Knowledge-Based Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem☆

Weishi Shao [a], Zhongshi Shao [b,*], Dechang Pi [c,d]

[a] School of Computer Science and Technology, Nanjing Normal University, Nanjing, China
[b] School of Computer Science, Shaanxi Normal University, Xi'an, China
[c] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
[d] Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China

ARTICLE INFO

Article history:
Received 11 July 2019
Received in revised form 8 January 2020
Accepted 13 January 2020
Available online xxxx

Keywords:
Distributed hybrid flow shop scheduling problem
Iterated greedy algorithm
Multi-neighborhood
Decomposition based heuristic
Makespan

ABSTRACT

As economic globalization, large manufacturing enterprises build production centers in different places to maximize profit. Therefore, scheduling problems among multiple production centers should be considered. This paper studies a distributed hybrid flow shop scheduling problem (DHFSP) with makespan criterion, which combines the characteristic of distributed flow shop scheduling and parallel machine scheduling. In the DHFSP, a set of jobs are assigned into a set of identical factories to process. Each job needs to be through same route with a set of stages, and each stage has several machines in parallel and at least one of stage has more than one machine. For solving the DHFSP, this paper proposes two algorithms: DNEH with smallest-medium rule and multi-neighborhood iterated greedy algorithm. The DNEH with smallest-medium rule constructive heuristic first generates a seed sequence by decomposition and smallest-medium rule, and then uses a greedy iteration to assign jobs to factories. In the iterated greedy algorithm, a multi-search construction is proposed, which applies the greedy insertion to the factory again after inserting a new job. Then, a multi-neighborhood local search is utilized to enhance local search ability. The proposed algorithms are evaluated by a comprehensive comparison, and the experimental results demonstrate that the proposed algorithms are very competitive for solving the DHFSP.

© 2020 Elsevier B.V. All rights reserved.

1. Introduction

Along with intense market competition and individual customer demand, the centralized manufacturing seems deficient to respond to the market requirements [1–3]. Consequently, amount of enterprises extend their production into distributed environment, and build factories or production centers in different places. This decentralized structure in manufacturing can be considered as a type of distributed manufacturing [4], which brings many advantages, such as increasing product quality, reducing manufacturing period, delivery costs and risks. However, the production tasks must be allocated into suitable factories, that is the distributed scheduling problem. Behnamian and Ghomi [5] considered that there are two types of distributed shop scheduling problems. The first is that all factories or workshops are included in the same company. Obviously, its objective is to maximize the whole company's profit. The virtual enterprise network is another type, which consists of a number of individual different companies where companies can operate more economically than operating individually. However, in the virtual enterprise network, the companies do not care much about the others. The first type is considered in this paper.

Hybrid flow shop scheduling problem (HFSP) is a complicated optimization decision problem in manufacturing industry, which has been verified to be a *NP*-hard problem in most case. In the HFSP, a set of n jobs have to be processed in a series of s stages. Each stage k has $m_k \geq 1$ machines in parallel and at least one of stages has more than one machine. The HFSP can be regarded as a mixture of flow shop scheduling and parallel machine scheduling. The major issues of this problem are assigning jobs to machines at each stage, and deciding the processing order of jobs assigned to each machine. As seen in Fig. 1, it shows a cast steel valve production line including two workshops in an enterprise that manufactures industrial equipment. All valves are cast first, and after undergoing inspection operation and mechanical tests, they enter the production line. The valve production has to be through turning operation, hole-drilling, grinding operation,

☆ No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.knosys.2020.105527.

* Corresponding author.
E-mail addresses: shaoweishi@hotmail.com (W. Shao), shaozhongshi@hotmail.com (Z. Shao), nuaacs@126.com (D. Pi).

https://doi.org/10.1016/j.knosys.2020.105527
0950-7051/© 2020 Elsevier B.V. All rights reserved.

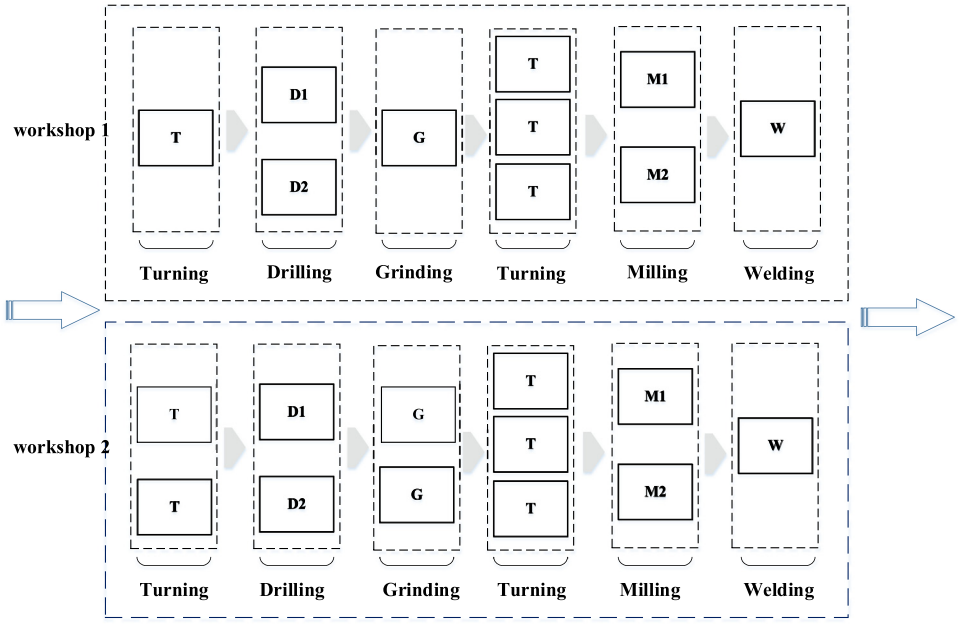

Fig. 1. The production process for cast steel valves.

welding operation. This processing procedure can be modeled as a distributed hybrid flow shop scheduling problem. There are a set of jobs that need to be allocated to a set of factories (workshops) to be processed, and each factory consists of a hybrid flow shop with several parallel machines. The distributed hybrid flow shop scheduling can be regarded as a mixture of distributed flow shop and parallel machine scheduling problem. This paper studies a simple variant, i.e., distributed hybrid flow shop scheduling with identical factories. Obviously, this problem consists of three sub-decisions: assigning jobs to factories, selecting machine for jobs, and determining the processing order on each machine.

The iterated greedy algorithm (IG) is a very simple and effective meta-heuristic, which is first proposed for solving the permutation flow shop scheduling problem [6]. It mainly performs two phases iteratively: destruction and construction. The destruction removes several jobs from the current individual, and the construction re-inserts these removed jobs into the sequence. A classic IG algorithm has very few control parameters, and it does not require specific knowledge of the problem as in sophisticated heuristic algorithms [7]. Due to its simplicity and effectiveness, the IG has been applied to solve amount of scheduling problems, such distributed flow shop scheduling [8–12], parallel batch machine scheduling problem [13], order scheduling [14], job shop scheduling [15].

Motivated by distributed factories and the realistic applications of hybrid flow shop scheduling problem, this paper studies the distributed hybrid flow shop scheduling problem. Compared to existing researches, the contribution of this paper can be summarized as follows. The DHFSP with makespan criterion is modeled, which is very common in many multinational manufacturers or multiple production lines that product like transformer, valve, paper, and so on. The research on DHFSP has more practical significance compared to DPFSP, since real production floors rarely use a single machine at each stage [16]. The duplicating parallel machines at stages can increase the throughput and capacity of the shop floor. We also establish the mathematical model of DHFSP. Then, a constructive heuristic (called DNEH_SMR) based on decomposition strategy is proposed, which first divides multiple stages to two-stages parallel machine scheduling, then constructs a seed sequence by using small–medium rule. Finally, the DNEH heuristic from literature is employed to assign jobs to factories. Due to the limited searching ability of heuristic, we also propose a meta-heuristic, i.e. iterated greedy (IG) algorithm. Since a superior meta-heuristic should make a tradeoff between intensification and diversification, we also adopt some techniques to balance them. The destruction and multi-search construction are used to generate a reconstructed solution whose purpose is to find some promising regions. Then, a multi-neighborhood local search including Insertion_between, Swap_between, Insertion_inner and Swap_inner is used to enhance intensification search ability. Furthermore, the proposed algorithms are also compared to other corresponding algorithms, and the experimental results shows that the proposed algorithms are competitive algorithms for solving the DHFSP.

The rest of this paper is organized as follows. Section 2 reviews recent literatures about distributed flow shops scheduling and distributed parallel machine scheduling. Section 3 describes the definition and mathematical model of DHFSP. Section 4 describes the detail of proposed DNEH_SMR. Section 5 shows each components of iterated greedy algorithm. Section 6 carries out the experiment results including parameter calibration, analysis of diversification strategy, intensification, instance factors, and comparison of other algorithms. Section 7 concludes this paper and proposes further research directions.

2. Literature review

Distributed manufacturing has been introduced in several enterprises in order to increase competitive advantages in the international economic area. For recent decade, the distributed scheduling in production area have attracted many scholars' attention due to its great application value. However, the research on distributed hybrid flow shop scheduling problem is very limited. Lin and Ying [17] first studied the distributed hybrid flow shop scheduling with multiprocessor tasks in which each job must be processed simultaneously by multiple parallel machines. A self-tuning iterated greedy algorithm incorporating an adaptive cocking decoding was presented. Hao et al. [18] considered the distributed hybrid flow shop scheduling with identical parallel machines. For this problem, a hybrid brain

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

storm optimization algorithm was presented in which a partial-mapped crossover is designed based on the property of distributed scheduling problem. Wang et al. [11] modeled a distributed hybrid flow shop scheduling with total tardiness, and presented an objective-driven decoding method and iterated greedy algorithm.

A feasible solution in DHFSP involves the factory that each job is assigned into, the machine that each job should be processed on at each stage, and the processing job order of each machine. Therefore, the distributed hybrid flow shop scheduling problem can be regarded as a combination of distributed flow shop scheduling and parallel machine scheduling. The following review summarizes the recent research works on these scheduling problems.

Naderi and Ruiz [8] first presented a distributed permutation flow shop scheduling problem (DPFSP) with makespan criterion. In the DPFSP, there are a set of jobs that are assigned to a set of factories, and each factory includes a permutation flow shop scheduling problem. The DPFSP includes two sub-decision problems: assigning jobs to suitable factory and deciding the order of job in factory. Naderi and Ruiz proposed six mathematical models and several heuristics. After that, a number of heuristics and meta-heuristics have been proposed for addressing the DPFSP, such as iterated greedy (IG) algorithm [9,19,20], genetic algorithm (GA) [21], scatter search (SS) algorithm, [22] estimation of distribution algorithm (EDA) [23], hybrid immune algorithm (HIA) [24], tabu search (TS) [25], variable neighborhood descent(VND) [26], chemical reaction optimization (CRO) [27], cuckoo search method [28]. Among these methods, the two-stage iterated greedy algorithm (IG2S) obtained the best results by testing the benchmark instances designed by Naderi and Ruiz [8]. The IG2S consists of two IG algorithms. After the first IG is finished, then the second IG will be performed for a limited amount of time. Additionally, Pan et al. [10] and Fernandez-Viagas et al. [29] discussed the DPFSP with total flow time criterion, and presented some problem properties and methods. Recently, there are many literatures that present a lot of variants of DPFSP by integrating some constraints or multi-objectives [30]. Lin et al. [31] integrated the no-wait constraint into DPFSP, in which each job could not wait between two consecutive operations. After that, Shao et al. [32] presented three iterated greedy algorithms, i.e. IG_VNS, IG_VND, IG_RNS, which have the best performance when solving this problem. Shao et al. [33] proposed a Pareto based EDA to solve the multi-objective distributed no-wait flow shop scheduling problem with makespan and total tardiness constraint. Like the above mentioned research, a number of researchers considered other constraints in the DPFSP, such as no-idle [34,35], blocking [36], assembly stage [37], limited buffers [38], etc. Moreover, Li et al. [39] considered a parallel batching distributed flow shop scheduling problem, which consists of two stages: the first stage includes a distributed flow shop scheduling problem, then each jobs is transferred and assembled in the second stage. Wang et al. [40] studied the energy-efficient DPFSP with the objectives of makespan and total energy consumption, and proposed a knowledge-based cooperative algorithm to solve it. Fu et al. [41] proposed a multi-objective brain storm optimization algorithm to solve the energy-conscious DPFSP with the total tardiness constraint.

Regarding another related distributed parallel machine scheduling problem (DPMSP), Behnamian [5] and Fatemi first considered a heterogeneous multi-factory production network scheduling (HMFPS). In the HMFPS, each factory includes several identical parallel machines, in which machines in different factories have different processing speed. The HMFPS should address two interdependent decisions: allocation of jobs to factories and determination of the production schedule at each factory. Moreover, a longest processing time-based heuristic and a genetic algorithm were proposed to solve the HMFSP. After that, Behnamian [42] considered the distributed parallel machine scheduling problem with different objectives in different factories. Some factories minimize the total processing cost and others maximize their production profits. They designed a multi-objective hybrid TS-VNS for addressing this problem. Behnamian [43] considered the case where factories are non-homogeneous, and assumed that each job can be transferred from its cluster to be processed in another factory, thus this contain can reduce workloads of factories. For deeper review of distributed scheduling problems in production, the readers can refer to literature [44] and [45].

Based on the above investigation, it can be concluded from following two points, the first is that more realistic complex production process should be modeled, however a number of researchers only integrate some constraints to classic scheduling problems, which may be not urgent problems to be solved or not suitable to real production. The second is that although some of methods have excellent effectiveness, these methods are not implemented easily for practitioners due to their complexity.

3. Problem statement

3.1. Problem definition

The distributed hybrid flow shop scheduling problem (DHFSP) is described as follows: a set J of jobs $J = \{1, 2, \ldots, n\}$ are assigned into a set F_s of factories $F_s = \{1, 2, \ldots, F\}$ to process. Each job has to be through same route with a set S of stages $S = \{1, 2, \ldots, s\}$. We permit some of stages can be skipped, i.e. there may be missing operation operations. Each stage k ($k \in S$) includes a set E_k of identical parallel machines $E_k = \{1, 2, \ldots, m_k\}$ ($m_k \geq 2$ at least for one stage). We denote Ω_i as the set of stages that the job $j \in J$ has to visit. Obviously, $1 \leq |\Omega_i| \leq s$. The process time $p_{i,k}$ of each job at each stages k is known. It should note that the classic HFSP contains three categories in terms of parallel machine type: (1) The HFSP with identical parallel machines, i.e. the job has same processing time on any parallel machines at each stage. (2) The HFSP with uniform parallel machines, i.e. the processing time of the same job on any parallel machines at each stage is inversely proportional to the processing speed of the machine. (3) The HFSP with unrelated parallel machines, i.e. the processing time of the job on any parallel machines at each stage is irrelevant, but depends on the match degree between job and machine. This paper considers the first case, and other cases can be easily obtained by extension.

Each job should be assigned to only one factory to process. Once a job has been assigned to a factory, this job should not transfer any other factories, and all of its stages should complete in this factory (suppose there is no broken-down machines). Furthermore, the following usual assumptions are also considered as the classical flow scheduling problem. Each machine can only process one job at a time. Each job can only be processed by one machine at a time. All jobs are independent and available for processing at time 0. No preemption is allowed when a job is being processed. There are infinite buffers between all stages, if a job needs a machine that is occupied, it waits indefinitely until it is available. Setup times for jobs and transportation time between two consecutive stages are included in the processing time at the corresponding stages or can be negligible.

The distributed hybrid flow shop scheduling is a decision-making process that should solve three sub-problems including assigning jobs to factories, machine selection for jobs, ordering jobs on machines. The task of DHFSP is to find a best scheduling solution to minimize the makespan among all factories. Naderi and Ruiz [8] have been proved the DPFSP with the makespan is a

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

NP-hard problem when $n > F$. The DPFSP can be considered as a special case of DHFSP, therefore the DHFSP is *NP*-hard problem too. Additionally, when $F = 1$, the DHFSP becomes a hybrid flow scheduling problem which is also a *NP*-hard problem, so devising a heuristic or meta-heuristic to solve this problem is highly desirable.

3.2. The mathematical model of DHFSP

Based on the model in literature [46], the mathematical model of DHFSP with makespan criterion is described as follows.

Parameters:

n Total number of job
s Total number of stage
F Total number of factory
i Index of job, $i = 1, 2, \ldots, n$
j Index of machine, $j = 1, 2, \ldots, m$
k Index of stage, $k = 1, 2, \ldots, S$
f Index of factory, $f = 1, 2, \ldots, F$
$p_{i,k}$ Processing time of job i at stage k
Ω_i Set of stages to be accessed by job i
E_k Set of parallel machines at stage k
m_k Number of parallel machines at stage k
L A very large positive number

Decision variables

$s_{i,k}$ Starting processing time of job i at stage k.

$$Y_{i,f} = \begin{cases} 1 & \text{if job } i \text{ is processed in factory } f \\ 0 & \text{otherwise} \end{cases} \quad (1)$$
$$i = 1, 2, \ldots, n, f = 1, 2, \ldots, F$$

$$X_{i,j,k} = \begin{cases} 1 & \text{if job } i \text{ is processed on machine } j \text{ at stage } k \\ 0 & \text{otherwise} \end{cases} \quad (2)$$
$$i = 1, 2, \ldots, n, j \in E_k, k \in \Omega_i$$

$$Z_{i,l,k} = \begin{cases} 1 & \text{if job } i \text{ precedes job } l \text{ at stage } k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$
$$i = 1, 2, \ldots, n; h, k \in \Omega_i$$

$$W_{i,l,k} = \begin{cases} 1 & \text{if job } j \text{ is to be processed at stage } h \\ & \text{immediately after its completion at stage } l \\ 0 & \text{otherwise} \end{cases} \quad (4)$$
$$i = 1, 2, \ldots, n; \quad l = 1, 2, \ldots, n; \quad k \in (\Omega_i \cap \Omega_l)$$

Objective:

$$f = \min\{\max_{i=1\ldots,n;\, k\in\Omega_i} \{s_{i,k} + p_{i,k}\}\} \quad (5)$$

Constraints:

Constraint sets (6) guarantee each job should be exactly at one factory.

$$\sum_{f=1}^{F} Y_{i,f} = 1, \quad i = 1, 2, \ldots, n \quad (6)$$

Constraint sets (7) ensure that each job must pass though all its stages and must be processed by exactly one machine at each stage.

$$\sum_{\substack{k\in\Omega_i \\ j\in E_k}} X_{i,j,k} = 1, \quad i = 1, 2, \ldots, n \quad (7)$$

Constraint sets (8) ensure that for two consecutive operations of a job, the next operation can only be started after the previous one is completed.

$$s_{i,h} - (s_{i,k} + p_{i,k}) - L \cdot (1 - W_{i,k,h}) \geq 0, \quad i = 1, 2, \ldots, n, \quad h, k \in \Omega_i, \quad j \in E_k \quad (8)$$

Table 1
The processing time of each job.

	Stage		
Job	Stage 1	Stage 2	Stage 3
J1	5	2	2
J2	4	3	4
J3	2	5	8
J4	5	5	6
J5	7	4	4
J6	3	3	2

Constraint sets (9) ensure that for any two different jobs, there exists an operation precedence.

$$Z_{l,i,k} + Z_{i,l,k} \leq 1, \quad i = 1, 2, \ldots, n, \quad l = 1, 2, \ldots, n, \quad k \in (\Omega_i \cap \Omega_l), \quad i \neq l \quad (9)$$

Constraint sets (10) ensure that for any jobs in the same factory that are assigned to the same machine, only when the preceding jobs is finished, the next job can be processed.

$$s_{l,k} - (s_{i,k} + p_{i,k}) - L \cdot (5 - Z_{l,i,k} - X_{i,j,k} - X_{l,j,k} - Y_{i,f} - Y_{l,f}) \geq 0 \quad (10)$$
$$z \in E_k, \quad k \in (\Omega_i \cap \Omega_l), \quad i = 1, 2, \ldots, n, \quad l = 1, 2, \ldots, n, \quad f = 1, 2, \ldots, F, \quad i \neq l$$

For large-sized optimization problem, constructive heuristics, meta-heuristics, approximation algorithms are often used since the mathematical model is difficult to solve by extract methods. This paper does not solve the model directly while use some constructive heuristics and a meta-heuristics to solve the problem. For deep reviews of other solution methods, literature [47–49] can be considered.

3.3. Solution representation and decoding strategy

The DHFSP consists of three sub-problems, i.e. assigning jobs to factories, selecting machine for jobs, ordering jobs on machines. If we adopt a directive encoding strategy, it will result in so large combinations of solutions. It will brings the difficulty for solving this problem. Therefore, this paper uses the permutation based encoding mechanism that is widely used in solving HFSP [50]. For the flow shop scheduling problems in distributed environment, the multi-permutations based encoding proposed by Naderi and Ruiz [22] is widely employed, in which each permutation represents the processing order of jobs in the corresponding factory. Regarding the HFSP, each permutation donates the processing order of jobs in the first stage. Then, the detail of encoding mechanism is shown as follows: There are a set of F lists $\boldsymbol{\pi} = [\boldsymbol{\pi}^1, \boldsymbol{\pi}^2, \ldots, \boldsymbol{\pi}^F]$, each of which $\boldsymbol{\pi}^f$ represents a factory. Each list contains the jobs assigned to each factory, and the sequence of jobs in the list represent the processing order of jobs in the first stage.

In order to decode the order of jobs at each stage, the earliest completion time rule (ECT) [51] is used in this paper. At the first stage, all jobs are processed according to their order of appearance in the solution representation $\boldsymbol{\pi}$. In the following stages, all jobs are sequenced in an ascending order according to their completion time at the previous stage. If some jobs have the same ready time at the beginning of stage t, $t = 2, \ldots, s$, we arrange them in the same relative order as in stage $t - 1$ [16]. Regarding the machine selection for jobs, the first available rule machine (FAM) [52,53] is employed, shown as the following procedure:

Step 1: Let $k = 1$, $C_{i,0} = 0$, $i = 1, 2, \ldots, n$.

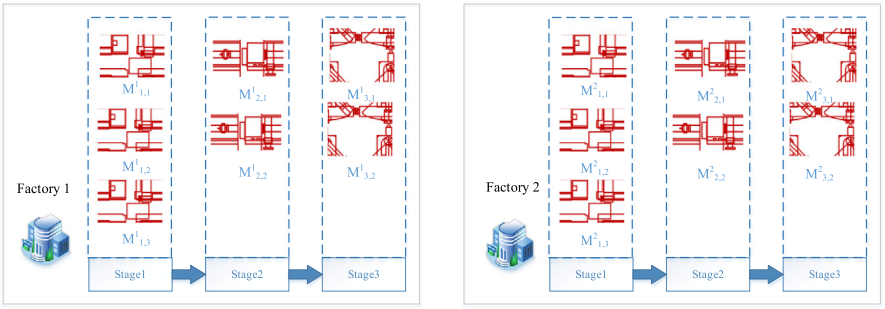

Fig. 2. The configuration of factories for an example.

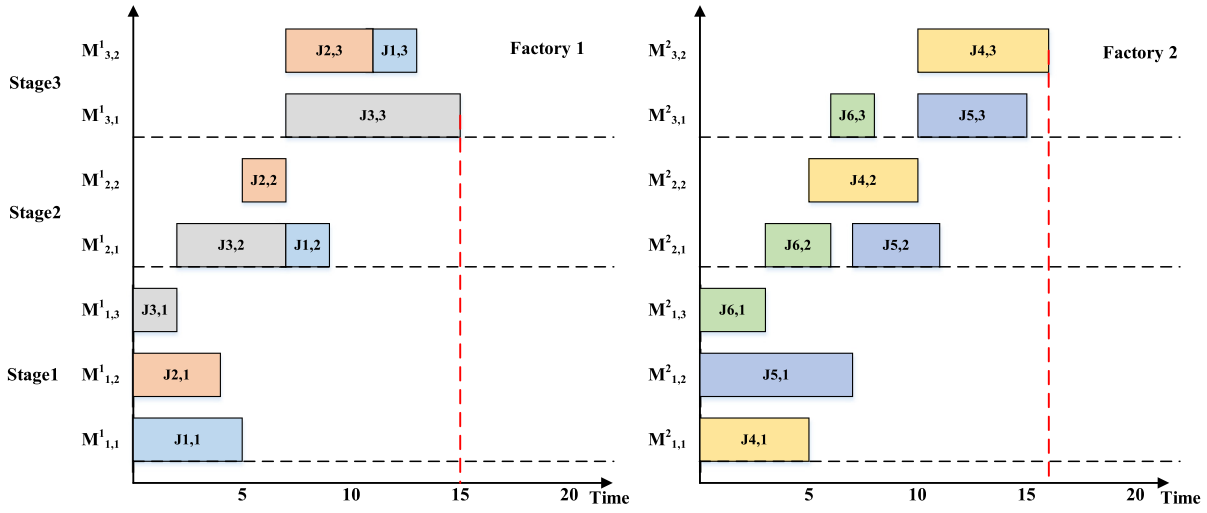

Fig. 3. The Gant char for DHFSP.

Step 2: Determine the earliest permitted time of each job J_i on machine j stage k, i.e. $\max r_j, C_{i,k-1}$, r_j is the releasing time of machine j, $C_{i,k-1}$ is the completion time of job i in stage $k-1$.

Step 3: For each job J_i, select the machine with minimum $\max r_j, C_{i,k-1} + p_{i,k}$.

Step 4: Update the completion time of job J_i at stage k and the releasing time of machine j.

Step 5: $k = k + 1$.

Step 6: Repeat step 2 to step 6, until all stages have been through.

In order to illustrate the DHFSP, encoding, and decoding mechanism clearly, we give a simple example here. Suppose $\boldsymbol{\pi} = [[1, 2, 3], [4, 5, 6]]$ represents a feasible solution. [1, 2, 3] denotes the processing order of job 1, 2, 3 at the first stage in the factory 1, i.e. $1 \rightarrow 2 \rightarrow 3$. Table 1 lists the processing time of each job at each stage. Fig. 2 displays the structure of each factory. Each factory includes three stages, and each stage includes several parallel machines. Table 2 shows the procedure of decoding. Fig. 3 shows a Gant chart for DHFSP.

4. Decomposition based heuristic for DHFSP

Jonhson's rule is a classic dispatch rule for two machines flow shop scheduling problem, which divides all jobs into two sets according some rules. Inspired by dividing strategies, this section proposes a decomposition based heuristic transferring multiple stages to two stages parallel machines scheduling problem. The heuristic, named DNEH_SMR, consists of three steps, shown as follows.

The first step divides $S = [S_1, S_2, \ldots, S_s]$ into two stages $S' = [S'_1, S'_2]$. Stage $S_1, S_2, \cdots, S_k$ make up the first stage S'_1, and the second stage S'_2 consists of the rest stages $S_{k+1}, S_{k+2}, \cdots, S_s$. The processing time of each job in S'_1 and S'_2 are defined as Eq. (11) and Eq. (12). If some jobs skip some stages, the processing time

Table 2
The procedure of decoding.

Stage	Previous completion time	Jobs processing order	Machine selection [J1, J2, J3] or [J4, J5, J6]	Releasing time of machines [$M^1_{1,1}$ to $M^1_{3,2}$] or [$M^2_{1,1}$ to $M^2_{3,2}$]	Starting processing time [J1, J2, J3] or [J4, J5, J6]	Completion time [J1, J2, J3] or [J4,J5,J6]
Stage 1		[1, 2, 3]	[$M^1_{1,1}$, $M^1_{1,2}$, $M^1_{1,3}$]	[5, 4, 2, 0, 0, 0, 0]	[0, 0, 0]	[5, 4, 2]
Stage 2	[5, 4, 2]	[3, 2, 1]	[$M^1_{2,1}$, $M^1_{2,2}$, $M^1_{2,1}$]	[5, 4, 2, 9, 7, 0, 0]	[7, 4, 2]	[9, 7, 7]
Stage 3	[9, 7, 7]	[3, 2, 1]	[$M^1_{3,2}$, $M^1_{3,2}$, $M^1_{3,1}$]	[5, 4, 2, 9, 7, 15, 13]	[11, 7, 7]	[13, 11, 15]
Stage 1		[4, 5, 6]	[$M^2_{1,1}$, $M^2_{1,2}$, $M^2_{1,3}$]	[5, 7, 3, 0, 0, 0, 0]	[0, 0, 0]	[5, 7, 3]
Stage 2	[5, 7, 3]	[6, 4, 5]	[$M^2_{2,2}$, $M^1_{2,1}$, $M^2_{2,1}$]	[5, 7, 3, 11, 10, 0, 0]	[5, 7, 3]	[10, 11, 6]
Stage 3	[10, 11, 6]	[6, 4, 5]	[$M^2_{3,2}$, $M^1_{2,1}$, $M^2_{3,1}$]	[5,7,3,11,10,15,16]	[10, 11, 6]	[16, 15, 8]

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

Algorithm 1 DNEH_SMR

Input : The processing time

Output: A feasible solution π

1: Let the dividing point $k = \lfloor s/2 \rfloor$

2: Compute the $t'_{i,1} = (1/k)\sum_{l=1}^{k} t_{i,l}$ and $t'_{i,2} = 1/(s-k)\sum_{l=k+1}^{s} t_{i,l}$, $i = 1, \cdots, n$

3: Sort all jobs in ascending order according to the mean processing time $t'_{i,2}$ in second stage, denoted as $[b_{\lambda 1}, b_{\lambda 2}, \cdots, b_{\lambda n}]$

4: Select the first and the middle job from $[b_{\lambda 1}, b_{\lambda 2}, \cdots, b_{\lambda n}]$ successively until all jobs have been checked, denoted as $[b_{\lambda 1}, b_{\lambda \lceil n/2 \rceil}, b_{\lambda 2}, b_{\lambda \lceil n \rceil + 1}, \cdots]$

5: **for** each job b_i **in** $[b_{\lambda 1}, b_{\lambda \lceil n/2 \rceil}, b_{\lambda 2}, b_{\lambda \lceil n \rceil + 1}, \quad]$ **do**

6: insert b_i into all possible positions in all factories of π, record the best position α and the factory $f*$ that result in the smallest makespan.

7: insert b_i into the position α in the factory $f*$.

8: **for** each job $\pi^{f*}(j)$ except b_i **in** π^{f*} **do**

9: remove $\pi^{f*}(j)$ and insert it to the best position that results in the smallest makespan in the factory $f*$.

10: **endfor**

11: **endfor**

of these stages are regarded as zero.

$$t'_{i,1} = \frac{1}{k}\sum_{l=1}^{k} t_{i,l}, \quad i = 1, \ldots, n \tag{11}$$

$$t'_{i,2} = \frac{1}{s-k}\sum_{l=k+1}^{s} t_{i,l}, \quad i = 1, \ldots, n \tag{12}$$

The second step sorts jobs according to their processing time to construct a complete job sequence. We improve *Large–medium rule* from [54] that is used to solve two stage parallel machine scheduling problem. Contrary to *Large–medium rule*, all jobs are sorted in ascending order according to the mean processing time $t'_{i,2}$ in second stage, the sorted sequence is defined as $[b_{\lambda 1}, b_{\lambda 2}, \ldots, b_{\lambda n}]$. Then a complete sequence is established by choosing the first and the middle job from $[b_{\lambda 1}, b_{\lambda 2}, \ldots, b_{\lambda n}]$ successively until all jobs are selected, i.e. $[b_{\lambda 1}, b_{\lambda \lceil n/2 \rceil}, b_{\lambda 2}, b_{\lambda \lceil n \rceil + 1}, \ldots]$. In total, there are also $s-1$ divided points, i.e. $k = 1, \ldots, s-1$. In order to reduce the computational time, this paper just selects one dividing point that is $k = \lfloor s/2 \rfloor$. This rule is named as *Small–medium rule* (SMR).

In order to further describe the SMR clearly, a simple example is given here. The processing time of each job is given in Table 1. The dividing point is $k = \lfloor s/2 \rfloor = \lfloor 3/2 \rfloor = 1$. The processing time of all jobs at two stages are $t'_1 = [5, 2]$, $t'_2 = [4, 3.5]$, $t'_3 = [2, 6.5]$, $t'_4 = [5, 5.5]$, $t'_5 = [7, 4]$, $t'_6 = [3, 2.5]$. Then, all jobs are sorted in ascending order according to the mean processing time $t'_{i,2}$ in second stage, i.e. [1, 6, 2, 5, 4, 3]. Finally, the first and the middle job from [1, 6, 2, 5, 4, 3] successively are selected to establish a complete sequence, i.e. [1, 5, 6, 4, 2, 3].

The third step assigns jobs from $[b_{\lambda 1}, b_{\lambda \lceil n/2 \rceil}, b_{\lambda 2}, b_{\lambda \lceil n \rceil + 1}, \ldots]$ into factories by using a greedy DNEH heuristic. The idea of DNEH (shorten for distributed NEH) coming from NEH2 proposed by Naderi and Ruiz [8] that has appeared in literature [32] and [9]. The DNEH first inserts the job into all possible positions in all factories, and finally inserts it into the position that results in the lowest makespan. Then for each job in this current factory that have included the new job, the iteration steps of the insertion are applied again, and the job is inserted into the best position. Comparing to NEH2, the DNEH can go through deeper solution space. The procedure of DNEH_SMR is shown as Algorithm 1.

5. Multi-neighborhood iterated greedy algorithm

The iterated greedy (IG) algorithm is a simple and effective type of meta-heuristics, which was first applied to solve scheduling problems by Ruiz and Stützle [6]. The framework of IG is shown as Algorithm 2. IG first employs NEH or its improved version to generate an initial solution, and then applies some local searches on the initial solution. The iteration procedure consists of four steps. The first step is Destruction, where some jobs of current solution are removed. Then there exist two partial sequences, i.e. the partial sequence including removed components π_d, and the other sequence π_r consists of rest jobs. The second step is Reconstruction that re-inserts jobs from π_d into π_r by using a greedy way. In the third step, local searches are applied again to the new reconstructed solution. Acceptance criterion is

Algorithm2 Classic iterated greedy algorithm

1: Generate an initial solution π_0

2: π = Local Search (π_0)

3: **while** termination criterion is not satisfied **do**

4: (π_D, π_r)= Destruction (π)

5: π' = Reconstruction (π_D, π_r)

6: π'' = Local Search (π')

7: π = Acceptance Criterion(π'', π_0)

8: **end while**

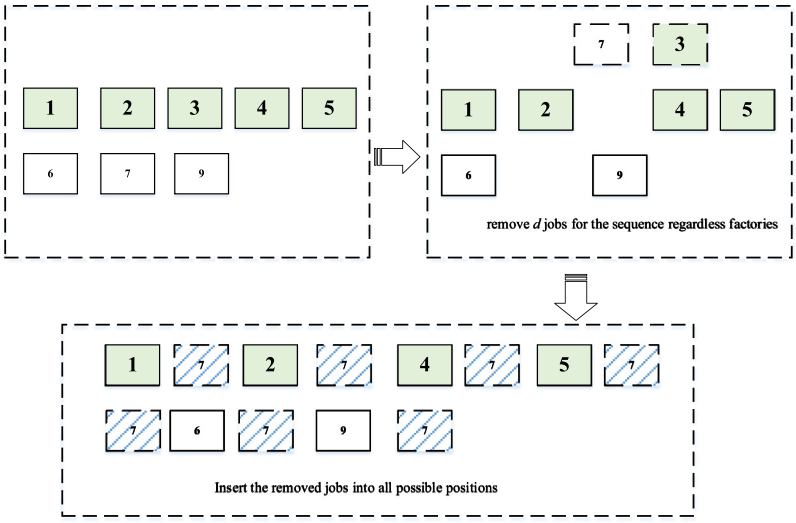

Fig. 4. An example for destruction and multi-search construction.

the last step, which decides whether this new solution is accepted to replace the current solution. If the new solution is better than the current solution, the current solution is replaced by the new solution; otherwise, the new solution is accepted according to a probability. The purpose of this operation is to explore more solution space. Although the procedure of IG is very simple, it also achieves diversification and intensification strategy. The destruction, reconstruction and acceptance criterion realize the leap of different neighbor, moreover the local search as intensification strategy achieves deep search of solution space.

5.1. Initial method

Many previous researches have stressed on the effect of a good initial solution for meta-heuristics. Nowadays, fewer meta-heuristics initialize a solution by random methods. Random solutions will result in decreasing convergence speed. Therefore, more and more meta-heuristics employ heuristics to generate initial solutions. High-performing initial solution can carry better information, and the search around it could be a promising region. IG starts with an initial solution which may partially influence the performance of IG. To preserve the quality of the initial solution, this paper employs DNEH_SMR as the initial method. In the experiment section, we still test the effect of other heuristics for distributed flow shop scheduling to verify the superiority of DNEH_SMR.

5.2. Diversification strategy

Diversification refers to the ability of visiting many and different search regions [55]. From the framework of the classic IG, it includes two diversification strategies: destruction and construction phase, acceptance criterion. The detail of acceptance criterion can be seen in Section 5.4. This section improves the destruction and construction from literature [20], shown as Algorithm 3. As seen in Fig. 4, in the destruction phase, d jobs regardless of factories are removed from the current solution in the selecting order. Let $\boldsymbol{\pi}_d$ denote the sequence of d jobs that have to be reinserted into the original partial sequence. The rest partial sequence is denoted by $\boldsymbol{\pi}_r$. The multi-search construction consists of two steps until a complete sequence of all n jobs is obtained. Firstly, a job $\pi_d(i)$ from $\boldsymbol{\pi}_d$ is re-inserted to all possible positions of the sequence of all factories, the best position is chosen. Let the chosen factory be denoted f_{rec}. In the second step, each job from the factory f_{rec} except $\pi_d(i)$ is extracted, and re-inserted the position of factory f_{rec} that results in minimum makespan. The multi-search construction goes deeper than the one used in literature [20], since it tries to go through more solution space.

5.3. Intensification strategy

Intensification means the ability of detection high-quality solutions in a small portion of the feasible search space and it is a powerful driver of the high-performing meta-heuristics [55]. Since the DHFSP will solve the machine selection, factory assignment, and job order on machines, the solution space of DHFSP is very large. Although some local search strategies, such as RLS2 [19], LS_insert, LS_swap [32], solution_combination method [22] have powerful exploitation ability for distributed flow shop scheduling, they may not be suitable to the DHFSP. These local search methods with complicated job moves may take much more computational time, it results in that the iteration of some meta-heuristics will be performed once in the limited running time. In order to make a tradeoff between exploitation ability and computational time, the following four local search methods are employed, which are classified two groups. The first group including Insertion_between and Swap_between is utilized to move jobs to other factories, whose purpose is to balance the processing capacity of factory. The second group including Insertion_inner and Swap_inner is used to change positions of jobs inside factory. Its purpose is to explore neighborhood space of critical factory. Let $C_{\max}$ denote the makespan of solution $\boldsymbol{\pi}$.

(i) Insertion_between. Each job in the factory $f_{\max}$ with the maximum makespan is extracted, and inserted into all possible positions of other factories. The position and the factory resulting in the minimum makespan are recorded as pos and f_{rec}. If both of the makespan of $f_{\max}$ and f_{rec} after extraction and insertion are better than $C_{\max}$, then the job is inserted to the position pos in factory f_{rec}, otherwise the next job in $f_{\max}$will be checked. The above procedure is performed until there is no improvement. It should note that if there are two factories that are have identical maximum makespan, the factory with more jobs is selected.

(ii) Insertion_inner. Each job in the factory $f_{\max}$ with the maximum makespan is extracted, and inserted into all other positions of $f_{\max}$. The position resulting in the minimum makespan is recorded as pos. If the makespan of $f_{\max}$ after extraction and insertion is better than $C_{\max}$, the job is inserted into pos; otherwise, the next job in $f_{\max}$will be checked. The above procedures is performed until there is no improvement.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

Algorithm 3 Destruction and multi-search construction

Input: A solution π, destruction size d
Output: A re-constructed feasible solution π
1: Randomly removed d jobs from π regardless of factories, and add them into π_d. The rest partial sequence is denoted as π_r.
2: **for** each job $\pi_d(i)$ **in** π_d **do**
3: $fitness$ = the minimum makespan by inserting $\pi_d(i)$ into all possible positions of all factories of π_r
4: record the factory and the position as f_{rec} and pos
5: insert $\pi_d(i)$ into position pos of factory f_{rec}
6: **for** each job $\pi_r^{f_{rec}}(j)$ **in** $\pi_r^{f_{rec}}$ **do**
7: **if** $\pi_r^{f_{rec}}(j)$!= $\pi_d(i)$ **then**
8: $fitness2$ = the minimum makespan by inserting $\pi_r^{f_{rec}}(j)$ into all positions of $\pi_r^{f_{rec}}$
9: record the position as $pos2$
10: **if** $fitness2 < fitness1$ **then**
11: insert $\pi_r^{f_{rec}}(j)$ into $pos2$ in $\pi_r^{f_{rec}}$
12: **end if**
13: **end if**
14: **end for**
15: $\pi = \pi_r$

(*iii*) Swap_between. Each job in the factory $f_{\max}$ with the maximum makespan is swapped with each jobs in other factories. The job and the factory resulting in the minimum makespan are recorded as *rec_job* and f_{rec}. If both of the makespan of $f_{\max}$ and f_{rec} after swapping are better than $C_{\max}$, then the job is swapped with *rec_job*; otherwise, the next job in $f_{\max}$will be checked. The above procedure is performed until there is no improvement.

(*iv*) Swap_inner. Each job in the factory $f_{\max}$ with the maximum makespan is swapped with other jobs of $f_{\max}$. The job resulting in the minimum makespan is recorded as *rec_job*. If both of the makespan of $f_{\max}$ after swapping is better than $C_{\max}$, then the job is swapped with *rec_job*, otherwise the next job in $f_{\max}$will be checked. The above procedures is performed until there is no improvement.

Algorithm 4 Multi-neighborhood local search

Input: A solution π
Output: A improved feasible solution π
1: $best = C_{\max}(\pi)$, $l_{\max} = 4$, $l = 1$
2: **while** $l \le l_{\max}$ **do**
3: **switch** l **do**
4: **case 1**: Insertion_between (π), break;
5: **case 2**: Swap_between (π), break;
6: **case 3**: Insertion_inner (π), break;
7: **case 4**: Swap_inner (π), break;
8: **end switch**
9: **if** $best < C_{\max}(\pi)$ **then**
10: $best = C_{\max}(\pi)$, $l = 1$
11: **else**
12: $l = l + 1$
13: **end if**
14: **end while**

Fig. 5 shows the examples for four local search methods. In order to provide convenience for further research, we give the detail of these local search methods in Appendix section. To integrate the above local search methods to IG, we employ a simple variable neighborhood descent framework [56,57] to organize them, shown as Algorithm 4. In Algorithm 4, if an improved solution is gotten with one of local search methods, then the loop return to the start of loop (i.e. $l = 1$); otherwise, the next local search is applied to the incumbent solution. The search procedure is completed when the maximum number of local search method meets.

5.4. Acceptance criterion

The proposed IG still adopts the simulated annealing based acceptance criterion [6] from classic IG to decide whether accept the new solution. The temperature T in the simulated annealing is shown as Eq. (13), where T_0 is a temperature parameter to be calibrated, which has been shown to be rather robust, and can works well from mostly any value different from zero and not overly high [9].

$$T = T_0 \times \frac{\sum_{i=1}^{n}\sum_{j=1}^{s} p_{i,j}}{10 \times n \times s} \tag{13}$$

Let π' denote a new solution produced by local search and π_{best} denote the current global best solution. π is the current solution. $r \in (0, 1)$ denotes a uniformly distributed random numbers. The acceptance criterion is as follows: if π' is better than π, then $\pi = \pi'$; otherwise, if $r \le \exp\{-(C_{\max}(\pi') - C_{\max}(\pi)))/T\}$, then $\pi = \pi'$. This acceptance criterion updates the current solution by accepting some worse solutions, it benefits to jump out a local optimum.

5.5. The framework of IG

Based on the above design, we propose a multi-neighborhood IG for solving the DHFSP. Firstly, DNEH_SMR is used to generate

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

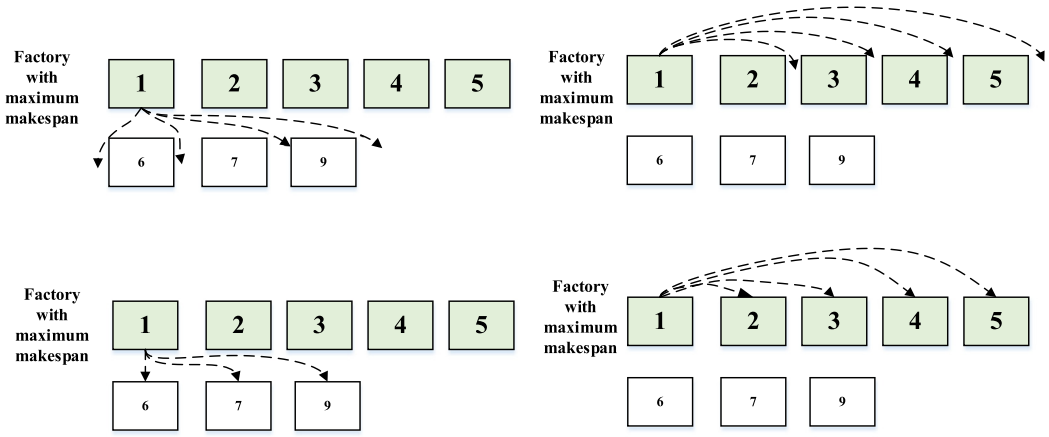

Fig. 5. The examples for four local search methods.

Algorithm 5 Iterated greedy algorithm for DHFSP

1: Initialize parameters and a solution π by using DNEH_SMR
2: $\pi^* = \pi$
3: **while** the termination condition is not met **do**
4: $\pi' = \pi$
5: Multi-neighborhood local search (π')
6: **if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
7: $\pi = \pi'$
8: **if** $C_{\max}(\pi') < C_{\max}(\pi^*)$ **then** $\pi^* = \pi'$ **end if**
9: **else**
10: **if** $random(0,1) < \exp(-(C_{\max}(\pi') - C_{\max}(\pi))/T)$ **then** $\pi = \pi'$ **end if**
11: **end if**
12: destruction and multi-search construction (π')
13: **end while**
14: return π^*

an initial solution. The iteration procedure of IG includes three operators: the first is the multi-neighborhood local search which uses four local search methods to improve solutions, i.e. Insertion_between, Swap_between, Insertion_inner, and Swap_inner. The second is destruction and multi-search construction. The destruction destroys the current solution by removing several jobs regardless of factories. Then we use multi-search construction to re-build this partial solution, which performs a greedy insertion again after inserting a job to the partial solution. Finally, the classic simulated annealing based acceptance criterion is adopted to decide whether the new solution is accepted. The procedure of IG for the DHFSP is shown as Algorithm 5.

6. Experimental results

In following sections, we carry out a comprehensive experiment to demonstrate the performance of proposed methods. Firstly, the proposed DNEH_SMR is compared to other heuristics. Secondly, the parameters of IG are calibrated by using Analysis of Variance (ANOVA) method. Then, we investigate the contribution and performance of main components of IG. Finally, the proposed IG is compared with other related algorithms, and the effect of instance factors on the experimental results are analyzed.

6.1. Experimental setup

Since this paper constitutes the first attempt at solving the DHFSP, there are no specific existing testing instance proposed in the literature. This section generates a set of 120 testing instances with $n = \{40, 60, 80, 100\}$ and $s = \{5, 10, 15\}$. Five instances are generated for each combination of n and s. The processing time are given by a discrete uniform distribution in the interval of [1, 99]. The number of identical parallel machines in each stage is generated uniformly in the interval of [1,5] as literature [55], where at least a stage has two or more parallel machines. The set of testing instances is classified into two groups. The first subset (1 ∼60, instance1) includes all stages, and the second instance set (61 ∼120, instances2) skips several stages, i.e. there may be missing operations. The number of factories f is from {2, 3, 4, 5, 6}. The average relative percentage deviation (ARPD) is employed to evaluate the quality of solutions, shown as Eq. (14).

$$ARPD = \frac{1}{R}\sum_{i=1}^{R}\frac{C_i - C_{opt}}{C_{opt}} \times 100\% \quad (14)$$

where R denotes the running times of a specific algorithm for a instance. Each algorithm is executed five replications for a given instance. C_i is the solution obtained by a specific algorithm in the ith experiments for a instance. C_{opt} is the best solution obtained

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

Table 3
The computational results of testing heuristics grouped by the number of factories (%).

Factory	DNEH_en	NEH2	DNEH + Dipak	DNEH_SMR
2	6.033	7.834	3.145	**2.978**
3	7.928	9.941	4.795	**3.674**
4	8.085	9.693	4.660	**3.592**
5	7.077	8.518	4.318	**2.699**
6	5.871	6.869	3.504	**2.191**
Total average	6.999	8.571	4.084	**3.027**

by any algorithms for that instance. All compared algorithms are carefully re-implemented by using Java language (JDK 1.8). It should be noted that since there are no specific algorithm for the DHFSP, some related algorithms used to solving other distributing flow shop scheduling are selected as referenced algorithms. The main structures of these algorithms are not changed, we just update the functions of computing objective and job assignment rule, and decoding. All considered algorithms are performed in the same experiment environment, i.e. a PC with Intel(R) Core(TM) i5-3470 CPU and 4G RAM. In order to make a fair comparison, each meta-heuristic adopt the same stopping condition, i.e. the maximum running time, $n^2 \times s \times F \times 0.1$ ms.

6.2. Comparison of heuristics

This section compares DNEH_SMR to DNEH + Dipak [32], NEH2 [8], DNEH_en [9]. DNEH + Dipak is proposed for distributed no-wait flow shop scheduling problem, in which a greedy insertion is applied again on the partial sequence after insertion a new job, and the seed sequence is generated by LPT rule. NEH2 only assigns the new job into factories by greedy insertion. DNEH_en randomly selects some jobs to perform greedy insertion again after inserting a new job. Table 3 lists the average relative percentage deviation values of testing heuristics grouped by the number of factories. From this table, it can be seen that DNEH_SMR obtains smaller ARPD values (3.027%) than other considered heuristics, i.e. DNEH_en (6.999%), NEH2 (8.571%), DNEH + Dipak (4.084%). Fig. 6 displays the means plot with 95% confidence interval for each heuristic. From this figure, it can be seen that there is no overlapping interval between DNEH_SMR and other heuristics in all scales of factory except $f = 2$. It demonstrates that the mean of DNEH_SMR is better than that of DNEH_en, NEH2, DNEH + Dipak significantly from statistical perspective. Compared to DNEH and NEH2, DNEH_SMR and DNEH+Dipak evaluate much more candidate solutions and exploit much space. However, more evaluations bring more computational cost than other heuristics. As seen in Fig. 7, DNEH_SMR, DNEH + Dipak, and DNEH_en take more computational time than NEH2 due to performing more greedy insertion. DNEH + Dipak and DNEH_en use LPT rule to generate the seed sequence, it may take lesser time than SMR rule. Additionally, the computational time of all heuristics reduce with the increasing number of factories. The main reason behind it may be that more factories would result in that each factory has lesser jobs to decode.

Since the initial method has an effect on the quality of solution, in the following experiment we employ different heuristics as the initial method to vary whether DNEH_SMR increases the performance of IG. The maximum running time of IG is set to $n^2 \times s \times F \times 0.1$ ms. Table 4 shows average relative percentage deviation values of proposed IG with different initial methods, i.e. DNEH_en, NEH2, random method, DNEH_SMR. The random method first randomly generates a seed job sequence, and then uses NEH2 and ECT rules to assign jobs to factories and machines. From Table 4, the total ARPD value of IG is 0.301% that is superior to IG_DNEH_en (0.404%), IG_NEH2 (0.464%), IG_random (0.464%).

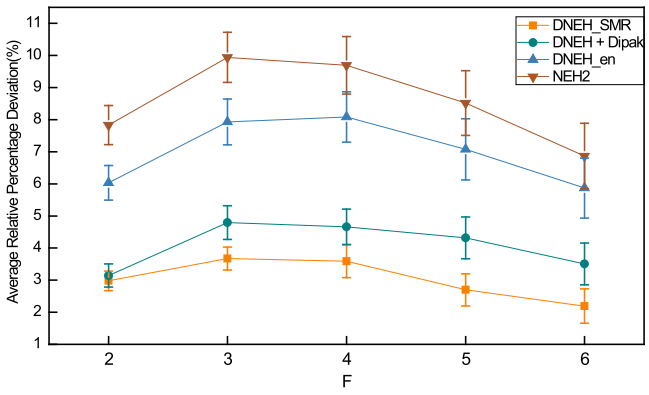

Fig. 6. Means plot with 95% confidence interval for heuristics.

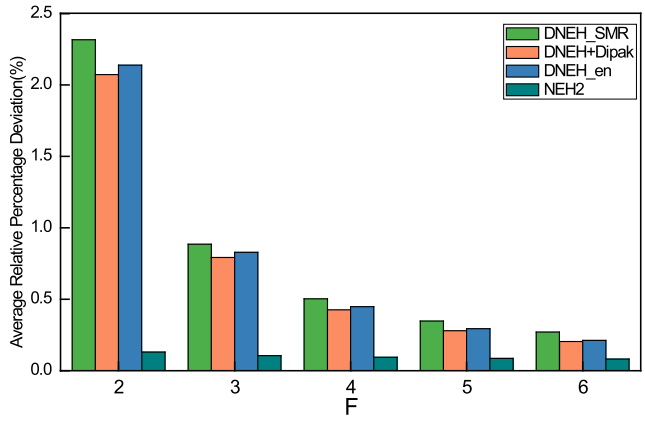

Fig. 7. The computational time of compared heuristics.

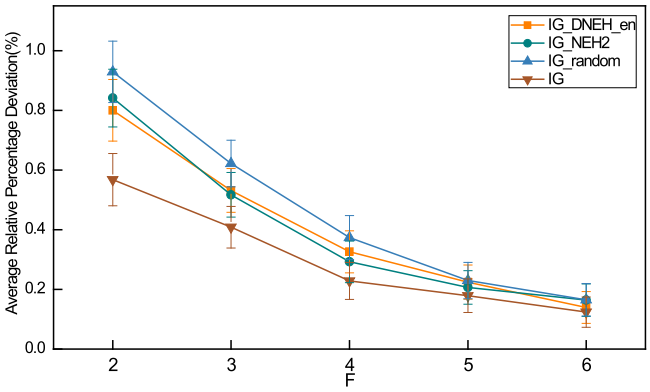

Fig. 8. Means plot and 95% confidence interval for IG with different initial methods.

From Fig. 8, the mean point of IG locates under other algorithms, and the gaps between each algorithm are gradually decreasing from $f = 2$ to $f = 6$. It demonstrates that as the number of factories increases, the influence of initial method will be reduces. To summary up, DNEH_SMR can bring an improvement for the performance of IG.

6.3. Parameter calibration

The parameters have an important effect on the performance of meta-heuristics. However, it does not mean all parameters can influence the searching results significantly because of using different searching mechanism. This section employs Analysis of

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

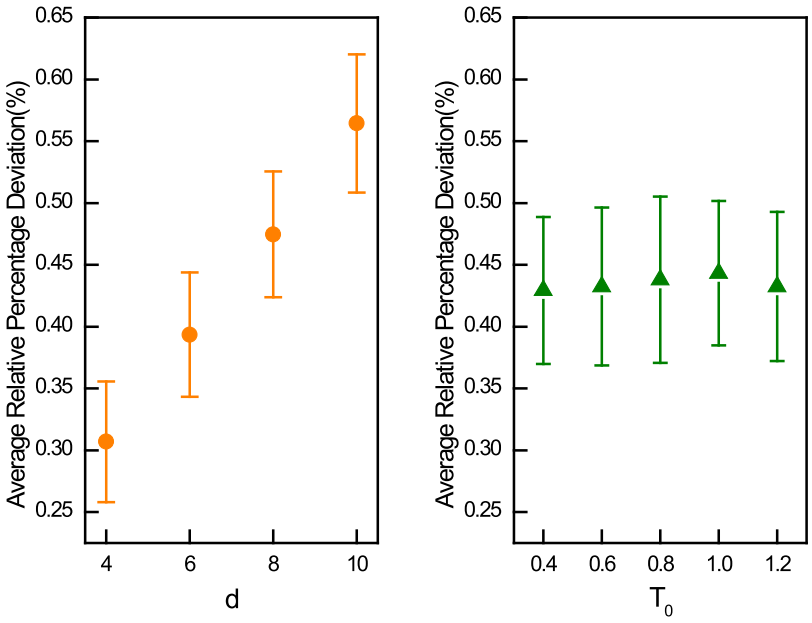

Fig. 9. Means plot with 95% confidence interval for d and T_0.

Table 4
The computational results of IG with different initial method grouped by the number of factories (%).

Factory	IG_DNEH_en	IG_NEH2	IG_random	IG
2	0.800	0.841	0.929	**0.568**
3	0.532	0.517	0.622	**0.408**
4	0.326	0.293	0.374	**0.228**
5	0.224	0.207	0.230	**0.179**
6	0.139	0.164	0.165	**0.124**
Total average	0.404	0.404	0.464	**0.301**

Variance method to analyze the influence of parameters on IG, and it selects the best parameters configuration according to the analyzed results. In order to avoid overfitting, we employ as set of randomly generated testing instances. The number of job, stage, and factory are $n = 30, 50, 70, 90$, $S = 5, 10$, $F = 3, 4, 5$. Each configuration has four instances where two instances include all processing stages, and the rest two instances randomly skip 20% stages. The processing time is given by a discrete uniform distribution in the interval of [1, 99]. The number of identical parallel machine at each stage is generated uniformly in the interval of [1,5]. The maximum running time is $n^2 \times s \times F \times 0.1$ ms.

The proposed IG algorithm includes two parameters: destruction size d and temperature adjusting parameter T_0. We carries out a full factorial design using following levels for these parameters: d: 4, 6, 8, 10 and $T_0 = 0.4, 0.6, 0.8, 1.0, 1.2$. The p-value of d is zero, which demonstrates that this parameter has the largest impact on IG. The destruction size controls searching diversification. If d is set to be too large, the destruction and construction phase become a random process. It may decrease search efficiency. If d has a too small value, the searching process may easily trap into local optimum. From Fig. 9, with the increasing size of d, the performance of IG gradually reduces. The best choice of d is 4. The effect of temperature adjusting parameter T_0 is not obvious whose p-value is larger than 0.05. The mean plot of T_0 also obtains the same results that the intervals of all candidate values are overlapping. However, it seems that $T_0 = 0.4$ provides better results among all candidate level. (See Table 5.)

Table 5
The ANOVA results of the parameters of IG.

Source	Sum. Sq.	d. f.	Mean Sq.	F	p-value
d	3.6441	3	1.2147	17.47	**0**
T_0	0.0108	4	0.00271	0.04	0.9971
$d * T_0$	0.3769	12	0.03141	0.45	0.9412

Table 6
The computational results of IG with different diversification strategies grouped by the number of factories (%).

Factory	IG_Des_Con	IG_Perturbation	IG
2	0.949	1.087	**0.568**
3	0.809	1.030	**0.408**
4	0.543	0.847	**0.228**
5	0.372	0.634	**0.179**
6	0.170	0.385	**0.124**
Total average	0.568	0.797	**0.301**

6.4. Analysis of diversification strategy

The destruction and construction strategy is an important part of IG, which determines diversification. In order to verify the contribution of improved destruction and construction, this section adopts different diversification strategies, i.e. destruction and construction from literature [19], perturbation mechanism from literature [58]. The perturbation mechanism randomly selects a job from one factory and inserts it into the best position of another plant that has been randomly selected. Table 6 lists the computational results of IG with different diversification strategies, in which the best values are marked in bold. As seen in Table 6, IG with improved destruction and construction yields the smallest total average relative percentage deviation (0.301%). For different number of factories, IG provides better results than other algorithms. From Fig. 10, the intervals of IG do not overlap with other algorithms except $f = 6$. It demonstrates that the contribution of destruction and construction from literature [19] and our improved version relatively close to each other with the increasing number of factory. From the above results, it can be concluded that the improved destruction and construction increases the performance of IG, and the reason may be that it goes deeper solution space than other diversification strategies.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

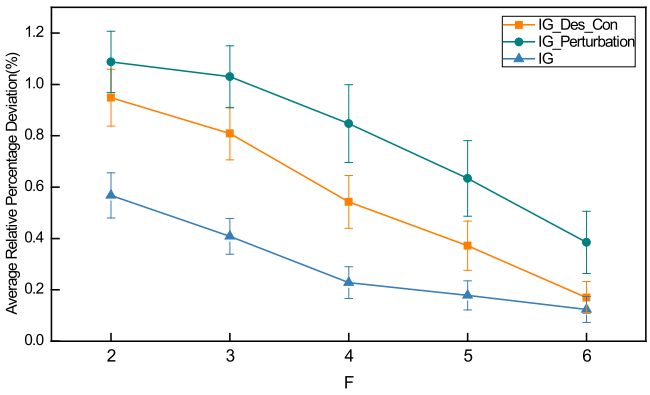

Fig. 10. Means plot with 95% confidence interval for IG with different diversification strategies.

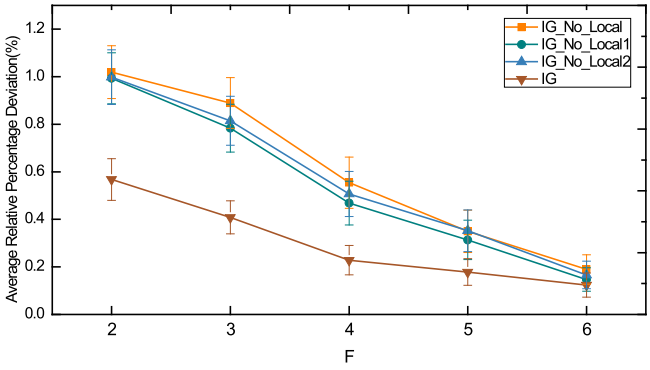

Fig. 11. Means plot with 95% confidence interval for IG with different local search methods.

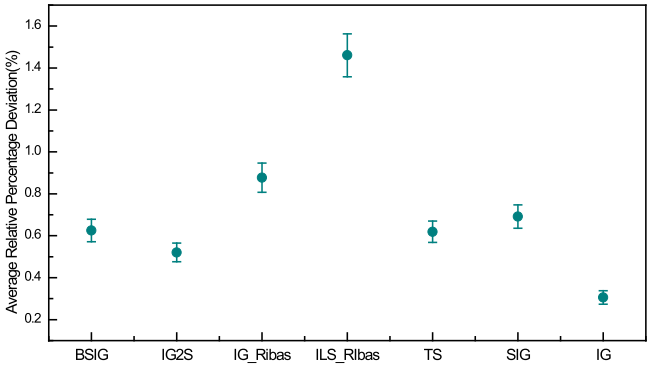

Fig. 12. Means plot with 95% confidence interval for testing algorithms.

6.5. Analysis of intensification strategy

This section carries out an experiment to verify whether the intensification strategy make an effective contribution to IG. Four versions of IG are run, i.e. IG_No_Local, IG_No_Local1, IG_No_Local2, IG. IG_No_local removing multi-neighborhood local search is used to demonstrate the effectiveness of multi-neighborhood local search. IG_No_Local1 and IG_No_Local2 remove job moving operators between factories (Insertion_between, Swap_between) and inside factory (Insertion_inner, Swap_inner) respectively. They are used to compare the performance of two classes of job moving operators. Table 7 shows the computational results of IG with different intensification strategies. It can be seen from this table that IG (0.301%) obtains much better results than IG_No_Local (0.600%), IG_No_Local1 (0.541%), IG_No_Local2 (0.567%), it demonstrates the effectiveness of combination of Insertion_between, Insertion_inner, Swap_between, Swap_inner. Compared IG_No_Local1 and IG_No_Local2, the job moving operators between factories produce slightly better results than that inside factories. From Fig. 11, the mean points of IG with multi-neighborhood local search locate under other algorithms, and there are no overlapping intervals between IG and other algorithms from $f = 2$ to $f = 5$. Therefore, comparison results indicate a clear advantage of the proposed intensification strategy.

Table 7
The computational results of IG with different intensification strategies grouped by the number of factories (%).

Factory	IG_No_Local	IG_No_Local1	IG_No_Local2	IG
2	1.020	0.994	0.998	**0.568**
3	0.889	0.784	0.814	**0.408**
4	0.554	0.469	0.507	**0.228**
5	0.350	0.314	0.352	**0.179**
6	0.189	0.147	0.166	**0.124**
Total average	0.600	0.541	0.567	**0.301**

Table 8
The computational results of each testing algorithm grouped by the number of factory, job, stage (%).

Factory	BSIG	IG2S	IG_Ribas	ILS_Ribas	TS	SIG	IG
2	0.915	0.985	1.349	1.758	0.856	0.973	0.568
3	0.864	0.724	1.238	1.941	0.796	0.961	0.408
4	0.546	0.405	0.920	1.538	0.611	0.711	0.228
5	0.583	0.289	0.681	1.216	0.507	0.472	0.179
6	0.218	0.190	0.198	0.860	0.324	0.340	0.124
Job							
40	0.436	0.337	0.524	0.865	0.456	0.405	0.178
60	0.569	0.444	0.812	1.321	0.586	0.594	0.259
80	0.750	0.630	1.095	1.843	0.743	0.843	0.360
100	0.745	0.670	1.078	1.814	0.691	0.923	0.425
Stage							
5	0.573	0.469	0.741	1.324	0.574	0.621	0.299
10	0.670	0.584	0.985	1.636	0.705	0.760	0.337
15	0.633	0.508	0.905	1.421	0.578	0.693	0.280
Total average	0.625	0.519	0.877	1.463	0.619	0.691	0.301

Table 9
Holm's procedure results of IG and other testing algorithms.

H_0	p	$\alpha/(k-i*+1)$	Holm's procedure: H_i
IG = ILS_Ribas	0.0000	0.0085	Reject
IG = IG_Ribas	0.0000	0.0102	Reject
IG = SIG	0.0000	0.0127	Reject
IG = BSIG	0.0000	0.0170	Reject
IG = TS	0.0000	0.0253	Reject
IG = IG2S	0.0000	0.0500	Reject

6.6. Comparison of other related algorithms

To evaluate the performance of IG, this section compares it to BSIG [19], IG2S [9], IG_Ribas, ILS_Ribas [58], TS [25], SIG [17]. BSIG, IG2S, IG_Ribas, ILS_Ribas, TS are proposed for DPFSP, and the SIG is proposed for the DHFSP with multiprocessors. Since all of above algorithms are not used to solve the DHFSP, we just repair decoding and objective function, and share most critical functions and algorithm structures. In fact, there are many superior methods for distributed flow shop scheduling problems, such as IG_RNS, IG_VND, IG_VND [32], SS [22], GK_LS [59]. In the experiment, we found that the iteration of these methods just run once and the running time exceeds limited running time more, so they degenerate into heuristics. Therefore, these methods may

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

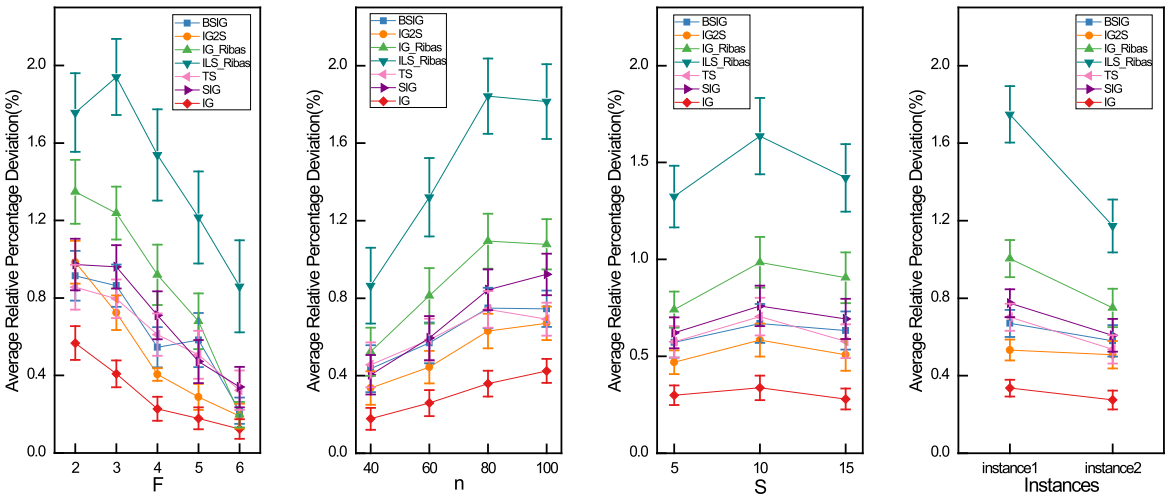

Fig. 13. Means plot with 95% confidence interval for the interaction between testing algorithms and instances factors, i.e., the number of job, stage, factory, instance.

Algorithm A.1 Insertion_between

Input: A solution π
Output: A improved feasible solution π
1: $\pi' = \pi$, $flag = true$, $best = C_{\max}(\pi')$
2: **while** $flag = true$ **do**
3: $flag = false$, $f_{\max}$ is set to the factory with maximum makespan among all factories
4: **for** each job $\pi'^{f_{\max}}(i)$ **in** $\pi'^{f_{\max}}$ **do**
5: $fitness$ = the minimum makespan by inserting $\pi'^{f_{\max}}(i)$ into all possible positions of all factories except $f_{\max}$.
6: record the factory and the position as f_{rec} and pos
7: $C_{\max}'$ = the makespan of $\pi'^{f_{\max}}$ extracting $\pi'^{f_{\max}}(i)$
8: **if** $fitness < best$ **and** $C_{\max}' < best$ **then**
9: remove $\pi'^{f_{\max}}(i)$ from $\pi'^{f_{\max}}$, and insert it into position pos of factory f_{rec}
10: $best = \max(fitness, C_{\max}')$, $flag = true$
11: **end if**
12: **end for**
12: **end while**
13: **if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
14: $\pi = \pi'$
15: **end if**

not be suitable to solve the DHFSP, since much more time is required in the local search phase and decoding. To make a fair comparison, each considered algorithm employs a same termination condition, i.e. maximum running time $n^2 \times s \times F \times 0.1$ ms, and the parameters of BSIG, IG2S, IG_Ribas, ILS_Ribas, TS, SIG are also calibrated by a full factorial design and ANOVA.

Table 8 lists the computational results of testing algorithms grouped by the number of factory, job, stage. From Table 8, it can be observed that IG obtains the best results among all compared algorithms in each scale of factory, job, stage. IG_Ribas and ILS_Ribas generate the worst results, and its reason may be their local search methods do not go deeper solution space. Fig. 12 shows the means plot with 95% confidence interval of BSIG, IG2S, IG_Ribas, ILS_Ribas, TS, SIG. From Fig. 12, no overlapping interval exists between IG and other algorithms, it demonstrates that there is a significance difference between them.

Furthermore, to further verify the statistical validity of the computational results obtained by these algorithms, Holm' procedure [60] is used to make multiple hypothesis testing. Table 9 lists Holm's procedure results of IG and other testing algorithms. $H_1, \ldots, H_k$ represent k hypotheses (this section supposes the results of IG have no significant difference from that of other algorithms). Let α denote the determined significant threshold for rejecting the null hypotheses. p_i is the p-value of the test H_i. Holm' procedure sorts k p-values in ascending order, represented by $p_1 \le p_2 \le \cdots \le p_k$. i^* is the smallest integer from 1 to k such that $p_{i^*} \ge \alpha/(k - i^* + 1)$. Holm's procedure rejects $H_1, \ldots, H_{i^*-1}$ and retains $H_{i^*}, \ldots, H_k$. From Table 9, the p-value for each hypotheses is less than $\alpha/(k - i * +1)$. It also demonstrates significance difference between IG and other algorithms. Therefore, it can be concluded that IG is an effective algorithm for solving DHFSP.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

Algorithm A.2 Insertion_inner

Input: A solution π
Output: A improved feasible solution π
1: $\pi' = \pi$, $flag = true$, $best = C_{\max}(\pi')$
2: **while** $flag = true$ **do**
3: $flag = false$, $f_{\max}$ is set to the factory with maximum makespan among all factories
4: **for** each job $\pi'^{f_{\max}}(i)$ **in** $\pi'^{f_{\max}}$ **do**
5: $fitness$ = the minimum makespan by inserting $\pi'^{f_{\max}}(i)$ into all other positions of $f_{\max}$
6: record the position as pos
8: **if** $fitness < best$ **then**
9: remove $\pi'^{f_{\max}}(i)$ from $\pi'^{f_{\max}}$, and insert it into position pos
10: $best = fitness$, $flag = true$
11: **end if**
12: **end for**
13: **end while**
14: **if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
15: $\pi = \pi'$
16: **end if**

Algorithm A.3 Swap_between

Input: A solution π
Output: A improved feasible solution π
1: $\pi' = \pi$, $flag = true$, $best = C_{\max}(\pi')$
2: **while** $flag = true$ **do**
3: $flag = false$, $f_{\max}$ is set to the factory with maximum makespan among all factories
4: **for** each job $\pi'^{f_{\max}}(i)$ **in** $\pi'^{f_{\max}}$ **do**
5: $fitness$ = the minimum makespan by swap $\pi'^{f_{\max}}(i)$ with all jobs of all factories except $f_{\max}$
6: record the factory and the job as f_{rec} and rec_job
7: $C_{\max}'$ = the makespan of $\pi'^{f_{\max}}$ swapping $\pi'^{f_{\max}}(i)$ and rec_job
8: **if** $fitness_{\min} < best$ **and** $C_{\max}' < best$ **then**
9: swap $\pi'^{f_{\max}}(i)$ and rec_job in factory f_{rec}
10: $best = C_{\max}(\pi')$, $flag = true$
11: **end if**
12: **end for**
13: **end while**
14: **if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
15: $\pi = \pi'$
16: **end if**

6.7. Analysis of instance factors

This section evaluates the relationship between the performance of algorithms and factors affecting characteristic of instances, like the number of job (n), factory (F), stage (s), instance. The ANOVA method is employed in this section, which shows a mean interaction plot with 95% confidence interval for each factor. As seen in Fig. 13, it can be drawn following conclusions.

(1) Each compared algorithm is sensitive to the number of factory, job, and instance. The mean plots of IG clearly lie under that of other algorithms, and there is only one overlapping interval between IG and other algorithms, i.e. $f = 6$. It demonstrates that IG provides significantly better results than others. Surprisingly, with increasing factories, the ARPD values of most algorithm decrease. When $f = 6$, the gaps between them relatively are close to each other.

(2) From the mean plot of interaction for job and algorithm, it can be observed that each algorithm performs worse as the number of job increase. However, IG is still the most competitive algorithm when solving large-scale instances. The DHFSP would become more complex as the number of job increase, since the solution space is expanded.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

Algorithm A.4 Insertion_inner

Input: A solution π

Output: A improved feasible solution π

1: $\pi' = \pi$, $flag = true$, $best = C_{\max}(\pi')$
2: **while** $flag = true$ **do**
3: $flag = false$, $f_{\max}$ is set to the factory with maximum makespan among all factories
4: **for** each job $\pi'^{f_{\max}}(i)$ **in** $\pi'^{f_{\max}}$ **do**
5: $fitness$ = the minimum makespan by swapping $\pi'^{f_{\max}}(i)$ with all other jobs in $f_{\max}$
6: record the job as rec_job
8: **if** $fitness < best$ **then**
9: swap $\pi'^{f_{\max}}(i)$ and rec_job
10: $best = fitness$, $flag = true$
11: **end if**
12: **end for**
13: **end while**
14: **if** $C_{\max}(\pi') < C_{\max}(\pi)$ **then**
15: $\pi = \pi'$
16: **end if**

(3) Regarding testing instance set, all algorithms obtain better results when running instances skipping some stages. Then, it can draw the same results that the higher the percentage of missing operations, the less harder the problem is.

(4) After checking the effect of the number of stages, we found no remarkable effects from these characteristic on the performance of algorithm. From the mean plot of stage, the difference between each levels of stages are not significant.

6.8. Implication for practitioners

Scheduling becomes more challenges when facing multiple or distributed factories. Inspired by the process of valve production, this paper models the distributed hybrid flow shop scheduling problem. This scheduling model can be extended to other practical problems by considering other constraints, such as assembly process, mixed no-wait, and transporting times. However, the following basic conditions that should be met by industrial case. (1) There are multiple factories or production lines that lies in different area. (2) The processed jobs must be through multiple stages. (3) Each stage includes a set of parallel machines. (at least one stage includes more than one machine). From the above experimental results, the proposed DNEH_SMR and the IG show better performance than other algorithms. Then they can be applied to solve other distributed scheduling problems, once their constraints are known. Furthermore, due to the distributed nature, the results of this paper can provide effective and efficient models and optimization methods to multi-processing centers, multi-workshops, multi-factories, cross-regional and multinational enterprises.

7. Conclusion and future work

In this paper, we address the distributed hybrid flow shop scheduling (DHFSP), which has receive little attention in literature, but it can be widely found in real world scenario, such as paper, photographic film, steel-making casting problems in multiple processing production centers. The DHFSP can be regarded as an integration of distributed permutation flow shop and parallel machines scheduling. To solve this problem, two algorithms including DNEH_SMR and IG are proposed. The DNEH_SMR is a constructive heuristic that employs a decomposition strategy and *smallest-medium rule* to build a seed sequence, and then it uses DNEH to assign jobs to factories. In the IG, a multi-search construction is presented which applies the greedy insertion to the current factory after inserting a new job. In the local search step, a multi-neighborhood local search integrated variable neighborhood descent framework is employed to improve reconstructed solution. A series of comparative studies are carried out to evaluate the performance of proposed algorithms. From the comparative results, it can draw the following conclusions. (1) The results of DNEH_SMR and IG are superior to other corresponding algorithms. (2) The diversification and intensification strategies have an important effect on the performance of IG. (3) The scale of local search should be controlled since oversize local search may decrease the effectiveness of meta-heuristics or turn it into a heuristic in the limited running time.

Regarding future work, the multi-objective DHFSP is an interesting direction, which should consider load balancing in each workshop, energy efficiency, production cost. Then, the DHFSP with rescheduling mechanism is also an important direction, since the distributed shop scheduling should have some fault-tolerant mechanisms to process emergency situations, like machinery breakdown, advancing delivery due, and so on.

CRediT authorship contribution statement

Weishi Shao: Funding acquisition, Investigation, Methodology, Software, Writing - original draft. **Zhongshi Shao:** Funding acquisition, Investigation, Writing - review & editing. **Dechang Pi:** Supervision, Writing - review & editing, Project administration.

Acknowledgments

This research is supported by the Natural Science Research of Jiangsu Higher Education Institutions of China (NO. 19KJB520042), Research Startup Fund of Shaanxi Normal University, Research Startup Fund of Nanjing Normal University.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

Appendix. Pseudo codes of local search method

See Algorithms A.1–A.4.

References

[1] Y. Hao, P. Helo, A. Shamsuzzoha, Virtual factory system design and implementation: integrated sustainable manufacturing, Internat. J. Systems Sci. 5 (2) (2018) 116–132.
[2] K. Li, X. Zhang, J.Y.T. Leung, B-y. Cheng, Integrated production and delivery withmultiple factories and multiple customers, Internat. J. Systems Sci. 4 (3) (2017) 219–228.
[3] S.A. Hoseini Shekarabi, A. Gharaei, M. Karimi, Modelling and optimal lot-sizing of integrated multi-level multi-wholesaler supply chains under the shortage and limited warehouse space: generalised outer approximation, Internat. J. Systems Sci. 6 (3) (2019) 237–257.
[4] M. Yazdani, S. Gohari, B. Naderi, Multi-factory parallel machine problems: Improved mathematical models and artificial bee colony algorithm, Comput. Ind. Eng. 81 (2015) 36–45.
[5] J. Behnamian, S.M.T. Fatemi Ghomi, The heterogeneous multi-factory production network scheduling with adaptive communication policy and parallel machine, Inform. Sci. 219 (2013) 181–196.
[6] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European J. Oper. Res. 177 (3) (2007) 2033–2049.
[7] Q.K. Pan, R. Ruiz, An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem, Omega 44 (2) (2014) 41–50.
[8] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Comput. Oper. Res. 37 (4) (2010) 754–768.
[9] R. Ruiz, Q.-K. Pan, B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem, Omega 83 (2019) 213–222.
[10] Q.-K. Pan, L. Gao, L. Wang, J. Liang, X.-Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, Expert Syst. Appl. 124 (2019) 309–324.
[11] J. Wang, L. Wang, An iterated greedy algorithm for distributed hybrid flow-shop scheduling problem with total tardiness minimization, in: 2019 IEEE 15th International Conference on Automation Science and Engineering, CASE, 2019, pp. 350–355.
[12] Q.-K. Pan, L. Gao, L. Xin-Yu, F.M. Jose, Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem, Appl. Soft Comput. 81 (2019) 105492.
[13] J.E.C. Arroyo, J.Y.T. Leung, R.G. Tavares, An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times, Eng. Appl. Artif. Intell. 77 (2019) 239–254.
[14] C.-C. Wu, T.-H. Yang, X. Zhang, C.-C. Kang, I.H. Chung, W.-C. Lin, Using heuristic and iterative greedy algorithms for the total weighted completion time order scheduling with release times, Swarm Evol. Comput. 44 (2019) 913–926.
[15] G. Deng, Q. Su, Z. Zhang, H. Liu, S. Zhang, T. Jiang, A population-based iterated greedy algorithm for no-wait job shop scheduling with total flow time criterion, Eng. Appl. Artif. Intell. 88 (2020) 103369.
[16] B. Naderi, R. Ruiz, M. Zandieh, Algorithms for a realistic variant of flowshop scheduling, Comput. Oper. Res. 37 (2) (2010) 236–246.
[17] K.-C. Ying, S.-W. Lin, Minimizing makespan for the distributed hybrid flowshop scheduling problem with multiprocessor tasks, Expert Syst. Appl. 92 (2018) 132–141.
[18] J. Hao, J. Li, Y. Du, M. Song, P. Duan, Y. Zhang, Solving distributed hybrid flowshop scheduling problems by a hybrid brain storm optimization algorithm, IEEE Access 7 (2019) 66879–66894.
[19] J.M. Framinan, A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 53 (4) (2015) 1111–1123.
[20] S. Lin, K. Ying, C. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, Int. J. Prod. Res. 51 (16) (2013) 5029–5038.
[21] J. Gao, R. Chen, A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem, Int. J. Comput. Intell. Syst. 4 (4) (2011) 497–508.
[22] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, European J. Oper. Res. 239 (2) (2014) 323–334.
[23] S.Y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, Int. J. Prod. Econ. 145 (1) (2013) 387–396.
[24] Y. Xu, L. Wang, S. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, Eng. Optim. 46 (9) (2014) 1269–1283.
[25] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (3) (2013) 1–11.
[26] R.B. Naderi, Variable Neighborhood Descent Methods for the Distributed Permutation Flowshop Problem, 2009.
[27] H. Bargaoui, O. Belkahla Driss, K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, Comput. Ind. Eng. 111 (2017) 239–250.
[28] J. Wang, L. Wang, J. Shen, A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem, in: 2016 IEEE Congress on Evolutionary Computation, CEC, 2016, pp. 2240–2246.
[29] V. Fernandez-Viagas, P. Perez-Gonzalez, J.M. Framinan, The distributed permutation flow shop to minimise the total flowtime, Comput. Ind. Eng. 118 (2018) 464–477.
[30] S. Cai, K. Yang, K. Liu, Multi-objective optimization of the distributed permutation flow shop scheduling problem with transportation and eligibility constraints, J. Oper. Res. Soc. China (2017) (Online).
[31] S.W. Lin, K.C. Ying, Minimizing makespan for solving the distributed no-wait flowshop scheduling problem, Comput. Ind. Eng. 99 (2016) 202–209.
[32] W. Shao, D. Pi, Z. Shao, Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms, Knowl.-Based Syst. 137 (2017) 163–181.
[33] W. Shao, D. Pi, Z. Shao, A Pareto-based estimation of distribution algorithm for solving multiobjective distributed no-wait flow-shop scheduling problem with sequence-dependent setup time, IEEE Trans. Autom. Sci. Eng. 16 (3) (2019) 1344–1360.
[34] C.-Y. Cheng, K.-C. Ying, H.-H. Chen, H.-S. Lu, Minimising makespan in distributed mixed no-idle flowshops, Int. J. Prod. Res. 57 (1) (2019) 48–60.
[35] W. Shao, D. Pi, Z. Shao, Local search methods for a distributed assembly no-idle flow shop scheduling problem, IEEE Syst. J. 13 (2) (2018) 1945–1956.
[36] I. Ribas, R. Companys, X. Tort-Martorell, An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem, Expert Syst. Appl. 121 (2019) 347–361.
[37] S. Hatami, R. Ruiz, C. Andrés-Romano, The distributed assembly permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (17) (2013) 5292–5308.
[38] G. Zhang, K. Xing, Differential evolution metaheuristics for distributed limited-buffer flowshop scheduling with makespan criterion, Comput. Oper. Res. 108 (2019) 33–43.
[39] J. Li, M. Song, L. Wang, P. Duan, Y. Han, H. Sang, et al., Hybrid artificial bee colony algorithm for a parallel batching distributed flow-shop problem with deteriorating jobs, IEEE Trans. Cybern. (2019) 1–15.
[40] J. Wang, L. Wang, A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop, IEEE Trans. Syst. Man Cybern. A (2018) 1–15.
[41] Y. Fu, G. Tian, A.M. Fathollahi-Fard, A. Ahmadi, C. Zhang, Stochastic multi-objective modelling and optimization of an energy-conscious distributed permutation flow shop scheduling problem with the total tardiness constraint, J. Cleaner Prod. 226 (2019) 515–525.
[42] J. Behnamian, Decomposition based hybrid VNS–TS algorithm for distributed parallel factories scheduling with virtual corporation, Comput. Oper. Res. 52 (2014) 181–191.
[43] J. Behnamian, Matheuristic for the decentralized factories scheduling problem, Appl. Math. Model. 47 (2017) 668–684.
[44] J. Behnamian, S.M.T.F. Ghomi, A survey of multi-factory scheduling, J. Intell. Manuf. 27 (1) (2016) 231–249.
[45] L. Wang, J. Deng, S. Wang, Survey on optimization algorithms for distributed shop scheduling, Control Decis. 31 (1) (2016) 1–10.
[46] J. Li, Q. Pan, P. Duan, An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping, IEEE Trans. Cybern. 46 (6) (2016) 1311–1324.
[47] A. Gharaei, S.A. Hoseini Shekarabi, M. Karimi, Modelling and optimal lot-sizing of the replenishments in constrained, multi-product and bi-objective EPQ models with defective products: Generalised cross decomposition, Internat. J. Systems Sci. (2019) 1–13.
[48] A. Gharaei, M. Karimi, S.A. Hoseini Shekarabi, Joint economic lot-sizing in multi-product multi-level integrated supply chains: Generalized benders decomposition, Internat. J. Systems Sci. (2019) 1–17.
[49] A. Gharaei, S.A. Hoseini Shekarabi, M. Karimi, E. Pourjavad, A. Amjadian, An integrated stochastic EPQ model under quality and green policies: generalised cross decomposition under the separability approach, Internat. J. Systems Sci. (2019) 1–13.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.

[50] R. Ruiz, J.A. Vázquez-Rodríguez, The hybrid flow shop scheduling problem, European J. Oper. Res. 205 (1) (2010) 1–18.
[51] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, European J. Oper. Res. 169 (3) (2006) 781–800.
[52] S.A. Brah, L.L. Loo, Heuristics for scheduling in a flow shop with multiple processors, European J. Oper. Res. 113 (1) (1999) 113–122.
[53] S. Wang, L. Wang, Y. Xu, G. Zhou, An estimation of distribution algorithm for solving hybrid flow-shop scheduling problem, Acta Automat. Sinica 38 (03) (2012) 437–443 (in Chinese).
[54] S. Wang, M. Liu, A genetic algorithm for two-stage no-wait hybrid flow shop scheduling problem, Comput. Oper. Res. 40 (4) (2013) 1064–1075.
[55] K. Peng, Q.-K. Pan, L. Gao, X. Li, S. Das, B. Zhang, A multi-start variable neighbourhood descent algorithm for hybrid flowshop rescheduling, Swarm Evol. Comput. 45 (2019) 92–112.
[56] P. Hansen, N. Mladenović, J. Moreno Pérez, Variable neighbourhood search: methods and applications, 4OR 6 (4) (2008) 319–360.
[57] F. Zhao, Y. Liu, Y. Zhang, W. Ma, C. Zhang, A hybrid harmony search algorithm with efficient job sequence scheme and variable neighborhood search for the permutation flow shop scheduling problems, Eng. Appl. Artif. Intell. 65 (2017) 178–199.
[58] I. Ribas, R. Companys, X. Tort-Martorell, Efficient heuristics for the parallel blocking flow shop scheduling problem, Expert Syst. Appl. 74 (2017) 41–54.
[59] J. Gao, R. Chen, Y. Liu, A knowledge-based genetic algorithm for permutation flowshop scheduling problems with multiple factories, Int. J. Adv. Comput. Technol. 4 (7) (2012) 121–129.
[60] S. Holm, A simple sequentially rejective multiple test procedure, Scand. J. Stat. 6 (2) (1979) 65–70.

Please cite this article as: W. Shao, Z. Shao and D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowledge-Based Systems (2020) 105527, https://doi.org/10.1016/j.knosys.2020.105527.