local search and batch reassign rule are effective. It also shows that the proposed algorithm is superior to the existing algorithms for *seru*-system scheduling problem.

The future work will focus on the adaptive cooperative optimization of different sub-problems. Learning-based mechanism will be considered in designing more powerful solution algorithms. It is also interesting to generalize the work to the multi-objective optimization of hybrid *seru*-system by considering total tardiness, labor time, and energy consumption [44].

Acknowledgement This study is supported by the National Science Fund for DistinguishedYoung Scholars of China (No. 61525304) and the National Natural Science Foundation ofChina (No. 61873328).

Declarations

Conflict of interest The authors declare that we have no conflict of interest.

References

1. Yin Y, Stecke KE, Swink M, Kaku I (2017) Lessons from *seru* production on manufacturing competitively in a high cost environment. J Oper Manag 49:67–76
2. Liu C, Stecke KE, Lian J, Yin Y (2014) An implementation framework for *seru* production. Int Trans Oper Res 21(1):1–19
3. Yin Y, Stecke KE, Li D (2017) The evolution of production systems from Industry 2.0 through Industry 4.0. Int J Prod Res 56(1–2):848–861
4. Takeuchi N (2006) *Seru* production system (*Seru* Seisan, in Japanese). JMA Management Center Tokyo
5. Stecke KE, Yin Y, Kaku I (2012) *Seru*: The organizational extension of JIT for a super-talent factory. Int J Strateg Decis Sci 3(1):105–118
6. Sakazume Y (2005) Is Japanese cell manufacturing a new system?: A comparative study between Japanese cell manufacturing and cell manufacturing. J Japan Indust Manag Associa 12:89–94
7. Yin Y, Kaku I, Stecke KE (2008) The evolution of *seru* production systems throughout Canon. Operations Management Education Review 2:35–39
8. Roth A, Singhal J, Singhal K, Tang CS (2016) Knowledge creation and dissemination in operations and supply chain management. Prod Oper Manag 25(9):1473–1488
9. Kaku I, Murase Y, Yin Y (2008) A study on human-task-related performances in converting conveyor assembly line to cellular manufacturing. Eur J Ind Eng 2(1):17–34
10. Kaku I, Gong J, Tang J, Yin Y (2009) Modeling and numerical analysis of line-cell conversion problems. Int J Product Res 47(8):2055–2078
11. Yu Y, Gong J, Tang J, Yin Y, Kaku I (2012) How to do assembly line-cell conversion? A discussion based on factor analysis of system performance improvements. Int J Prod Res 50(18):5259–5280
12. Yu Y, Tang J, Sun W, Yin Y, Kaku I (2013) Reducing worker(s) by converting assembly line into a pure cell system. Int J Prod Econ 145(2):799–806
13. Yu Y, Tang J, Gong J, Yin Y, Kaku I (2014) Mathematical analysis and solutions for multi-objective line-cell conversion problem. Eur J Oper Res 236(2):774–786
14. Liu C, Dang F, Li W, Lian J, Evans S, Yin Y (2015) Production planning of multi-stage multi-option *seru* production systems with sustainable measures. J Clean Prod 105:285–299
15. Sun W, Wu Y, Lou Q, Yu Y (2019) A cooperative coevolution algorithm for the *seru* production with the minimizing makespan. IEEE Access 7:5662–5670
16. Yılmaz ÖF (2019) Operational strategies for *seru* production system: a bi-objective optimisation model and solution methods. Int J Prod Res 58(11):3195–3219
17 Yılmaz ÖF (2020) Attaining flexibility in *seru* production system by means of Shojinka: An optimization model and solution approaches. Comput Oper Res 119:104917

18. Zhang Z, Shao L, Yin Y (2020) PSO-based algorithm for solving lot splitting in unbalanced *seru* production system. Int J Ind Syst Eng 35(4):433–450
19. Kaku I, Gong J, Tang J, Yin Y (2008) A mathematical model for converting conveyor assembly line to cellular manufacturing. Ind Eng Manag Syst 7(2):160–170
20. Liu C, Li W, Lian J, Yin Y (2012) Reconfiguration of assembly systems: from conveyor assembly line to *seru*s. J Manuf Syst 31(3):312–325
21. Yu Y, Sun W, Tang J, Wang J (2017) Line-hybrid *seru*-system conversion: models, complexities, properties, solutions and insights. Comput Ind Eng 103(1):282–299
22. Potter M, Jong K (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29
23. Yang Z, Tang K, Yao X (2008) Large scale evolutionary optimization using cooperative coevolution. Inf Sci 178(15):2985–2999
24. Omidvar MN, Li X, Xin Y (2011) Smart use of computational resources based on contribution for cooperative co-evolutionary algorithms. In: 13th Annual Genetic and Evolutionary Computation Conference (GECCO), Dublin, Ireland
25. Omidvar MN, Li X, Mei Y, Yao X (2014) Cooperative co-evolution with differential grouping for large scale optimization. IEEE Trans Evol Comput 18(3):378–393
26. Shang R, Wang Y, Wang J, Jiao L, Wang S, Qi L (2014) A multi-population cooperative coevolutionary algorithm for multi-objective capacitated arc routing problem. Inf Sci 277:609–642
27. Trunfio GA, Topa P, Wąs J (2016) A new algorithm for adapting the configuration of subcomponents in large-scale optimization with cooperative coevolution. Inf Sci 372:773–795
28. Hu XM, He FL, Chen WN, Zhang J (2017) Cooperation coevolution with fast interdependency identification for large scale optimization. Inf Sci 381:142–160
29. Yang Q, Chen WN, Zhang J (2018) Evolution consistency based decomposition for cooperative coevolution. IEEE Access 6:51084–51097
30. Li L, Fang W, Mei Y, Wang Q (2020) Cooperative coevolution for large-scale global optimization based on fuzzy decomposition. Soft Comput 25(5):3593–3608
31. Yang M, Zhou A, Li C, Yao X (2021) An efficient recursive differential grouping for large-scale continuous problems. IEEE Trans Evol Comput 25(1):159–171
32. Panait L (2010) Theoretical convergence guarantees for cooperative coevolutionary algorithms. Evol Comput 18(4):581–615
33. Larrañaga P, Lozano J (2001) Estimation of distribution algorithms: A new tool for evolutionary computation. Springer Science & Business Media, Berlin
34. Jarboui B, Eddaly M, Siarry P (2009) An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. Comput Oper Res 36(9):2638–2646
35. Wang S, Wang L, Liu M, Xu Y (2013) An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. Int J Prod Econ 145(1):387–396
36. Ceberio J, Irurozki E, Mendiburu A (2014) A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. IEEE Trans on Evol Comput 18(2):286–300
37. Du Y, Li J, Luo C, Meng L (2021) A hybrid estimation of distribution algorithm for distributed flexible job shop scheduling with crane transportations. Swarm Evol Comput 62:100861
38. Sun B, Wang L (2021) An estimation of distribution algorithm with branch-and-bound based knowledge for robotic assembly line balancing. Complex Intell Syst 7(3):1–14
39. Cheng R, He C, Jin Y, Yao X (2018) Model-based evolutionary algorithms: a short survey. Complex Intell Syst 4:283–292
40. Gao K, Cao Z, Zhang L, Chen Z, Pan Q (2019) A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. IEEE-CAA J Automatica Sin 6(4):904–916
41. Gao K, He Z, Huang Y, Duan P, Suganthan P (2020) A survey on meta-heuristics for solving disassembly line balancing, planning and scheduling problems in remanufacturing. Swarm Evol Comput 57:100719
42. Montgomery DC (2017) Design and analysis of experiments. John wiley & sons, New York
43. Davis L (1985) Applying adaptive algorithms to epistatic domains. Proceedings of the International Joint Conference on Artificial Intelligence 85:162–164
44. Gao KZ, Huang Y, Sadollah A, Wang L (2020) A review of energy-efficient scheduling in intelligent production systems. Complex Intell Syst 6(2):237–249