Journal of Ambient Intelligence and Humanized Computing (2023) 14:11123–11138
https://doi.org/10.1007/s12652-022-04392-w

ORIGINAL RESEARCH

Reinforcement learning iterated greedy algorithm for distributed assembly permutation flowshop scheduling problems

Kuo-Ching Ying[1] · **Shih-Wei Lin**[2,3,4]

Received: 27 January 2022 / Accepted: 30 August 2022 / Published online: 19 September 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
Integrating component and final assembly production plans is critical to optimizing the global supply chain production system. This research extends the distributed assembly permutation flowshop scheduling problem to consider unrelated assembly machines and sequence-dependent setup times. A mixed-integer linear programming (MILP) model and a novel metaheuristic algorithm, called the Reinforcement Learning Iterated Greedy (RLIG) algorithm, are proposed to minimize the makespan of this problem. The RLIG algorithm applies a multi-seed hill-climbing strategy and an $\varepsilon-$greedy selection strategy that can exploit and explore the existing solutions to find the best solutions for the addressed problem. The computational results, as based on extensive benchmark instances, show that the proposed RLIG algorithm is better than the MILP model at solving tiny-size problems. In solving the small- and large-size test instances, RLIG significantly outperforms the traditional iterated greedy algorithm. The main contribution of this work is to provide a highly effective and efficient approach to solving this novel scheduling problem.

Keywords Scheduling · Distributed manufacturing · Distributed assembly permutation flowshop · Sequence-dependent setup times · Reinforcement learning algorithm

1 Introduction

The energetic applications of Industry 4.0 have attracted the attention of researchers in various fields. Recently, booming global demand and the rapid development of advanced communication technologies, such as 5G and the Internet of Things, have driven the development of the distributed manufacturing system (DMS) and accelerated the pace of production globalization. The application of DMS can effectively manage complex production decisions for the supply chain system to reduce production costs, shorten response times, increase production flexibility, improve product quality, and reduce management risk (Renna 2012). Production planning must integrate the production schedules of each member of the supply chain to enhance the performance of the DMS (Rossit et al. 2019). The distributed scheduling problem (DSP) is a vital issue in the globalization of production scheduling, which can help facilitate autonomous manufacturing and take a step towards Industry 4.0. Therefore, DSP has received increasing attention from researchers and practitioners (Lin and Ying 2013; Ying and Lin 2017; Cheng et al. 2019; Pourhejazy et al. 2021) and is becoming one of the most critical developments in the field of production scheduling.

DSP was first proposed by Jia et al. (2002), who presented a web-based scheduling system for a distributed manufacturing environment. Since then, DSP has been one of the prime examples of growing recognition, motivated by the fact that advanced communication systems can be used to make efficient production decisions in various DMSs (Ying et al. 2022). Many pioneering works on various DSPs have been proposed and become widely used in the automotive, consumer electronics, and other globalized manufacturing

✉ Shih-Wei Lin
swlin@mail.cgu.edu.tw

Kuo-Ching Ying
kcying@ntut.edu.tw

1 Department of Industrial Engineering and Management, National Taipei University of Technology, Taipei, Taiwan

2 Department of Information Management, Chang Gung University, Taoyuan, Taiwan

3 Department of Industrial Engineering and Management, Ming Chi University of Technology, Taipei, Taiwan

4 Department of Emergency Medicine, Keelung Chang Gung Memorial Hospital, Keelung 204, Taiwan

industries; for example, Chan et al. (2005, 2006) presented genetic algorithms for DSPs in five multi-factory models and a flexible manufacturing system environment; Naderi and Ruiz (2010) proposed the distributed permutation flowshop scheduling problem (DPFSP) and proposed six mixed integer linear programming (MILP) models, two simple factory assignment rules, and 14 algorithms to minimize the makespan among factories. The research boundaries of DSP were extended to distributed jobshop scheduling problems by Jia et al. (2007), De Giovanni and Pezzella (2010), and Naderi and Azab (2014), and to the distributed parallel-machine scheduling problem by Behnamian (2014). Based on the seminal study by Naderi and Ruiz (2010), some extensions of the basic DPFSP setting have been proposed for other modern industrial manufacturing systems. Among these pioneering research works, the distributed assembly permutation flowshop scheduling problem with flexible assembly (DAPFSP-FA, Zhang et al. 2018) is one of the most significant problems.

Sequence-dependent setup times are the essential production factors considered in many practical DMSs. Moreover, unrelated parallel machines have more general settings than the uniform parallel machines used in the assembly stage of DAPFSP-FA. However, to the best of our knowledge, there has been no research on the DAPFSP-FA with unrelated assembly machines and sequence-dependent setup times, which is a significant production setting in a variety of industries. These facts inspired our research to extend DAPFSP-FA to include unrelated assembly machines and sequence-dependent setup times for a more general and practical globalized production scheduling mode. As shown in Fig. 1, the DMS of DAPFSP-FASDST has the two stages of production and assembly. The first stage (production) consists of a distributed production system with g homogeneous factories/shops capable of handling all jobs/components, and each factory/shop is a permutation flowshop equipped with m machines disposed of in series. The second stage (flexible assembly) is equipped with q unrelated parallel machines that are functionally equivalent but have different processing speeds for different products. In line with a predefined assembly plan,n components/jobs are assembled into t products, where each component/job belongs to only one product. Each component/job $j(j \in \{1, 2, ..., n\})$ must be assigned and produced in one of the factories, and the processing time of component/job $j(j \in \{1, 2, ..., n\})$ on the production machine PM_i $(i \in \{1, 2, ..., m\})$ is $P_{j,i}$. The processing sequence of the components/jobs assigned to the same factory/shop remains unchanged on each machine. A product can be assembled when all of its jobs/components have been completed in the production stage. Each product p $(p = 1, 2, ..., t)$ requires a single assembly operation on any one of the assembly machines, denoted as AM_s $(s = 1, 2, ..., q)$, with corresponding assembly time $AP_{p,s}$, depending on the assembly machine to which it is assigned. The speeds of the assembly machines are not in constant proportion to each other. Before processing each component/job, a job sequence-dependent setup time,$S_{j',j,i}$ $(j' = 0, 1, ..., n;$ $j = 1, 2, ..., n; j' \neq j; i = 1, 2, ..., m)$, is incurred when job j is processed immediately after job j' on production machine i. Similarly, the machines at the assembly stage have product sequence-dependent setup times, denoted as $AS_{p',p,s}$ $(p' = 0, 1, ..., t; p = 1, 2, ..., t; p' \neq p; s = 1, 2, ..., q)$, if product p is assembled on assembly machine s immediately after product p'. The objective is to generate the corresponding production sequence of jobs/components in each factory/shop and the corresponding assembly sequence in each assembly machine, in order to minimize the makespan.

Using the three-field notation proposed by Framinan et al. (2019), the addressed DAPFSP-FASDST is signified as $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$, where $DF_m \rightarrow R_m$ denotes that the DMS has two stages. The first stage, i.e., the production stage, consists of distributed flowshops, and the second stage, i.e., the flexible assembly stage, is equipped with unrelated parallel machines; *prmu* designates that the shop type of each component production factory/shop in the production stage is a permutation flowshop; ST_{sd} represents that sequence-dependent setup times are considered in both the production and assembly stages;

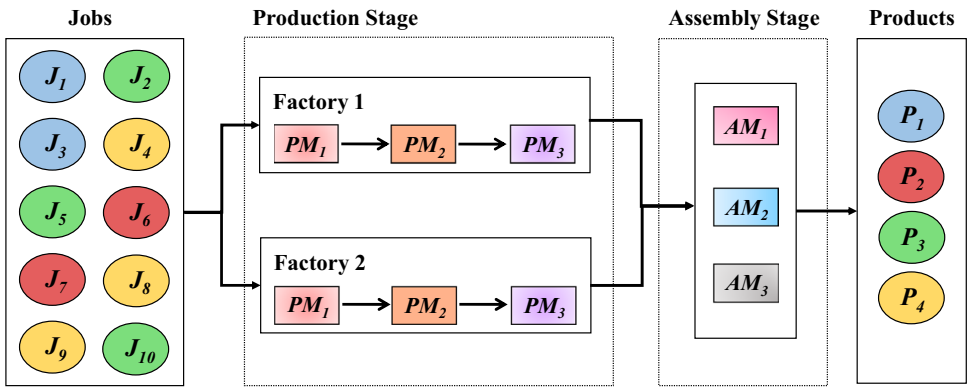


Fig. 1 Schematic diagram of DAPFSP-FASDST

and $C_{\max}$ indicates that the objective of the schedule is to minimize the makespan. Without the production stage, the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem is reduced to the solution of the unrelated parallel machine scheduling problem with sequence-dependent setup times, which is an *NP*-hard problem in the strong sense (Ying et al. 2012). Therefore, we can easily conclude that the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem is also an *NP*-hard problem in the strong sense.

In addition, the following basic assumptions and limitations are asserted for the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem addressed in this study:

- Once a component/product is assigned to a factory/machine, the planner cannot interrupt the processing/assembly operation of a component/product or transfer it to other factories/machines.
- Each component/product can only be processed on one production/assembly machine at a time, and each production/assembly machine can handle only one component/product at a time. Once the process has begun on a production/assembly machine, it cannot be interrupted until it is completed.
- The ready times of all components/jobs are assumed to be zero; i.e., the components/jobs are available at the beginning of the planning horizon.
- The component/product orders are independent of each other; therefore, the sequence of the component/product does not affect the production/assembly time.
- The production/assembly machine is available throughout the planning horizon, and there are no maintenance issues, failures, or other issues throughout the planning horizon.
- The number of factories/machines, the number of components/products, the production/assembly times, and the job/product sequence-dependent setup times are given by deterministic nonnegative integers.
- Excluding the trivial cases, the number of components/jobs is larger than the number of production/assembly machines.

Given the originality of the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem, a MILP model and a novel IG-based metaheuristic, called the reinforcement learning iterated greedy (RLIG) algorithm, are presented in this study to minimize the makespan. The main novelty and contributions of this study are to fill the research gap on the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem and offer a practical method for the promotion of real-world applications in a variety of industries.

2 Literature review

The DPFSP has been extended to various production settings, such as DMSs with heterogeneous factories (Li et al. 2020a, 2020b; Meng and Pan 2021), group scheduling (Pan et al. 2022), hybrid flowshops (Ying and Lin 2018; Li et al. 2020c; Lei and Wang 2020), and integrated assembly-production flowshops (Lin 2018; Wu et al. 2018, 2019; Lei et al. 2021). In addition, DPFSPs have been integrated with a number of practical features that facilitate their application in the real world; blocking conditions (Zhang et al. 2018; Li et al. 2019; Zhao et al. 2020), limited buffer constraints (Zhang and Xing 2019), no-wait (Lin and Ying 2016; Komaki and Malakooti 2017; Li et al. 2020a), no-idle (Ying et al. 2017; Cheng et al. 2019; Zhao et al. 2021), customer order-priority (Meng et al. 2019), time window constraints (Jing et al. 2020), machine-breakdowns (Wang et al. 2016), and preventive maintenance (Mao et al. 2021) are notable examples.

This section reviews the most important literature regarding the two well-known branches of the DPFSP, *i.e.*, the distributed assembly permutation flowshop scheduling problem (DAPFSP, Hatami et al. 2013) and the distributed assembly permutation flowshop scheduling problem with flexible assembly (DAPFSP-FA, Zhang et al. 2018). Regarding the first branch of DPFSPs, Hatami et al. (2013) first added an assembly stage to DPFSP to form DAPFSP, which has the two stages of production and assembly. In the first stage (production), all jobs/components are assigned to a distributed permutation flowshop manufacturing system, and upon completing all the jobs/components of a particular product, product assembly is conducted in the second stage (assembly) using an assembly machine. This setting usually occurs when different manufacturers supply the components and an assembly center in the parent company processes the final products. Hatami et al. (2013) proposed three construction heuristics and a variable neighborhood descent (VND) metaheuristic to minimize the makespan of DAPFSPs. The following year, Hatami et al. (2014) extended the DAPFSP problem to include sequence-dependent setup time constraints and presented two constructive heuristics for solving the distributed assembly permutation flowshop scheduling problem with sequence-dependent setup times. After that, Hatami et al. (2015) proposed two construction heuristic algorithms and an iterated greedy (IG)-based algorithm to solve this problem, and their computational results revealed that the IG-based algorithm performed better than the VND algorithm. To further improve the solution quality of the algorithms presented by Hatami et al. (2013), Li et al. (2015) presented a genetic algorithm (GA), Lin and Zhang (2016) presented a hybrid biogeography-based

optimization (HBBO) algorithm, and Wang and Wang (2016) proposed an estimation of the distribution algorithm-based memetic algorithm (EDAMA) to solve DAPFSPs. Their experimental results showed that GA is superior to the three construction heuristic algorithms of Hatami et al. (2013), and HBBO and EDAMA outperform VND. Following these studies, Lin et al. (2017) presented a backtracking search hyper-heuristic (BS-HH) algorithm, and their results showed that BS-HH provides better solution quality and a shorter computation time than VND, HBBO, and EDAMA. Then, Pan et al. (2019) presented two VND metaheuristics and an IG-based approach for minimizing the makespan of DAPFSPs, and the experimental results of 810 benchmark test problems confirmed that their approaches significantly outperform VND, GA, HBBO, EDAMA, BS-HH, and other existing algorithms. Huang et al. (2021) considered the DAPFSP problem with the criterion of total flow time and proposed an IG-based algorithm (gIGA) to solve it, and their computational results showed that gIGA is superior to the seven algorithms available in the literature. These breakthrough studies on DAPFSPs could effectively and efficiently coordinate the production components and final assembly scheduling to optimize the global supply chain system. More recently, Pourhejazy et al. (2022) extended the limited literature on DAPFSPs to the distributed two-stage production-assembly scheduling problems, proposed a meta-Lamarckian-based iterated greedy algorithm and compared it with the current-best-performing algorithm in the literature. The computational results revealed that this approach is a strong benchmark algorithm for solving this problem. Although this setting is widely used in globalized production, very little attention has been paid to this branch of the problem.

Regarding the second branch of DPFSPs, Zhang et al. (2018) first introduced DAPFSP-FA, which extends the assembly stage of DAPFSP from a single machine to multiple machines. In the first stage (production), all jobs/components are first assigned to a distributed permutation flowshop manufacturing system. As existing globalization production systems usually require greater flexibility during the assembly stage to accommodate the paradigms of concurrent and mass production, the product assembly work is carried out directly after the production stage with a uniform parallel machine in the second stage (assembly). DAPFSP-FAs are widely used in many industries, especially globalized manufacturing companies, such as automotive and consumer electronics, which have many outsourced components. Zhang et al. (2018) proposed two metaheuristics, the hybrid variable neighborhood search (HVNS) and hybrid particle swarm optimization (HPSO), for minimizing the makespan on the DAPFSP with flexible assembly and sequence-independent setup times, and the extensive numerical results revealed that the performance of HPSO is superior to that of HVNS. Subsequently, Zhang et al. (2020) further investigated this new generation of scheduling problems and proposed a MILP model and a memetic algorithm to solve the same problem more effectively than HPSO. More recently, Yang and Xu (2021) proposed the distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery to coordinate the supply chain scheduling of production and transportation, and presented four heuristics algorithms, one VND algorithm, and two IG-based approaches (IG_desP and IG_desJ) to minimize the total cost of delivery and tardiness. The computational results revealed that IG_desJ provides higher quality solutions among the approaches. Pourhejazy et al. (2021) developed another IG-based algorithm, named iterated greedy with a product destruction mechanism (IG_{PD}), to solve the DAPFSP with flexible assembly and sequence-independent setup times, and the computational results showed that IG_{PD} significantly outperforms the HPSO and HVNS methods. More recently, Ying et al. (2022) extended DAPFSP to consider flexible assembly and sequence-independent setup times in a supply chain-like setting and proposed a MILP model, a constructive heuristic, and customized metaheuristic algorithms to solve this problem. The computational results showed that the proposed algorithms outperformed the best performing algorithms and yielded the best-known solutions in almost all benchmark instances. As only a handful of articles have been published, and all of them were submitted in the last five years, further studies are needed to support the development of this new branch of DPFSPs.

3 Methodology

3.1 MILP model

In this sub-section, a novel MILP model for solving small-size $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problems was formulated. For this purpose, the various indices, parameters, and decision variables are defined, as follows.

3.1.1 Indices

j', j: Job, $j \in \{1, 2, ..., n\}$ and $j' \in \{0, 1, ..., n\}$, where 0 represents the dummy job.

i: Machine, $i \in \{1, 2, ..., m\}$ at the production stage.

f: Factory/Shop, $f \in \{1, 2, ..., g\}$ at the production stage.

p, p': Product, $p, p' \in \{0, 1, 2, ..., t\}$, where 0 represents the dummy product.

s: Machine, $s \in \{1, 2, ..., q\}$ at the assembly stage.

3.1.2 Parameters

n: Number of jobs to be processed.

m: Number of available production machines.

g: Number of production factories.

t: Number of products.

q: Number of available assembly machines ($q \le t$).

$P_{j,i}$: Processing time of job j on the production machine i.

$S_{j',j,i}$: Setup time of job j on the production machine i when it is processed after job j'.

$AP_{p,s}$: Assembly time of product p on the assembly machine s.

$AS_{p',p,s}$: Setup time of product p on the assembly machine s when it is processed after product p'.

G_p: The jobs associated with product p.

M: A sufficiently large constant.

3.1.3 Decision variables

$C_{k,i,f}$: Completion time of the job in position k on machine i at factory/shop f.

$X_{j,k,f}$: Binary variable that takes value 1 if job j occupies position k in factory/shop f, and 0 otherwise.

$Y_{p',p,s}$: Binary variable; = 1 if product p is assembled after product p' on assembly machine s; = 0, otherwise.

$Z_{p,s}$: Binary variable; = 1 if product p is dispatched to assembly machine s; = 0, otherwise.

$AR_{p,s}$: Starting time of assembling the product p on the assembly machine s.

$AC_{p,s}$: Completion time of assembling product p on the assembly machine s.

This study proposed a position-based MILP model for the $DF_m \to R_m|prmu, ST_{sd}|C_{\max}$ problem, as follows.

Minimize C_{max}

subject to

$$\sum_{k=1}^{n}\sum_{f=1}^{g} X_{j,k,f} = 1,\ j = 0,\ 1,\ ...,\ n \tag{1}$$

$$\sum_{j=1}^{n} X_{j,k,f} \le 1,\ k = 1,\ 2,\ ...,\ n; f = 1,\ 2,\ ...,\ g \tag{2}$$

$$C_{1,1,f} = \sum_{j=1}^{n} X_{j,1,f} \cdot (p_{j,1} + S_{0,j,1}),\ f = 1,\ 2,\ ...,\ g \tag{3}$$

$$C_{k,i,f} \ge C_{k,i-1,f} + \sum_{j=1}^{n} X_{j,k,f} \cdot P_{j,i} + (\sum_{j=1}^{n} X_{j,k,f} - 1) \cdot \mathrm{M},\quad k = 1,\ 2,\ ...,\ n; i = 2,\ 3,\ ...,\ m; f = 1,\ 2,\ ...,\ g \tag{4}$$

$$C_{k,i,f} \ge C_{k-1,i,f} + P_{j,i} + S_{j',j,i} + (X_{j,k,f} + X_{j',k-1,f} - 2) \cdot \mathrm{M},\quad k = 1,2,\ ...,\ n; i = 1,\ 2,\ ...,\ m; f = 1,\ 2,\ ...,\ g;\quad j' = 0,\ 1,\ ...,\ n;, j = 1,\ 2,\ ...,\ n; j \ne j' \tag{5}$$

$$\sum_{j=1}^{n} X_{j,k-1,f} \ge \sum_{j=1}^{n} X_{j,k,f}, k = 1,2,\ ...,\ n; f = 1,\ 2,\ ...,\ g \tag{6}$$

$$\sum_{\substack{p'=0\\p'\ne p}}^{t} Y_{p',p,s} - \sum_{\substack{p'=0\\p'\ne p}}^{t} Y_{p,p',s} = 0;\ p = 1,2,\cdots,n; s = 1,2,\cdots,q, \tag{7}$$

$$\sum_{s=1}^{q}\sum_{p'=0,p'\ne p}^{t} Y_{p',p,s} = 1\,; p = 1,\ ...,\ t \tag{8}$$

$$\sum_{s=1}^{q}\sum_{p=1,p'\ne p}^{t} Y_{p',p,s} \le 1\,; p' = 1,\ 2,\ ...,\ t \tag{9}$$

$$\sum_{p=1}^{t} Y_{0,p,s} = 1,\ s = 1,\ 2,\ ...,\ q \tag{10}$$

$$Y_{p',p,s} + Y_{p,p',s} \le 1\,,\ p' = 1,\ 2,\ ...,\ t-1; p > p'; s = 1,\ 2,\ ...,\ q \tag{11}$$

$$Z_{p,s} - \sum_{p'}^{t} Y_{p',p,s} = 0; p = 1,\cdots t; s = 1,\cdots q; \tag{12}$$

$$AR_{p,s} \ge C_{k,m,f} + (X_{j,k,f} + \mathrm{Z}_{p,s} - 2) \cdot \mathrm{M},\quad p = 1,\ ...,\ t; s = 1,\ 2,\ ...,\ q; j \in G_p;\quad k = 1,\ 2,\ ...,\ n;\ ; f = 1,\ 2,\ ...,\ g \tag{13}$$

$$AC_{p,s} \ge AR_{p,\mathrm{s}} + AP_{p,s} + (\mathrm{Z}_{p,s} - 1) \cdot \mathrm{M},\quad s = 1,\ 2,\ ...,\ q; p = 0,\ 1,\ ...,\ t \tag{14}$$

$$AC_{p,s} \ge AC_{p',s} + AS_{p',p,s} + AP_{p,s} + \left(Y_{p',p,s} - 1\right) \cdot \mathrm{M}\,,\quad s = 1,\ 2,\ ...,\ q; p,\ p' = 0,\ 1,\ ...,\ t; p \ne p' \tag{15}$$

$$C_{max} \ge AC_{p,s},\ p = 0,\ 1,\ ...,\ t; s = 1,\ 2,\ ...,\ q \tag{16}$$

$$C_{k,i,f} \ge 0\,; k = 1,\ 2,\ ...,\ n; i = 1,\ 2,\ ...,\ m; f = 1,\ 2,\ ...,\ g \tag{17}$$

$$C_{0,i,f} = 0\,; i = 1,\ 2,\ ...,\ m; f = 1,\ 2,\ ...,\ g \tag{18}$$

$$X_{j,k,f} \in \{0,1\},\ j = 0,\ 1,\ ...,\ n; i = 1,\ 2,\ ...,\ m; f = 1,\ 2,\ ...,\ g \tag{19}$$

$$X_{0,0,f} = 1, f = 1, 2, ..., g \tag{20}$$

$$Y_{l,k,s} \in \{0,1\} l, k = 0, 1, ..., t; l \neq k; s = 1, 2, ..., q \tag{21}$$

$$Z_{p,s} \in \{0,1\}, p = 0, 1, ..., t; s = 0, 1, ..., q; \tag{22}$$

$$C_{j,i} \geq 0; j = 0, 1, ..., n; i = 1, 2, ..., m \tag{23}$$

$$AR_{k,s} \geq 0; k = 0, 1, ..., t; s = 1, 2, ..., q \tag{24}$$

$$AC_{k,s} \geq 0; k = 0, 1, ..., t; s = 1, 2, ..., q \tag{25}$$

Constraint sets 1 and 2 restrict that a component/job can be assigned to only one position in a factory/shop, and a position in each factory/shop can be assigned one component/job at most. Constraint set 3 computes the completion times of the first component/job to be processed on the first machine in each factory/shop. Constraint sets 4 and 5 set the inequality relationships between the completion times of the components/jobs on successive machines and priorities, respectively. Constraint set 6 ensures that the positions of the assigned components/jobs in each factory/shop are consecutive. Constraint set 7 guarantees that the products are assembled on each machine sequentially. Constraint sets 8 to 11 specify that a product can have only one predecessor and at most one successor. Constraint set 12 establishes a link between the two binary decision variables of $Y_{p',p,s}$ and $Z_{p,s}$. Constraint set 13 defines the start times of the products at the assembly machines. Constraint set 14 defines the inequality relations between the completion times and start times of the products at the assembly machines. Constraint set 15 defines the inequality relation between the completion time of a product and its predecessor. Constraint set 16 calculates the maximum completion time of the products, i.e., the makespan. Finally, constraint sets 17 to 25 define the range for each decision variable. This MILP model has a large number of binary variables, which makes it superior in terms of both model size and computational complexity.

3.2 Proposed metaheuristic

In recent decades, various optimization algorithms have been developed to solve complex problems (Vanchinathan and Valluvan 2018; Vanchinathan and Selvaganesan 2021; Vanchinathan et al. 2021; Khalifa et al. 2021). IG-based algorithms (Pourhejazy et al. 2021, 2022) have gained recognition among researchers due to their effectiveness and efficiency in solving various DAPFSPs. Considering that the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem is *NP*-hard in the strong sense, this work proposed a novel IG-based metaheuristic called RLIG to solve this complex supply-chain-integrated scheduling problem.

The flowchart of the RLIG is shown in Fig. 2. First, a set of initial solutions $\mathbf{\Pi} = \{\Pi^1, \Pi^2, ..., \Pi^{N_{\max}}\}$ are generated using a specific constructive heuristic. Second, let Π_{best} be the best solution among $\{\Pi^1, \Pi^2, ..., \Pi^{N_{\max}}\}$ and $C_{\max}(\Pi_{best})$ is the objective function value of the best solution Π_{best}. For each iteration, a solution Π^N is chosen from $\{\Pi^1, \Pi^2, ..., \Pi^{N_{\max}}\}$ by applying the $\varepsilon-$greedy strategy and setting it as the incumbent solution, $\Pi_{incumbent}$. Then, apply the destruction and reconstruction operator to generate a new solution Π_{new}. Furthermore, the acceptance criteria are used to assess whether Π^N and Π_{best} are replaced by Π_{new}. The Boltzmann function, $Exp(-100 * \Delta E)$, is used to determine whether or not Π^N is replaced by an inferior new solution, which may contribute to the search procedure escaping from the local optimum. A random number, $r \in (0, 1)$, is produced for this process. If $r < Exp(-100 * \Delta E)$, Π^N is replaced by $\Pi_{new'}$; otherwise, $\Pi_{new'}$ must be discarded. At the end of each iteration, the fitness values of solutions $\{\Pi^1, \Pi^2, ..., \Pi^{N_{\max}}\}$ are updated. If the computational time exceeds the maximum allowable computation time, the RLIG procedure is terminated. Following the termination of the RLIG procedure, the (near) global optimal schedule can be derived using the Π_{best}. The RLIG algorithm is explained in more detail in the following subsections.

3.2.1 Solution representation

In this study, solution Π is represented by a single vector $(S_{\text{Stage 1}}|S_{\text{Stage 2}})$, where $S_{\text{Stage 1}}$ and $S_{\text{Stage 2}}$ are the sequence of jobs/components of the production stage and the sequence of products of the assembly stage, respectively. $S_{\text{Stage 1}}$ consists of a permutation of n jobs/components divided into g divisions by $g - 1$ "0", where the f^{th} $(f = 1, 2, ..., g)$ division represents the processing sequence of jobs/components in the f^{th} factory. Similarly, $S_{\text{Stage 2}}$ consists of a permutation of t products divided into q divisions by $q - 1$ "0", where the s^{th} $(s = 1, 2, ..., q)$ division represents the processing sequence of the products on the s^{th} assembly machine.

For example, solution $\Pi = (S_{\text{Stage 1}}|S_{\text{Stage 2}}) =$(2, 3, 5, 0, 1, 4, 6 | 1, 2, 0, 3) can be decoded, as follows. In the production stage, six jobs/components are assigned to two factories. The process sequences of jobs/components for factories 1 and 2 are 2–3–5 and 1–4–6, respectively. Then, these components are assigned and assembled into three products in the assembly stage using two assembly machines. The assembly sequences of the products for assembly machines 1 and 2 are 1–2 and 3, respectively.

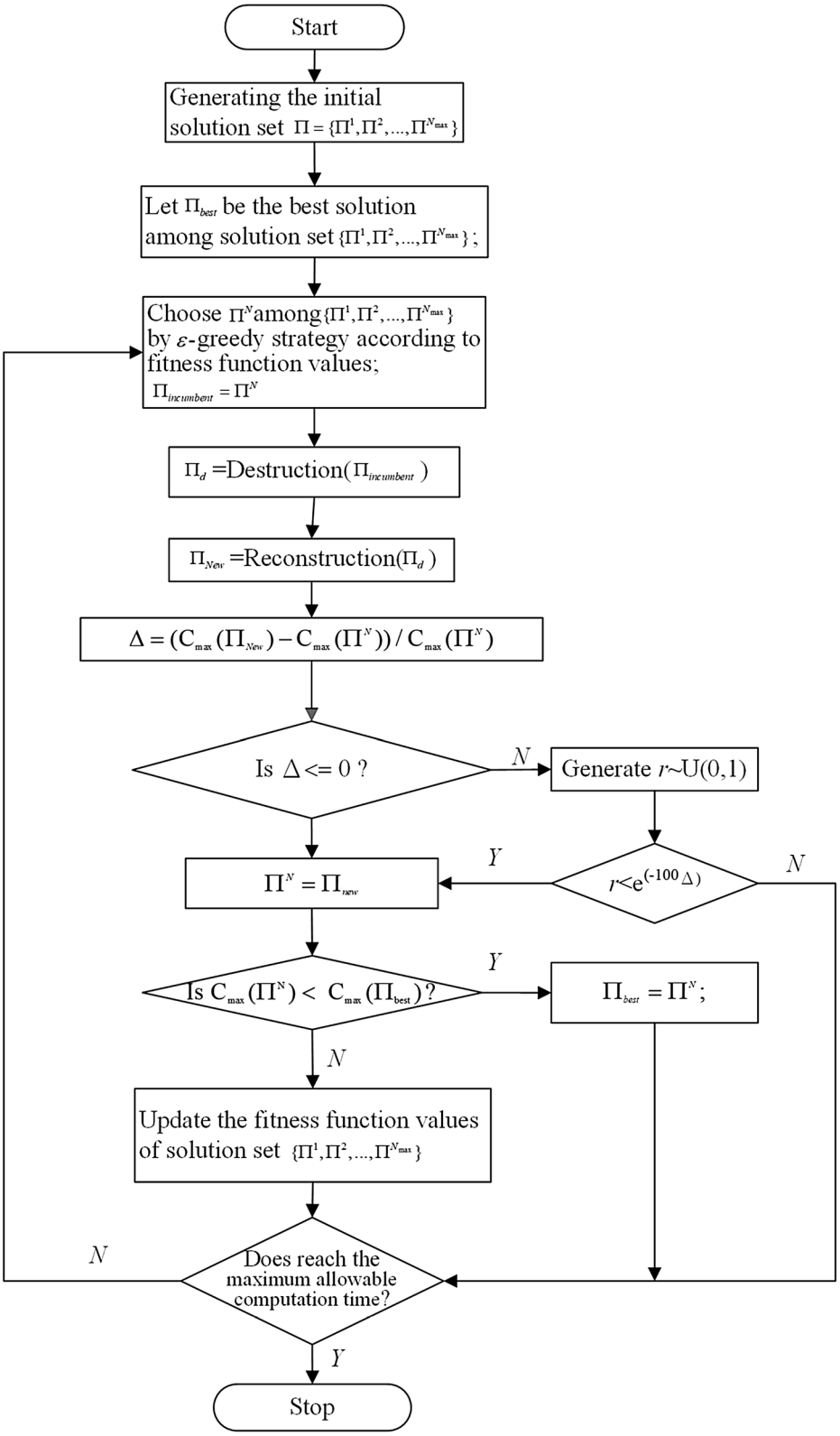


Fig. 2 Flowchart of the RLIG algorithm

3.2.2 Main steps of RLIG

The main steps of RLIG are, as follows:

3.2.2.1 Step 1: generate an initial solution set Step 1.1: Sort t products in non-ascending order of the total processing times of their jobs/components on all production machines,$\sum_{j\in G_p}\sum_{i=1}^{m} P_{j,i}$, and insert them into the initial list of products,$L_P := \{P_{[1]}, ..., P_{[t]}\}$, where $P_{[p]}$ denotes the product in the position p ($p = 1, 2, ..., t$). Then, sort the corresponding jobs/components of each product in L_P in non-ascending order of their total processing times on all production machines,$\sum_{i=1}^{m} P_{j,i}$, and insert them into the initial list of jobs/components, $L_C := \{\pi_{[1]}, ..., \pi_{[n]}\}$, where $\pi_{[j]}$ denotes the job in the position $j(j = 1, 2, ..., n)$.

Step 1.2: Initialize the partial sequence of the production stage, $S_{\text{Stage 1}} := \{\pi_{[1]}\}$, and remove the corresponding job/component from L_C.

Step 1.3: Sequence the first job/component in L_C with the minimum makespan in the respective position in the current partial job/component sequence, $S_{\text{Stage 1}}$, and remove this job/component from L_C. Repeat this procedure until all jobs/components in L_C are sequenced and a complete schedule of jobs/components, $S_{\text{Stage 1}}$, is yielded.

Step 1.4: Initialize the partial sequence of the assembly stage, $S_{\text{Stage 2}} := \{P_{[1]}\}$, and remove the corresponding product from L_P.

Step 1.5: Sequence the first product in L_P with the minimum makespan in the respective position in the current partial product sequence, $S_{\text{Stage 2}}$, and remove this product from L_P. Repeat this procedure until all products in L_P are sequenced and a complete schedule of products, $S_{\text{Stage 2}}$, is yielded.

Step 1.6: Let $\Pi^N := (S_{\text{Stage 1}}|S_{\text{Stage 2}}), \forall N = 1, ..., N_{\max}$, and $\Pi_{best} := (S_{\text{Stage 1}}|S_{\text{Stage 2}})$.

3.2.2.2 Step 2: ε−greedy destruction phase Step 2.1: Apply the ε−greedy strategy to select solution Π^N from the current solution set,$\{\Pi^1, \Pi^2, ..., \Pi^{N_{\max}}\}$, and set it as the incumbent solution, $\Pi_{incumbent}$. When applying the ε−greedy strategy, there is a probability of $1-\varepsilon$ $(0 < \varepsilon < 1)$ selecting Π_{best} and a probability of ε selecting a solution from other $N_{\max} - 1$ solutions using the roulette wheel selection technique.

Step 2.2: Select a product from the assembly machine with the maximum makespan among all assembly machines and randomly select $\beta - 1$ $(\beta \le t)$ products from other assembly machines. Remove the jobs/components of the selected products from $\Pi_{incumbent}$ to Π_d in their selecting order, and set the residual partial solution of $\Pi_{incumbent}$ after removing the jobs/components of the selected products as Π_p. To avoid removing too many jobs/components, each job/component of the removed products has a 50% probability of being removed.

3.2.2.3 Step 3: reconstruction phase Reinsert the jobs/components in Π_d into Π_p one by one until a new complete solution, Π_{new}, is reconstructed. When inserting a job/component in Π_d into Π_p, evaluate and select the best one among all possible positions.

3.2.2.4 Step 4: acceptance criteria Use the following criteria to assess whether or not Π^N and Π_{best} are replaced by Π_{new}:

IF $C_{\max}(\Pi_{new}) \le C_{\max}(\Pi^N)$, set $\Pi^N := \Pi_{new}$.

ELSE_IF generate $\Pi_{new'}$ by swapping/inserting the components/products of Π_{new}.

IF $C_{\max}(\Pi_{new'}) \le C_{\max}(\Pi^N)$, set $\Pi^N := \Pi_{new'}$

ELSE_IF $C_{\max}(\Pi_{new'}) > C_{\max}(\Pi^N)$, generate $r \sim$ U (0,1);

IF $r < Exp(-100 \cdot \Delta E)$,, set $\Pi^N := \Pi_{new'}$.

Otherwise, reject $\Pi_{new'}$.

IF $C_{\max}(\Pi^N) \le C_{\max}(\Pi_{best})$, set $\Pi_{best} := \Pi^N$.

Here, $C_{\max}(\bullet)$ is the makespan of a specific solution $(\bullet)$; $r \in [0, 1]$ denotes a random number that is randomly generated from the uniform distribution U(0,1); and $\Delta E = [C_{\max}(\Pi_{new'}) - C_{\max}(\Pi_{incumbent})]/[C_{\max}(\Pi_{incumbent})]$.

3.2.2.5 Step 5: update the fitness function values Apply the following formula to update the fitness function value of Π^N:

$$fit^N_{iter} = \frac{n^N - 1}{n^N} \cdot fit^N_{iter-1} + \frac{1}{n^N} \cdot [C_{\max}(\Pi_{incumbent}) - C_{\max}(\Pi_{new})]$$

where fit^N_{iter} denotes the fitness function value of Π^N used in the next iteration of the roulette wheel selection procedure, and n^N means the cumulative selected times of Π^N in the solution procedure.

3.2.2.6 Step 6: stop criterion Repeat Steps 2 to 5 until the maximum allowable computation time, $T_{\max}$, is reached. In this study, $T_{\max} = \tau \cdot (g + m + q) \cdot (n + t)/1000$ (CPU time in seconds), in which τ is a parameter that controls the maximum allowed computation time.

The proposed RLIG algorithm benefits from both the multi-seed hill-climbing strategy and the ε−greedy selection strategy. The multi-seed hill-climbing strategy can effectively improve the diversification of traditional hill-climbing, which is one of the simplest and oldest local search technologies (Ying and Lin 2020, 2022). As the use of multi-seed can prevent the search from being trapped in the local optima, this mechanism usually leads to relatively high-quality results (Lin et al. 2011, 2013; Lin and Ying 2013; 2015). To implement the multi-seed hill-climbing strategy, in Step 1, $N_{\max}$ initial solutions were generated using the famous NEH constructive heuristic (Nawaz et al. 1983) to

increase diversification and escape local optimums. Steps 2 and 3 are tailored $\varepsilon-$greedy destruction and reconstruction mechanisms that are repeatedly used to improve the incumbent and best solutions found so far. The RLIG algorithm removed the selected products and 50% of the associated jobs/components from the incumbent solution using the $\varepsilon-$greedy selection strategy in the deconstruction mechanism. The $\varepsilon-$greedy selection strategy is one of the most popular exploration methods used in reinforcement learning (Maqbool et al. 2012). This mechanism can be used as a perturbation mechanism to better balance the dilemma between exploration and exploitation in solving many combination optimization problems (Guo et al. 2020), as it allows more effective searching during the iterative process through reinforcement learning, thus, escaping the local optimum. Although the greedy algorithm has a strong exploitation capability, it neglects 'inferior' incumbent solutions, which can be significantly improved in the future; therefore, the main point of the $\varepsilon-$greedy selection strategy is to adjust the utilization rate of the greedy or random algorithm by the probability ε ($\varepsilon < 1$). After that, the reconstruction mechanism was applied to insert the removed jobs/components back into the remaining partial solution one by one, in order to generate a new complete solution. In Step 4, the acceptance criteria were applied to judge whether Π^N and Π_{best} are updated by Π_{new} or the neighborhood solution $\Pi_{new'}$ as generated by swap or insertion operations. In Step 5, the fitness function value of Π^N, which is used in the next iteration of the roulette wheel selection procedure, was updated according to the formula modified from the $\varepsilon-$greedy algorithm. To ensure a fair comparison with different compared algorithms, the iterative solution procedure was repeated until the termination condition, i.e., the maximum allowed computation time, is reached.

4 Numerical analysis

The following section presents the experimental results to demonstrate the performance of RLIG and the MILP model in solving the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem. All compared algorithms were programmed using Visual C++(2019) and executed on a PC with the following specifications: an Intel® Xeon® E3-1245 3.7 GHz V6 processor with 64 GB of RAM, and a Windows 10 operating system. The numerical analysis began with a description of the dataset, performance measurement, and calibration parameters of the algorithm, followed by the solution of the MILP formulation using Gurobi version 9.0, and concluded with a discussion of the performance of RLIG in the context of solving test problems of different sizes.

4.1 Test problem set

The numerical experiments in this study were performed using the benchmark instances created by Zhang et al. (2018). The dataset contained tiny-, small- and large-sized test instances. The tiny-size test instances were generated based on $n = 10$ jobs, $f = \{2, 3\}$ factories, $m = \{2, 3\}$ machines, $t = \{4, 5\}$ products, and $q = \{2, 3, 4\}$ assembly machines, and one test instance was generated for each of the $1 \times 2^3 \times 3 = 24$ combinations, thus, 24 tiny-size test instances were used in this study. The small-size test instances were generated based on $n = \{20, 24, 30\}$ jobs, $f = \{2, 3\}$ factories, $m = \{2, 3\}$ machines, $t = \{6, 8\}$ products, and $q = \{2, 3, 4\}$ assembly machines, and ten test instances were generated for each of the $3 \times 2^3 \times 3 = 72$ combinations; therefore, $72 \times 10 = 720$ small-size test instances were used in this study. The large-size test instances were generated based on $n = \{100, 200\}$ jobs, $f = \{6, 8\}$ factories, $m = \{5, 10\}$ machines, $t = \{30, 40\}$ products, and $q = \{6, 8\}$ assembly machines, and ten test instances were generated for each of the $2^5 = 32$ combinations; hence, $32 \times 10 = 320$ large-size test instances, as created by Zhang et al. (2018), were used in this study. To evaluate the performance of the compared algorithms on larger test instances, we generated $32 \times 10 = 320$ very large-size test instances based on $n = \{500, 1000\}$ jobs,$f = \{10, 20\}$ factories,$m = \{12, 16\}$ machines,$t = \{30, 40\}$ products, and $q = \{60, 80\}$ assembly machines, and ten test instances were generated for each of the $2^5 = 32$ combinations, therefore, a total of 640 large-size test instances were used in this study. Following the approach of Zhang et al. (2018), six small-size and six large-size test instances were generated to calibrate the parameters of RLIG, and the processing times of the jobs/components, the assembly times of the products, and the sequence-dependent setup times were all randomly generated from uniform distribution U[1, 99] and U[1, 20], respectively.

4.2 Parameter calibration

As with many metaheuristics, the setting of the parameter values affected the performance of the RLIG algorithm. In this study, four levels for each of the $N_{\max}$,β,ε, and τ parameters were considered for the calibration of the RLIG algorithm, where $N_{\max}$ is the number of solutions in the solution set; ε is the selection probability used in the $\varepsilon-$greedy selection strategy; β is the number of products to be removed in the destruction phase; and τ is a parameter controlling the maximum allowable computation time. A summary of the calibration configurations can be found in Table 1. The calibration experiments were performed with 12 test instances considering 4^4 possible scenarios of parameter values. In order to efficiently obtain the best setting for each parameter,

Table 1 Orthogonal array and the obtained *ARPD*s

No	$N_{\max}$	β	ε	τ	*ARPD*
1	3	2	0.1	30	1.0251
2	3	3	0.2	40	1.0463
3	3	4	0.3	50	1.2908
4	3	5	0.4	60	1.3015
5	4	2	0.2	50	0.9364
6	4	3	0.1	60	0.9510
7	4	4	0.4	30	1.3727
8	4	5	0.3	40	1.4267
9	5	2	0.3	60	0.8652
10	5	3	0.4	50	1.0071
11	5	4	0.1	40	1.3084
12	5	5	0.2	30	1.4834
13	6	2	0.4	40	0.9209
14	6	3	0.3	30	1.1595
15	6	4	0.2	60	1.1979
16	6	5	0.1	50	1.3367

Table 2 *ARPD*s obtained by different levels of each parameter

Setting	$N_{\max}$	β	ε	τ
1	1.1659	0.9369	1.1553	1.2601
2	1.1717	1.0410	1.1660	1.1756
3	1.1660	1.2924	1.1855	1.1427
4	1.1537	1.3871	1.1505	1.0789
Range	0.0180	0.4502	0.0350	0.1812
Rank	4	1	3	2

type B of the Taguchi L16 orthogonal experimental design was employed, including 16 combinations of parameter settings that underwent 20 runs for each random test instance, resulting in a total of $20 \times 16 = 320$ runs. To evaluate the performance of different scenarios of parameter values, we used the average relative percentage deviation (*ARPD*) of the obtained makespan value, which was calculated using the following equation, as the desired performance measure:

$$ARPD = \left(\sum_{r=1}^{12} \frac{C_{\max}^{ave}(\pi) - C_{\max}^{best}(\pi)}{C_{\max}^{best}(\pi)} \times 100 \right) / 12 \tag{26}$$

where $C_{\max}^{ave}(\pi)$ denotes the average makespan value for a particular parameter combination obtained by the algorithm in one run for a particular instance, and $C_{\max}^{best}(\pi)$ represents the best makespan value obtained by the algorithm when considering all 16 parameter combinations and 20 runs for the same test instance.

A comparison of the *ARPD*s obtained using different levels of each parameter is presented in Table 2. According to the range of the *ARPD*s,β was the most significant parameter among the four parameters. Under a fixed maximal allowed computation time, the larger the value of β, the more products and their associated jobs/components would be selected and removed from the incumbent solution, resulting in more time spent in the subsequent reconstruction phase. In other words, it would require more computation time to perform one iteration of the algorithm, resulting in fewer total iterations. Consequently, if the maximum computation time was fixed, a larger β value would result in fewer candidate solutions being evaluated and the best solution not being found. Among the four parameters, the allowed maximum computation time,τ, was the second significant parameter. When the value of τ was increased, better solutions could be obtained with more computation time. Compared with β and τ, parameters $N_{\max}$ and ε had no significant impacts on the performance of RLIG. According to the parameter calibration experimental results, the parameter values of $N_{\max}$,β,ε, and τ were set to 6, 2, 0.4, and 60, respectively. In related studies (Pan et al 2019; Zhang et al. 2020; Huang et al. 2021), the maximal computational time is set to $\tau \cdot m \cdot n/1000$ seconds, in which τ is a parameter that controls the maximum allowed computation time, and n and m is the number of jobs and number of machines, respectively. In this study, m and n are expanded to $(n + t)$ and $(g + m + q)$. Based on these estimates, the maximum allowable computation time of the RLIG algorithm in the following experiments was set to $T_{\max} = 60 \cdot (g + m + q) \cdot (n + t)/1000$(CPU time in seconds) to achieve an equilibrium between solution quality and computing time.

4.3 Numerical results

To evaluate the performance of the compared algorithms from a broader perspective, the average relative percentage deviation (*ARPD*) was used to measure solutions obtained by different algorithms. The formula for calculating *ARPD* is, as follows:

$$ARPD = \sum_{i=1}^{n} \frac{C_{\max}(\pi) - C_{\max}^{best}(\pi)}{C_{\max}^{best}(\pi)} / n \times 100 \tag{27}$$

where $C_{\max}(\pi)$ is the makespan value obtained by a single run of the algorithm for a given test instance, and $C_{\max}^{best}(\pi)$ denotes the best makespan value obtained by all compared algorithms performing five runs for that test instance. An algorithm with smaller *ARPD*s would indicate that it could produce higher quality solutions than the compared algorithms. Finally, the paired *t*-test was performed to determine whether the differences between the computational results of RLIG and IG were statistically significant.

To the best of our knowledge, in existing literature, IG-based algorithms are by far the best meta-heuristics for solving DAPFSP-FA problems. Therefore, this study used the IG (i.e., the RLIG algorithm without the reinforcement

learning mechanism) as the benchmark algorithms to evaluate the performance of RLIG. In order to compare the performance of RLIG with that of IG (i.e., the RLIG algorithm without the reinforcement learning mechanism) and the MILP model for solving the 24 tiny-size instances, the makespan value of the best solution (C_{max}), the average makespan value (Ave.C_{max}), and the average computation time (T(s)) for five runs of each instance was summarized, as shown in Table 3. The MILP model was solved using Gurobi version 9.0, which is one of the strongest and widely used solvers with a maximum computation time of 7200 s. As can be seen in Table 3, RLIG and IG achieved the same best solutions for each tiny-size instance. The solution quality of IG was as good as that of RLIG, except that the average makespan value of test instance T14, as obtained with IG, was slightly worse than that of RLIG. As shown in Table 3, both IG and RLIG yielded the same optimal solutions for the 19 test instances that could be optimally solved within 7200 s using the MILP model. Regarding the remaining five test instances that could not be optimally solved within 7200 s using the MILP model, the best solutions, as obtained using both IG and RLIG, were superior to the feasible solutions obtained using the MILP model. Since the computation times required by the RLIG and IG algorithms were significantly shorter than those of the MILP model, we could conclude that these algorithms were more appropriate for solving tiny-size problems.

In order to compare the performance of RLIG with that of IG for solving the small- and large-size test instances, the average makespan value of the best solution (C_{max}), the average makespan value of five runs (Ave.C_{max}), and the average computation time (T(s)) for five runs of each subset test instance with different configurations are summarized in Table 4. In order to evaluate the performance of RLIG and IG from a broader perspective, the best *ARPD* values and total average *ARPD* values of their obtained makespan values with different configurations of the small- and large-size test instances are summarized in Table 5. As shown in Tables 4 and 5, RLIG was superior to IG for all examined workloads and production scales, except for the subset of test instances $n = 20$ and $q = 2$. The larger the production scale and workload, the greater the differences between RLIG and IG performance.

The overall computational results were visually compared with various configurations, as shown in Fig. 3. As

Table 3 Computational results of tiny-scale instances (best in **bold**)

						MILP			IG			RLIG		
Instance	n	f	m	p	q	C_{max}	LB	T (s)	C_{max}	Ave.C_{max}	T (s)	C_{max}	Ave.C_{max}	T (s)
T01	10	2	2	4	2	**363**	363	2970.68	**363**	**363.0**	5.04	**363**	**363.0**	5.04
T02	10	2	2	4	3	**329**	329	5639.15	**329**	**329.0**	5.88	**329**	**329.0**	5.88
T03	10	2	2	4	4	**348**	348	4184.19	**348**	**348.0**	6.72	**348**	**348.0**	6.72
T04	10	2	2	5	2	**259**	259	2063.90	**259**	**259.0**	5.40	**259**	**259.0**	5.40
T05	10	2	2	5	3	415	212	7200.10	**410**	**410.0**	6.30	**410**	**410.0**	6.30
T06	10	2	2	5	4	**339**	153	7200.12	**339**	**339.0**	7.20	**339**	**339.0**	7.20
T07	10	2	3	4	2	**386**	386	5098.02	**386**	**386.0**	5.88	**386**	**386.0**	5.88
T08	10	2	3	4	3	**421**	215	7200.09	**421**	**421.0**	6.72	**421**	**421.0**	6.72
T09	10	2	3	4	4	**347**	244	7200.10	**347**	**347.0**	7.56	**347**	**347.0**	7.56
T10	10	2	3	5	2	**404**	181	7200.08	**404**	**404.0**	6.30	**404**	**404.0**	6.30
T11	10	2	3	5	3	412	167	7200.09	**410**	**410.0**	7.20	**410**	**410.0**	7.20
T12	10	2	3	5	4	479	183	7200.10	**478**	**478.0**	8.10	**478**	**478.0**	8.10
T13	10	3	2	4	2	**344**	344	1129.94	**344**	**344.0**	5.88	**344**	**344.0**	5.88
T14	10	3	2	4	3	**269**	263	7200.09	**269**	**269.0**	6.72	**269**	269.2	6.72
T15	10	3	2	4	4	**210**	210	1167.91	**210**	**210.0**	7.56	**210**	**210.0**	7.56
T16	10	3	2	5	2	**294**	294	6071.49	**294**	**294.0**	6.30	**294**	**294.0**	6.30
T17	10	3	2	5	3	**286**	286	6641.62	**286**	**286.0**	7.20	**286**	**286.0**	7.20
T18	10	3	2	5	4	**256**	256	5948.52	**256**	**256.0**	8.10	**256**	**256.0**	8.10
T19	10	3	3	4	2	**237**	237	839.69	**237**	**237.0**	6.72	**237**	**237.0**	6.72
T20	10	3	3	4	3	**322**	322	5420.35	**322**	**322.0**	7.56	**322**	**322.0**	7.56
T21	10	3	3	5	4	346	211	7200.15	**335**	**335.0**	9.00	**335**	**335.0**	9.00
T22	10	3	3	5	2	**335**	328	7200.15	**335**	**335.0**	7.20	**335**	**335.0**	7.20
T23	10	3	3	5	3	315	210	7200.12	**311**	**311.0**	8.10	**311**	**311.0**	8.10
T24	10	3	3	5	4	**306**	292	7200.15	**306**	**306.0**	9.00	**306**	**306.0**	9.00

Table 4 Results analysis for small- and large-size test instances (best in **bold**)

Configuration		IG			RLIG		
		$C_{\max}$	Ave.$C_{\max}$	T (s)	$C_{\max}$	Ave.$C_{\max}$	T (s)
n	20	**543.24**	544.56	12.96	543.30	544.59	12.96
	24	633.67	635.84	14.88	**633.53**	635.78	14.88
	30	765.14	767.77	17.76	**765.11**	767.77	17.76
	100	1244.93	1248.69	172.20	**1244.86**	1248.63	172.20
	200	2119.38	2124.12	299.70	**2114.98**	2119.57	299.70
	500	3097.08	3104.66	1453.80	**3095.50**	**3102.55**	1453.80
	1000	5555.40	5560.36	2889.32	**5554.09**	**5557.51**	2889.32
f	2	757.59	759.60	14.25	**757.58**	759.67	14.25
	3	537.11	539.18	16.15	**537.05**	539.09	16.15
	6	1864.05	1868.36	224.85	**1861.66**	1866.18	224.85
	8	1500.26	1504.44	247.05	**1498.18**	1502.03	247.05
	15	4009.03	4014.90	2030.11	**4008.11**	**4013.14**	2030.11
	20	4643.45	4650.12	2313.02	**4641.49**	**4646.92**	2313.01
m	2	613.82	615.78	14.25	**613.76**	615.74	14.25
	3	680.88	683.00	16.15	**680.88**	683.02	16.15
	6	1482.88	1487.05	210.90	**1481.06**	1484.90	210.90
	8	1881.43	1885.76	261.00	**1878.78**	1883.31	261.00
	12	4803.36	4810.13	2073.16	**4801.71**	**4807.50**	2073.16
	16	3849.12	3854.88	2269.96	**3847.89**	**3852.56**	2269.96
t	6	648.58	650.58	14.72	**648.54**	650.60	14.72
	8	646.12	648.20	15.68	**646.09**	648.16	15.68
	30	1681.68	1685.88	226.80	**1679.45**	1683.69	226.80
	40	1682.64	1686.93	245.10	**1680.39**	1684.51	245.10
	60	4332.67	4338.42	2120.71	**4330.64**	**4335.34**	2120.71
	80	4319.81	4326.60	2222.41	**4318.96**	**4324.72**	2222.41
q	2	**655.36**	657.70	13.30	655.40	657.62	13.30
	3	642.81	644.72	15.20	**642.73**	644.76	15.20
	4	643.88	645.74	17.10	**643.81**	645.76	17.10
	6	1725.51	1730.09	231.72	**1723.26**	1727.75	231.72
	8	1609.90	1613.60	243.00	**1607.68**	1611.36	243.00
	12	4329.21	4335.66	2097.46	**4328.31**	**4333.55**	2097.46
	16	4323.26	4329.36	2245.66	**4321.28**	**4326.51**	2245.66

depicted in Fig. 3, the RLIG algorithm performed better with different production scales, such as the number of machines, factories, products, and assembly machines. When testing different workloads and production scales, IG performed slightly better than RLIG for $n = 20$ and $q = 2$ test instances; however, as the workload and production scales were increased, the visual analysis of the computational results clearly revealed that RLIG was superior to IG.

In order to verify whether the differences between the computational results of RLIG and IG were statistically significant, paired t-testing was performed, as shown in Table 6. The t-values in Table 6 confirmed at the confidence interval of 0.05 that RLIG was significantly better than IG, indicating that the multi-seed hill-climbing strategy and $\epsilon-$greedy selection strategy could improve the performance of IG and help RLIG find the best solution for the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem.

5 Discussion

This study presented a MILP model and a highly effective and efficient RLIG algorithm to solve the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem. The analytical results, as based on extensive benchmark instances, show that the proposed RLIG algorithm is better than the MILP model in solving tiny-size problems. In addition, it significantly outperformed the IG algorithm in solving small and large test instances.

Table 5 *ARPDs* for small- and large-size test instances (best in **bold**)

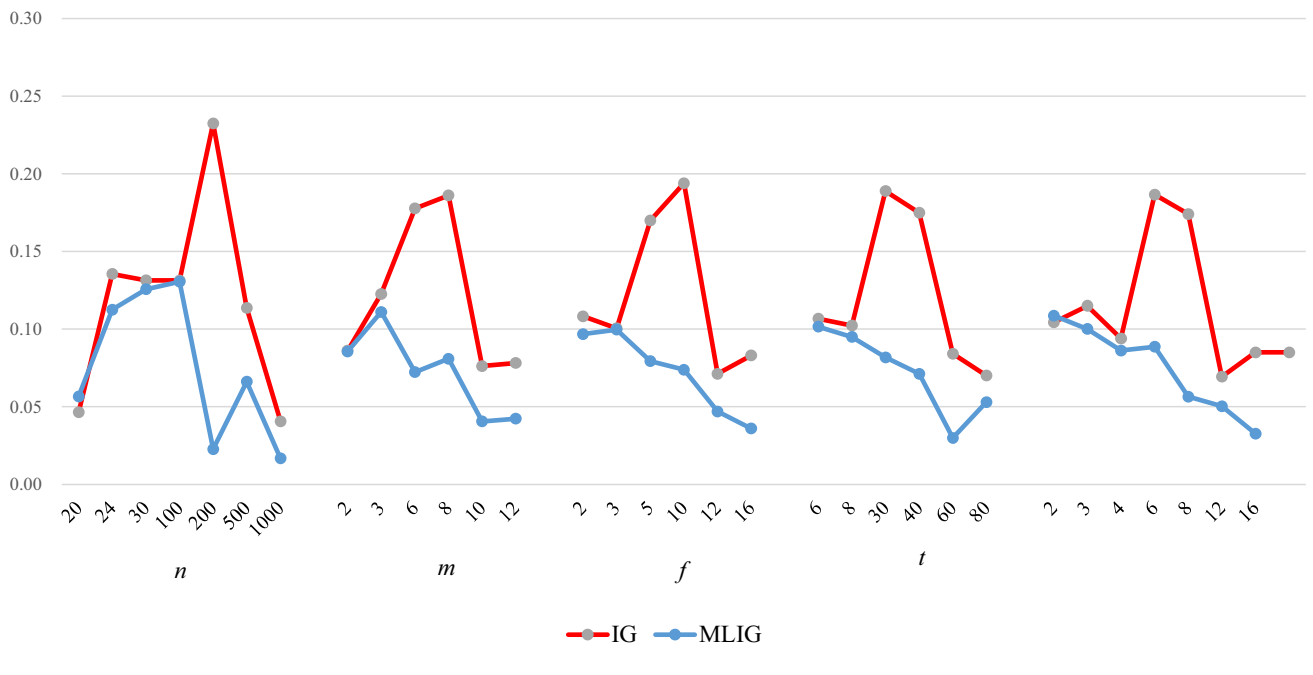
Configuration		IG		RLIG	
		Best ARPD	Ave. ARPD	Best ARPD	Ave. ARPD
n	20	**0.046**	**0.301**	0.057	0.305
	24	0.135	0.490	**0.113**	**0.477**
	30	0.131	0.487	**0.126**	**0.484**
	100	0.131	0.437	**0.130**	**0.435**
	200	0.232	0.464	**0.023**	**0.242**
	500	0.114	0.359	**0.066**	**0.296**
	1000	0.041	0.131	**0.017**	**0.079**
f	2	0.108	0.436	0.086	0.357
	3	0.101	0.416	0.111	0.486
	6	0.170	0.464	**0.072**	**0.331**
	8	0.194	0.437	**0.081**	**0.346**
	15	0.071	0.242	**0.041**	**0.187**
	20	0.083	0.247	**0.042**	**0.188**
m	2	**0.086**	**0.349**	0.097	0.425
	3	0.123	0.503	**0.100**	**0.418**
	6	0.178	0.427	**0.079**	**0.350**
	8	0.186	0.474	**0.074**	**0.327**
	12	0.076	0.241	**0.047**	**0.203**
	16	0.078	0.249	**0.036**	**0.172**
t	6	0.107	0.421	**0.102**	0.423
	8	0.102	0.431	**0.095**	**0.421**
	30	0.189	0.456	**0.082**	**0.352**
	40	0.175	0.445	**0.071**	**0.325**
	60	0.084	0.244	**0.030**	**0.162**
	80	0.070	0.246	**0.053**	**0.212**
q	2	**0.104**	0.470	0.109	**0.452**
	3	0.115	**0.418**	**0.100**	0.424
	4	0.094	0.390	**0.086**	**0.389**
	6	0.187	0.470	**0.089**	**0.363**
	8	0.174	0.418	**0.056**	**0.298**
	12	0.069	0.242	**0.050**	**0.196**
	16	0.085	0.248	**0.033**	**0.179**

The advantage of the proposed RLIG algorithm is that it applies a multi-seed hill-climbing strategy, which is a framework to achieve diversification for preventing the search from being trapped in a local optimum, to generate the initial solution set founded on the NEH constructive heuristic. Furthermore, this study embedded an $\varepsilon-$greedy selection strategy in the destruction phase of RLIG, which is used in reinforcement learning to formalize the notion of decision making under uncertainty. The $\varepsilon-$greedy selection strategy can combine the random algorithm and the IG-based algorithm to handle the exploration and exploitation dilemma through reinforcement learning during the iterative process. While traditional IG-based algorithms have strong exploitation ability, they easily get stuck in the local optimum. The $\varepsilon-$greedy selection strategy is a perturbation mechanism that uses knowledge reinforcement, as learned from historical search results, to adjust the exploration and exploitation of the RLIG algorithm and help the algorithm escape the local optimum. Without the multi-seed hill-climbing strategy and the $\varepsilon-$greedy selection strategy, the IG-based algorithm may be localized in a small region of the solution space, and thus, be trapped in a local optimum, which eliminates the possibility of finding an optimal solution.

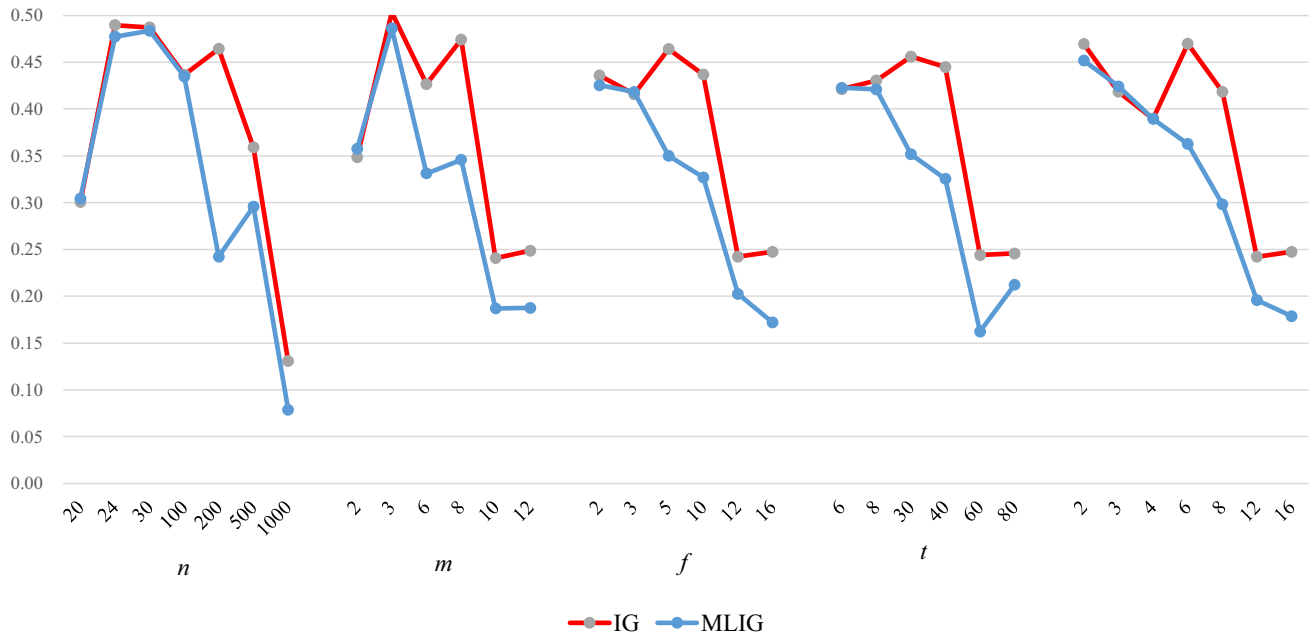
The drawback of the proposed RLIG algorithm is that the multi-seed hill-climbing strategy and the $\varepsilon-$greedy selection strategy may not work very well for small test instances, e.g., the subset test instances of $n = 20$ and $q = 2$. The possible reasons are that the diversification, as resulted from the multi-seed hill-climbing strategy, and the perturbation, as resulted from the $\varepsilon-$greedy selection strategy, are not able to exploit and explore the existing solutions to find the best solution for small test instances. The major limitation of the proposed RLIG algorithm is the computation time required for very large problems, e.g., $n = 500$ and 1000, which needs to be improved to make it more suitable for solving real-time scheduling problems. An additional limitation of this research is that the operational parameters were assumed to be deterministic.

6 Conclusions and future research directions

Globalization and the rapid development of communication technology have accelerated the development of DMSs, and scheduling literature has evolved significantly with the integration of production and assembly operations in DMSs. This study addressed the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem, which has wide application in globalized production but has never been explored. Given the novelty of the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem, we first contribute to the literature by developing a MILP model for the problem to fill the research gap of the scheduling theory. Considering that the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem is *NP*-hard in the strong sense, an effective and efficient RLIG algorithm was also presented to solve this problem. The RLIG algorithm uses a multi-seed hill-climbing strategy and an $\varepsilon-$greedy selection strategy that could exploit and explore the existing solutions to find the best solutions for the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem. Extensive numerical analysis with 1360 test instances shows that RLIG can achieve high quality solutions under different production scales and workloads, which makes it a viable approach for use in various industries with supply-chain-integrated scheduling requirements and a strong benchmark algorithm for optimizing the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem.



(a) Best *RPD*



(b) Average *RPD*

Fig. 3 Visual analysis of the computational results

Future research can be extended in several directions. First, as this study is the first theoretical advance made on the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem, further studies are desirable, and in particular, the development of sophisticated exact, approximation, and heuristic algorithms to solve this problem. Second, the DAPFSP-FASDST problem with other performance criteria, e.g. the total weighted completion time, the total weighted tardiness, and the number of tardy jobs, deserves further study. Third, the DAPFSP-FASDST problem with process-related constraints,

Table 6 Statistical test of significance for the benchmark results for *RPD*

Best *RPD* among five runs		Average *RPD* among five runs	
RLIG *vs.*	IG	RLIG *vs*	IG
Difference	0.03643	Difference	0.042
Degree of freedom	1360	Degree of freedom	1360
t-value	4.718	t-value	8.500
P-value	0.0000	P-value	0.0000

such as blocking, no-wait, and no-idle, in the distributed flowshops of the production stage is worthy of further investigation. Fourth, a corresponding stochastic model of the $DF_m \rightarrow R_m|prmu, ST_{sd}|C_{\max}$ problem is needed to bridge the gap between theoretical progress and industrial practice with non-deterministic operational parameters. Fifth, further investigation is essential to solve multi-objective DAPFSP-FASDST. Finally, the related problems of production and assembly stages with additional or heterogeneous shop types, e.g., non-permutation flowshops, jobshops, and openshops, deserve further discussion.

Funding Ministry of Science and Technology, Taiwan, MOST109-2221-E-027-073, Kuo-Ching Ying, MOST109-2410-H-182-009MY3, Shih-Wei Lin, Chang Gung Memorial Hospital, Linkou, BMRPA19, Shih-Wei Lin.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

Behnamian J (2014) Decomposition based hybrid VNS–TS algorithm for distributed parallel factories scheduling with virtual corporation. Comput Oper Res 52:181–191

Chan FTS, S.H. Chung SH, and P.L.Y. Chan. (2005) An Adaptive genetic algorithm with dominated genes for distributed scheduling problems. Expert Syst Appl 29:364–371

Chan FTS, Chung SH, Chan PLY (2006) Application of genetic algorithms with dominant genes in a distributed scheduling problem in flexible manufacturing systems. Int J Prod Res 44:523–543

Cheng CY, Ying KC, Chen HH, Lu HS (2019) Minimizing makespan in distributed mixed no-idle flowshops. Int J Prod Res 57:48–60

De Giovanni L, Pezzella F (2010) An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. Eur J Oper Res 200:395–408

Framinan JM, Perez-Gonzalez P, Fernandez-Viagas V (2019) Deterministic assembly scheduling problems: a review and classification of concurrent-type scheduling models and solution procedures. Eur J Oper Res 273:401–417

Guo F, Li Y, Liu A, an Z. Liu, Z. (2020) A reinforcement learning method to scheduling problem of steel production process. J Phys Conf Ser 1486:072035

Hatami S, Ruiz R, Andrés-Romano C (2013) The distributed assembly permutation flowshop scheduling problem. Int J Prod Res 51:5292–5308

Hatami S, Ruiz R, Romano CA (2014) "Two simple constructive algorithms for the distributed assembly permutation flowshop scheduling problem", managing complexity. Int J Prod Res 51:139–45

Hatami S, Ruiz R, Andrés-Romano C (2015) Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. Int J Prod Econ 169:76–88

Huang YY, Pan QK, Huang JP, Suganthan PN, Gao L (2021) An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem. Comput Ind Eng 152:107021

Jia HZ, Fuh JYH, Nee AYC, Zhang YF (2002) 2002, "Web-based multi-functional scheduling system for a distributed manufacturing environment." Concurr Eng 10:27–39

Jia HZ, Fuh JYH, Nee AYC, Zhang YF (2007) Integration of genetic algorithm and gantt chart for job shop scheduling in distributed manufacturing systems. Comput Ind Eng 53:313–320

Jing XL, Pan QK, Gao L, Wang YL (2020) An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows. Appl Soft Comput 96:106629

Khalifa HAE, Alodhaibi SS, Kumar P (2021) Solving constrained flow-shop scheduling problem through multistage fuzzy binding approach with fuzzy due dates. Adv Fuzzy Syst 2021:6697060

Komaki M, Malakooti B (2017) General variable neighborhood search algorithm to minimize makespan of the distributed no-wait flow shop scheduling problem. Prod Eng Res Devel 11:315–329

Lei D, Wang T (2020) Solving Distributed two-stage hybrid flowshop scheduling using a shuffled frog-leaping algorithm with memeplex grouping. Eng Optim 52:1461–1474

Lei D, Su B, Li M (2021) Cooperated teaching-learning-based optimisation for distributed two-stage assembly flow shop scheduling. Int J Prod Res 59:7232–7245

Li W, Li J, Gao K, Han Y, Niu B, Liu Z, Sun Q (2019) Solving robotic distributed flowshop problem using an improved iterated greedy algorithm. Int J Adv Rob Syst 2019:1–16

Li H, Li X, Gao L (2020a) A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem. Appl Soft Comput 100:106946

Li Y, Li X, Gao L, Meng L (2020b) An Improved artificial bee colony algorithm for distributed heterogeneous hybrid flowshop scheduling problem with sequence-dependent setup times. Comput Ind Eng 147:106638

Li Y, Li X, Gao L, Zhang B, Pan QK, Tasgetiren MF, Meng L (2020c) A discrete artificial bee colony algorithm for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. Int J Prod Res 59:1–20

Li X., X. Zhang, M. Yin, and J. Wang. 2015, "A Genetic Algorithm for the Distributed Assembly Permutation Flowshop Scheduling Problem," In: 2015 IEEE Congress on Evolutionary Computation, pp. 3096–3101.

Lin WC (2018) "Minimizing the makespan for a two-stage three-machine assembly flow shop problem with the sum-of-processing-time based learning effect. Discret Dyn Nat Soc 2018:1–15

Lin J, S. Zhang S. (2016) An Effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. Comput Ind Eng 97:128–136

Lin SW, Ying KC (2013) Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm. Comput Oper Res 40:1625–1647

Lin SW, Ying KC (2015) A multi-point simulated annealing heuristic for solving multiple-objective unrelated parallel machine scheduling problem. Int J Prod Res 53:1065–1076

Lin SW, Ying KC (2016) Minimizing makespan for solving the distributed no-wait flowshop scheduling problem. Comput Ind Eng 99:202–209
Lin SW, Ying KC, Lu CC, Gupta JND (2011) Applying multi-start simulated annealing to schedule a flowline manufacturing cell with sequence dependent family setup times. Int J Prod Econ 130:246–254
Lin SW, Ying KC, Huang CY (2013) Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. Int J Prod Res 51:5029–5038
Lin J, Wang ZJ, Li X (2017) A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. Swarm Evol Comput 36:124–135
Mao J, Pan Q, Miao Z, Gao L (2021) An Effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. Expert Syst Appl 169:114495
Maqbool SD, Ahamed TI, Ali SQ, Pazheri FR, Malik NH (2012) Comparison of pursuit and ε-Greedy algorithm for load scheduling under real time pricing. 2012 IEEE International Conference on Power and Energy (PECon). Kota Kinabalu, Malaysia, pp 515–519
Meng T, Pan QK (2021) A distributed heterogeneous permutation flowshop scheduling problem with lot-streaming and carryover sequence-dependent setup time. Swarm Evol Comput 60:100804
Meng T, Pan QK, Wang L (2019) A Distributed permutation flowshop scheduling problem with the customer order constraint. Knowl-Based Syst 184:104894
Naderi B, Azab A (2014) Modeling and heuristics for scheduling of distributed job shops. Expert Syst Appl 41:7754–7763
Naderi B, Ruiz R (2010) The distributed permutation flowshop scheduling problem. Comput Oper Res 37:754–768
Nawaz M Jr, Enscore E, Ham I (1983) A Heuristic algorithm for the M-machine, N-job flow-shop sequencing problem. Omega 11:91–95
Pan QK, Gao L, Li XY, Jose FM (2019) Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. Appl Soft Comput 81:105492
Pan QK, Gao L, Wang L (2022) An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems. IEEE Trans Cybern Forthcom. https://doi.org/10.1109/TCYB.2020.3041494
Pourhejazy P, Cheng CY, Ying KC, Lin SY (2021) Supply chain-oriented two-stage assembly flowshops with sequence-dependent setup times. J Manuf Syst 61:139–154
Pourhejazy P, Cheng CY, Ying KC, Nam NH (2022) Meta-lamarckian-based iterated greedy for optimizing distributed two-stage assembly flowshops with mixed-setups. Ann Oper Rese Forthc. https://doi.org/10.1007/s10479-022-04537-2
Renna P. 2012, Production and Manufacturing System Management: Coordination Approaches and Multi-site Planning. IGI Global.
Rossit DA, Tohmé F, Frutos M (2019) "Industry 4.0: smart scheduling. Int J Prod Res 57:3802–3813
Vanchinathan K, Selvaganesan N (2021) Adaptive fractional order PID controller tuning for brushless DC motor using artificial bee colony algorithm. Results Control Optim 4:100032
Vanchinathan K, Valluvan KR (2018) "A metaheuristic optimization approach for tuning of fractional-order PID controller for speed control of sensorless BLDC motor. J Circ Sys Comput 27:1850123
Vanchinathan K, Valluvan KTR, Gnanavel C, Gokul C, Albert JR (2021) An improved incipient whale optimization algorithm based robust fault detection and diagnosis for sensorless brushless dc motor drive under external disturbances. Int Trans Electr Energy Syst 31:e13251
Wang SY, Wang L (2016) An Estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. IEEE Trans Syst Man Cybern Syst 46:139–149
Wang K, Huang Y, Qin H (2016) A Fuzzy Logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. J Oper Res Soc 67:68–82
Wu CC, Chen JY, Lin WC, Lai K, Liu SC, Yu PW (2018) A two-stage three-machine assembly flow shop scheduling with learning consideration to minimize the flowtime by six hybrids of particle swarm optimization. Swarm Evol Comput 41:97–110
Wu CC, Chen JY, Lin WC, Lai K, Bai D, Lai SY (2019) A two-stage three-machine assembly scheduling flowshop problem with both two-agent and learning phenomenon. Comput Ind Eng 130:485–499
Yang S, Xu Z (2021) The distributed assembly permutation flowshop scheduling problem with flexible assembly and batch delivery. Int J Prod Res 59:4053–4071
Ying KC, Lin SW (2017) Minimizing makespan in distributed blocking flowshops using hybrid ierated greedy algorithms. IEEE Access 5:15694–15705
Ying KC, Lin SW (2018) Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks. Expert Syst Appl 92:132–141
Ying KC, Lin SW (2020) Solving no-wait job-shop scheduling problems using a multi-start simulated annealing with bi-directional shift timetabling algorithm. Comput Ind Eng 146:106615
Ying KC, Lin SW (2022) Minimizing total completion time in no-wait jobshops using a backtracking multi-start simulated annealing algorithm. Comput Ind Eng 169:108238
Ying KC, Lee ZJ, Lin SW (2012) Makespan minimization for scheduling unrelated parallel machines with setup times. J Intell Manuf 23:1795–1803
Ying KC, Lin SW, Cheng CY, He CD (2017) Iterated Reference Greedy Algorithm for Solving Distributed No-idle Permutation Flowshop Scheduling Problems. Comput Ind Eng 110:413–423
Ying KC, Pourhejazy P, Cheng CY, Syu RS (2022) "Supply chain-oriented permutation flowshop scheduling considering flexible assembly and setup times. Int J Prod Res Forthc. https://doi.org/10.1080/00207543.2020.1842938
Zhang G, Xing K (2019) Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion. Comput Oper Res 108:33–43
Zhang G, Xing K, Cao F (2018) Scheduling distributed flowshops with flexible assembly and set-up time to minimise makespan. Int J Prod Res 56:3226–3244
Zhang G, Xing K, Zhang G, He Z (2020) Memetic algorithm with meta-lamarckian learning and simplex search for distributed flexible assembly permutation flowshop scheduling problem. IEEE Access 8:96115–96128
Zhao F, Zhao L, Wang L, Song H (2020) An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan Criterion. Expert Syst Appl 160:113678
Zhao F, Zhang L, Cao J, Tang J (2021) A Cooperative water wave optimization algorithm with reinforcement learning for the distributed assembly no-idle flowshop scheduling problem. Comput Ind Eng 153:107082

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

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.