International Journal of Industrial Engineering Computations 15 (2024) 553–568

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

An improved iterated greedy algorithm for distributed mixed no-wait permutation flowshop problems with makespan criterion

Chuan-Chong Li[a], Yuan-Zhen Li[a*] and Lei-Lei Meng[a]

[a]*School of Computer Science, Liaocheng University, Shandong Liaocheng, 252000, P.R. China*

C H R O N I C L E

Article history:
Received July 20 2023
Received in Revised Format October 12 2023
Accepted December 26 2023
Available online December 26 2023

Keywords:
Distributed flowshop scheduling
Flowshop
Iterated greedy algorithm
Mixed no-wait

A B S T R A C T

The distributed permutation flowshop scheduling is a critical issue in various industries, involving jobs allocation and scheduling among multiple flowshops. This paper extends the research to explore the Distributed Mixed No-Wait Permutation Flowshop Scheduling Problems (DMNWPFSP) with minimizing makespan. The innovation lies in an optimized mathematical model, hybrid heuristic algorithms, an improved iterated greedy algorithm (IIG), and high-quality solutions. Extensive experimental results demonstrate the effectiveness and superiority of the proposed IIG in terms of scheduling quality, computational efficiency, and robustness compared to existing approaches. The outcomes of this work contribute to the field of distributed flowshop scheduling, providing valuable insights for practitioners seeking to enhance production efficiency and competitiveness.

1. Introduction

In the era of globalization, industrial collaboration has become increasingly diversified (De Giovanni & Pezzella, 2010). The development of technology has fundamentally transformed the landscape of manufacturing. These technologies offer unprecedented connectivity, real-time data exchange, and intelligent decision-making capabilities. Companies are changing their production strategies to enhance competitiveness and mitigate risks. The traditional centralized production model has transitioned into distributed manufacturing, with numerous factories established globally. Distributed manufacturing offers scalability, flexibility, and high reliability, optimizing resource utilization for geographically dispersed enterprises and factories (Lu, Liu, Zhang, & Yin, 2022). Scheduling plays a crucial role in various industries and has a significant impact on operational efficiency and overall performance. Efficient scheduling minimizes makespan, improves resource utilization, and ensures timely task completion (Jia, Fuh, Nee, & Zhang, 2007) with limited resources, interdependencies, and coordination among stages. Researchers develop various algorithms to optimize scheduling based on criteria like makespan, throughput, and workload balance (Li, Pan, Gao, et al., 2021). Effective scheduling optimizes resources, streamlines processes, and enhances competitiveness (Yang, Wang, & Xu, 2022). One of the extensively researched subjects is the distributed permutation flowshop scheduling problem(DPFSP), which has a wide range of application backgrounds (Pan, Gao, Xin-Yu, & Jose, 2019). One prominent variant for the DPFSP is the DMNWPFSP, which has garnered significant interest from researchers and practitioners (Pan, Fatih Tasgetiren, & Liang, 2008; Shao, Shao, & Pi, 2021). In actual manufacturing process, both waiting and no-wait constraints need to be considered simultaneously (Cheng, Ying, Li, & Hsieh, 2019). This unique characteristic presents new challenges in scheduling and resource allocation, necessitating innovative approaches to optimize production efficiency and meet real-world production requirements (Allahverdi, Ng, Cheng, & Kovalyov, 2008). In the canned food processing industry, operations such as procurement, sorting, trimming, cleaning, peeling, and shelling are typically carried out in the early stage. These operations are all regular operation and there can be waiting between them. Other operations such as adding syrup, degassing, sealing, sterilizing, and refrigerating are usually performed after the ingredients

* Corresponding author
E-mail liyuanzhen@lcu.edu.cn (Y.-Z. Le)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2024 Growing Science Ltd.
doi: 10.5267/j.ijiec.2023.12.007

have undergone preliminary processing and are placed into cans. These operations need to be carried out immediately after the preliminary processing to ensure the quality and safety of the ingredients. Another example is the pharmaceutical industry, where the production of medications typically involves multiple processes, including raw material mixing, reactions, filtration, drying, granulation, packaging, etc. Some processes need to be no-waiting to ensure the quality and safety of the medications. For example, processes such as mixing, reactions, filtration, and drying need to be performed sequentially without waiting. It is important to note that whether a factory needs to adopt a mixed no-wait processing approach depends on its specific circumstances and requirements. The above example serves as an illustration, and actual decisions should be based on comprehensive evaluations of factors such as production processes, resource utilization, cost-effectiveness, and technological feasibility (Li, Pan, & Mao, 2016; Li, Pan, He, et al., 2022).

This study focuses on the DMNWPFSP with the criterion of completion time (Wang, Li, Ruiz, & Sui, 2018). In this study, we introduce a mixed integer linear programming (MILP) model aimed at addressing the challenges posed by DMNWPFSP. The MILP model provides a systematic and rigorous approach to addressing the complexity of the problem. An improved iterative greedy algorithm (IIG) is proposed, which significantly improves the effectiveness of the heuristic algorithm. Notably, enhancements are observed in the generation of initial solutions, the utilization of evolutionary strategies, and the implementation of local search operations. As a result of these improvements, the IIG algorithm produces high-quality solutions, which is a key advance in solving DMNWPFSP. And the algorithm was verified through experimentation. Experimental results show that the IIG algorithm performs well under the DMNWPFSP instance and can efficiently find high-quality solutions. This further solidifies the practicality and usefulness of the algorithm in practical applications.

In Section 2 of this paper, a comprehensive review of the literature related to DMNWPFSP is presented. Section 3 introduces the DMNWPFSP, providing its definition, examples, and highlighting its key characteristics. Following that, Section 3 provides a detailed description of the improved Iterated Greedy (IG) algorithm. In Section 4, we discuss the fine-tuning of parameters for the IIG algorithm to achieve optimal performance. To assess its effectiveness, Section 5 presents a thorough comparison of the IIG algorithm against four state-of-the-art approaches. Lastly, Section 6 concludes this paper by summarizing the main findings and offers valuable suggestions for future research directions.

2. Literature Review

Although not been extensively studied, the DMNWFSP is an extension of the DPFSP and the No-Wait Permutation Flowshop Scheduling Problem (NWPFSP), both of which have been well-researched. Therefore, we will focus on DPFSP and NWFSP in this section. The DPFSP deals with a scenario where f identical factories are organized in flowshops configuration with m machines. A crucial challenge is to determine the allocation of n jobs to the appropriate factory and their processing sequence in each factory. Nader and Ruiz (Naderi & Ruiz, 2010) first studied the DPFSP. They proposed 6 MILP models, 2 assignment rules, and 14 heuristic algorithms. Gao and Chen (2011) proposed a GA-based algorithm for the DPFSP. It minimizes makespan using specialized operators and efficient local search. Extensive experiments on Taillard instances (TAILLARD, 1990) show significant improvements over existing methods. Gao *et al*. (2013) introduce an advanced Tabu search algorithm which utilizes a novel Tabu strategy and enhanced local search. It is worth noting that the Tabu search algorithm exhibits higher efficiency compared to the hybrid genetic algorithm (Gao & Chen, 2011). Victor and Jose (Fernandez-Viagas & Framinan, 2014) presented a bounded-search iterated greedy algorithm for solving the DPFSP. Wang *et al.* (2016) introduced a hybrid discrete cuckoo search (HDCS) algorithm for DPFSP. It uses permutation encoding and a specialized variable neighborhood search (VNS) for efficient exploration. Simulations confirm the HDCS's effectiveness compared to existing algorithms on various instances. Wang *et al*. (2013) introduced an Estimation of Distribution Algorithm for the DPFSP. They developed a probabilistic model to characterize the solution space and effectively explored high-quality solutions by updating the model using improved solutions. Deng and Wang (2017) introduced the Competitive Memetic Algorithm (CMA) as a solution to the multi-objective DPFSP. The CMA utilizes two populations, objective-specific operators, and a competitive mechanism. Pan *et al*. (2019) proposed four intelligent optimization algorithms and three heuristics to solve the DPFSP with minimizing the total flowtime. Meng et al. (2022) and Meng, Zhang, Ren, Zhang, & Lv (2020) utilized their proposed MILP models and constraint programming approaches to address the minimization of makespan in the distributed flexible job shop scheduling problems (DFJSP) and the distributed hybrid flow shop scheduling (DHFSP) with sequence-dependent setup times. In their study, Li *et al*. (2021) explored the DPFSP with mixed no-idle constraints. They also introduced a total delay rule to address the problem (Li, Pan, Ruiz, & Sang, 2022). Their research aims to find efficient algorithms to solve these complex scheduling problems. Additionally, Rossi and Nagano (2021) proposed a MILP formulation in their work. Their algorithm employs an IG strategy designed to optimize the solution to the problem and provides an efficient solution.

The NWFSP is a significant variant of the flowshop scheduling problem and has become a hot topic (Pei, Zhang, Zheng, & Wan, 2019). Researchers are devoted to solving the NWFSP by developing innovative algorithms and heuristic methods to improve production efficiency and resource utilization. The findings in this field provide valuable guidance for scheduling decisions in practical production scenarios. RöCK proof that the NWPFSP is NP-hard when the number of machines exceed 2. As the scale of the problem increases, traditional methods such as branch and bound or mixed integer programming become impractical. To address this, researchers have devised efficient heuristic algorithms. Aldowaisan and Allahverdi (2004) introduced novel heuristics for the m-machine no-wait flowshop problem with total completion time criterion. Subsequently, a hybrid discrete particle swarm optimization (HDPSO) algorithm (Pan, Wang, Tasgetiren, & Zhao, 2007) was proposed for the NWPFSP. It combines a simplified calculation method for makespan, and a fusion of discrete particle swarm optimization (DPSO) and local search. Experimental results demonstrate the superiority of HDPSO over single DPSO and existing HPSO

algorithms in terms of quality, robustness, and efficiency. A Discrete Harmony Search (DHS) algorithm (Gao, Pan, & Li, 2011) was proposed to minimize total flow time for the NWPFSP. It incorporates a new heuristic for harmony initialization and novel pitch adjustment rules. Experimental results validate DHS's effectiveness in solving the NWPFSP. Ye *et al.* (2017) presented an average idle time heuristic to optimize the NWPFSP with makespan criterion. Deng *et al.* (2020) formulated the NWPFSP based on the total process time criterion, and proposed a population-based iterated greedy algorithm (PBIG) to solve the sorting subproblem. Recently, to effectively solve the distributed heterogeneous NWPFSP, Li *et al*. (2021) proposed four neighborhood search operators and used them in the discrete artificial bee colony algorithm for searching neighborhoods in the employed bee stage and the onlooker bee stage. Miyata and Nagano (2021) introduced the distributed wait-free flowshop scheduling problem with sequence-dependent setup time and maintenance operations to minimize makespan. Allali *et al*. (2022) presented three naturally inspired meta-heuristics: genetic algorithm, artificial bee colony algorithm and migratory bird optimization algorithm for the distributed NWPFSP.

From the above overview, DPFSP and NWPFSP are studied by many people. However, there are no research on the DPFSP considering mixed no-wait constraints. In this paper, the DMNWPFSP is studied. A mathematical model is established, and a heuristic algorithm is proposed. This paper presents the IIG algorithm and validates its practicality. To assess the high efficiency of the IIG algorithm, we conducted a comparative study with four state-of-the-art algorithms from the existing literature.

3. Problem Description

The DMNWFSP can be succinctly explained as follows. A total of *n* jobs are required to be processed, which will be distributed among *f* factories with the same series of processing machines. The key objective is to determine the optimal job assignment for each factory for efficient scheduling. In each factory, a job is processed along the same route, for example, following the same route from machine 1 to machine m.In DMNWFSP, the processing time required by job *j* (where *j* belongs to the set $\{1,2,\dots,n\}$) on machine i (where i belongs to the set $\{1,2,\dots,m\}$) is denoted by $P_{i,j}$. Due to production process limitations, machines are divided into two types: no-wait machines and regular ones. The set of regular machines is denoted as $\mathbb{M}^w$, and the number of regular machines is denoted as $\xi = |\mathbb{M}^w|$. Regular machines divide no-wait machines into several groups, which is denoted as $\cup_{r=1}^{q} \mathbb{M}^r$. In the context of the problem, we define $\mathbb{M}^r$ as the r^{th} group of no-wait machines, and q denotes the number of such no-wait machine groups. A no-wait machine group consists of at least two machines, i.e., $|\mathbb{M}^r| \geq 2, r \in \{1,2,\dots,q\}$. It is obvious that $\mathbb{M}^r \cap \mathbb{M}^{r'} = \emptyset (r \neq r')$. For example, assume M_2 and M_3, M_6–M_8, are no-wait machines in a ten-machine DMNWFSP. Therefore, the machine group can be divided into $\mathbb{M}^1 = \{M_2, M_3\}$, $\mathbb{M}^2 = \{M_6, M_7, M_8\}$, $\mathbb{M}^w = \{M_1, M_4, M_5, M_9, M_{10}\}$.
Table **1** lists of Symbols.

Table 1
Symbol definition.

Notations	
$\boldsymbol{n}$	Total number of jobs.
$\boldsymbol{m}$	Number of machines per factory.
$\boldsymbol{f}$	Number of factories.
$\boldsymbol{j}$	Index for jobs, $j \in \{1,2,\dots,n\}$.
$\boldsymbol{i}$	Index for machines, $i \in \{1,2,\dots,m\}$.
$\boldsymbol{k}$	Index for factories,$k \in \{1,2,\dots,f\}$.
$\boldsymbol{J}$	The set of jobs, $J = \{J_1, J_2, \dots, J_n\}$.
$\boldsymbol{M}$	The set of machines, $M = \{M_1, M_2, \dots, M_m\}$.
$\boldsymbol{F}$	The set of factories, $F = \{F_1, F_2, \dots, F_f\}$.
$\boldsymbol{l}$	Index for job positions in each factory, $l \in \{1,2,\dots,n_k\}$.
$\boldsymbol{O_{i,j}}$	The operation of job *j* on machine *i*.
$\boldsymbol{P_{i,j}}$	The processing time of $O_{i,j}$.
$\mathbb{M}^{\boldsymbol{w}}$	Set of regular machines.
$\boldsymbol{\xi}$	Number of regular machines, $\xi = |\mathbb{M}^w|$.
$\boldsymbol{q}$	Number of no-wait machine groups.
$\mathbb{M}^r$	The r^{th} group of no-wait machines, $r = \{1,2,\dots,q\}$.
$\mathbb{M}^a$	The set of the first machine in all no-wait machine group, $\mathbb{M}^a = \{\mathbb{M}^1_{[1]}, \mathbb{M}^2_{[1]}, \dots \mathbb{M}^r_{[1]}\}$.
$\boldsymbol{C_{i,k,l}}$	The completion time of the job in position *l* on machine *i* in factory *k*.
$\boldsymbol{C_{max}}$	The completion time of a scheduling.
Decision variables	
$\boldsymbol{X_{j,k,l}}$	Binary variable: This binary variable takes the value 1 if job *j* is assigned to location *l* in factory *k*; otherwise, it takes the value 0.

3.1 The Mixed-Integer Linear Programming Model

To solve DMNWFSP, we propose the MILP model.

Objective:

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

subject to:

$$\sum_{l=1}^{n}\sum_{k=1}^{f} X_{j,k,l} = 1\,, \forall j \tag{2}$$

$$\sum_{j=1}^{n} X_{j,k,l} \leq 1, \forall l, k \tag{3}$$

$$C_{1,k,1} = \sum_{j=1}^{n} X_{j,k,1}\, P_{1,j}, \forall j, k \tag{4}$$

$$C_{i,k,l} \geq C_{i,k,l-1} + \sum_{j=1}^{n} X_{j,k,l}\, P_{i,j}, \forall i, k, l > 1 \tag{5}$$

$$C_{i,k,l} \geq C_{i-1,k,l} + \sum_{j=1}^{n} X_{j,k,l}\, P_{i,j}, \forall M_j \in \mathbb{M}^w \cup \mathbb{M}^a \tag{6}$$

$$C_{i,k,l} = C_{i-1,k,l} + \sum_{j=1}^{n} X_{j,k,l}\, P_{i,j}, \forall M_j \in M - \mathbb{M}^w - \mathbb{M}^a \tag{7}$$

$$C_{max} \geq C_{i,k,l}, \forall i, k, l \tag{8}$$

$$C_{i,k,l} > 0, \forall i, k, l \tag{9}$$

$$X_{j,k,l} \in \{0,1\}, \forall j, k, l \tag{10}$$

The goal is to minimize C_{max}, which is described by Eq. (1). Constraint set (2) ensures that each job occurs only once in all factories. Constraint set (3) ensure that each position is occupied by at most one job. Constraint set (4) represents the completion time of the job at the first position on the first machine. The constraint set (5) enforces that a job can only start processing on a particular machine after a previous job on the same machine has completed. This ensures that jobs are sequenced correctly on individual machines. The set of constraints (6) specifies that each operation of a job must start only after its previous operation has completed. This ensures that jobs are processed in the correct sequence and follow their specific route through the machine. The constraint set (7) guarantees that there is no waiting time between any two consecutive no-waiting machines. This restriction plays an important role in satisfying the problem properties. Constraint set (8) defines the completion time of a schedule. Constraint sets (9) and (10) define the value ranges of the intermediate and decision variables.

3.2 An Illustrative Example

We consider an example in which there are two factories ($f = 2$), four machines ($m = 4$), and eight jobs ($n = 8$). Where $\mathbb{M}^1 = \{M_2, M_3\}$, $\mathbb{M}^{\mathrm{w}} = \{M_1, M_4\}$. The feasible solution of DMNWFSP is as follows: For factory 1, the job schedule is $x_{1,1,1} = x_{3,1,2} = x_{5,1,3} = x_{7,1,4} = 1$, while the other decision variables for that factory are set to zero. This means that jobs 1, 3, 5, and 7 are processed sequentially in factory 1. Likewise, for factory 2, the job schedule is $x_{2,2,1} = x_{4,2,2} = x_{6,2,3} = x_{8,2,4} = 1$, and all other decision variables for factory 2 are zero. Therefore, jobs 2, 4, 6, and 8 are processed sequentially in factory 2. Among them, the time of $\boldsymbol{O_{i,j}}$ is shown in Table 2. By implementing this solution, we ensure that the constraints are met.

Table 2
Processing times $P_{i,j}$ of jobs on machines 1-4

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
M_1	3	3	3	6	6	5	6	5
M_2	2	3	2	5	7	3	5	5
M_3	4	4	5	5	5	2	4	4
M_4	7	5	5	2	5	2	4	4

In Fig. 1, the Gantt chart illustrates the DMNWPFSP constraints. When processing jobs on $M_2(\mathbb{M}^1_{[1]})$ and $M_3(\mathbb{M}^1_{[2]})$, there

cannot be jobs waiting for processing, that is, the waiting time between machines is 0. As a result, the makespan of the scheduling is calculated as $C_{4,1,5} + p_{4,7} = 29 + 4 = 33$.

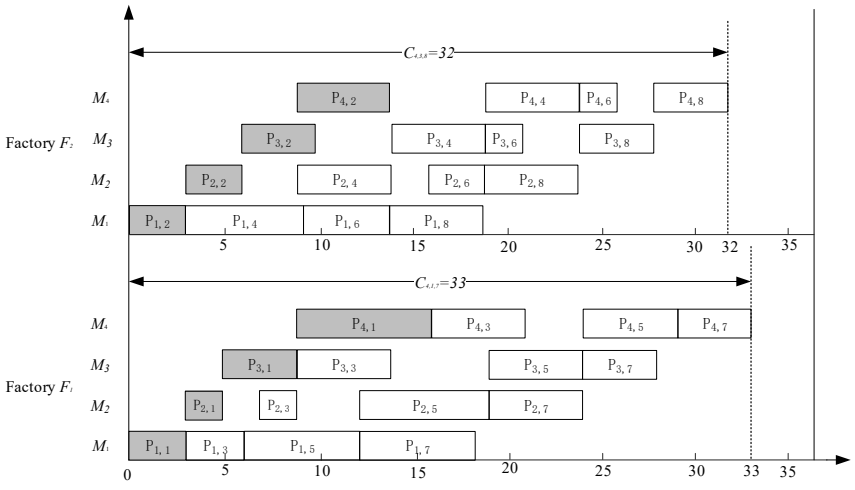

Fig. 1. Gantt chart of a solution for the illustrative example

3.3 Proposed Metaheuristics

The Iterated Greedy algorithm (IG) has shown good performance for various scheduling problems and their variants (Deng et al., 2020; Ruiz, Pan, & Naderi, 2019; Glass et al., 1994). The IG is a simple but effective algorithm with two solutions: the best solution and the current solution. Some researchers have proposed IG methods to solve the DPFSP and achieved good results (Li et al., 2019; Li, Pan, Ruiz, et al., 2022). We have made appropriate modifications on the basic IG algorithm based on the characteristics of the considered problem. Generally, the IG is divided into four stages: initialization stage, destruction stage, construction stage, and acceptance criterion. In the first stage, we first generate an initial solution using the heuristic algorithms described in Section 3.5. Then, we perform subsequent operations on this initial solution. During the two-step process of the second and third stages of the algorithm, certain elements are selected and removed from the current solution and temporarily stored, which is known as the destruction stage. The extracted elements are then reinserted into a solution containing only some of the elements. This reinsertion takes place according to predefined rules, and this process is called the construction stage. Once the construction is complete, acceptance criteria come into play. The acceptance criteria evaluate the new solution obtained in the current iteration and determines what to do with it. Depending on the circumstances, the algorithm may choose to accept the new solution as is or discard it entirely. In addition, in order to enhance the prevent from falling into local optimum, local search operators based on shift, swap and hybrid operators are introduced. This local search operator enables the algorithm to explore different regions of the solution space, thereby improving the quality of the obtained solutions. In the following sections, these modifications will be elaborated to gain a comprehensive understanding of their respective roles in enhancing the effectiveness of the algorithm.

3.4 Solution Representation

A reasonable solution representation should be able to fully express the constraints of the problem and the structure of the solution, while reducing the complexity of the search space as much as possible (Pan, Gao, Xin-Yu, et al., 2019). A two-dimensional array is used to represent jobs assigned to factories (Pan, Gao, Wang, et al., 2019). The array contains f rows, each representing job sequences $\pi_k = (\lambda_k)$ assigned to factory k. Jobs within each sequence are arranged in the order of processing. Hence, a solution is denoted as $sol = (\pi_1, \pi_2, \ldots, \pi_f)$, using a sequence-based notation commonly employed in scheduling literature due to its intuitive and easily programmable nature (Pan, Zhao, & Qu, 2008). The solution for the example in section 3.2 can be expressed as $sol = (\pi_1, \pi_2)$, where $\pi_1 = (1,3,5,7)$, and $\pi_2 = (2,4,6,8)$.

3.5 Heuristic Algorithm

The quality of randomly generated solutions is generally not high. Heuristics based on the problem characteristics are crucial. The Distributed SDH+Dipak algorithm (DSD) shown in **Algorithm 1** combines the initialization method of the SDH algorithm with the Dipak algorithm (Trietsch & Baker, 1993). The job with the largest standard deviation of all machining times is inserted preferentially in the SDH+Dipak algorithm to ensure the quality of the initial solution. First, the standard deviation σ_j of each job at all machining times is calculated (Line 1). $\bar{P}_j$ is the average processing time of job j on all machine, calculated as the total processing time of job j divided by the number of machines. A temporary job permutation λ is obtained according to the non-descending order of σ_j (Line 2). The first f jobs in λ are put into f empty factories (Lines 3 and Lines 4). The remaining jobs in λ are removed in turn and try to insert the optimal location (Lines 5-18). When there are two or more

jobs in each factory, except for the newly inserted job in factory k, all jobs in that factory are inserted into the best position among the other π_{k^*} positions (Lines 14-17). Finally, the completion time C_{max} are updated.

Algorithm 1. DSD

1: Compute $\sigma_j = \sqrt{\frac{\sum_{i=1}^{m}(P_{i,j}-\overline{P_j})}{m}}$ for each job $j \in N$;
2: Generate job permutation λ=(λ_1,λ_2,…,λ_n) according to non-descending order of σ_j;
3: π=(π_1,π_2,…,π_f),where $\pi_k = (\lambda_k)$, $k = 1,2, \dots, f$;
4: Remove jobs λ_1,λ_2,…,λ_f from λ;
5: **While** sizeof(λ) > 0 **do** //SDH+Dipak enumeration procedure
6: Extract the first job j from λ;
7: **for** $k = 1$ to f **do**
8: Test job j at all possible positions of π_k;
9: Δ_k = minimum increase of makespan;
10: ξ_k = position resulting in Δ_k;
11: **endfor**
12: $k^* = \arg(\min \Delta_k)$;
13: Insert job j at position ξ_{k^*} of π_{k^*};
14: **If** sizeof(π_k) > 2 **do**
15: All job except j in factory k^* are extracted sequentially, try to insert into other positions in factory k^* and inserted into the best position;
16: **endif**
17: **endwhile**
18: **return** $sol = (\pi_1, \pi_2, \dots, \pi_f)$;

The DNEH algorithm (Pan, Gao, Wang, et al., 2019) can also generate high-quality initial solutions which is shown in **Algorithm 2**. Initially, the total processing times for each job on every machine, denoted as T_i, are computed (line 1). Subsequently, a temporary job arrangement λ, is created based on the non-descending order of T_i (line 2). The first f jobs from λ are then chosen and allocated to f available factories, resulting in the formation of π_k (lines 3 and 4). The remaining jobs in λ are sequentially removed and inserted in their best positions (Lines 5-14). Finally, the completion time C_{max} is updated accordingly.

Algorithm 2. DNEH

1: Compute $T_i = \sum_{i=1}^{m} P_{i,i}$ for each job $i \in N$;
2: Generate job permutation λ=(λ_1,λ_2,…,λ_n) according to non-descending order of T_i;
3: π=(π_1,π_2,…,π_f),where $\pi_k = (\lambda_k), k = 1,2, \dots, f$;
4: Remove jobs λ_1,λ_2,…,λ_f from λ;
5: **While** sizeof(λ) > **0 do**
6: Extract the first job j from λ;
7: **for** $k = 1$ to f **do**
8: Test job j at all possible positions of π_k;
9: Δ_k = minimum increase of makespan;
10: ξ_k = position resulting in Δ_k;
11: **endfor**
12: $k^* = \arg(\min \Delta_k)$;
13: Insert job j at position ξ_{k^*} of π_{k^*};
14: **endwhile**
15: **return** $sol = (\pi_1, \pi_2, \dots, \pi_f)$;

In the IIG algorithm, the parameter "$init$" determines which heuristic algorithm to use. Specifically, when $init = 0$, the DNEH algorithm is used, and when $init = 1$, the DSD algorithm is employed for initialization.

3.6 Reference-based Local Search

RLS (Reference Local Search) is a reference-based local search algorithm that has demonstrated favorable performance in solving diverse problems (Pan, Gao, Wang, et al., 2019; Pan & Ruiz, 2014). This method employs local search techniques to explore new solutions within a confined search space, thereby avoiding the exhaustive exploration of the entire solution space. After obtaining the initial solution via a heuristic algorithm, RLS extracts jobs one by one according to the job order given in sol and reinserts them in all possible locations to form a new solution. Until all jobs have been tried and no further improvements can be made.

Algorithm 3. RLS(sol)

$\lambda = \emptyset$;
for $k = 1$ to f **do**
  Append all jobs in factory k to λ;
endfor
$Counter = 0; j = 1$;
While $Counter < n$ **do**
  $sol^{-\lambda_j}$ = partial solution generated by extracting job λ_i from sol;
  sol' = best solution generated after reinserting job λ_i to $sol^{-\lambda_j}$;
  if $C_{max}(sol') < C_{max}(sol)$ **then**
    $sol = sol'$; Counter = 0;
  else
    Counter = Counter + 1;
  endif
  $j = (j + 1)\%n + 1$;
endwhile
return $sol = (\pi_1, \pi_2, \dots, \pi_f)$;

3.7 Destruction Stage

The destruction stage is used to perturb solutions to provide diversity. In general, the IG algorithms randomly select jobs from the current solution to remove during the destruction stage. This paper proposes a novel removal strategy for key factories, defined as those with the longest completion time. The strategy involves randomly selecting half $d/2$ of the jobs from the critical factory and half $d/2$ of the jobs from the remaining factories, and then removing these selected jobs from the solution. The removed jobs are placed in a temporary sequence ω. For this process, please refer to **Algorithm 4**.

Algorithm 4. Destruction(sol, d)

$\omega = \emptyset$;
Find the critical factory F_c with the maximum makespan in sol;
Randomly extract $d/2$ jobs in factory F_c;
Append these $d/2$ jobs to ω;
While sizeof(ω) $< d$ **do**
  Randomly select a factory k;
  Randomly extract a job J_j in factory k;
  Append job J_j to ω;
endwhile
return sol and ω;

3.8 Construction Stage

During the reconstruction phase, the jobs in the temporary sequence ω are sorted according to the sorting method in the **Algorithm 1**. Then, each job in the sequence ω is inserted into the solution one by one. Finally, we choose the best positions p, where each job is inserted to minimize the increase in makespan, and repeat this process until all removed jobs are inserted. The details of this stage are presented in **Algorithm 5**.

Algorithm 5. Construction(sol, ω)

Compute $\sigma_j = \sqrt{\frac{\sum_{i=1}^{m}(P_{i,j} - \overline{P_j})}{m}}$ for job j in ω;
Generate job permutation λ=(λ_1,λ_2,…,λ_d) according to non-descending order of σ_j;
for $start = 1$ to sizeof(ω) **do**
  Extract the first job j from λ;
  for $k = 1$ to f **do**
    Test job j at all possible positions of π_k;
    Δ_k = minimum increase of makespan;
    ξ_k = position resulting in Δ_k;
  endfor
  $k^* = \arg(\min_{k=1,2,\dots,f} \Delta_k)$;
  Insert job j at position ξ_{k^*} of π_{k^*};
endfor
return sol;

3.9 Local Search

High-quality local search leads to better solutions without searching the entire solution space. Based on the characteristics of the DMNWPFSP and the characteristics of the IG algorithm, we define two operators, namely, Shift and Swap.

（1） The operation process of Shift can be shown by **Algorithm 6**. It randomly selects a job j_1 in the critical factory F_c and moves job j_1 to another randomly selected position p in all factories.

Algorithm 6. Shift(sol)

1: Find the critical factory F_c with the maximum makespan in sol;
2: Randomly select a job j_1 in factory F_c;
3: Randomly select a factory k
4: Randomly select a position p in factory k;
5: Shift job j_1 to position p;
6: **return** sol;

（2） The Swap operator shown in **Algorithm 7** randomly selects a job j_1 in the critical factory F_c and another random job j_2 in all factories and then swaps jobs j_1 and j_2.

Algorithm 7. Swap(sol)

1: Find the critical factory F_c in sol;
2: Randomly select a job j_1 in factory F_c;
3: Randomly select a factory k;
4: Randomly select a job j_2 in factory k;
5: Swap job j_1 and job j_2;
6: **return** sol;

To search across a wider range of regions, the LocalSearchShift algorithm (presented in Algorithm 8) and the LocalSearchSwap algorithm (presented in Algorithm 9) are proposed to repeatedly operater (Shift and Swap) on the solution *sol*. After each operation, the current solution is compared with the old one. If the new solution is better, the local search will save it as the current best solution. Otherwise, the solution *sol* will continue to be operated on until the loop ends after *OperNumber* times, where *OperNumber* is a parameter.

Algorithm 8 LocalSearchShift(sol)

1: **for** $i = 0$ to *OperNumber* **do** to
2: $sol' = sol$;
3: $sol' =$ Shift(sol'); %(use Algorithm 6)
4: **if** $C_{max}(sol') < C_{max}(sol)$ **do**
5: $sol = sol'$;
6: **endif**
7: **endfor**

Algorithm 9. LocalSearchSwap(sol)

1: **for** $i = 0$ to *OperNumber* **do** to
2: $sol' = sol$;
3: $sol' =$ Swap(sol'); %(use Algorithm 7)
4: **if** $C_{max}(sol') < C_{max}(sol')$ **do**
5: $sol = sol'$;
6: **endif**
7: **endfor**

As shown in Algorithm 10, Hybrid Local Search (HLS) is a combination of shift and swap operators used in the local search algorithm to maintain diversity during the search. The algorithm randomly selects between LocalSearchShift and LocalSearchSwap algorithms for neighborhood search with a probability of 0.5 for each. This allows the search to explore different neighborhood structures and can avoid getting stuck in local optima. By using a combination of operators, the algorithm can exploit the strengths of each operator while minimizing their weaknesses. Overall, the use of HLS can improve algorithm performance

Algorithm 10. HLS(sol)

```
r=(double)random(0,1);
if r < 0.5 do
  LocalSearchShift(sol);          %(use Algorithm 8)
Else
  LocalSearchSwap(sol);           %(use Algorithm 9)
endif
```

3.10 Acceptance Criteria

Acceptance criteria play a vital role in determining whether to accept a new solution as a starting point for the next iteration. In the case of the constant temperature T (Stützle) simulated annealing algorithm, if the objective function value is inferior, the acceptance criterion shown in Equation (11) is applied.

$$T = TempFactor \times \frac{\sum_{j=1}^{n}\sum_{i=1}^{m} \boldsymbol{P_{i,j}}}{10 \times m \times n} \tag{11}$$

where *TempFactor* is the control parameter.

Algorithm 11. AcceptanceCriterion(sol', sol)

```
if C_max(sol') < C_max(sol) do
  sol = sol';
  if C_max(sol') < C_max(sol_best) do
    sol_best = sol';
  endif
else
  if rand(0,1) < exp ((C_max(sol) − C_max(sol')/T)) do   %(Calculate T use Formula (11))
    sol = sol';
  endif
endif
return;
```

3.11 Flow Chart for the IIG Algorithm

The specific process of the IIG algorithm is summarized in Fig. 2.

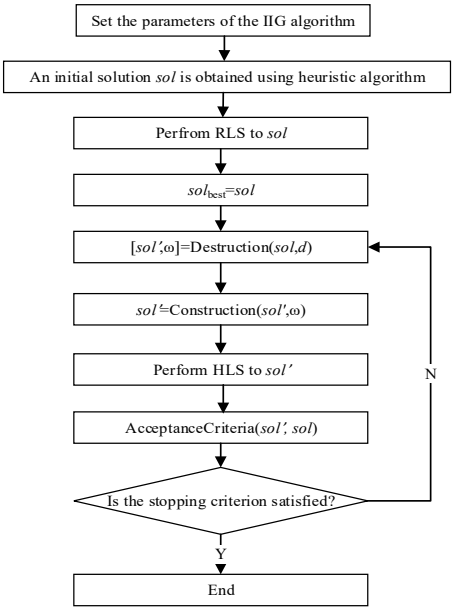

Fig. 2. The flowchart of the IIG for the DMNWPFSP

Step 1: Set parameters, including d, *TempFactor* , *Init* and *OperNumber*;
Step 2: An initial solution (denoted as sol) is generated.
Step 3: Perform Reference-based local search to the solution sol.
Step 4: The optimal solution sol_{best} is recorded.
Step 5: Destruction: remove d jobs, put them in ω, and store the new solution in sol'。
Step 6: Construction: insert the jobs in ω into sol'.
Step 7: Perform HLS to sol'.
Step 8: Acceptance Criterion: decide whether to accept the new solution.
Step 9: If the stop condition is met, the IIG algorithm ends; otherwise, it returns to step 5.

4. Experimental Calibration

For achieving the best performance of the IIG algorithm, we conducted calibration experiments to fine-tune its parameters. We utilized the testing methodology proposed in the literature (Cheng, Ying, Chen, & Lu, 2018) to generate instances for evaluation. A test instance consisted of n jobs and f factories, with m machines in each factory, where $f \in \{2, 4, 6\}$, $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. Below are the families of no-wait machines of the seven types generated from the paper (Pan & Ruiz, 2014). 1) Series 1: The top 50% of machines are subject to no-wait constraints, while the remaining 50% operate as regular machines. 2) Series 2: The bottom 50% of machines are assigned no-wait constraints. 3) Series 3: Machines are alternately assigned regular constraints and no-wait constraints in a sequential manner. 4) Series 4: 25% of machines are randomly selected and given no-wait constraints. 5) Series 5: 50% machines are randomly selected, no waiting time for operation. 6) Series 6: 75% of machines are randomly assigned with no-wait constraints. 7) Series 7: All machine constraints are no-wait. In each instance, the no-wait machine group categories were randomly generated to maximize the accuracy of the experiment.

Table 3 provides details of the example parameters used in the experiments. The IIG algorithm was implemented using Microsoft Visual Studio 2019 and the C++ programming language, with all optimization flags enabled. The computer used for the experiments runs on the Windows 10 Pro operating system and is equipped with a quad-core Intel i7-12700 2.1 GHz processor and 16 GB of RAM. Using Relative Percentage Increment (RPI) as an indicator allows us to intuitively select the most suitable algorithm or optimize parameters to obtain better solutions. RPI is calculated using the following Eq. (12):

$$RPI = \frac{C_{max} - C_{max}^{*}}{C_{max}^{*}} \times 100\% \tag{12}$$

The parameter settings mentioned above include two parameters: C_{max} and C_{max}^{*}. Here, C_{max} represents the performance metric of the algorithm on a specific instance, while C_{max}^{*} represents the best value of that metric achieved by optimizing or tuning on the same instance. This RPI measure allows for a more comprehensive assessment of the algorithm's performance (Li, Gao, Meng, Jing, & Zhang, 2023).

Table 3
Parameters of instances.

Number of jobs	n	{20,50,100,200,500}
Number of machines	m	{5,10,20}
Number of factories	f	{2,4,6}
The processing time of $O_{i,j}$	$P_{i,j}$	[0,100]
Seven different families of no-wait machines	Family_x	{1,2,3,4,5,6,7}

The IIG involves four parameters that may affect its performance: d, *TempFactor*, *Init* and *OperNumber*. The value of d represents the number of jobs removed in Section 3.7. *TempFactor* represents the probability of accepting a solution in Section 3.10. The *Init* parameter indicates the selection of the DNEH heuristic algorithm, or the DSD heuristic algorithm described in Section 3.5. *OperNumber* represents the number of times the selected operator is applied as described in Section 3.9. For the IIG algorithm, we first determine an approximate range for the four parameters through preliminary experiments. Subsequently, we conduct calibration experiments to find the optimal value for each parameter. The four parameters of the IIG algorithm are set as follows: *d:* 2, 4 and 6, *TempFactor*: 0.4, 0.6 and 0.8, *Init:* 0 and 1 and *OperNumber:* 50, 60 and 70. Therefore, there are 3×3×2×3 = 54 configurations for the IIG algorithm.

In the calibration experiment, there are three values for variable f (2, 4, 6), and nine combinations for variables m and n: 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×5, 200×10, and 200×20. There are total of 3×9=27 combinations. Five instances are generated for each combination. Finally, we obtained a total of 3×9×5=135 benchmark instances. These 135 instances calibrated the proposed IIG algorithm. For each instance, the experiment was run two times independently. The CPU time, when the IIG algorithm terminates, was $v \times m \times n$ ms, where v=90. Therefore, there are 135×2×54 = 14580 RPI results. In the experiments presented in this paper, we employed the method of experimental design and analysis of variance (ANOVA) (Li, Pan, & Tasgetiren, 2014; Zhang et al., 2020) to analyze the obtained results. ANOVA is a widely-used statistical technique for comparing the average differences among multiple groups. Our focus in the analysis was on the algorithm type, rather than the problem size factors (n, f, and m). In ANOVA, the algorithm type is considered as the controlled factor, while the variables

f and *m* are treated as uncontrolled variables. The ANOVA results are shown in Table 4, *d*, *TempFactor* , *Init* and *OperNumber* cause the response variables to differ statistically significantly; that is, to optimize and improve the performance of the IIG algorithm, it is crucial to carefully select and adjust these parameters.

Table 4
ANOVA table for the experiment on tuning the parameters of IIG

Main effects					
A:*d*	959.084	2	479.542	1113.50	0.0000
B:*TempFactor*	173.222	2	86.6108	201.11	0.0000
C:*Init*	5.69317	1	5.69317	13.22	0.0003
D:*OperNumber*	0.78477	2	0.392385	0.91	0.4021
RESIDUAL	6275.59	14572	0.430661		
TOTAL (CORRECTED)	7414.37	14579			

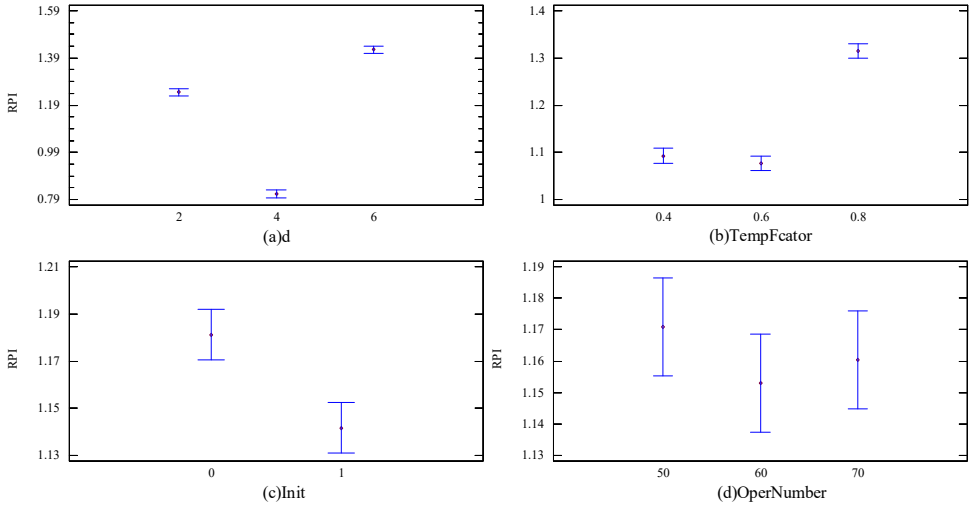

Fig. 3. Means and 95.0% Tukey HSD confidence intervals

It can be seen from Fig. 3 (a) that the optimal value of *d* should be 4. Fig. 3 (b) shows that *TempFactor* should be fixed to 0.6. As illustrated in Fig. 3 (c), when comparing the performance of the DSD algorithm and the DNEH algorithm, it is observed that the DSD algorithm is more effective in solving the DMNWPFSP problem. Therefore, the parameter selection of *Init* should be 1. The number of operations on the selection factor during the local search, should be 60, as shown in Fig. 3 (d). According to the analysis of the experimental results, the four parameters of the IIG algorithm are set as follows: $d = 4$, *TempFactor* = 0.6, *Init* = 1 and *OperNumber* = 60.

5. Experimental Results

The proposed IIG algorithm was evaluated by a large number of numerical comparisons. In the final experiment, $f \in \{2, 4, 6\}$ and there are 12 combinations of 20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, 500×20 for *n* and *m*. Each combination contained ten instances, which resulted in a total of 12×10×3=360 instances. The dataset and the kind of no-wait machine group for the final experiment were generated using the methods in Section 4. We compared the proposed IIG with EA (Fernandez-Viagas, Perez-Gonzalez, & Framinan, 2018), IG (Pan, Gao, Wang, et al., 2019), DABC (Pan, Gao, Wang, et al., 2019), and ILS (Pan, Gao, Wang, et al., 2019). All algorithms are coded using the same environment as Section 4. We note that the four competitive algorithms compared did not consider the mixed no-wait constraint, and their initial optimization goal was not the completion time. Therefore, we made necessary modifications to these algorithms to fit the DMNWPFSP. In addition, we only made changes to the necessary code to try to maintain all details of the competitive algorithms to ensure and achieve the performance they should have. The parameters of all competitive algorithms are tabulated in Table 5.

Table 5
Parameter setting for IG, ILS, DABC, and EA.

EA	$\gamma = 5$;
DABC	$PS = 5$; $\Xi = 0$;
ILS	$\tau = 3$; $a = 0$; $\varpi = 20$; $\Xi = 0$; $\beta = 0.7$;
IG	$d = 7$; $a = 0$; $\beta = 0.6$;

In the experiments conducted in this paper, we utilized the same operating environment as described in Section 4. All algorithms ran the same termination time and output results. The termination time was $v \times m \times n$ ms, where v took the 3 values of 30, 60 and 90, and m and n please refer to **Table 1**, respectively. The comparison of the results of all algorithms under these three termination conditions comprehensively demonstrated the performance of all algorithms. All algorithms independently ran five times to solve each of the 360 instances. Therefore, 360× 5 × 3 × 5= 27000 tests are conducted. The RPI was used as a performance indicator for comparison.

In Tables 6, 7, and 8, we provide the average (ARPI) values for the algorithms. Specifically, **Table 6** shows the values obtained by the algorithm by plant, machine, and job classification at a termination time of 30mn ms. The IIG algorithm exhibits superior performance, as indicated by its high ARPI value. This is followed by the IG algorithm, which ranks second in terms of ARPI. Furthermore, it can also be seen that the IIG algorithm consistently exhibits superior performance across different categories.

Table 6
ARPI value at 30mn ms termination time (The optimal ARPI is in bold)

type	DABC	EA	ILS	IG	IIG
***f*=2**	2.606	2.525	2.842	1.955	**0.653**
***f*=4**	3.049	3.182	3.825	2.631	**0.657**
***f*=6**	3.522	3.489	4.302	3.183	**0.787**
***n*=20**	0.849	0.388	0.584	0.203	**0.073**
***n*=50**	2.749	2.880	3.731	2.333	**0.724**
***n*=100**	4.219	4.391	5.091	3.525	**0.926**
***n*=200**	4.673	4.926	5.602	4.403	**1.092**
***n*=500**	3.912	3.957	4.456	4.085	**1.034**
***m*=5**	2.665	2.693	3.267	2.203	**0.597**
***m*=10**	3.347	3.361	3.961	2.772	**0.769**
***m*=20**	3.065	3.051	3.646	2.676	**0.703**
MEAN	3.059	3.065	3.656	2.590	**0.699**

In our study, we conducted a multivariate analysis of variance to assess the significance of various factors presented in Table 6. These factors include algorithm type, the size of the job, number of machines and factories.

Fig. **4** presents the statistics for multiple comparisons at a termination time of 30mn ms. The results of our analysis indicate significant differences in the ARPI values among different algorithms. Specifically, the IIG algorithm demonstrated the highest performance, followed by the IG algorithm. In the study, we observed that although the ILS algorithm is relatively weak, compared with the other two algorithms, the performance gap was not statistically significant. These findings provide valuable insights for further exploration of algorithm performance and demonstrate the utility of the IIG algorithm for the problems under consideration.

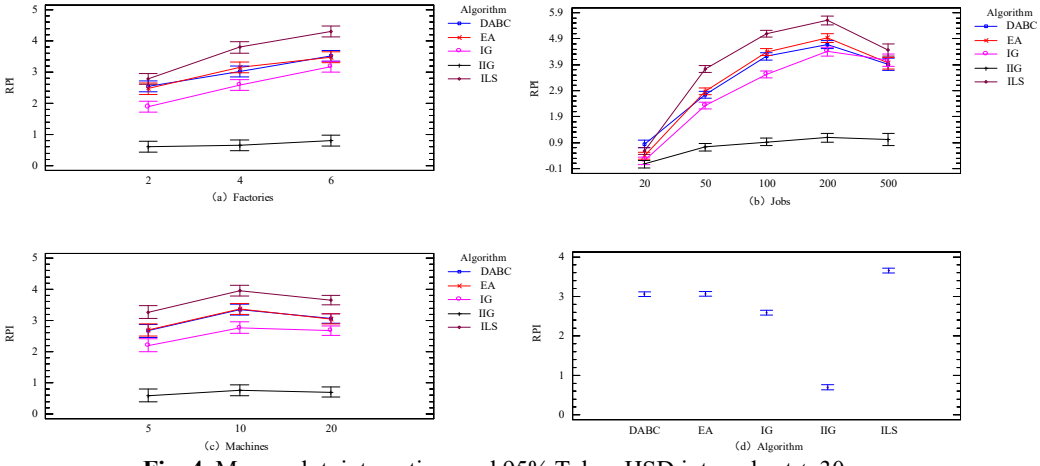

Fig. 4. Means plot, interaction, and 95% Tukey HSD intervals at t=30mn ms

Tables 7 and 8 show the ARPI values of various factors at termination time of 60mn and 90mn ms, respectively. Figs 8 and 9 illustrate the mean and interaction plots at termination times of 60mn ms and 90mn ms, respectively. The ranking of all algorithms is still maintained, with IIG continuing to outperform other comparison algorithms by a significant margin. The ARPI differences of all algorithms are still significant.

Table 7
ARPI value at 60*mn* ms termination time (The optimal ARPI is in bold)

type	DABC	EA	ILS	IG	IIG
***f*=2**	2.350	2.249	2.557	1.631	**0.375**
***f*=4**	2.790	2.864	3.560	2.287	**0.382**
***f*=6**	3.270	3.162	4.105	2.874	**0.455**
***n*=20**	0.773	0.335	0.431	0.125	**0.030**
***n*=50**	2.492	2.344	3.359	1.976	**0.467**
***n*=100**	3.764	3.967	4.802	3.110	**0.567**
***n*=200**	4.388	4.659	5.392	3.872	**0.585**
***n*=500**	3.772	3.841	4.332	3.794	**0.487**
***m*=5**	2.380	2.330	3.003	1.918	**0.371**
***m*=10**	3.079	3.010	3.706	2.411	**0.453**
***m*=20**	2.837	2.814	3.411	2.354	**0.385**
MEAN	2.803	2.758	3.408	2.264	**0.404**

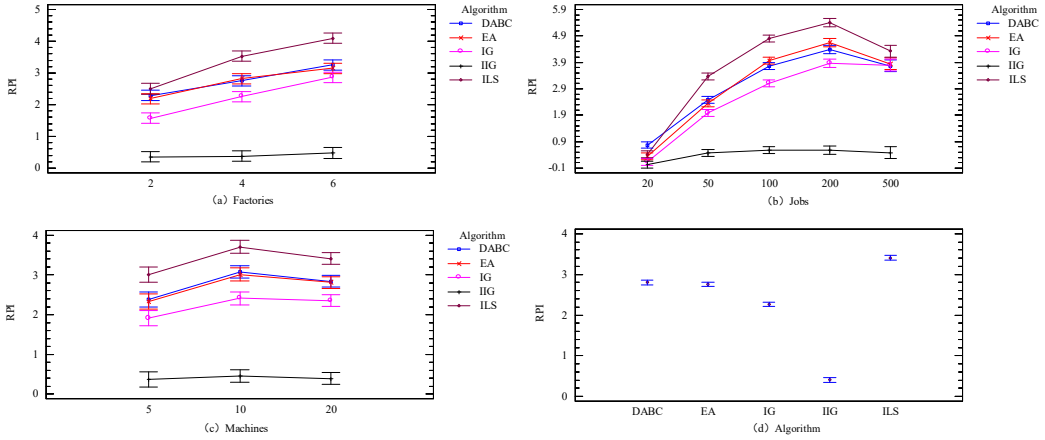

Fig. 5. Means plot, interaction, and 95% Tukey HSD intervals at *t*=60*mn* ms

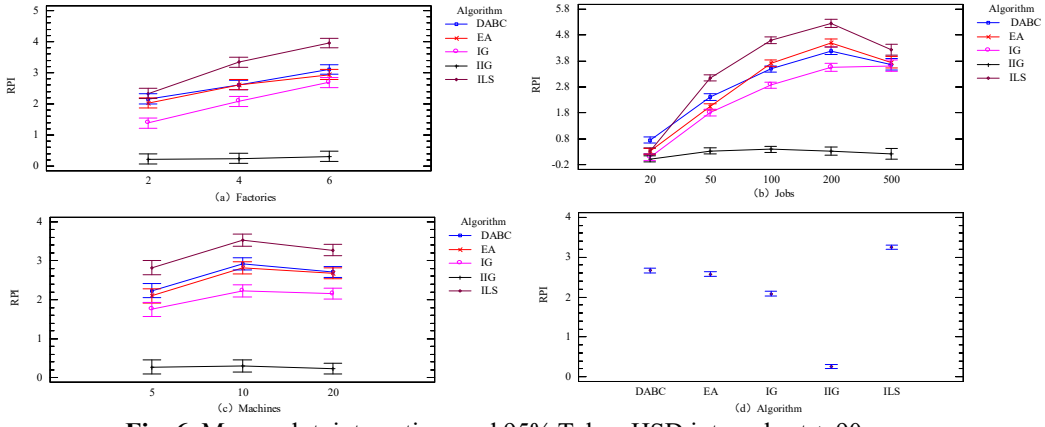

Fig. 6. Means plot, interaction, and 95% Tukey HSD intervals at *t*=90*mn* ms

Table 8
ARPI value at 90*mn* ms termination time (The optimal ARPI is in bold)

type	DABC	EA	ILS	IG	IIG
***f*=2**	2.219	2.095	2.391	1.437	**0.241**
***f*=4**	2.651	2.667	3.377	2.113	**0.246**
***f*=6**	3.129	2.970	3.976	2.699	**0.288**
***n*=20**	0.750	0.314	0.331	0.083	**0.020**
***n*=50**	2.403	2.034	3.146	1.803	**0.334**
***n*=100**	3.494	3.716	4.599	2.868	**0.389**
***n*=200**	4.193	4.492	5.253	3.558	**0.326**
***n*=500**	3.668	3.749	4.240	3.618	**0.216**
***m*=5**	2.235	2.102	2.818	1.753	**0.262**
***m*=10**	2.927	2.815	3.534	2.230	**0.289**
***m*=20**	2.716	2.672	3.278	2.163	**0.231**
MEAN	2.666	2.577	3.248	2.083	**0.258**

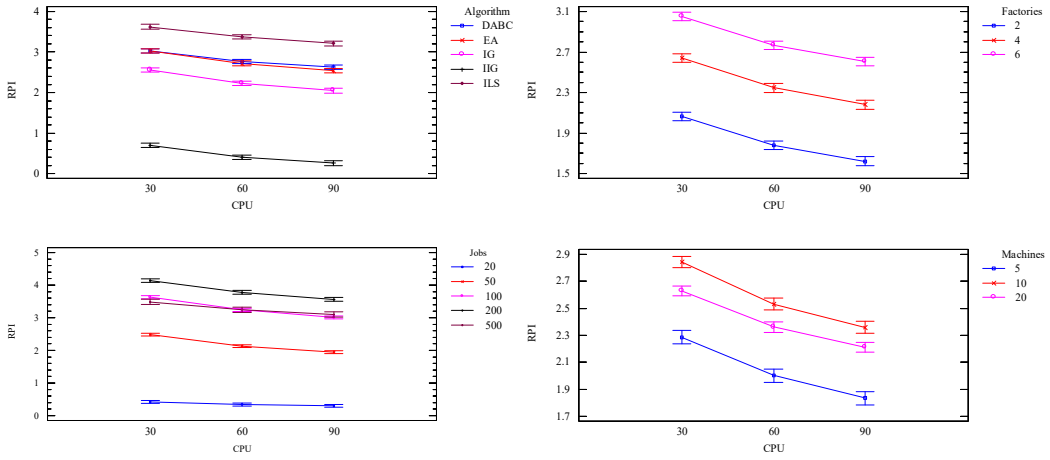

Fig. 7. Interaction of CPU time and various influencing factors and 95% Tukey HSD intervals

Fig. 7 presents the performance comparison of all algorithms under different termination times. As the problem complexity rises, particularly for larger instances, the algorithm's performance typically declines due to the increased complexity of the search space, making it more challenging to find optimal solutions. However, the results clearly demonstrate that with longer termination times, all algorithms achieve improved performance. Notably, the proposed IIG algorithm consistently outperforms the four contrasting algorithms in solving the DMNWPFSP with makespan criterion, even with longer termination times. This highlights the excellent performance and superiority of the IIG algorithm in solving the complexity of DMNWPFSP and obtaining high-quality solutions.

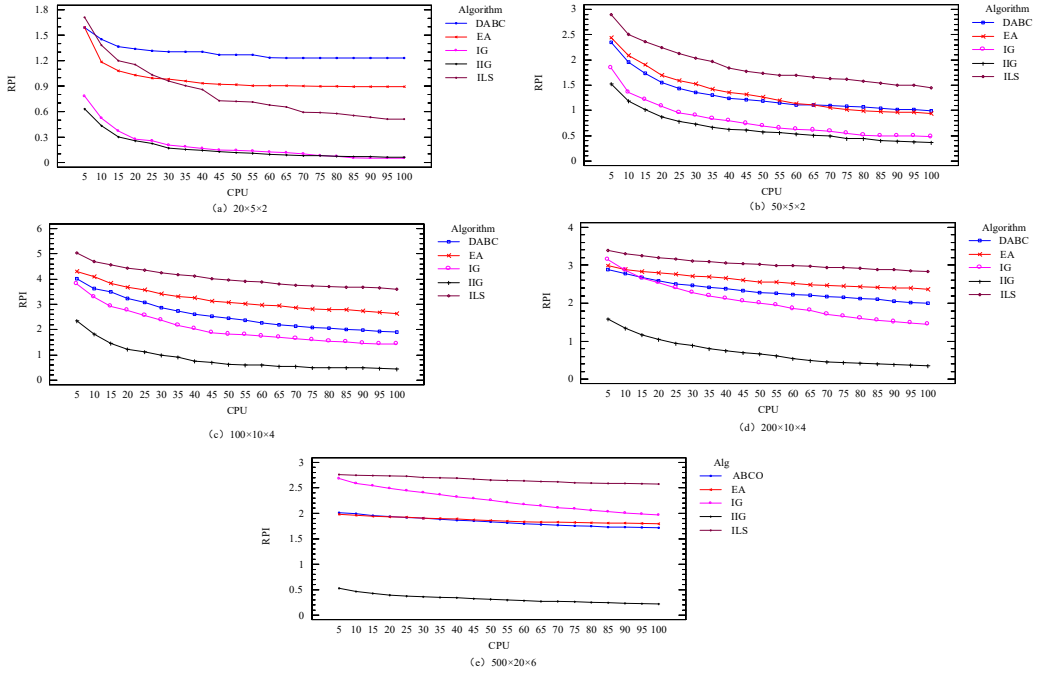

Fig. 8. Evolutionary curve for all algorithms.

To further demonstrate how the IIG algorithm evolves over time and how its performance changes, we conducted a series of experiments and analyzed the IIG algorithm evolution curve. To ensure the fairness and practicality of the experiments, we designed 5 different combinations of $n \times m \times f$ with different scales, which are 20×5×2, 50×5×2, 100×10×4, 200×10×4, and 500×20×6, respectively. We generated 10 instances for each combination. All algorithms run independently 5 times to solve each instance. We used CPU time to measure the efficiency of the IIG algorithm. **Fig. 8** shows the changing trend of the average RPI value of all algorithms. As the scale of the instances increases, the time taken by all algorithms to reach a relatively stable peak performance also increases. Moreover, the superior performance of the IIG algorithm becomes more and more evident as compared to the other algorithms. Therefore, in our experiments, we set the CPU termination time of the algorithm to three different values: $v \in \{30, 60, 90\}$. For the largest instance of 500×20×6, all algorithms can reach a relatively stable performance at v=90.

6. Conclusions

This paper introduces an IIG algorithm designed to address the challenges of the DMNWPFSP. The algorithm incorporates critical factory destruction and employs an iterative local search mechanism. Initially, the unique characteristics of the DMNWPFSP are thoroughly examined, and a location-based mathematical model is formulated to describe it. Subsequently, the IIG algorithm utilizes the DSD heuristic algorithm to generate a better initial solution. Then, by utilizing the framework of the IG algorithm and integrating a powerful local search mechanism, the solution space is thoroughly explored, enhancing its ability to find better solutions. The efficacy of the IIG algorithm is extensively evaluated through a series of numerical experiments. Performance evaluation and comparison show that the IIG algorithm achieves better solutions, demonstrating its superiority in solving the given problem and outperforming the other four recently proposed algorithms. Furthermore, the IIG algorithm consistently produces high-quality feasible solutions, even under varying stopping conditions.

Future research in the field of DPFSP should prioritize gaining a comprehensive understanding of the problem, as well as the development of advanced evolutionary algorithms and the design of metaheuristic algorithms tailored specifically for DPFSP. In terms of algorithms, future work should be combined with other algorithms to improve exploration capabilities. Furthermore, the problem can be extended to the distributed flexible flowshop.

Acknowledgments

This research is partially supported by the National Natural Science Foundation of China 52205529, the Natural Science Foundation of Shandong Province (ZR2021QE195), and the Discipline with Strong Characteristics of Liaocheng University --Intelligent Science and Technology under Grant 319462208.

References

Aldowaisan, T., & Allahverdi, A. (2004). New heuristics for m-machine no-wait flowshop to minimize total completion time. *Omega, 32*(5), 345-352.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research, 187*(3), 985-1032.

Allali, K., Aqil, S., & Belabid, J. (2022). Distributed no-wait flow shop problem with sequence dependent setup time: Optimization of makespan and maximum tardiness. *Simulation Modelling Practice and Theory, 116*.

Cheng, C.-Y., Ying, K.-C., Chen, H.-H., & Lu, H.-S. (2018). Minimising makespan in distributed mixed no-idle flowshops. *International Journal of Production Research, 57*(1), 48-60.

Cheng, C.-Y., Ying, K.-C., Li, S.-F., & Hsieh, Y.-C. (2019). Minimizing makespan in mixed no-wait flowshops with sequence-dependent setup times. *Computers & Industrial Engineering, 130*, 338-347.

De Giovanni, L., & Pezzella, F. (2010). An Improved Genetic Algorithm for the Distributed and Flexible Job-shop Scheduling problem. *European Journal of Operational Research, 200*(2), 395-408.

Deng, G., Su, Q., Zhang, Z., Liu, H., Zhang, S., & Jiang, T. (2020). A population-based iterated greedy algorithm for no-wait job shop scheduling with total flow time criterion. *Engineering Applications of Artificial Intelligence, 88*.

Deng, J., & Wang, L. (2017). A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm and Evolutionary Computation, 32*, 121-131.

Fernandez-Viagas, V., & Framinan, J. M. (2014). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 53*(4), 1111-1123.

Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. M. (2018). The distributed permutation flow shop to minimise the total flowtime. *Computers & Industrial Engineering, 118*, 464-477.

Gao, J., & Chen, R. (2011). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems, 4*(4), 497-508.

Gao, J., Chen, R., & Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 51*(3), 641-651.

Gao, K.-z., Pan, Q.-k., & Li, J.-q. (2011). Discrete harmony search algorithm for the no-wait flow shop scheduling problem with total flow time criterion. *The International Journal of Advanced Manufacturing Technology, 56*(5-8), 683-692.

Glass, C. A., Gupta, J. N. D., & Potts, C. N. (1994). Lot streaming in three-stage production processes. *European Journal of Operational Research, 75*, 378-394.

Jia, H. Z., Fuh, J. Y. H., Nee, A. Y. C., & Zhang, Y. F. (2007). Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. *Computers & Industrial Engineering, 53*(2), 313-320.

Li, J.-Q., Pan, Q.-K., & Tasgetiren, M. F. (2014). A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Applied Mathematical Modelling, 38*(3), 1111-1132.

Li, J.-Q., Pan, Q.-K., & Mao, K. (2016). A Hybrid Fruit Fly Optimization Algorithm for the Realistic Hybrid Flowshop Rescheduling Problem in Steelmaking Systems. *IEEE Transactions on Automation Science and Engineering, 13*(2), 932-949.

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. *International Journal of Advanced Robotic Systems, 16*(5).

Li, H., Li, X., & Gao, L. (2021). A discrete artificial bee colony algorithm for the distributed heterogeneous no-wait flowshop scheduling problem. *Applied Soft Computing, 100*.

Li, Y.-Z., Pan, Q.-K., Li, J.-Q., Gao, L., & Tasgetiren, M. F. (2021). An Adaptive Iterated Greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems. *Swarm and Evolutionary Computation* (Vol. 63).

Li, Y.-Z., Pan, Q.-K., Gao, K.-Z., Tasgetiren, M. F., Zhang, B., & Li, J.-Q. (2021c). A green scheduling algorithm for the distributed flowshop problem. *Applied Soft Computing, 109*.

Li, Y.-Z., Pan, Q.-K., He, X., Sang, H.-Y., Gao, K.-Z., & Jing, X.-L. (2022). The distributed flowshop scheduling problem with delivery dates and cumulative payoffs. *Computers & Industrial Engineering, 165*.

Li, Y.-Z., Pan, Q.-K., Ruiz, R., & Sang, H.-Y. (2022). A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Knowledge-Based Systems* (Vol. 239).

Li, Y.-Z., Gao, K., Meng, L., Jing, X.-L., & Zhang, B. (2023). Heuristics and metaheuristics to minimize makespan for flowshop with peak power consumption constraints, *International Journal of Industrial Engineering Computations, 14*(2), 221-238.

Lu, C., Liu, Q., Zhang, B., & Yin, L. (2022). A Pareto-based hybrid iterated greedy algorithm for energy-efficient scheduling of distributed hybrid flowshop. *Expert Systems with Applications, 204*.

Meng, L., Gao, K., Ren, Y., Zhang, B., Sang, H., & Chaoyong, Z. (2022). Novel MILP and CP models for distributed hybrid flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 71*.

Meng, L., Zhang, C., Ren, Y., Zhang, B., & Lv, C. (2020). Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & Industrial Engineering, 142*.

Miyata, H. H., & Nagano, M. S. (2021). Optimizing distributed no-wait flow shop scheduling problem with setup times and maintenance operations via iterated greedy algorithm. *Journal of Manufacturing Systems, 61*, 592-612.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research, 37*(4), 754-768.

Pan, Q.-K., Fatih Tasgetiren, M., & Liang, Y.-C. (2008). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research, 35*(9), 2807-2839.

Pan, Q.-K., Gao, L., Wang, L., Liang, J., & Li, X.-Y. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications, 124*, 309-324.

Pan, Q.-K., Gao, L., Xin-Yu, L., & Jose, F. M. (2019). Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. *Applied Soft Computing, 81*.

Pan, Q.-K., & Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega, 44*, 41-50.

Pan, Q.-K., Wang, L., Tasgetiren, M. F., & Zhao, B.-H. (2007). A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology, 38*(3-4), 337-347.

Pan, Q.-K., ZHAO, B.-H., & Qu, Y.-g. (2008). Heuristics for the No-Wait Flow Shop Problem with Makespan Criterion. *Chinese Journal of Computers, 31*(7), 1147-1154.

Pei, Z., Zhang, X., Zheng, L., & Wan, M. (2019). A column generation-based approach for proportionate flexible two-stage no-wait job shop scheduling. *International Journal of Production Research, 58*(2), 487-508.

RöCK, H. (1984). The Three-Machine No-Wait Flow Shop Is NP-Complete. *Journal of the ACM, 31*(2), 336-345.

Rossi, F. L., & Nagano, M. S. (2021). Heuristics and iterated greedy algorithms for the distributed mixed no-idle flowshop with sequence-dependent setup times. *Computers & Industrial Engineering, 157*.

Ruiz, R., Pan, Q.-K., & Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega, 83*, 213-222.

Shao, W., Shao, Z., & Pi, D. (2021). Effective constructive heuristics for distributed no-wait flexible flow shop scheduling problem. *Computers & Operations Research, 136*.

Stützle, T. Applying Iterated Local Search to the Permutation Flow Shop Problem. *Technical Report AIDA-98-04, FG Itellektik, FB Informatik, TU Darmstadt.*

TAILLARD. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research, 47(1)*, 65–74.

Trietsch, D., & Baker, K. R. (1993). Basic Techniques for Lot Streaming. *Operations Research, 41*(6), 1065-1076.

Wang, J., Wang, L., & Shen, J. (2016). A Hybrid Discrete Cuckoo Search for Distributed Permutation Flowshop Scheduling Problem. *IEEE Congress on Evolutionary Computation*, 2240-2246.

Wang, S.-y., Wang, L., Liu, M., & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics, 145*(1), 387-396.

Wang, Y., Li, X., Ruiz, R., & Sui, S. (2018). An Iterated Greedy Heuristic for Mixed No-Wait Flowshop Problems. *IEEE Trans Cybern, 48*(5), 1553-1566.

Yang, S., Wang, J., & Xu, Z. (2022). Real-time scheduling for distributed permutation flowshops with dynamic job arrivals using deep reinforcement learning. *Advanced Engineering Informatics, 54*.

Ye, H., Li, W., & Abedini, A. (2017). An improved heuristic for no-wait flow shop to minimize makespan. *Journal of Manufacturing Systems, 44*, 273-279.

Zhang, B., Pan, Q.-K., Gao, L., Meng, L.-L., Li, X.-Y., & Peng, K.-K. (2020). A Three-Stage Multiobjective Approach Based on Decomposition for an Energy-Efficient Hybrid Flow Shop Scheduling Problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50*(12), 4984-4999.