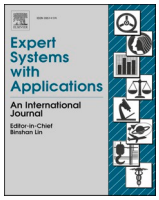
Expert Systems With Applications 233 (2023) 120830

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



An iterated greedy algorithm for distributed flowshops with tardiness and rejection costs to maximize total profit

Zhen Lin [a], Xue-Lei Jing [b,*], Bao-Xian Jia [a,*]

[a] School of Computer Science and Technology, Liaocheng University, Liaocheng 252000, PR China
[b] Network Information Center, Liaocheng University, Liaocheng 252000, PR China

ARTICLE INFO

Keywords:
Distributed flowshops
Iterated greedy algorithm
Total profit
Local search

ABSTRACT

Distributed permutation flowshop scheduling problems (DPFSP) have become research hotspots. However, a DPFSP with tardiness and rejection costs (DPTR), which has important practical significance, has not yet been addressed. As manufacturers are most concerned about the maximization of the total profit, this study addresses the DPFSP to maximize the total profit. A mixed-integer linear programming model is used to describe the DPTR. An iterated greedy algorithm named IG_TR is proposed. IG_TR generates initial solutions using an improved NEH heuristic using four rules based on the deadline, due date, tardiness and rejection costs. For IG_TR, a dynamic number of jobs are removed from factories in the destruction phase. In the reconstruction phase, the removed jobs and those waiting to be processed are selected to be inserted into appropriate positions in the factories. Two local search methods are designed to improve the quality of solutions generated in the reconstruction phase. During the acceptance phase, a restart mechanism is proposed to avoid falling into local optimal. The experimental results of the component analysis demonstrate the effectiveness of IG_TR. Several comprehensive experiments show that IG_TR outperforms five related algorithms.

1. Introduction

With economic globalization, multi-factory cooperation is considered the mainstream trend of current industrial production (Ruiz, Pan, & Naderi, 2019, Fu et al., 2021). The multi-factory production method is a type of distributed scheduling that has recently gained prominence in current research (Naderi, & Ruiz, 2010, Deng, & Wang, 2017). Makespan and total flowtime criteria are commonly used in the distributed permutation flowshop scheduling problem (DPFSP) literature. The maximization of total profit is one of the most important objectives for manufacturers. In a real manufacturing system, tardiness and rejection costs are incurred because of order changes, late deliveries, resource constraints, and other issues. However, the maximization of total profit for a DPFSP with tardiness and rejection costs (DPTR) has not been investigated in the literature. Therefore, this study investigates the DPTR, which has important practical significance. A typical example is the aerospace industry, where it is difficult for a single company to produce an aircraft, and multiple suppliers are typically required to deliver many jobs. When a company fails to complete a job by its due date but can complete it by its deadline, the job is delivered late and a tardiness costs is incurred. When a company completes a job after its deadline, the job is rejected, and a rejection costs is incurred. Another example is the electronics manufacturing industry. Customers want the next generation of electronic products as soon as possible because new products are being updated quickly. As the capacity of the company/companies cannot meet the demand of customers, it will lead to an imbalance of supply and demand. The reasons for this phenomenon are raw material shortage, insufficient resources, equipment failure, etc. These factors cause the production requirements of customers to be unmet, resulting in some orders being delayed or even unable to be completed.

Among the scheduling problems involving tardiness and rejection costs, the single-machine scheduling problem with generalized due dates is one of the main research directions in this field (Mosheiov, Oron, & Shabtay, 2021, Gerstl, & Mosheiov, 2020). Unlike the above studies, in this study, each job has its due date, tardiness and rejection costs; however, all jobs have a common deadline. As the single-machine scheduling problem with tardiness is NP-hard (Steiner, & Zhang, 2011), the DPTR is also NP-hard because the DPTR is composed of multiple factories corresponding to multiple machines. As a result, one of the most promising algorithms for solving the DPTR is a *meta*-heuristic. The iterated greedy (IG) algorithm is one of the most successful

* Corresponding authors.
E-mail addresses: 1051708812@qq.com (Z. Lin), jingxuelei@lcu.edu.cn (X.-L. Jing), jiabaoxian@lcu.edu.cn (B.-X. Jia).

https://doi.org/10.1016/j.eswa.2023.120830
Received 25 September 2022; Received in revised form 10 June 2023; Accepted 11 June 2023
Available online 30 June 2023

meta-heuristics for scheduling problems (Dubois-Lacoste, Pagnozzi, & Stützle, 2017), which has also been applied to the DPFSP (Wang, & Wang, 2019). Therefore, we propose an IG algorithm named IG_TR to solve the DPTR. In the phase of generating the initial solution, we improve the NEH heuristic (Nawaz, Enscore, & Ham, 1983) and name the improved NEH heuristic PNEH. A new feature of the DPTR is that jobs that cannot be scheduled for production are placed in an alternative sequence. Therefore, we present some strategies for optimizing the solution in the destruction and reconstruction phases, employ five operators to further optimize the solution, and propose a restart mechanism to avoid falling into local optimal in the acceptance phase.

The rest of this paper is organized as follows. Section 2 reviews the relevant literature related to the DPFSP and flowshop scheduling problems with tardiness and rejection costs. Section 3 describes the DPTR. Section 4 details the IG_TR algorithm. Section 5 explains the calibration experiment. In Section 6, comprehensive comparisons are given for the IG_TR algorithm with five competitive algorithms from related literature. Finally, Section 7 gives conclusions.

2. Literature review

To our knowledge, the DPTR has not been mentioned in the literature. Hence, we review related contributions, including the DPFSP and flowshop scheduling with tardiness and rejection costs.

2.1. Distributed permutation flowshop scheduling problems

The makespan and total flowtime criteria have been extensively discussed for the DPFSP in the literature. Other objectives or multiple objectives have also been studied for the DPFSP.

To minimize the makespan, Naderi and Ruiz (2010) presented six different mathematical models, 14 heuristics, and two variable neighborhood descent (VND) methods VNDa and VNDb. They used 720 large test cases to validate these models and algorithms. Gao and Chen (2011) presented a hybrid genetic algorithm (HGA) with new crossover and mutation operators and updated solutions for 692 large test cases. Gao et al. (2012) proposed a VND method (VND(B&B)). Wang et al. (2013) designed a probability model, proposed an estimation of the distribution algorithm (EDA), and updated solutions for 589 large test cases. Gao et al. (2013) presented a tabu search (TS) algorithm and updated solutions for 472 large test cases. Lin et al. (2013) presented an improved IG (MIG) that outperformed the HGA and TS algorithms. Xu et al. (2014) proposed a hybrid immunization algorithm by hybridizing local search operators with an immune search operator and updated solutions for 585 large test cases. Naderi and Ruiz (2014) presented a scatter search (SS) algorithm that outperformed the HGA, MIG, TS, VNDa, VNDb and VND(B&B) algorithms. Fernandez-Viagas and Framinan (2015) presented a bounded-search IG algorithm with two local search methods (BSIG) that outperformed the MIG, EDA, and TS algorithms. Zhao et al. (2022) proposed a memetic discrete differential evolution (MDDE) algorithm, which mainly introduces four adaptive local search structures. For a DPFSP with a setup time, Huang et al. (2020) presented an IG algorithm that can not only avoid falling into local optimal but can also effectively utilize the obtained information and diversified search area; Huang et al. (2021) devised a discrete artificial bee colony (DABC) algorithm. For a DPFSP with a customer order, Meng et al. (2019) proposed three algorithms, namely, IG, DABC and VND algorithms, with the IG algorithm being the most efficient. For a DPFSP with an uncertain processing time, Jing et al. (2021) proposed IG and iterated local search (ILS) algorithms. Then, Jing et al. (2022) presented an improved IG algorithm to solve the DPFSP with an uncertain processing time and a carryover sequence-dependent setup time. Mao et al. (2021) investigated a DPFSP with preventive maintenance and proposed a multi-start IG algorithm. Meng and Pan (2021) investigated a DPFSP with non-identical factories and proposed five heuristics and an enhanced DABC algorithm.

To minimize the total flowtime, Fernandez-Viagas et al. (2018) designed an evolutionary algorithm (EA). Pan et al. (2019) presented four algorithms, including DABC, SS, ILS, and IG algorithms. Mao et al. (2022) proposed a hash map-based modal algorithm for a DPFSP with preventive maintenance.

Other objectives or multiple objectives have been studied for the DPFSP. Pan, Gao, Li and Wu (2022) presented an improved Jaya algorithm to solve a lot-streaming distributed flowshop scheduling problem with makespan and total energy consumption objectives. Rifai et al. (2021) studied a distributed re-entrant permutation flowshop scheduling problem with three objectives of makespan, tardiness and production cost, and proposed a multi-objective adaptive large neighborhood search algorithm. Wang, Gao, Li and Tasgetiren (2020) investigated a DPFSP with a setup time to minimize two objectives of makespan and energy consumption, and presented a multi-objective whale swarm algorithm. Wang and Wang (2020) presented a knowledge-based cooperative algorithm for a DPFSP with makespan and total energy consumption objectives. Fu et al. (2019) investigated a DPFSP considering the total tardiness constraint and designed a multi-objective brainstorm optimization algorithm (MOBSO) to minimize the makespan and total energy consumption. Lu et al. (2023) investigated a DPFSP with limited buffers to minimize the makespan and total energy consumption and proposed a Pareto-based collaborative multi-objective optimization algorithm. Chen et al. (2023) investigated an energy-efficient distributed blocking flowshop scheduling problem (DBFSP) to minimize the makespan and total energy consumption and presented a knowledge-based iterated Pareto greedy algorithm. Li et al. (2022) proposed an IG algorithm and nine heuristics for a DPFSP with delivery dates to maximize the total payoff.

From the above review, the DPFSP has become a research hotspot. However, the DPTR, a DPFSP with tardiness and rejection costs to maximize the total profit, has not yet been addressed. The literature review of the DPFSP is shown in Table 1.

2.2. Flowshop scheduling problems with tardiness and rejection

For scheduling problems associated with tardiness, Braglia and Grassi (2009) designed a heuristic for a flowshop scheduling problem with makespan and maximum tardiness. Allahverdi et al. (2018) solved a no-wait flowshop scheduling problem using both simulated annealing and insertion algorithms to minimize the makespan and total tardiness. Jing et al. (2020) presented an IG algorithm with idle time insertion for the DPFSP with due windows to minimize total weighted earliness and tardiness. Khare and Agrawal (2020) proposed five solution schemes to solve a DPFSP with total tardiness, including two heuristics, a mathematical model, an IG algorithm, and a population-based hybrid discrete Harris hawk optimization algorithm. Pan, Ruiz and Alfaro-Fernandez (2017) investigated a hybrid flowshop problem with due windows to minimize total weighted earliness and tardiness. Huang et al. (2023) investigated a distributed assembly permutation flowshop scheduling problem (DAPFSP) and proposed an effective memetic algorithm to minimize the total tardiness. Li et al. (2022) similarly considered delivery dates when solving the DPSFP to maximize the total payoff.

For scheduling problems associated with rejection, Gerstl and Mosheiov (2020) investigated a single-machine scheduling problem to increase the number of on time jobs and decrease rejection costs. Mor et al. (2020) presented a pseudo-polynomial dynamic programming algorithm for a single-machine batch scheduling problem with an upper bound on the total permitted rejection costs to minimize the makespan, total completion time and total weighted completion time. For the parallel machine scheduling problem, Ou et al. (2015) proposed a near linear-time heuristic to minimize the total penalty of rejected jobs, and Li et al. (2015) proposed a 2-approximation algorithm to minimize the makespan with the total penalty of rejected jobs at a given bound. Liu and Lu (2020) presented three approximation algorithms for single-machine and parallel machine scheduling problems to minimize the

Table 1
The DPFS and related problems.

Criterion	References	Constraints	Algorithms
makespan	Naderi and Ruiz (2010)	DPFSP	14 heuristics, VNDa, VNDb
	Gao and Chen (2011)	DPFSP	GA_LS
	Gao et al. (2012)	DPFSP	VND(B&B)
	Wang et al. (2013)	DPFSP	EDA
	Gao et al. (2013)	DPFSP	TS
	Lin et al. (2013)	DPFSP	MIG
	Xu et al. (2014)	DPFSP	HIA
	Naderi and Ruiz (2014)	DPFSP	SS
	Fernandez-Viagas and Framinan (2015)	DPFSP	BSIG
	Zhao et al. (2022)	DPFSP	a heuristic, MDDE
	Huang et al. (2020)	DPFSP with a setup time	an IG
	Huang et al. (2021)	DPFSP with a setup time	3 heuristics, a DABC
	Meng et al. (2019)	DPFSP with a customer order	3 heuristics, a DABC, a VND, an IG
	Jing et al. (2021)	DPFSP with an uncertain processing time	an IG, an ILS
	Jing et al. (2022)	DPFSP with an uncertain processing time and a setup time	an IG
	Mao et al. (2021)	DPFSP with preventive maintenance	an IG
	Meng and Pan (2021)	DPFSP with no-identical factories	a DABC
total flowtime	Fernandez-Viagas et al. (2018)	DPFSP	EA
	Pan et al. (2019)	DPFSP	a DABC, a SS, an ILS, an IG
	Mao et al. (2022)	DPFSP with preventive maintenance	HMMA
multi-objective	Pan, Gao, Li and Wu (2022)	DPFSP with lot-streaming	Jaya
	Rifai et al. (2021)	DPFSP with re-entrant	IMOALNS
	Wang, Gao, Li and Tasgetiren (2020)	DPFSP with a setup time	MOWSA
	Wang and Wang (2020)	DPFSP	KCA
	Fu et al. (2019)	DPFSP with total tardiness constraint	MOBSO
	Lu et al., 2023	DPFSP with limited buffers	CMOA
	Chen et al. (2023)	DBFSP	KBIPG
total payoff	Li et al. (2022)	DPFSP	9 heuristics, an IG

maximum completion time and total penalties. Fu et al. (2019) investigated a DPFSP with a total tardiness constraint and proposed an MOBSO to minimize the makespan and total energy consumption.

For scheduling problems with both tardiness and rejection costs, Cordone and Hosteins (2019) investigated a single-machine scheduling problem with a deadline and proposed a branch-and-bound algorithm to minimize two objectives of total tardiness and total rejection costs. Mosheiov et al. (2021) investigated a single-machine scheduling problem with generalized due dates and an upper bound on the total permitted rejection costs and proposed a pseudo-polynomial dynamic programming algorithm to minimize the total late work and rejection costs.

From the above review, flowshop scheduling problems with tardiness and rejection costs have attracted increasing attention from scholars. However, the DPTR with the maximization of total profit has not been reported in the literature. The DPTR has many similarities with the problems studied in the above literature; however, it also has its problematic properties. In the DPTR, not all jobs may be processed, and if the expected completion time of a new job exceeds the deadline, it will not be processed. Therefore, we need to propose effective methods for solving the DPTR. A literature review of flowshop scheduling problems with tardiness and rejection costs is shown in Table 2.

3. Problem description

The DPTR is described as follows. There are n jobs $J = \{J_1, J_2, ..., J_n\}$ to be processed by f identical factories, where each factory has m machines $M = \{M_1, M_2, ..., M_m\}$. The jobs in each factory are processed on the m machines in the same order, i.e., from the first machine to the last machine. Each job has a processing time on each machine. Each job is released at time 0. At any time, each machine can process no more than one job, and each job can be processed by no more than one machine. All jobs have a common deadline Φ. The due date of job J_j is denoted as ψ_j. The completion time of J_j is denoted as C_j. If $C_j \leqslant \psi_j$, J_j is delivered on time, and the profit J_j is denoted as ρ_j. If $\psi_j < C_j \leqslant \Phi$, a tardiness costs τ_j of J_j is incurred, and $\tau_j = (C_j - \psi_j) \times \omega_j$, where ω_j denotes the unit tardiness weight of J_j, and the profit of J_j is $\rho_j - \tau_j$. If $C_j > \Phi$, a rejection costs θ_j of J_j is incurred, and J_j is assigned to the alternative sequence μ and is not scheduled. In other words, the job is rejected, and the profit of

Table 2
The flowshop scheduling problems with tardiness and rejection costs.

Criterion	References	Constraints	Algorithms
multi-objective (makespan and total tardiness)	Braglia and Grassi (2009)	flowshop scheduling	MOGLS
multi-objective (makespan and total tardiness)	Allahverdi et al. (2018)	no-wait flowshop scheduling	AA
total weighted earliness and tardiness	Jing et al. (2020)	DPFSP with due windows	IGITE
total tardiness	Khare and Agrawal (2020)	DPFSP	2 heuristics, HDHHO, IG
total weighted earliness and tardiness	Pan, Ruiz and Alfaro-Fernandez (2017)	hybrid flowshop problems with due windows	IGT, ILST
total tardiness	Huang et al. (2023)	DAPFSP	EMA
total payoff	Li et al. (2022)	DPFSP	9 heuristics, IG
the number of jobs completed exactly on time	Gerstl and Mosheiov (2020)	single machine scheduling	1 heuristic, a branch and bound algorithm
multi-objective (makespan, total completion time and total weighted completion time)	Mor et al. (2020)	single machine lot scheduling	pseudo-polynomial dynamic programming algorithms
total rejection penalty	Ou et al. (2015)	parallel machine scheduling	a near linear-time heuristic
makespan	Li et al. (2015)	parallel machine scheduling	2-approximation algorithm
multi-objective (makespan and total penalties)	Liu and Lu (2020)	single machine scheduling/ parallel machine scheduling	approximation algorithm
multi-objective (makespan and total energy consumption)	Fu et al. (2019)	DPFSP	MOBSO
total tardiness and rejection costs	Cordone and Hosteins (2019)	single-machine scheduling	a branch and bound algorithm
total late work and rejection costs	Mosheiov et al. (2021)	single-machine scheduling	pseudo-polynomial dynamic programming solution algorithm

J_j is $-\theta_j$. The total profit of all jobs is denoted as Ξ. The objective of the DPTR is to maximize Ξ. The formulation of the DPTR is described as follows.

Parameters and constants:

f: total number of factories.
n: total number of jobs.
m: total number of machines in each factory.
k: index for factories, $k = 1,2,\ldots,f$.
j: index for jobs, $j = 1,2,\ldots,n$.
i: index for machines, $i = 1,2,\ldots,m$.
F: set of factories,$F = \{F_1, F_2, \ldots, F_f\}$
J: set of jobs,$J = \{J_1, J_2, \ldots, J_n\}$
M: set of machines,$M = \{M_1, M_2, \ldots, M_m\}$
Φ: common deadline for all jobs.
$P_{i,j}$: processing time for J_j on M_i.
ψ_j: due date of J_j.
ρ_j: profit of J_j when it is delivered on time.
ω_j: unit tardiness weight of J_j when it is delivered overdue.
θ_j: rejection costs of J_j when it is rejected.
μ: the alternative sequence.
SLP: a sufficiently large positive number.
J_0: a dummy job, which requires no processing time.

Decision variables:

Ξ: total profit of all jobs.
n_k: total number of jobs in factory F_k.
q: index for the dummy job J_0 and all jobs,$q = 0, 1, \ldots, n$
$C_{i,j}$: completion time of J_j on M_i.
C_j: completion time of J_j,$C_j = C_{m,j}$
τ_j: tardiness costs of J_j when it is delivered overdue,$\tau_j = (C_j - \psi_j) \times \omega_j$

$X_{q,j,k}$: binary variable, which is 1 if the job J_j is processed immediately after the job J_q in the factory F_k, and 0 otherwise, where $j \neq q$.

$Y_{j,k}$: binary variable, which is 1 if job J_j is processed in the factory F_k, and 0 otherwise.

Z_j: binary variable, which is 1 if job J_j is in the alternative sequence μ, and 0 otherwise.

Objective function:

$$\mathbf{max}\Xi = \sum_{j=1}^{n}\left[\rho_j \times (1 - Z_j) - max(0, \tau_j) - \theta_j \times Z_j\right] \quad (1)$$

s.t.

$$\sum_{q=0,q\neq j}^{n}\sum_{k=1}^{f} X_{q,j,k} + Z_j = 1, j = 1, 2, \ldots, n \quad (2)$$

$$\sum_{k=1}^{f} Y_{j,k} + Z_j = 1, j = 1, 2, \ldots, n \quad (3)$$

$$\sum_{q=0,q\neq j}^{n}\left(X_{q,j,k} + X_{j,q,k}\right) \leqslant 2 \times Y_{j,k}\ ,\ j = 1, 2, \ldots, n, k = 1, 2, \ldots, f \quad (4)$$

$$\sum_{j=1,j\neq q}^{n}\sum_{k=1}^{f} X_{q,j,k} \leqslant 1, q = 1, 2, \ldots, n \quad (5)$$

$$\sum_{k=1}^{f} X_{0,j,k} \leqslant 1, j = 1, 2, \ldots, n \quad (6)$$

$$\sum_{k=1}^{f}\left(X_{j,q,k} + X_{q,j,k}\right) \leqslant 1, j = 1, 2, \ldots, n-1, q > j \quad (7)$$

$$X_{q,j,k} = 1 \rightarrow Y_{q,k} \wedge Y_{j,k} = 1, q = 0, 1, \ldots, n, j = 1, 2, \ldots, n, j \neq q, k = 1, 2, \ldots, f \quad (8)$$

$$C_{0,j} = 0, j = 1, 2, \ldots, n \quad (9)$$

$$Z_j = 0 \rightarrow C_{i,j} \geqslant C_{i-1,j} + P_{i,j}, j = 1, 2, \ldots, n, i = 1, 2, \ldots, m \quad (10)$$

$$Z_j = 0 \rightarrow C_{i,j} \geqslant C_{i,q} + P_{i,j} + \left(\sum_{k=1}^{f} X_{q,j,k} - 1\right) \times SLP, j = 1, 2, \ldots, n, q = 0, 1, \ldots, n, q \neq j, i = 1, 2, \ldots, m \quad (11)$$

$$Z_j = 1 \rightarrow C_{i,j} = 0, j = 1, 2, \ldots, n, i = 1, 2, \ldots, m \quad (12)$$

$$C_{m,j} \leqslant \Phi, j = 1, 2, \ldots, n \quad (13)$$

$$X_{q,j,k} \in \{0, 1\}, q = 0, 1, \ldots, n, j = 1, 2, \ldots, n, j \neq q, k = 1, 2, \ldots, f \quad (14)$$

$$Y_{j,k} \in \{0, 1\}, j = 1, 2, \ldots, n, k = 1, 2, \ldots, f \quad (15)$$

$$Z_j \in \{0, 1\}, j = 1, 2, \ldots, n \quad (16)$$

The objective function (1) aims to maximize the total profit of all jobs. Constraint (2) means that each job must be at one position in a factory or the alternative sequence. Constraint (3) means that each job must be assigned to a factory or the alternative sequence. Constraint (4) indicates that each job processed in a factory can be a predecessor or a successor. Constraint (5) indicates that each job has at most one successor. Constraint (6) indicates that each factory has at most one dummy job. Constraint (7) ensures that no job can be the predecessor and successor of any other job in any factory simultaneously. Constraint (8) indicates that when a job is a successor of another job in a factory, both jobs must be processed in the same factory. Constraint (9) defines a virtual machine for facilitating Constraint (10). Constraint (10) means that the start time of any job processed on any machine must be greater than the completion time of the job on the previous machine. Constraint (11) ensures that if the job J_j is the successor of job J_q, J_j cannot be processed on machine M_i before J_q finishes processing on M_i. Constraint (12) defines that the completion time of any job in the alternative sequence on any machine is 0. Constraint (13) ensures that the completion time of any job on the last machine is less than the common deadline. Constraints (14)–(16) define three binary variables.

An example instance is illustrated in Table 3. The instance involves 10 jobs and 2 identical factories, with each factory containing 2 machines, and the common deadline is $\Phi = 217$. IBM ILOG CPLEX Studio software offers the best solution for the instance by compiling the mathematical model, and Fig. 1 shows the scheduling Gantt chart for the best solution. Through experimental tests, the commercial solver (IBM ILOG CPLEX Studio software) takes 23 s to obtain the best solution for the example instance, and the proposed IG_TR algorithm takes only 0.4 s to obtain the best solution. After conducting extensive experiments, we found that when the number of jobs is 11, it takes 15 min to obtain the optimal solution by CPLEX software, and the proposed IG algorithm only takes 0.44 s to obtain the optimal solution. However, when the number of jobs exceeds 11, the CPLEX software is unable to obtain the optimal solution within a reasonable time.

A solution for the DPTR comprises two parts. One part is a two-dimensional array $\pi = \{\pi_1, \pi_2, \ldots, \pi_f\}$ for the processed jobs, where $k = 1,2,\ldots,f$, $\pi_k = \{\pi_{k,1}, \pi_{k,2}, \ldots, \pi_{k,n_k}\}$ denotes the sequence of jobs processed in the factory F_k, and r denotes the index for a job of π_k, $r = 1, 2, \ldots, n_k$, $\pi_{k,r} \in J$. The other part is a one-dimensional array $\mu = \{\mu_1, \ldots, \mu_{tna}\}$ for unprocessed jobs, namely, the alternative sequence, where tna denotes the total number of jobs in the alternative sequence, e represents the index for jobs of μ, $e = 1,2,\ldots,tna$, and $\mu_e \in J$ is a job that is unassigned to any factory. $tna = 0$ indicates that there is no job in the alternative sequence, $\mu = \{\}$. Note that when all jobs are processed, $\mu = \{\}$ is an empty set. For the instance, the solution is $\{\pi, \mu\}$, where $\pi = \{\pi_1, \pi_2\}$, $\pi_1 = \{2, 10, 5, 9, 3\}$, $\pi_2 = \{6, 8, 7\}$, and $\mu = \{1, 4\}$. The process of calculating the total profit is shown as follows.

Table 3
Datum for the example instance.

J_j	1	2	3	4	5	6	7	8	9	10
$p_{1,j}$	52	28	44	89	27	36	79	62	20	61
$p_{2,j}$	17	83	31	66	16	70	34	71	32	21
ρ_j	1794	5994	1500	5270	731	2226	4294	5054	2496	2296
ω_j	8	2	1	5	4	4	4	2	4	5
θ_j	897	3996	1500	2635	1462	4452	4294	5054	1664	2296
ψ_j	141	156	87	186	156	162	122	184	165	108

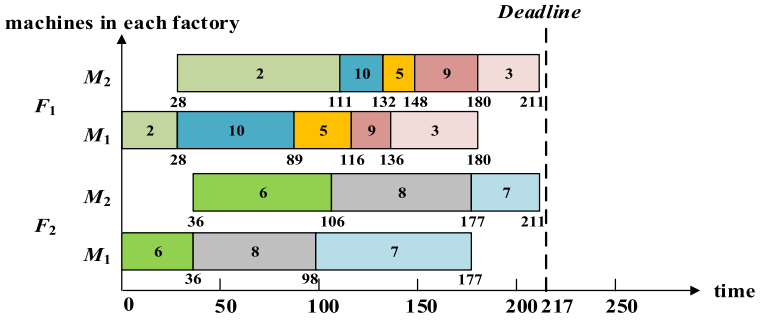


Fig. 1. A scheduling Gantt chart for the instance in Table 3.

$\Xi = [1794 \times (1–1) – 0–897 \times 1] + [5994 \times (1–0) – 0–3996 \times 0] + [1500 \times (1–0) – 124–1500 \times 0] + [5270 \times (1–1) – 0–2635 \times 1] + [731 \times (1–0) – 0–1462 \times 0] + [2226 \times (1–0) – 0–4452 \times 0] + [4294 \times (1–0) – 356–4294 \times 0] + [5054 \times (1–0) – 0–5054 \times 0] + [2496 \times (1–0) – 60–1664 \times 0] + [2296 \times (1–0) – 120–2296 \times 0] = 20399.$

4. Propose an iterated greedy algorithm

The IG algorithm is effective in solving the DPFSP (Pan et al., 2019). The conventional IG algorithm typically involves five phases: initialization solution, destruction, reconstruction, local search, and acceptance. The local search phase is optional. The NEH heuristic and its variants are typically used to generate initial solutions. During the destruction phase, some jobs are extracted from the current solution. The extracted jobs are reinserted into the solution during the reconstruction phase. The local search phase improves the solution obtained in the reconstruction phase. The acceptance phase determines whether the solution obtained in the current iteration should be accepted. An IG algorithm named IG_TR is proposed to solve the DPTR. The above five phases and the complete pseudocode for IG_TR are described in detail as follows.

4.1. Initialization solution phase

The NEH heuristic is one of the most useful heuristics for solving flowshop scheduling problems (Rad, Ruiz, & Boroojerdian, 2009). The conventional NEH heuristic prioritizes jobs on the basis of the largest processing time (LPT) rule, which gives high priority to jobs with the largest processing time. A seed sequence is generated by sorting all jobs according to the LPT rule. The jobs in the seed sequence are then removed in turn and placed in a position where the best objective value is obtained. In addition, the shortest processing time (SPT), the earliest due date (EDD), and RANDOM (all jobs are randomly sorted) rules are commonly used to generate seed sequences. Here we improve the NEH heuristic for the DPTR and call the improved heuristic PNEH. Algorithm 1 shows the pseudocode for PNEH.

Algorithm 1. PNEH (π, μ)

```
β← a seed sequence by a rule
π_k←{ },k = 1,2,...,f
π←{π_1, π_2, ..., π_k}
for loc = 1 to n do
for k = 1 to f do
    test job β[loc] in all possible positions of F_k
    EE_k is the minimum incremental tardiness costs by inserting β[loc] into F_k,
    CF_{k,β[loc]} is the completion time of F_k by inserting β[loc] into its best position
pos_k is the position where EE_k is obtained
if CF_{k,β[loc]} > Φ then
    EE_k = +∞
endif
endfor
k* ← arg(min_{k=1}^{f}(EE_k))
if EE_{k*} = +∞ then
    put β[loc] into μ
else
    insert β[loc] into π_{k*} at position pos_{k*}
endif
endfor
Output π, μ
```

A rule called PPUT is proposed, which calculates the unit profit of job J_j by $\rho_j / \sum_{i=1}^{m} P_{i,j}$ and gives the highest priority to the jobs with the highest unit profit. By comparing the above five rules through experiments, the best rule PPUT is selected to generate the seed sequence for PNEH. First, 720 test instances were generated, as described in Section 6. Then, seed sequences are generated following the above five rules, and objective function values are derived using PNEH. Relative deviation index (RDI) values are used, as in the method in Section 5, and there are $720 \times 5 = 3600$ RDI values. The analysis of variance (ANOVA) was used to test 3600 RDI values, and finally, mean plots are obtained, as shown in Fig. 2. From Fig. 2, the PPUT rule outperforms LPT, SPT, EDD and RANDOM rules.

PNEH differs from NEH2 (Naderi, & Ruiz, 2010) in that it calculates whether the completion time of the last job in a factory exceeds the deadline after inserting a new job. If the completion time of the last job

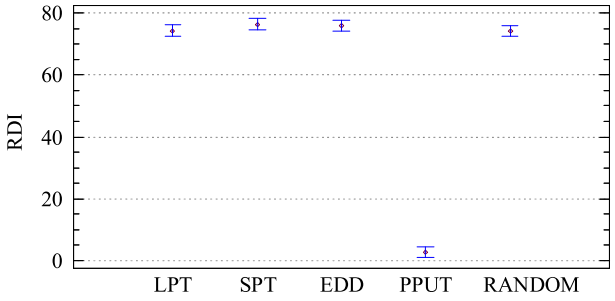


Fig. 2. Means Plots of 5 rules.

in the factory exceeds the deadline, the new job is not allowed to be processed. PNEH attempts to insert the new job into a factory that can accommodate it, calculates the incremental tardiness costs for each factory, and selects the factory with the smallest incremental tardiness costs after inserting the new job. Here is an example to illustrate PNEH. There are two factories, each factory containing two machines and six jobs. The common deadline for all jobs is 209. The processing times, due dates, unit tardiness weights, and profits for all jobs are listed in Table 4.

The best position for incremental tardiness costs of job J_j in the factory F_k is denoted as $\sigma_{k,j}$. The steps are shown as follows.

Step (1) The PPUT rule calculates the unit time profit for each job. A seed sequence $\beta = \{4, 5, 2, 1, 6, 3\}$ is generated according to the PPUT rule.

Step (2) As shown in Fig. 3 a), job 4 is the first job in β, and there is no difference between inserting it into F_1 and F_2. Therefore, job 4 is inserted into F_1 according to the factory sequence.

Step (3) Remove job 5 from β when job 5 is placed at the best position in F_1, $\sigma_{1,5} = +\infty$ ($\sigma_{k,j}$ is set to infinity when the completion time of the factory exceeds the deadline). Thus, job 5 is assigned to F_2 because $\sigma_{2,5} = 0$, as depicted in Fig. 3 b).

Step (4) Job 2 is taken out from β, and attempts are made to insert it into all possible positions of both factories, $\sigma_{1,2} = 448$ and $\sigma_{2,2} = 378$. Assign job 2 to F_2, as depicted in Fig. 3 c).

Step (5) Attempts are made to insert job 1 into all possible positions of both factories, $\sigma_{1,1} = +\infty$, $\sigma_{2,1} = +\infty$. Since the completion times of the job in both factories exceed the deadline, as shown in Fig. 3 d), the job is appended to the alternative sequence μ, $\mu = \{1\}$.

Step (6) Job 6 is tried to insert into all possible positions of both factories, $\sigma_{1,6} = 23$ and $\sigma_{2,6} = +\infty$. Therefore, job 6 is assigned to F_1, as shown in Fig. 3 e).

Step (7) Job 3 is inserted into the two factories to find the position with the lowest tardiness costs, as shown in Fig. 3 f). Since $\sigma_{1,3} = +\infty$ and $\sigma_{2,3} = +\infty$, append job 3 to the alternative sequence μ, $\mu = \{1, 3\}$, as shown in Fig. 3 g).

To save computational effort, the PNEH and other algorithms use an acceleration method. There are n_k jobs $\pi_k = \{\pi_{k,1}, ..., \pi_{k,n_k}\}$ in a factory F_k, $r = 1, 2, ..., n_k$, $\pi_{k,r} \in J$ is the r^{th} job in F_k. A new job can be inserted into $n_k + 1$ possible positions in F_k. The completion time of $\pi_{k,r}$ on M_i is denoted as $C_{i,\pi_{k,r}}$, the tardiness costs of $\pi_{k,r}$ is denoted as $\tau_{\pi_{k,r}}$, and the tardiness costs of all jobs in F_k is denoted as $S[\pi_k]$. After the new job is inserted into F_k, the tardiness costs of all jobs in F_k is denoted as $S[\pi_k^{'}]$. The incremental tardiness costs of all jobs in F_k is denoted as EE_k. The calculation formulas are as follows.

$$C_{1,\pi_{k,1}} = P_{1,\pi_{k,1}} \tag{17}$$

$$C_{i,\pi_{k,1}} = C_{i-1,\pi_{k,1}} + P_{i,\pi_{k,1}}, i = 2, 3, ..., m \tag{18}$$

$$C_{1,\pi_{k,r}} = C_{1,\pi_{k,r-1}} + P_{1,\pi_{k,r}}, r = 2, 3, ..., n_k \tag{19}$$

$$C_{i,\pi_{k,r}} = \max\{C_{i,\pi_{k,r-1}}, C_{i-1,\pi_{k,r}}\} + P_{i,\pi_{k,r}}, i = 2, 3, ..., m, r = 2, 3, ..., n_k \tag{20}$$

$$\tau_{\pi_{k,r}} = \max\{C_{m,\pi_{k,r}} - \psi_{\pi_{k,r}}, 0\} \times \omega_{\pi_{k,r}}, r = 1, 2, ..., n_k \tag{21}$$

$$S[\pi_k] = \sum_{r=1}^{n_k} \tau_{\pi_{k,r}} \tag{22}$$

$$EE_k = S[\pi_k^{'}] - S[\pi_k] \tag{23}$$

Formulas (17) and (18) calculate the completion times for the first job in F_k on M_1 and other machines, respectively. Formulas (19) and (20) calculate completion times for other jobs in F_k on M_1 and other machines, respectively. Formula (21) represents the tardiness costs. Formula (22) represents the total tardiness costs of all jobs in F_k. Formula (23) represents the incremental tardiness costs of all jobs in F_k after inserting the new job. The steps of the acceleration method are as follows.

Step (1) Calculate $C_{i,\pi_{k,r}}$, $\tau_{\pi_{k,r}}$ and $S[\pi_k]$, and then save the results.

Step (2) After the new job is inserted into the r^{th} position of F_k, $S[\pi_k^{'}] = S[\pi_k] + \sum_{r=r}^{n_k+1} \tau_{\pi_{k,r}^{'}}$, calculate EE_k using formula (23).

Step (3) Repeat Step (2) until the new job has been attempted to be inserted into all possible positions.

Step (4) Insert the new job into the best position among all factories or the alternative sequence.

4.2. Destruction phase

During the destruction phase, d jobs are taken out from π, and the removed jobs are appended to the alternative sequence μ. Through experiments, we found that better results can be obtained by varying d with dynamic sizes. The value of d is randomly generated by a uniform distribution within the range of $[2, d_c + 2]$, and d_c denotes a parameter that should be calibrated. Algorithm 2 shows the pseudocode for the destruction phase.

Algorithm 2. Destruction (π, μ, d_c)

```
d←rand(2, d_c + 2)
if d > Sm do // Sm is the number of all the jobs in π
    d = Sm ÷ 2
endif
for num = 1 to d do
    extract a job randomly from π, append the job to μ
endfor
Output π, μ
```

4.3. Reconstruction phase

During the reconstruction phase, first, sort the jobs in the alternative sequence according to certain rules, and then insert the jobs one after the other into π. The rules for sorting jobs are the LPT rules in descending order (DLPT), the EDD rule in ascending order (ADD), the PPUT rule in descending order (DUP), and the RANDOM rule. The parameter Ru should be calibrated, *and* $Ru = 1$, 2, 3, and 4 denote DLPT, ADD, DUP, and RANDOM, respectively. This process is similar to the part of PNEH that assigns jobs. The pseudocode for the reconstruction phase is shown

Table 4
Datum for the PNEH example instance.

J_j	1	2	3	4	5	6
$p_{1,j}$	54	83	55	71	77	36
$p_{2,j}$	79	32	81	99	56	70
ρ_j	3192	4370	1768	9180	5320	1802
ω_j	2	7	3	1	3	9
θ_j	3192	4370	3536	3060	5320	1802
ψ_j	175	138	146	183	165	114

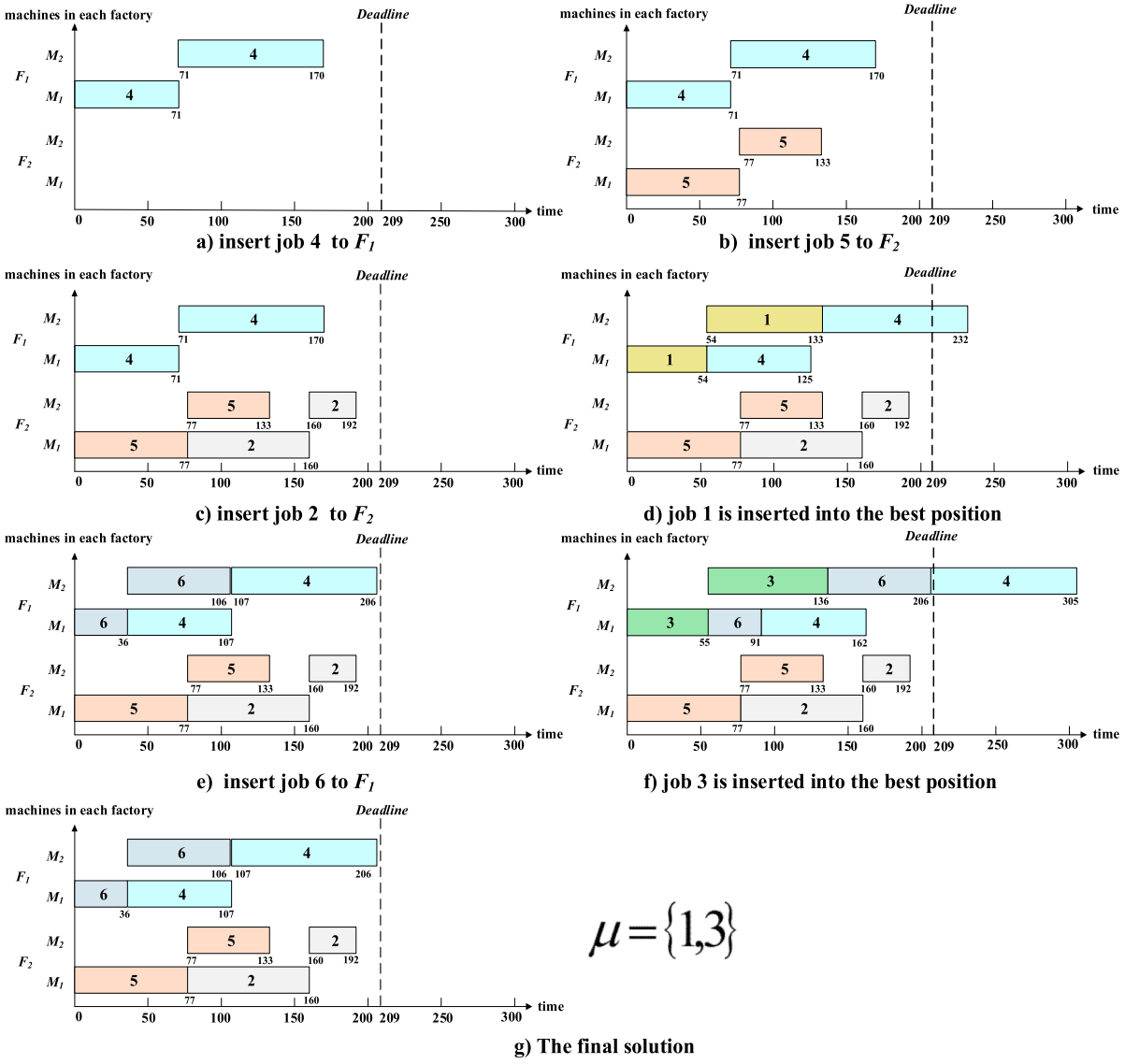


Fig. 3. PNEH inserts jobs for an example by Gantt Chart.

in Algorithm 3.

Algorithm 3. Reconstruction (π, μ, Ru)

for $loc = 1$ **to** tna **do**
  $\mu' \leftarrow$ Sort all jobs in μ by Ru
  for $k = 1$ **to** f **do**
    test job $\mu'[loc]$ in all possible positions of π_k
    EE_k is minimum incremental tardiness costs by inserting $\mu'[loc]$, $CF_{k,\mu'[loc]}$ is the completion time of F_k by inserting $\mu'[loc]$
    pos_k is the position where EE_k is obtained
    if $CF_{k,\mu'[loc]} > \Phi$ **then**
      $EE_k = +\infty$
    endif
  endfor
  $k^* \leftarrow \arg\left(\min_{k=1}^{f}(EE_k)\right)$
  if $EE_{k^*} = +\infty$ **then**
    do not remove job $\mu'[loc]$ from sequence μ'
  else
    remove job $\mu'[loc]$ from sequence μ' and insert it to π_{k^*} at pos_{k^*}
  endif
endfor
$\pi, \mu \leftarrow \pi_{k^*}, \mu'$
Output π, μ

4.4. Local search phase

During the local search phase, the solution obtained in the reconstruction phase should be optimized using local search methods. To prevent the completion time of each factory from exceeding Φ, an operator OptimizationSol is proposed to be used in all local search methods. The OptimizationSol operator collects the jobs completed after the deadline and assigns them to the alternative sequence.

Shift and Swap operators are like Jing et al. (2021), and the OptimizationSol operator is added to solve the DPTR. The Shift operator includes Shift Inside Factory (SIF) and Shift Between Factories (SBF). SIF selects a job from a random position in a factory and moves it to another random position in that factory. SBF selects a job from a random position in a factory and moves it to a random position in another factory. The pseudocode for the Shift operator is given in Algorithm 4. *OrigFac* refers to a factory from which a job will be removed, and *InsFac* refers to a factory into which the removed job will be inserted. Both *OrigFac* and *InsFac* contain at least one job, and they may be the same factory. *OrigPos* refers to a random position of *OrigFac*, and *InsPos* refers to a

random position of *InsFac*.

Algorithm 4. Shift (π, μ)

while $OrigFac = InsFac$ and $OrigPos = InsPos$ **do**
  select *OrigFac*,*InsFac*,*OrigPos*,*InsPos*
endwhile
if $OrigFac = InsFac$ **then**
  $\pi' \leftarrow \text{SIF}(\pi, OrigFac, OrigPos, InsPos)$
else
  $\pi' \leftarrow \text{SBF}(\pi, OrigFac, InsFac, OrigPos, InsPos)$
endif
$\pi, \mu \leftarrow \text{OptimizationSol}(\pi', \mu')$
Output π, μ

The Swap operator includes Swap Inside Factory (SWIF) and Swap Between Factories (SWBF). SWIF randomly selects a factory fa_1 with more than two jobs, then selects two jobs from two positions po_1 and po_2 of that factory at random, and exchanges their positions. SWBF selects two factories with at least one job randomly and selects one job in each factory randomly to exchange their positions. Algorithm 5 shows the pseudocode for the Swap operator.

Algorithm 5. Swap (π, μ)

while $fa_1 = fa_2$ and $po_1 = po_2$ **do**
  select fa_1,fa_2,po_1,po_2
endwhile
if $fa_1 = fa_2$ **then**
  $\pi' \leftarrow \text{SWIF}(\pi, fa_1, po_1, po_2)$
else
  $\pi' \leftarrow \text{SWBF}(\pi, fa_1, fa_2, po_1, po_2)$
endif
$\pi, \mu \leftarrow \text{OptimizationSol}(\pi', \mu')$
Output π, μ

Another Swap operator is Swap between a factory and the alternative sequence (SWFA). SWFA randomly selects a factory *Fac* containing at least one job, selects a job from a random position Pos_1 of that factory and a job from a random position Pos_2 of the alternative sequence, swaps the two jobs, and updates the solution. Algorithm 6 shows the pseudocode for the SWFA operator.

Algorithm 6. SWFA (π, μ)

if $sizeof(\mu) > 0$ **then**
  Randomly select *Fac*,Pos_1,Pos_2
  $\pi', \mu' \leftarrow$ Swap two jobs at Pos_1 of *Fac* and Pos_2 of μ
  $\pi, \mu \leftarrow \text{OptimizationSol}(\pi', \mu')$
endif
Output π, μ

An operator HybridOperator is proposed, which chooses one of the Shift, Swap, and SWFA operators at random. For the above four operators, a parameter *Op* is obtained to be calibrated, $Op = 1$, 2, 3, and 4 refer to Shift, Swap, SWFA, and HybridOperator, respectively. An operator LocalSearch1 is employed by the above four operators. The pseudocode for LocalSearch1 is given in Algorithm 7.

Algorithm 7. LocalSearch1 $(\pi, \mu, \Xi, Op, nPers)$

for count $= 1$ to *nPers* **do**
$\pi' \leftarrow \pi, \mu' \leftarrow \mu, \Xi' \leftarrow \Xi$
switch *the value of Op* **do**
  case $Op = 1$ **do**
    $\pi', \mu' \leftarrow \text{Shift}(\pi', \mu')$// **Algorithm 4**
  case $Op = 2$ **do**
    $\pi'', \mu'' \leftarrow \text{Swap}(\pi', \mu')$// **Algorithm 5**
  case $Op = 3$ **do**
    $\pi', \mu'' \leftarrow \text{SWFA}(\pi', \mu')$// **Algorithm 6**
  case $Op = 4$ **do**
    $\pi'', \mu'' \leftarrow \text{HybridOperator}(\pi', \mu')$
  endcase
endswitch
the new total profit Ξ'' is calculated
if $\Xi'' > \Xi$ **then**

(*continued on next column*)

(*continued*)

Algorithm 7. LocalSearch1 $(\pi, \mu, \Xi, Op, nPers)$

$\pi \leftarrow \pi'', \mu \leftarrow \mu'', \Xi \leftarrow \Xi''$
endif
endfor
Output π, μ, Ξ

If the solution has not been optimized by LocalSearch1, an operator LocalSearch2 is used. If the alternative sequence is not empty, LocalSearch2 selects a key factory *Keyfac* with the lowest profit and selects a random position *Keypos* from *Keyfac*. Furthermore, it selects a random position *Altpos* from the alternative sequence. SWFA is applied to *Keyfac* and the alternative sequence. If the alternative sequence is empty, LocalSearch2 selects a key factory with the lowest profit $Keyfac_1$ and a factory with the highest tardiness costs $Keyfac_2$. Randomly select a position $Keypos_1$ from $Keyfac_1$ and a position $Keypos_2$ from $Keyfac_2$, apply SBF to $Keyfac_1$ and $Keyfac_2$, and update the solution. Algorithm 8 shows the pseudocode for LocalSearch2.

Algorithm 8. LocalSearch2 (π, μ)

if $tna > 0$ **then**
  select *Keyfac*,*Keypos*,*Altpos*
$\pi', \mu' \leftarrow \text{SWFA}(\pi, \mu)$
else
  select $Keyfac_1$,$Keyfac_2$,$Keypos_1$,$Keypos_2$
  $\pi' \leftarrow \text{SBF}(\pi, Keyfac_1, Keyfac_2, KeyPos_1, KeyPos_2)$
endif
$\pi, \mu \leftarrow \text{OptimizationSol}(\pi', \mu')$
Output π, μ

4.5. Acceptance phase

In the literature, the acceptance criterion of IG algorithms typically adopts a simulated annealing algorithm. However, the simulated annealing algorithm has a limited effect on IG_TR; thus, a restart scheme is proposed to avoid falling into local optimal. The pseudocode for the restart scheme is shown in Algorithm 9. A new operator named ChanceTwoFac selects two factories with the smallest total profit, takes out their jobs, and puts them in the alternative sequence. Then, ChanceTwoFac sorts the jobs in the alternative sequence by the PPUT rule and inserts them into all factories, as in Section 4.1.

Algorithm 9. Restart-scheme (π, μ, d_c)

$cha \leftarrow rand(1,3)$//*arandomlygeneratedintegerin*$U[1,3]$
switch *the value of cha* **do**
  case $cha = 1$ **do**
    ChanceTwoFac(π, μ)
  case $cha = 2$ **do**
    LocalSearch2(π, μ)//**Algorithm 8**
  case $cha = 3$ **do**
$\pi', \mu' \leftarrow \text{Destruction}(\pi, \mu, dc)$ //**Algorithm 2**
    $\pi, \mu \leftarrow \text{Reconstruction}(\pi', \mu')$//**Algorithm 3**
endcase
endswitch
Output π, μ

4.6. Complete pseudocode

The complete pseudocode for the IG_TR algorithm is given in Algorithm 10, where π^* and μ^* are denoted as the best solution, where Ξ^* represents the best value calculated by formula (1).

Algorithm 10. IG_TR $(\pi, \mu, d_c, nPers, Op, Ru)$

$\pi, \mu, \Xi \leftarrow \text{PNEH}(\pi, \mu)$//**Algorithm 1**
$\pi, \mu, \Xi \leftarrow \text{LocalSearch1}(\pi, \mu)$//**Algorithm 7**
$\pi^* \leftarrow \pi, \mu^* \leftarrow \mu, \Xi^* \leftarrow \Xi$
while (time limit is not exceeded) **do**
$\pi', \mu' \leftarrow \text{Destruction}(\pi, \mu, d_c)$ //**Algorithm 2**
  $\pi'', \mu'' \leftarrow \text{Reconstruction}(\pi', \mu', Ru)$//**Algorithm 3**
  $\pi''', \mu''', \Xi' \leftarrow \text{LocalSearch1}(\pi'', \mu'', Op, nPers)$//**Algorithm 7**
if $\Xi' = \Xi$ **then**

(*continued on next page*)

(*continued*)

Algorithm 10. IG_TR $(\pi, \mu, d_c, nPers, Op, Ru)$

```
    π'',μ'' ← LocalSearch2(π''',μ''') //Algorithm 8
    π''←π'', μ''←μ'', the new total profit Ξ'' is calculated, Ξ'←Ξ''
endif
if Ξ' > Ξ then
    π←π'', μ←μ'', Ξ←Ξ'
if Ξ > Ξ* then
    π*←π, μ*←μ, Ξ*←Ξ
endif
else
π, μ ← Restart − scheme(π, μ, d_c) //Algorithm 9
endif
endwhile
Output π*, μ*, Ξ*
```

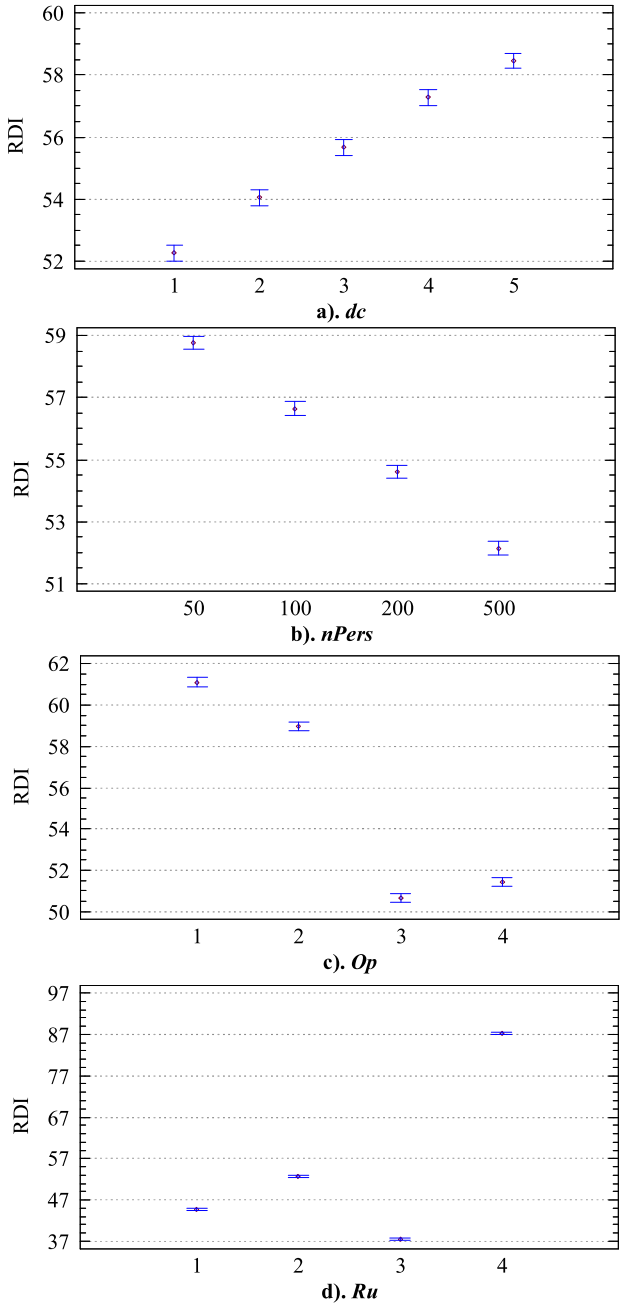


Fig. 4. Means plots for four parameters of the IG_TR.

5. Experimental calibration

Design of experiments and ANOVA are frequently used in the scheduling literature (Pan, Gao, Li, & Gao, 2017), and the two methods are used to calibrate the parameters of the IG_TR algorithm by 50 instances, as generated in the next Section. All algorithms were coded in the C++ language on Microsoft Visual Studio 2015 software. All codes were run on Windows 7 with 2 processors (Intel(R) Core(TM) i7-9700 K 3.60 GHz), 3.99 GB RAM, and the maximum-elapsed CPU time was 20 *nm* milliseconds. All results were analyzed by ANOVA, as shown in Fig. 4.

The IG_TR algorithm comprises four parameters d_c, *nPers*, *Op*, and *Ru*. *nPers* denotes the number of runs for LocalSearch1, and *Ru* denotes the four methods of sorting jobs in the alternative sequence mentioned in Section 4.3. The levels of the four parameters are as follows. d_c is at five levels 1, 2, 3, 4, and 5; *nPer* is at four levels 50, 100, 200, and 500; *Op* is at four levels 1, 2, 3, and 4; *Ru* is at four levels 1, 2, 3, and 4. With the combination of the four parameters, there are $5 \times 4 \times 4 \times 4 = 320$ different combinations. Each combination solves 50 instances, and each instance runs ten independent replications; thus, there are $320 \times 10 = 3200$ results for each instance. The RDI is shown in formula (24).

$$RDI(R_i) = \frac{100 \times (R^* - R_i)}{R^* - R_{worst}} \tag{24}$$

where R^* represents the best value among 3200 results for each instance, R_{worst} represents the worst value among 3200 results for each instance, and R_i denotes an objective value for a solution. The smaller the value of RDI is, the closer the solution is to the best solution.

The total number of RDI values is $3200 \times 50 = 160{,}000$. From Fig. 4, d_c, *nPers*, *Op*, and *Ru* are statistically significant. Fig. 4 a) shows that there is optimal performance when the parameter $d_c = 1$, indicating that too many jobs should not be removed in the destruction phase, resulting in little disturbance in the reconstruction phase. Thus, new solutions can maintain some of the excellent properties of the best solution. Fig. 4 b) shows that a higher number of local search runs generates better solutions; here, the maximum number of runs is chosen to be 500 by considering the total number of IG iterations and some previous studies. Fig. 4 c) shows that the proposed SWFA operator performs best because the profit of the factory is also related to the rejection costs. Jobs with high unit profits are previously preferred for processing. By swapping the jobs in a factory and the alternative sequence, the jobs with larger rejection costs can be selected for processing, thereby optimizing the current solution. Fig. 4 d) shows that ranking the jobs of the alternative sequence in the reconstruction phase is necessary. Because the low profit per unit processing time of the jobs will take a long time, the solution can be optimized by giving priority to the higher profit per unit processing time of the jobs.

Because the smaller the RDI is, the closer the solution is to the best solution, set $d_c = 1$, $nPers = 500$, $Op = 3$, and $Ru = 3$.

6. Computational evaluation

As the DPTR has been not discussed in the literature, test instances are to be generated. We employ the 720 test instances (Naderi, & Ruiz, 2010) by adding a common deadline, due date, profit, unit tardiness weight, and rejection costs for each job. The 720 test instances include the total number of factories *f*, ranging between 2, 3, 4, 5, 6, and 7, the total number of jobs *n*, ranging between 20, 50, 100, 200, and 500, and the total number of machines *m*, ranging between 5, 10, and 20.

The unit tardiness weight for each job ω_j is a randomly generated integer in *U*[1,9](), and the common deadline is generated as formula (25).

$$\Phi = C_{\max}^{PNEH} \times 0.8 \tag{25}$$

where $C_{\max}^{PNEH}$ denotes the makespan obtained by the NEH heuristic based

on the LPT rule.

With $C_{\max}^{PNEH}$, formulas (26) and (27) are generated as follows.

$$rd_j = C_{\max}^{PNEH} \times 0.3 + rand\left(C_{\max}^{PNEH} \times 0.3, C_{\max}^{PNEH} \times 0.7\right) \tag{26}$$

$$\psi_j = \max\left(rd_j, SUM\left(P_{i,j}\right) \times (1 + H/100)\right) \tag{27}$$

In formula (26), rd_j denotes a random due date of job J_j according to Jing et al. (2020). In formula (27), the due date ψ_j is to be the greater of the two values, where $H = U$ [1,10]() and $SUM(P_{i,j})$ is the sum of processing times for J_j on all machines.

The rejection costs θ_j for J_j is generated, as expressed in formulas (28) and (29).

$$Dep_j = \left(minDep_j + rand\left(maxDep_j - minDep_j + 1\right) + rand(1,5)\right) \times SUM\left(P_{i,j}\right) \tag{28}$$

$$\theta_j = \max\left(Dep_j \times (rand(0,1) + 1), \omega_j \times \left(\Phi - \psi_j\right)\right) \tag{29}$$

where Dep_j denotes an intermediate variable: $minDep_j = 10$ and $maxDep_j = 20$. The rejection costs is greater than or equal to the biggest tardiness costs, and $rand(0,1)$ denotes a randomly generated integer in $U[0,1]$. The profit ρ_j for J_j is generated, as shown in formula (30).

$$\rho_j = \max\left(Dep_j \times (rand(0,2) + 1), \omega_j \times \left(\Phi - \psi_j\right)\right) \tag{30}$$

Note that ρ_j is greater than the biggest tardiness costs, and $rand(0,2)$ denotes a random integer in $U[0,2]$.

6.1. Validity of the IG_TR components

To investigate the effectiveness of the components of the IG_TR algorithm, two versions of the IG_TR algorithm named IG_1 and IG_2 are used for comparison. IG_1 is the IG_TR algorithm that does not sort the jobs in the alternative sequence during the reconstruction phase. IG_2 is the IG_TR algorithm that does not apply the local search phase. Other components of the algorithm have been validated by the previous experiments; thus, comparisons are not performed here.

The IG_TR, IG_1, and IG_2 algorithms are used to solve the 720 instances, and they are executed ten times independently for each instance. To ensure that the algorithms have the same running time, we set the maximum-elapsed CPU time for each algorithm at 20 *mn* milliseconds. The average RDI values (ARDI) grouped by the number of factories, jobs, and machines are shown in Table 5.

From Table 5, the IG_TR algorithm performs best, except that the IG_1 algorithm slightly outperforms it at $n = 20$. In addition, Table 5 shows that the IG_2 algorithm significantly underperforms the other two algorithms. The results show that the job sorting method in the reconstruction phase and the local search phase are important in optimizing the proposed algorithm. Overall, the IG_TR algorithm is the best.

Table 5
ARDI values of the IG_TR, IG_1 and IG_2 (minimum values are in italics and bold).

type	IG_TR	IG_1	IG_2
$f = 2$	***12.783***	28.755	100.000
$f = 3$	***14.117***	23.504	100.000
$f = 4$	***14.853***	23.240	100.000
$f = 5$	***14.489***	21.117	99.167
$f = 6$	***12.423***	19.146	91.667
$f = 7$	***12.136***	17.799	87.500
$n = 20$	16.786	***12.356***	85.556
$n = 50$	***13.612***	17.743	100.000
$n = 100$	***11.327***	23.375	100.000
$n = 200$	***10.775***	30.787	100.000
$n = 500$	***14.875***	45.124	100.000
$m = 5$	***15.108***	20.236	97.778
$m = 10$	***12.917***	21.302	97.500
$m = 20$	***12.922***	24.241	94.667
Mean	***13.467***	22.260	96.389

A multifactor ANOVA is employed to test whether the datum shown in Table 5 is statistically significant. Fig. 5 a) shows mean plots of the IG_TR, IG_1, and IG_2 algorithms with 95% Tukey's honestly significant difference (Tukey, 1954) confidence intervals, and overlapping intervals mean statistically insignificant differences. Fig. 5 b)–d) show the interactions between the algorithms and the number of jobs, machines, and factories, respectively. From Fig. 5 a), the differences shown in Table 5 are statistically significant, and the IG_TR algorithm performs best. As Fig. 5 b)–d) show that the IG_1 algorithm outperforms IG_TR at $n = 20$, and in the other cases, the IG_TR algorithm performs best in all factory, job, and machine configurations, respectively. In order of best to worst, the overall mean ARDI values obtained by the algorithms are as follows: IG_TR, IG_1 and IG_2.

6.2. Effectiveness of IG_TR

To test the effectiveness of the proposed algorithm, it is compared with five state-of-the-art algorithms, namely, TWIG (Ruiz, Pan, & Naderi, 2019), BSIG (Fernandez-Viagas, & Framinan, 2015), DABC, ILS (Pan et al., 2019), and EA (Fernandez-Viagas, Perez-Gonzalez, & Framinan, 2018), from closely related scheduling studies. By sharing most codes of the IG_TR algorithm, such as generating initial solutions, acceleration method, and some local search operators, these five algorithms can solve the DPTR. Table 6 shows the parameters used by these five algorithms.

All algorithms are used to solve the 720 instances by the maximum-elapsed CPU time at 20, 40, and 60 *mn* milliseconds, respectively, and the results are shown in Tables 7, 8, and 9. The maximum and minimum values are obtained for each instance at all CPU times. The results show that the ARDI values of all algorithms decrease as the CPU time increases, which indicates that the quality of the solutions gradually improves over time.

A multifactor ANOVA is employed to test whether the datum shown in Table 7, Table 8, and Table 9 are statistically significant. Fig. 6 shows the mean plots of the algorithms. Fig. 7 a)–d) show the interactions between the algorithms with the number of CPU times, machines, jobs, and factories, respectively. These statistics are statistically significant. As shown in Fig. 7, the IG_TR algorithm outperforms the other algorithms. Fig. 7 a) shows that the IG_TR algorithm outperforms the other algorithms, and the results of all algorithms are gradually improved as the CPU time increases. Fig. 7 b) shows that the IG_TR algorithm performs best except at $n = 500$, and the results of the EA algorithm progressively worsen as the number of jobs increases and are not significantly different from those of BSIG and ILS. In order of best to worst, the overall mean ARDI values obtained by the algorithms are as follows: IG_TR, TWIG, DABC, EA, ILS, and BSIG. Fig. 7 c) shows that the IG_TR algorithm performs the best in all factory configurations. Fig. 7 d) shows that the best performance of TWIG at $f = 2$, and IG_TR algorithm slightly underperforms TWIG at $f = 2$. Overall, the IG_TR algorithm outperforms the other five existing algorithms, which supports the conclusions in Table 7, Table 8, and Table 9. Nevertheless, overall, the IG_TR algorithm performs best.

Here, a test instance is chosen at random. The evolution of solutions for the six algorithms is given in Fig. 8, where the horizontal axis manifests time (s), and the vertical axis means the objective function value. Fig. 8 shows that the IG_TR algorithm outperforms the other algorithms.

6.3. Association between total profit and tardiness and rejection costs

The six algorithms are used to solve the 720 instances, and the total profit, tardiness costs, rejection costs, and the sum of tardiness and rejection costs (total costs) are obtained for each instance, respectively. The six algorithms are executed ten times independently for each instance by the maximum-elapsed CPU time at 20, 40, and 60 *mn*

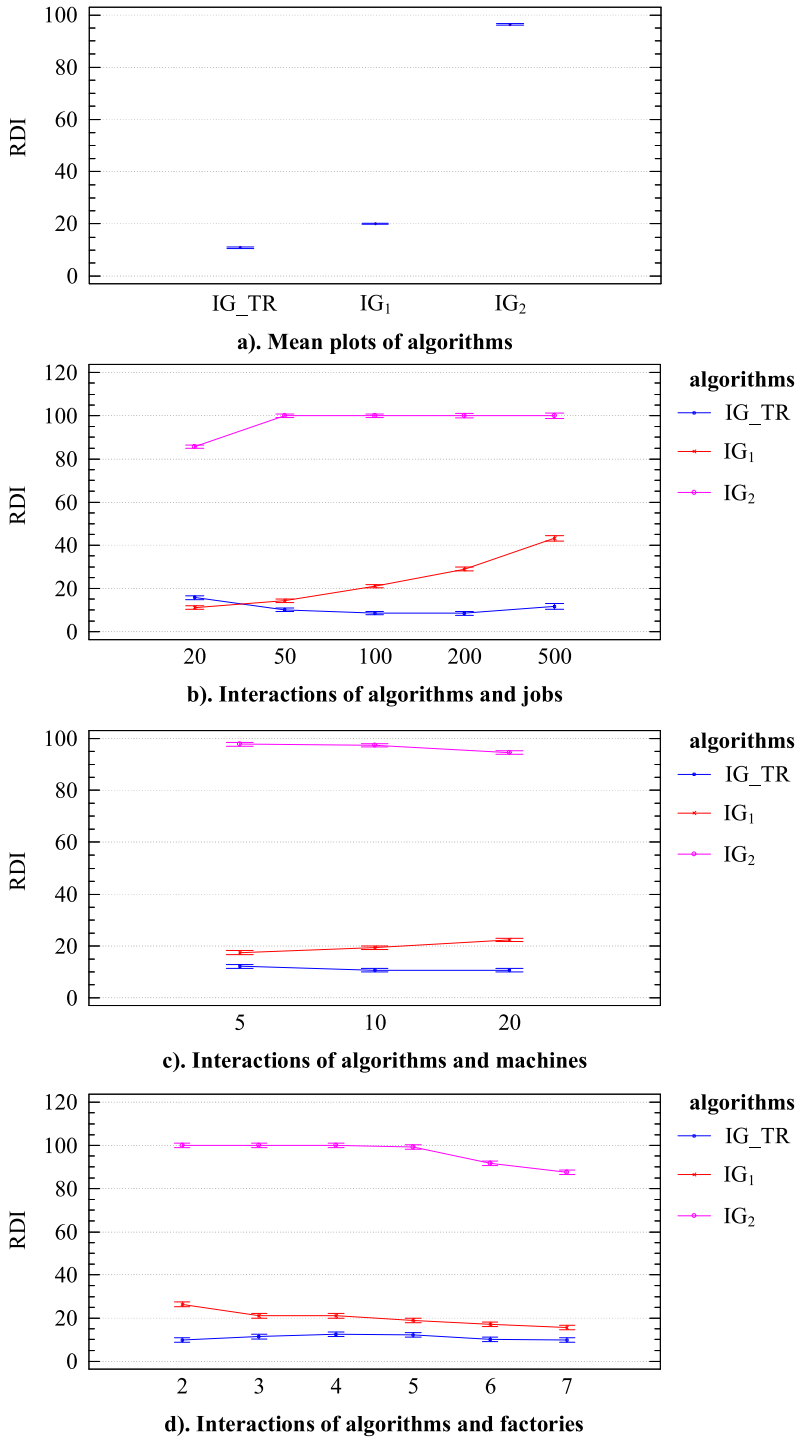


Fig. 5. Means plots and interactions for the IG_TR, IG_1 and IG_2.

milliseconds, respectively. Therefore, there are a total of 129,600 datum. For each instance, the maximum total profit, minimum tardiness costs, minimum rejection costs, and minimum total costs are generated from ten times by each algorithm. If the maximum profit occurs more than once in these ten times, the result generated by the first time is selected. When the maximum total profit is obtained for each instance by each algorithm, T_{num} represents the number of instances where the minimum tardiness costs are obtained simultaneously; R_{num} represents the number of instances where the minimum rejection costs are obtained simultaneously; TR_{num} represents the number of instances where the minimum total costs are obtained simultaneously. Table 10 shows the values of T_{num}, R_{num}, and TR_{num} for the six algorithms.

From Table 10, total profit and tardiness costs, rejection costs, and total costs for the BSIG algorithm have very strong connection, they link each other nearly. Total profit and tardiness costs, rejection costs, and total costs for the EA and ILS algorithms have strong connection. There are moderate correlations between total profit and rejection costs, and total costs for the TWIG algorithm, and the correlation between total profit and tardiness costs increases with the increase of CPU time. There are certain correlations between total profit and tardiness costs, rejection costs, and total costs for the DABC and IG_TR algorithms. In summary, Table 10 shows that the correlations between total profit and

Table 6
Parameters used by the algorithms for comparison.

Algorithms	Parameters	Means of parameters	value
TWIG	d	In the first stage, the number of jobs to remove in the destruction	4
	d_2	In the second stage, the number of jobs to remove in the destruction	2
	ρ	The proportion of CPU time that is given to the first stage	0.95
	T	A constant temperature	0.2
	OperType	A parameter for selecting a local search method	VND(a)R
BSIG	T	A constant temperature	0.1
	d	The number of jobs to remove in the destruction	3
	L	A parameter in improvement phase	30
DABC	*PS*	Population size	20
	OperType	A parameter for selecting a local search method	SWFA
	Ξ	Insertion (0), pairwise interchange (1), hybrid (2)	1
EA	γ	Population size	20
ILS	*PLen*	A parameter in improvement phase	3
	nPerSols	A parameter related to the iteration of the algorithm	20
	T	A constant temperature	0.3
	OperType	A parameter for selecting a local search method	HybridOperator

Table 7
ARDI at 20*mn* milliseconds (minimum values are in italics and bold).

type	EA	DABC	BSIG	ILS	TWIG	IG_TR
$f = 2$	92.102	47.312	95.917	90.057	31.244	***30.642***
$f = 3$	93.820	54.899	98.408	96.238	33.362	***22.472***
$f = 4$	91.490	55.002	98.818	97.082	37.460	***19.643***
$f = 5$	87.158	53.572	98.982	97.363	41.870	***21.256***
$f = 6$	80.547	51.935	96.730	92.906	43.291	***24.894***
$f = 7$	77.979	54.327	94.303	88.579	46.787	***28.164***
$n = 20$	57.985	41.465	94.238	88.777	65.706	***36.376***
$n = 50$	96.906	55.842	99.409	98.932	30.937	***14.270***
$n = 100$	97.500	51.917	98.575	97.377	30.911	***16.336***
$n = 200$	98.005	56.424	98.060	92.682	29.114	***21.245***
$n = 500$	93.009	73.574	93.530	83.825	***27.139***	50.708
$m = 5$	88.184	44.811	97.586	95.929	44.802	***20.666***
$m = 10$	86.116	48.772	97.725	94.846	38.155	***21.596***
$m = 20$	87.435	60.914	96.532	91.455	36.200	***29.152***
Mean	87.183	52.841	97.193	93.704	39.002	***24.512***

Table 8
ARDI at 40*mn* milliseconds (minimum values are in italics and bold).

type	EA	DABC	BSIG	ILS	TWIG	IG_TR
$f = 2$	91.479	44.460	95.713	89.172	***23.860***	27.264
$f = 3$	93.356	51.859	98.227	95.312	26.760	***18.970***
$f = 4$	91.422	52.859	98.662	96.212	31.639	***16.325***
$f = 5$	86.346	50.128	98.897	96.711	36.372	***18.450***
$f = 6$	80.169	49.457	96.651	92.771	38.242	***21.991***
$f = 7$	77.116	51.962	94.233	88.281	42.012	***25.439***
$n = 20$	57.361	41.465	94.230	88.666	64.960	***33.721***
$n = 50$	96.291	55.561	99.400	98.965	25.405	***10.747***
$n = 100$	96.769	49.374	98.522	97.254	22.884	***13.194***
$n = 200$	97.775	50.253	97.892	91.976	20.310	***18.570***
$n = 500$	92.966	61.744	92.524	78.315	***17.402***	46.749
$m = 5$	86.915	42.875	97.537	95.727	38.954	***16.717***
$m = 10$	86.005	45.718	97.658	94.614	32.538	***18.727***
$m = 20$	87.002	57.991	96.305	90.257	30.151	***26.363***
Mean	86.648	50.121	97.064	93.076	33.147	***21.406***

Table 9
ARDI at 60*mn* milliseconds (minimum values are in italics and bold).

type	EA	DABC	BSIG	ILS	TWIG	IG_TR
$f = 2$	91.062	43.251	95.622	88.742	***19.544***	25.617
$f = 3$	93.114	51.221	98.171	95.050	23.271	***17.153***
$f = 4$	90.572	51.503	98.648	95.763	28.247	***14.488***
$f = 5$	85.525	48.717	98.863	96.585	33.548	***17.236***
$f = 6$	79.616	48.270	96.623	92.036	35.393	***20.747***
$f = 7$	76.837	50.260	94.211	88.277	39.273	***24.111***
$n = 20$	56.372	41.063	94.234	88.471	64.694	***32.385***
$n = 50$	95.779	55.934	99.400	98.955	22.627	***8.962***
$n = 100$	96.313	48.204	98.524	97.287	18.366	***11.766***
$n = 200$	97.566	48.028	97.816	91.716	14.884	***16.940***
$n = 500$	92.927	54.784	92.169	75.335	***11.723***	45.487
$m = 5$	86.390	42.190	97.543	95.949	35.883	***14.973***
$m = 10$	85.516	44.706	97.626	94.522	29.246	***17.246***
$m = 20$	86.443	56.210	96.228	89.395	26.785	***24.960***
Mean	86.121	48.870	97.023	92.742	29.879	***19.892***

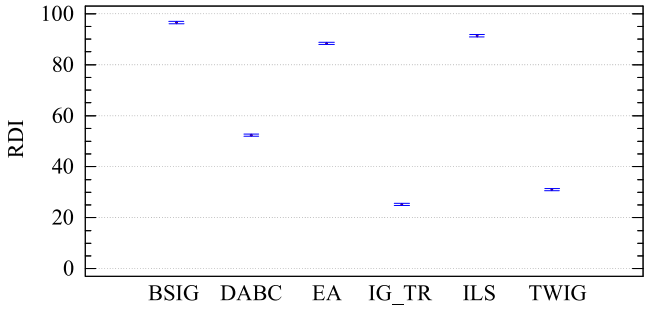


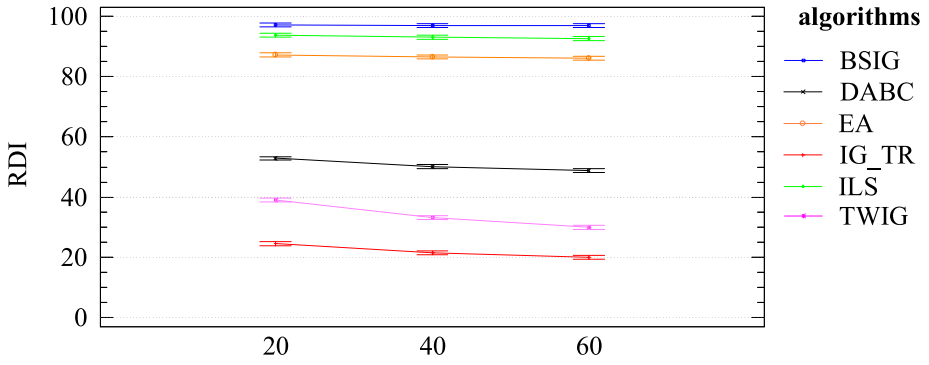
Fig. 6. Means plots for BSIG, DABC, EA, IG_TR, ILS, TWIG.

tardiness costs, rejection costs, and total costs vary among different algorithms. The reason for this is that total profit is not only related to tardiness and rejection costs, but also to the final processed jobs and the scheduling order of the final processed jobs.

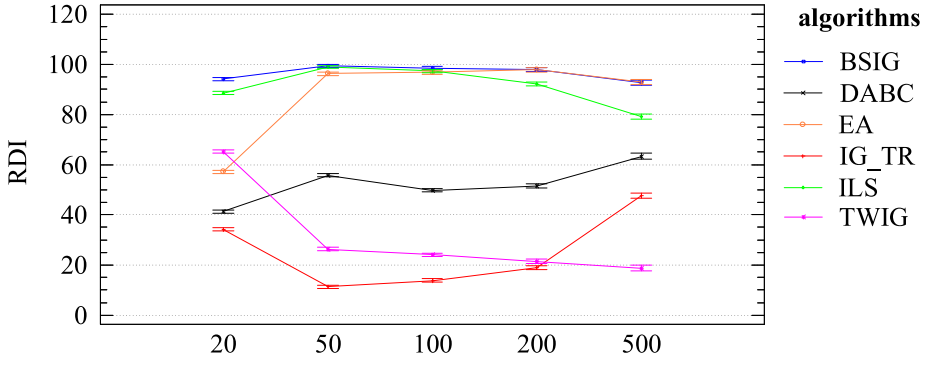
7. Conclusions

In a real-world production environment, it is possible to have too many orders signed and insufficient capacity to complete all jobs on time. This study investigates the DPTR. To our best knowledge, this is the first reported work solving such a problem. An IG algorithm, IG_TR, is proposed to maximize the total profit. The IG_TR algorithm is compared with five algorithms from closely related studies. Extensive comparative evaluations show that the IG_TR algorithm outperforms the five comparison algorithms. The reasons for the better performance of the IG_TR algorithm are as follows. In the initialization phase of the IG_TR algorithm, we improve the NEH heuristic by proposing a sorting rule for the seed sequence and a method for assigning jobs. In the destruction phase, we make the number of jobs extracted from the factory vary dynamically within a given range. In the reconstruction phase, we rank the jobs in the alternative sequences and improve the job assignment method. In the local search phase, we propose two local search methods. The first local search method performs an exchange operation between jobs in the factory and alternative sequence, and the second method optimizes the key factory. Finally, we propose a restart scheme as an acceptance criterion to jump out of the local optimum.

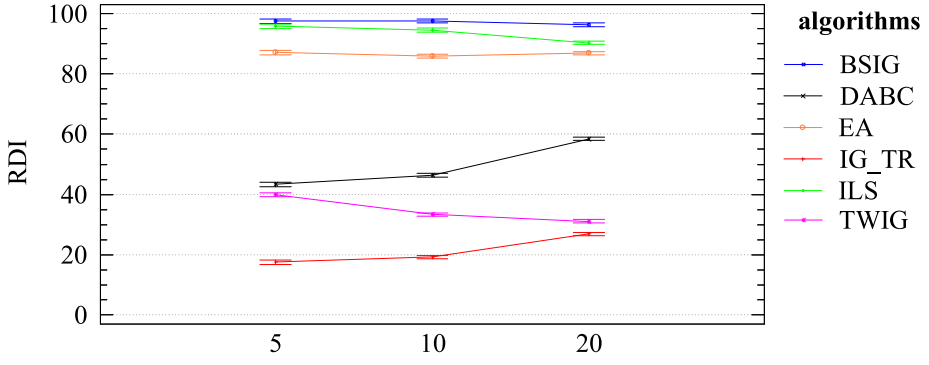
From the comparison results of the IG_TR and other algorithms, we found that the IG_TR algorithm needs further optimization in its ability to handle a large number of jobs ($n = 500$). The underlying reason is that there is a slight disturbance in the reconstruction phase. This can be enhanced in subsequent work by designing higher perturbations or suitable local search methods for a large number of jobs ($n = 500$). In the future, we intend to employ the IG_TR algorithm to solve scheduling problems in more complicated and realistic production environments,



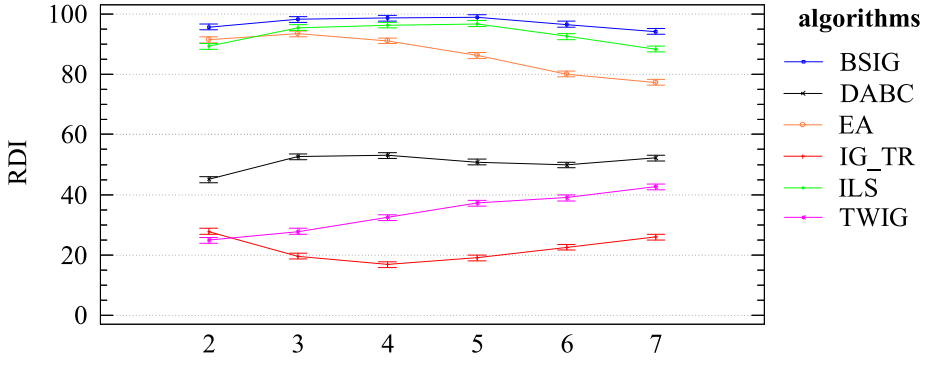
a). Interactions of algorithms and CPU times



b). Interactions of algorithms and jobs



c). Interactions of algorithms and machines



d). Interactions of algorithms and factories

Fig. 7. Interactions for BSIG, DABC, EA, IG_TR, ILS, TWIG.

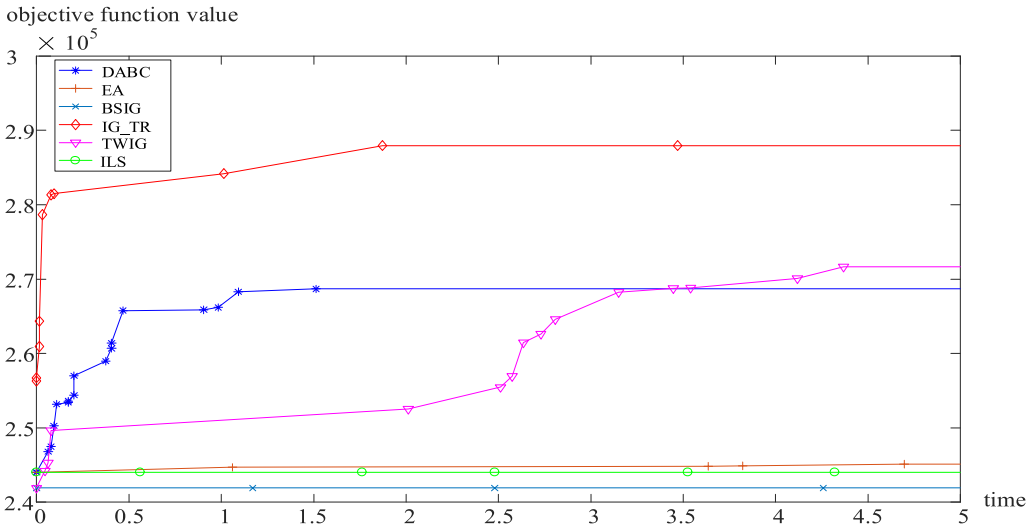


Fig. 8. Evolution of solutions for the 276th test instance.

Table 10
Association between total profit and tardiness and rejection costs.

CPU	Type	BSIG	EA	ILS	TWIG	DABC	IG_TR
20	T_{num}	715	546	563	111	97	101
	R_{num}	720	656	615	453	331	296
	TR_{num}	719	650	616	450	316	295
40	T_{num}	716	537	556	255	103	101
	R_{num}	720	666	624	424	296	282
	TR_{num}	719	662	617	421	297	283
60	T_{num}	717	539	568	424	95	111
	R_{num}	720	657	627	430	324	272
	TR_{num}	719	654	628	428	302	273

such as distributed flowshop grouping scheduling (Pan, Gao and Wang, 2022), robust distributed flowshop scheduling, distributed flowshop scheduling with setup and release times, and hybrid flowshop scheduling (Zhang et al., 2020, Lu et al., 2022), and we will employ orthogonal learning design (Ma, Cheng, & Shi, 2021) to calibrate the parameters of algorithms.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This research is partially supported by the National Natural Science Foundation of China 62273221 and 61973203.

References

Allahverdi, A., Aydilek, H., & Aydilek, A. (2018). No-wait flowshop scheduling problem with two criteria; total tardiness and makespan. *European Journal of Operational Research, 269*(2), 590–601.

Braglia, M., & Grassi, A. (2009). A new heuristic for the flowshop scheduling problem to minimize makespan and maximum tardiness. *International Journal of Production Research, 47*(1), 273–288.

Chen, S., Pan, Q. K., Gao, L., Miao, Z. H., & Peng, C. (2023). Energy-efficient distributed heterogeneous blocking flowshop scheduling problem using a knowledge-based iterated Pareto greedy algorithm. *Neural Computing and Applications, 35*, 6361–6381.

Cordone, R., & Hosteins, P. (2019). A bi-objective model for the single-machine scheduling problem with rejection costs and total tardiness minimization. *Computers & Operations Research, 102*, 130–140.

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

Dubois-Lacoste, J., Pagnozzi, F., & Stützle, T. (2017). An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem. *Computers & Operations Research, 81*, 160–166.

Fernandez-Viagas, V., & Framinan, J. M. (2015). 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 flowshop to minimise the total flowtime. *Computers & Industrial Engineering, 118*, 464–477.

Fu, Y. P., Tian, G. D., Fathollahi-Fard, A. M., Ahmadi, A., & Zhang, C. Y. (2019). Stochastic multi-objective modelling and optimization of an energy-conscious distributed permutation flowshop scheduling problem with the total tardiness constraint. *Journal of Cleaner Production, 226*, 515–525.

Fu, Y. P., Hou, Y. S., Wang, Z. F., Wu, X. W., Gao, K. Z., & Wang, L. (2021). Distributed scheduling problems in intelligent manufacturing systems. *Tsinghua Science and Technology, 26*(5), 625–645.

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

Gao, J., Chen, R., Deng, W., & Liu, Y. (2012). Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm. *Journal of Computational Information Systems, 8*(5), 2025–2032.

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*(3–4), 641–651.

Gerstl, E., & Mosheiov, G. (2020). Single machine scheduling to maximize the number of on-time jobs with generalized due-dates. *Journal of Scheduling, 23*, 289–299.

Huang, J. P., Pan, Q. K., & Gao, L. (2020). An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 59*, Article 100742.

Huang, J. P., Pan, Q. K., Miao, Z. H., & Gao, L. (2021). Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times. *Engineering Applications of Artificial Intelligence, 97*, Article 104016.

Huang, Y. Y., Pan, Q. K., & Gao, L. (2023). An effective memetic algorithm for the distributed flowshop scheduling problem with an assemble machine. *International Journal of Production Research, 61*(6), 1755–1770.

Jing, X. L., Pan, Q. K., Gao, L., & Wang, Y. L. (2020). An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows. *Applied Soft Computing, 96*, Article 106629.

Jing, X. L., Pan, Q. K., & Gao, L. (2021). Local search-based metaheuristics for the robust distributed permutation flowshop problem. *Applied Soft Computing, 105*(2), Article 107247.

Jing, X. L., Pan, Q. K., Gao, L., & Wang, L. (2022). An effective iterated greedy algorithm for a robust distributed permutation flowshop problem with carryover sequence-dependent setup time. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 52*(9), 5783–5794.

Khare, A., & Agrawal, S. (2020). Effective heuristics and metaheuristics to minimise total tardiness for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 7266–7282.

Li, W., Li, J., Zhang, X., & Chen, Z. (2015). Penalty cost constrained identical parallel machine scheduling problem. *Theoretical Computer Science, 607*, 181–192.

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*(3), Article 107961.
Lin, S. W., Ying, K. C., & Huang, C. Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research, 51*(15–16), 5029–5038.
Liu, P., & Lu, X. (2020). New approximation algorithms for machine scheduling with rejection on single and parallel machine. *Journal of Combinatorial Optimization, 40*(1), 1–24.
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*, Article 117555.
Lu, C., Huang, Y., Meng, L., Gao, L., Zhang, B., & Zhou, J. (2023). A Pareto-based collaborative multi-objective optimization algorithm for energy-efficient scheduling of distributed permutation flow-shop with limited buffers. *Robotics and Computer-Integrated Manufacturing, 74*, Article 102277.
Ma, L., Cheng, S., & Shi, Y. (2021). Enhancing learning efficiency of brain storm optimization via orthogonal learning design. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 51*(11), 6723–6742.
Mao, J. Y., Pan, Q. K., Miao, Z. H., & Gao, L. (2021). An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance. *Expert Systems with Applications, 169*, Article 114495.
Mao, J. Y., Pan, Q. K., Miao, Z. H., Gao, L., & Chen, S. (2022). A hash map-based memetic algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance to minimize total flowtime. *Knowledge-Based Systems, 242*, Article 108413.
Meng, T., Pan, Q. K., & Wang, L. (2019). A distributed permutation flowshop scheduling problem with the customer order constraint. *Knowledge-Based Systems, 184*(15), Article 104894.
Meng, T., & Pan, Q. K. (2021). A distributed heterogeneous permutation flowshop scheduling problem with lot-streaming and carryover sequence-dependent setup time. *Swarm and Evolutionary Computation, 60*, Article 100804.
Mor, B., Mosheiov, G., & Shapira, D. (2020). Single machine lot scheduling with optional job-rejection. *Journal of Combinatorial Optimization, 41*, 1–11.
Mosheiov, G., Oron, D., & Shabtay, D. (2021). Minimizing total late work on a single machine with generalized due-dates. *European Journal of Operational Research, 293*, 837–846.
Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research, 37*(4), 754–768.
Naderi, B., & Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research, 239*(2), 323–334.
Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91–95.
Ou, J., Zhong, X., & Wang, G. (2015). An improved heuristic for parallel machine scheduling with rejection. *European Journal of Operational Research, 241*(3), 653–661.
Pan, Q. K., Gao, L., Li, X. Y., & Gao, K. Z. (2017). Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times. *Applied Mathematics & Computation, 303*, 89–112.
Pan, Q. K., Ruiz, R., & Alfaro-Fernandez, P. (2017). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research, 80*, 50–60.
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., & Wang, L. (2022). An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems. *IEEE Transactions on Cybernetics, 52*(7), 5999–6012.
Pan, Y., Gao, K., Li, Z., & Wu, N. (2022). *Solving Biobjective Distributed Flow-Shop Scheduling Problems With Lot-Streaming Using an Improved Jaya Algorithm*. IEEE Transactions on Cybernetics.
Rad, S. F., Ruiz, R., & Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *Omega, 37*(2), 331–345.
Rifai, A. P., Mara, S., & Sudiarso, A. (2021). Multi-objective distributed reentrant permutation flow shop scheduling with sequence-dependent setup time. *Expert Systems with Applications, 183*, Article 115339.
Ruiz, R., Pan, Q. K., & Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega, 83*, 213–222.
Steiner, G., & Zhang, R. (2011). Revised delivery-time quotation in scheduling with tardiness penalties. *Operations Research: The Journal of the Operations Research Society of America, 59*(6), 1504–1511.
Tukey, J. W. (1954). Some selected quick and easy methods of statistical analysis. *Transactions of the New York Academy of Sciences, 16*(2), 88–97.
Wang, G. C., Gao, L., Li, X. Y., & Tasgetiren, M. F. (2020). Energy-efficient distributed permutation flow shop scheduling problem using a multi-objective whale swarm algorithm. *Swarm and Evolutionary Computation, 57*, Article 100716.
Wang, J. J., & Wang, L. (2019). An iterated greedy algorithm for distributed hybrid flowshop scheduling problem with total tardiness minimization. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)* (pp. 350–355). IEEE.
Wang, J. J., & Wang, L. (2020). A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50*, 1805–1819.
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.
Xu, Y., Wang, L., Wang, S., & Liu, M. (2014). An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Engineering Optimization, 46*(9), 1269–1283.
Zhang, B., Pan, Q. K., Gao, L., Meng, L. L., & Peng, K. (2020). A three-stage multiobjective approach based on decomposition for an energy-efficient hybrid flowshop scheduling problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 50*(12), 4984–4999.
Zhao, F. Q., Hu, X. T., Wang, L., & Li, Z. K. (2022). A memetic discrete differential evolution algorithm for the distributed permutation flow shop scheduling problem. *Complex & Intelligent Systems, 8*, 141–161.