Applied Soft Computing 112 (2021) 107750

Contents lists available at ScienceDirect

Applied Soft Computing

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

An optimal block knowledge driven backtracking search algorithm for distributed assembly No-wait flow shop scheduling problem

Fuqing Zhao [a,*], Jinlong Zhao [a], Ling Wang [b], Jianxin Tang [a,*]

[a] School of Computer and Communication Technology, Lanzhou University of Technology, Lanzhou 730050, China
[b] Department of Automation, Tsinghua University, Beijing, 10084, China

ARTICLE INFO

Article history:
Received 17 January 2021
Received in revised form 1 July 2021
Accepted 21 July 2021
Available online 26 July 2021

Keywords:
Distributed assembly No-wait flow shop scheduling
Job block knowledge
Constructive heuristics
Backtracking search algorithm

ABSTRACT

The distributed assembly flow shop scheduling problem (DAFSP) is an important scenario in manufacturing system. In this paper, an optimal block knowledge driven backtracking search algorithm (BKBSA) is proposed to solve the distributed assembly No-wait flow shop scheduling problem (DANWFSP) with the objective of minimizing the completion time of assembly process. In BKBSA, three constructive heuristics are proposed to generate a competitive initial solution. Block-shifting based on block knowledge is embedded in the mutation strategy of BKBSA. The proposed block-shifting ensures that the optimal subsequence of a candidate solution is not destroyed in the mutation operation. The similarity between candidate solutions is utilized as feedback indicator to control the utilization of block-shifting. In addition, the VND algorithm based on factory-to-factory is proposed to further improve the optimal solution. Finally, the BKBSA and the other three state-of-the-art algorithms for DANWFSP are tested on 810 large-scale instances and 900 small-scale instances. The statistical analysis results show that BKBSA is an effective algorithm to solve DANWFSP.

1. Introduction

Economic globalization has promoted the international division of labor and the development of the world market, and has realized the optimal allocation of global resources and the deep integration of global interests. Under the background of globalization, the competition among enterprises is increasingly fierce. Efficient production mode is an effective means for enterprises to improve production efficiency and maintain their competitiveness. Therefore, the distributed assembly scheduling problem based on multiple production centers is becoming more and more popular.

Assembly scheduling problems are mainly divided into production stage and assembly stage [1]. In the production phase, all the jobs that make up the product are assigned to the factory for production. However, reasonable production scheduling in the production stage plays an important role in improving the efficiency of assembly scheduling problem [2]. Compared with traditional production scheduling, distributed production scheduling not only shortens the production cycle of products, but also reduces the risks faced by enterprises to produce products [3]. Therefore, distributed production scheduling has attracted the attention of researchers in recent years.

In this paper, distributed assembly No-wait flow shop scheduling problem (DANWFSP) is considered. DANWFSP is a combination of NWFSP and assembly issues. DANWFSP considers three sub-problems: job assignment problem, job processing order problem and product assembly order problem. Therefore, the DANWPSP is more complex than the NWFSP. The innovation points of this paper are as follows.

(1) Three constructive heuristics are proposed to generate a competitive initial solution.

(2) A block-shifting method based on the knowledge of optimal block is proposed to ensure the optimal subsequence of a candidate solution is not destroyed.

(3) A factory-based variable neighborhood descent algorithm is proposed to further improve the optimal solution.

The rest of this article is described below. The Section 2 introduces the relevant literature. The Section 3 introduces the definition of DANWFSP, Section 4 introduces the proposed BKBSA algorithm, and Section 5 is the experiment and discussion. Conclusions and future work are presented in Section 6.

2. Literature review

Scheduling is an optimization process, which has been studied by many scholars in recent years. A discrete Distributed Gray Wolf optimizer (DGWO) was used in [4] to provide a solution for scheduling of dependent tasks to virtual machines in cloud computing environments. The discrete DGWO algorithm uses

* Corresponding authors.
E-mail addresses: Fzhao2000@hotmail.com (F. Zhao), 1409776617@qq.com (J. Zhao), wangling@tsinghua.edu.cn (L. Wang), 582971672@qq.com (J. Tang).

https://doi.org/10.1016/j.asoc.2021.107750
1568-4946/

the maximum-order value (LOV) method to transform the continuous candidate solutions generated by DGWO into discrete candidate solutions. The DGWO algorithm was tested and compared with the Particle Swarm Optimization (PSO) and Gray Wolf Optimizer. Experimental results show that the DGWO algorithm can assign tasks to virtual machines faster than other algorithms. A variant of the island-based Cuckoo Search with highly disruptive polynomial mutation iCSPM called Discrete iCSPM with Opposition-based Learning Strategy (DiCSPM) was proposed in [5] for scheduling workflows in cloud computing environments. The effectiveness of the proposed algorithm is verified by experiments, and compared with the optimal resource selection algorithm, PSO algorithm and Gray Wolf Optimizer in two scheduling algorithms under balanced and unbalanced working flows. The overall experimental and statistical results show that DICSPM can solve the workflow scheduling problem in cloud computing environment faster than the other compared algorithms. A hybrid immune algorithm (HIA) was first proposed by Xu and Wang [6] to solve the distributed permutation flow shop scheduling problem (DPFSP) based on maximum completion time C_{max}. In this paper, the author proposes a local search algorithm with four search operators according to the characteristics of the problem. Then a special crossover operator is designed for DPFSP. In addition, crossover operator and immune operator are applied in HIA framework. The effectiveness of the proposed algorithm is verified with the existing heuristic algorithm on 420 small instances and 720 large instances. Compared to the best-known results, new best solutions for 17 small-sized problems and 585 large-sized problems were obtained by HIA. Naderi and Ruiz [7] proposed a scatter search (SS) method to optimize the C_{max} of DPFSP. In SS, the authors use a reference set of complete and partial solutions, as well as techniques such as restarts and local searches. The computational results are accompanied by sound statistical analysis using design of experiments and analysis of variance techniques. Results indicate that the proposed SS outperforms all existing methods by a wide statistical margin, including methods that have been proposed very recently. Ribas [8] first proposed the distributed blocking flow shop scheduling problem (DBFSP), and proposed certain constructive heuristic algorithms to solve the C_{max} of DBFSP. Lin and Ying [9] proposed the distributed No-idle permutation flow shop scheduling (DNIPFSP) for the first time. In this paper, an iterated reference greedy algorithm (IRG) was used to solve the C_{max} of DNIPFSP. The validity of IRG algorithm is verified by comparing with the results of IG algorithm. Lin and Ying [10] proposed the distributed no-wait flow shop scheduling problem (DNWFSP) for the first time and presented the mixed integer programming model (MIP) of DNWFSP. In this paper, the author proposes an iterated cocktail greedy algorithm (ICG) to solve the C_{max} of DNWFSP. ICG includes two self-adjusting mechanisms and a cocktail destruction mechanism. Through statistical analysis of experimental results, the proposed ICG is an effective method to solve DNWFSP. Shao and Pi [11] used the proposed iterated Greedy algorithm (IG) to solve DNWFSP with C_{max} criterion. In this paper, the author first proposes several speed-up methods based on the properties of DNWFSP. Secondly, an improved NEH heuristic algorithm is proposed to generate the initial solution. Finally, four local search methods are proposed according to the four neighborhood structures of factory assignment and job order adjustment. The validity of the proposed algorithm is verified by testing 720 large examples. Chen and Wang [12] proposed a collaborative optimization algorithm (COA) to solve the efficient DNIPFSP for simultaneous optimization of C_{max} and total energy consumption (TEC). In COA, the two heuristic algorithms cooperate to initialize the population. Second, multiple search operations collaborate in a competitive manner to enhance exploration adaptively. Thirdly, different local search strategies are designed for dominated individuals and non-dominated individuals to strengthen the exploitation ability. Fourth, the speed regulation strategy of non-critical operation is designed to improve the TEC. The effectiveness of COA was verified by a large number of experimental results. Wang and Wang [13] proposed a knowledge-based collaborative algorithm (KCA) for energy-efficient scheduling of distributed permutation flow shop (EEDPFSP) with C_{max} and TEC as optimization indexes. A collaborative initialization method and a knowledge-based search operator are proposed. Through a large number of experiments, KCA is an effective method to solve EEDPFSP. Shao and Shao [14] propose two algorithms for distributed hybrid flow shop scheduling problem (DHFSP) based on C_{max} criteria, namely DNEH with smallest-medium rule (DNEH_SMR) and multi-neighborhood iterated greedy algorithm. 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. Shao and Pi [15] proposed a hybrid enhanced Discrete fruit fly optimization algorithm (HEDFOA) to solve DBFSP based on C_{max} criteria. The proposed HEDFOA employed the multi-swarm strategy of fruit fly. In the initialization, an effective constructive heuristic named EHPF2 was developed based on a new assignment rule of jobs and an insertion-based improvement procedure to initialize a common central location for all fruit fly swarms. Then, an effective insertion-based neighborhood operator was developed to implement the smell-based foraging. In the vision-based foraging, a local search was embedded to enhance the exploitation ability of algorithm in local region. Meanwhile, a simulated annealing-like acceptance criterion was used to help algorithm escape from the local optimum. Wang and Gao [16] proposed a multi-objective whale swarm algorithm (MOWSA) to solve EEDPFSP using C_{max} and TEC as optimization indicators. In this paper, the author tries to find a trade-off between makespan and total energy consumption. By comparing with other comparison algorithms in this paper, MOWSA has the most significant effect.

In recent years, the combination of distributed flow shop scheduling problem and assembly problem is increasing. Hatami and Ruize [1] proposed the distributed Assembly permutation flow shop scheduling problem (DAPFSP) by combining PFSP and assembly problem for the first time. In DAPFSP, the optimized indicator is the product assembly completion time $C_{max}(\pi_P)$. In addition, the author proposes six constructive heuristics and two local search algorithms to solve DAPFSP. Zhang and Lin [17] proposed a hybrid biogeography-based optimization algorithm (HBBO) to solve DAPFSP with $C_{max}(\pi_P)$ as the optimization indicator. In HBBO, the author first adopts the path relinking heuristic algorithm as the local search strategy of the product in the migration phase. Second, the insertion-based heuristic is used to determine the job sequence for each product during the mutation phase. Finally, a new local search method is designed according to the characteristics of the problem. The effectiveness of HBBO is verified by statistical analysis of 910 small instances and 810 large instances. Wang and Wang [18] proposed a memetic algorithm (MA) based on estimation of distribution algorithm (EDA) to solve DAPFSP with $C_{max}(\pi_P)$ as the optimization indicator. In MA, the author proposes a local search strategy based on critical path. The validity of MA is verified by experimental results on 1710 examples. Sang and Pan [2] proposed three discrete invasive weed optimization algorithms (DIWO) to solve DAPFSP with total flow time as the optimization indicator. In DIWO, the author introduces a

local search method based on product and job. The effectiveness of the proposed algorithm is verified by comparing with other algorithms. Pan and Gao [19] proposed a novel DAPFSP. In this paper, each factory contains a flow shop for job processing and an assembly line for product assembly. The optimization indicator of this paper is $C_{max}(\pi_P)$. In addition, three constructive heuristic algorithms and two variable neighborhood search (VNS) algorithms combined with IG algorithm are proposed to solve DAPFSP. Finally, the validity of the proposed algorithm is verified by testing on 810 examples. Ferone and Hatami [20] proposed a biased-randomized iterated local search algorithm (BR-ILS) to solve DAPFSP. Shao and Pi [21] first proposed the Distributed assembly no-idle flow shop scheduling problem (DANIFSP) and used ILS and VNS to solve this problem. In this paper, the author proposes a new job assignment rule and three constructional heuristic algorithms. Experimental results show that the proposed algorithm is superior to other advanced algorithms. Shao and Shao [22] proposed the Distributed Assembly Blocking flow shop scheduling problem (DABFSP). In this paper, the author proposes a constructive heuristic algorithm based on job assignment rules and product-based insertion process. Then, four perturbation operators and a variable neighborhood search based on critical job are proposed. Finally, the effectiveness of the proposed algorithm is verified in 810 examples.

Heuristic method is a kind of effective method to solve distributed flow shop scheduling problem. Backtracking search algorithm (BSA) is a new heuristic algorithm, which was first proposed by Civicioglu [23]. BSA focuses on the historical experience to guide the evolution of the population, so that it is easier to form a better evolutionary trend in the evolutionary process. In addition, historical information is the necessary information to form feedback, and plays an important role in the evolutionary algorithms based on knowledge learning. Therefore, BSA has been widely used in various engineering optimization problems in recent years. Such as economic emission dispatch problem [24], fuzzy multiproduct multistage scheduling problem [25], soil parameter identification problems [26], joint replenishment problem [27], the echo state network time series prediction optimization problem [28], casting heat treatment charge plan problem [29], chaos system parameter identification problem [30], flood forecasting problem [31], photovoltaic model parameters estimation problem [32], color images with multilevel threshold problem [33], energy-efficient permutation flow shop scheduling problem [34], distributed assembly permutation flow shop scheduling problem [35].

According to the above literature review, distributed production scheduling with assembly process has been the focus of research in recent years due to its practicality. Researchers have considered different assembly scheduling problems, including distributed assembly permutation flow-shop scheduling problem, distributed assembly no-idle flow-shop scheduling problem, distributed assembly blocking flow-shop scheduling problem. Lots of efficient heuristics and meta-heuristics have been designed to solve these scheduling problems. However, few researches focus on distributed assembly no-wait flow-shop scheduling problem. Design methods to solve such problems according to the characteristics of the problems are also rarely proposed. This paper presents a BSA based on the knowledge of the optimal block (BKBSA) to solve DANWFSP with $C_{max}(\pi_P)$ as the optimization indicator.

3. Distributed assembly No-wait flow shop scheduling problem

3.1. Notation definition

The definitions of the variables and parameters mentioned in this paper are recorded in Table 1.

Table 1
Notation.

Notation	Definition
n	Number of jobs
m	Number of machines in each production factory
F	Number of production factories
s	Number of products
F_A	The assembly factory
M_A	The assembly machine
i	The index of the job, $i = 0, 1, 2, \ldots, n.0$ is a virtual job.
j	The index of the machine, $j = 1, 2, \ldots, m$.
h	The index of the product, $h = 0, 1, 2, \ldots, n.0$ is a virtual product.
f	The index of the production factory, $f = 1, 2, \ldots, F$.
$O_{i,j}$	Process of job i on machine j
$t_{i,j}$	Processing time of job i on machine j
t_h	Assembly time of product h
N_h	Number of jobs composing product h
f_n	Number of jobs allocated to factory f
π^f	The sequence of jobs assigned to factory f, $\pi^f = \left\{\pi^f(1), \pi^f(2), \ldots, \pi^f(f_n)\right\}$.
π_J	A feasible scheduling job sequence, $\pi_J = \{\pi_1, \pi_2, \ldots, \pi_n\}$
π_P	A feasible scheduling product sequence, $\pi_P = \{\pi_1, \pi_2, \ldots, \pi_s\}$
$S_{i,j}$	Start processing time of job i on machine j
$C_{i,j}$	The completion time of job i on machine j
$S^f_{i,j}$	Start processing time of job i on machine j in factory f
$C^f_{i,j}$	The completion time of job i on machine j in factory f
R_h	Release time of product h in the production factory
S_h	Start assembly time of product h in assembly factory
C_h	Assembly completion time of product h in assembly factory
k	The index of a certain job in the job sequence
l	The index of a certain product in the product sequence
Max	A sufficiently large positive number
$X_{k,i}$	Binary variables. If job k is the direct predecessor of job i, the value of the binary variable is 1, otherwise the value of the binary variable is 0
$P_{l,h}$	Binary variables. If product l is the direct predecessor of product h, the value of the binary variable is 1, otherwise the value of the binary variable is 0
$G_{i,h}$	Binary variables. If job i belongs to product h, the value of the binary variable is 1, otherwise the value of the binary variable is 0
$X^f_{i,k}$	Binary variables. If job i is at position K in factory f, the value of the binary variable is 1, otherwise the value of the binary variable is 0
$C_{max}(\pi_P)$	The maximum assembly completion time of the product sequence in the assembly factory
$\{J_1, J_2, \ldots, J_n\}$	Job sequence
$\{M_1, M_2, \ldots, M_m\}$	Machine sequence
$\{F_1, F_2, \ldots, F_F\}$	Factory sequence
$\{P_1, P_2, \ldots, P_s\}$	Product sequence

3.2. Definition of optimization

Optimization problem is to find a set of parameter values to maximize or minimize some performance indexes of the system under certain constraint conditions. The application of optimization problem is widespread in industry, society, economy, management and other fields. Optimization problems can be divided

into many types according to the properties of objective function, constraint function and value of optimization variable. Each type of optimization problem has its specific solution method according to its different properties.

Without loss of generality, let the optimization problem considered be:

$$min\rho = f(X) \tag{1}$$

$$s.t.X \in S = \{X|g_i(X) \leq 0 \quad i = 1, 2, \ldots, m\} \tag{2}$$

where, $\rho = f(X)$ is the objective function. $g_i(X)$ is the constraint function. S is the constraint domain. X is the n-dimensional optimization variable. Usually, the maximization problem is easily translated into the minimization problem $\rho = -f(X)$. When, $f(X)$ and $g_i(X)$ are linear function, and $X \geq 0$, the above optimization problem is linear programming problem. When at least one of the functions in $f(X)$ and $g_i(X)$ is nonlinear, the above problem is nonlinear programming problem. When the optimization variable X is only an integer value, the above problem is an integer programming problem. In particular, when X can only be 0 or 1, the above problem is a 0–1 integer programming problem. Since integer programming problem belongs to combinatorial optimization category, its computation amount increases exponentially with the increase of variable dimension, so there exists "dimension disaster" problem. When the constraint space restricted by $g_i(X) \leq 0$ $(i = 1, 2, \ldots, m)$ is the whole n-dimensional Euclidean space, the above optimization problem is unconstrained optimization problem.

3.3. Definition of DANWFSP

Distributed assembly No-wait flow shop scheduling problem (DANWFSP) is the process of assembling the job sequence produced in the distributed No-wait flow shop into products in the assembly shop. DANWFSP is divided into two stages as shown in Fig. 1. The production stage is to produce all the Jobs that make up the product in the production factory. The description of the production stage is as follows. There are n jobs $\{J_1, J_2, \ldots, J_n\}$ to be allocated to F factory $\{F_1, F_2, \ldots, F_F\}$ for processing. Each factory is a No-wait flow shop containing m identical machines $\{M_1, M_2, \ldots, M_m\}$. The processing path of all jobs is the same. In other words, the jobs assigned to each factory are processed sequentially on m machines. In a certain period of time, a process O_{ij} of a job is processed on only one machine. Each machine only processes one job at a time. In order to satisfy the No-waiting condition, there is no waiting time between successive processes of the same job. In order to satisfy the distributed conditions, once a job is assigned to a specific factory f, all processes belonging to the job are only performed in factory f. In other words, the job J is not allowed to be transferred in the factory during the production process. In addition, the following assumptions must be met. All jobs are independent of each other. Each job is able to start processing at time 0. The setup time of the machine and the transmission time of the job are ignored. There is an infinite buffer between the two stages.

The assembly stage is to assemble n jobs generated in the production stage into s products in the assembly factory F_A. There is only one machine M_A in the assembly factory F_A. Each product $P = \{J_1, J_2, \ldots, J_{N_h}\}$ consists of N_h jobs and satisfy $\sum_{h=1}^{s} N_h = n$. The product is released from the production factory only when all the jobs that make up the product have been produced. The latter product is only assembled after the current product is assembled. In summary, the task of DANWFSP is to assign jobs to production factories based on certain allocation criteria and determine the processing sequence of jobs on all machines, and determine the optimal processing sequence of the products in the assembly factory to make a certain index optimal. DANWFSP is used in steelmaking, food processing, chemical and pharmaceutical manufacturing processes. In the process of steel-making and continuous casting in iron and steel production enterprises, no waiting time of molten steel is expected in order to realize the production mode of hot feeding and hot charging and reduce the temperature drop of molten steel in the air [10,11].

3.4. Mixed integer linear programming model

The mixed integer linear programming (MILP) model utilizes mathematical methods to model the requested problem to simplify the difficulty of the requested problem. The mixed integer linear programming model has been widely used in flow shop scheduling problems in recent years [21]. In this paper, the goal of optimization is to minimize the maximum assembly completion time of the product in the assembly factory. The MILP model of DANWFSP is described as follows.

Objective:

$$minC_{max}(\pi_P) \tag{3}$$

Subjective to:

$$\sum_{k=0,k\neq i}^{n} X_{k,i} = 1 \quad \forall i \tag{4}$$

$$\sum_{i=0,k\neq i}^{n} X_{k,i} \leq 1 \quad \forall k \tag{5}$$

$$X_{k,i} + X_{i,k} \leq 1 \quad \forall i \in \{1, 2, \ldots, n-1\}, \quad i > k \tag{6}$$

$$\sum_{k=1}^{n}\sum_{f=1}^{F} X_{i,k}^{f} = 1, \quad \forall i \tag{7}$$

$$C_{k,j}^{f} = C_{k,j-1}^{f} + \sum_{i=1}^{n} X_{i,k}^{f} \cdot t_{i,j}, \quad \forall k, j, f \tag{8}$$

$$C_{k,j}^{f} \geq C_{k,j-1}^{f} + \sum_{i=1}^{n} X_{i,k}^{f} \cdot t_{i,j}, \quad \forall k, j, f \tag{9}$$

$$\sum_{l=0,l\neq h}^{s} P_{l,h} = 1 \quad \forall h \tag{10}$$

$$\sum_{h=1,l\neq h}^{s} P_{l,h} \leq 1 \quad \forall l \tag{11}$$

$$P_{l,h} + P_{h,l} \leq 1 \quad \forall l \in \{1, 2, \ldots, s-1\}, \quad h > l \tag{12}$$

$$C_h \geq \left(C_{i,m} \cdot G_{i,h}\right) + t_h \tag{13}$$

$$C_h \geq C_l + t_h + \left(P_{l,h} - 1\right) \cdot Max \quad \forall l, s \tag{14}$$

$$C_{max}(\pi_P) \geq C_h \tag{15}$$

$$C_{i,j} \geq 0 \quad \forall i, j \tag{16}$$

$$S_{i,j} \geq 0 \quad \forall i, j \tag{17}$$

$$C_{i,j}^{f} \geq 0 \quad \forall i, j, f \tag{18}$$

$$S_{i,j}^{f} \geq 0 \quad \forall i, j, f \tag{19}$$

$$C_h \geq 0 \quad \forall h \tag{20}$$

$$S_h \geq 0 \quad \forall h \tag{21}$$

$$X_{i,k}^{f} \in \{0, 1\} \quad \forall i, k, f \tag{22}$$

$$X_{k,i} \in \{0, 1\} \quad \forall i, k \tag{23}$$

$$P_{l,h} \in \{0, 1\} \quad \forall l, h \tag{24}$$

$$G_{i,h} \in \{0, 1\} \quad \forall i, h \tag{25}$$

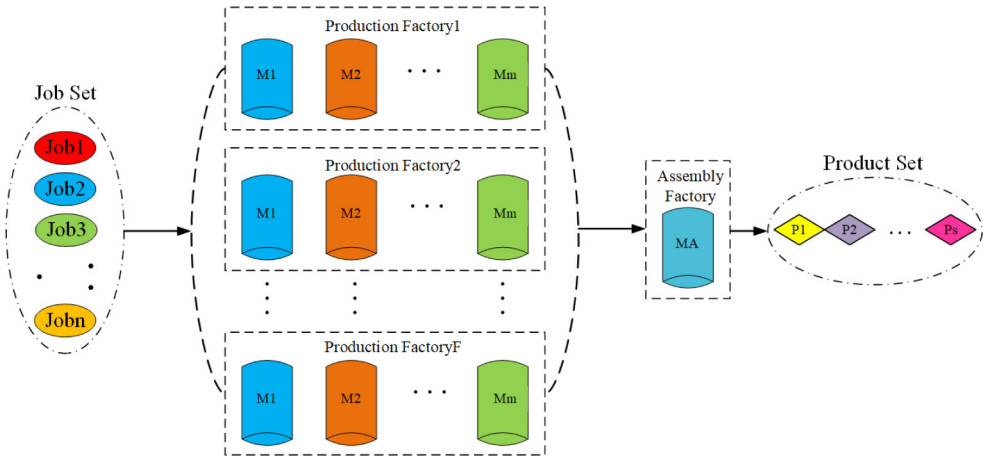


Fig. 1. Distributed assembly No-wait flow shop scheduling problem.

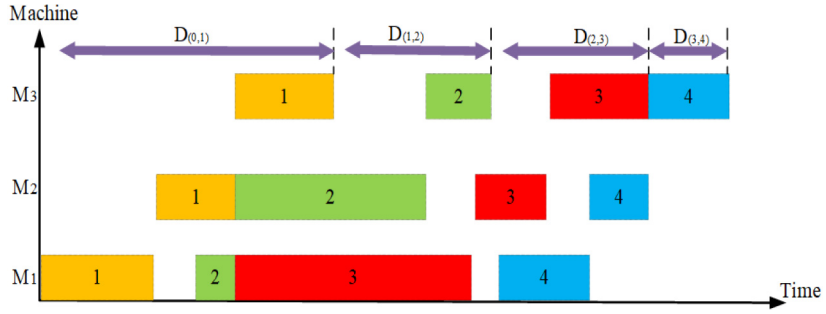


Fig. 2. Difference in completion time between adjacent jobs.

Constraint (3) is the optimization index in this paper. Constraint (4) guarantees that starting from the second job, every job must have one and only one direct predecessor. Constraint (5) ensures that each job has at most one immediate successor. Constraint (6) ensures that in a feasible scheduling job sequence, each job only appears in one position. Constraint (7) ensures that a job is only assigned to one factory and only appears in one position in the feasible scheduling job sequence of this factory. Constraint (8) ensures that the different processes of the same job are continuous, which means that the constraint of no waiting is satisfied. Constraint (9) ensures that the same job only starts the next process when the previous process is completed. Constraint (10) guarantees that starting from the second product, every product must have one and only one direct predecessor. Constraint (11) ensures that each product has at most one immediate successor. Constraint (12) ensures that in a feasible scheduling product sequence, each product only appears in one position. Constraint (13) ensures that the product is assembled in the assembly factory only when all the jobs that make up the product have been processed. Constraint (14) ensures that the next product is only assembled when the previous product has been assembled on the assembly machine. Constraint (15) is the maximum assembly completion time of the product. Constraints (16)–(19) ensure that the start time and completion time of each job are non-negative. Constraints (20)–(21) ensure that the start time and completion time of each product are non-negative. Constraints (22)–(25) define certain binary variables in this paper.

3.5. The calculation method of makespan of DANWFSP

The calculation process of maximum assembly completion time $C_{max}(\pi_P)$ of DANWFSP is completed in two steps. The first step is to calculate the completion time of the last job belonging to each product in the production factory. This time is the release time of each product. The second step is to utilize the release time and assembly time of each product to calculate the start assembly time and assembly completion time of each product in the assembly factory. In the assembly factory, the time for the last product to be completed on the assembly machine is the maximum assembly completion time $C_{max}(\pi_P)$ of DANWFSP. The optimization target of this paper is to minimize the value of $C_{max}(\pi_P)$. Assume that $\pi^f = \{\pi^f(1), \pi^f(2), \ldots, \pi^f(f_n)\}$ is the sequence of jobs assigned to production factory f during the production phase. f_n is the number of jobs allocated to factory f and satisfies $\sum_{f=1}^{F} f_n = n$. The completion time of job i on machine j is denoted by $C_{i,j}$. The first step is to calculate the completion time $C_{i,m}$ of the job in each production factory on the last machine. In production factory the operation of the same job on two adjacent machines must satisfy the no wait constraint. In other words, different operations of the same job are continuous. Therefore, the different operations of the same job are regarded as a whole, and the completion time of each job is calculated by the incremental solution method of the completion time difference $D_{(i,i+1)}$ of adjacent jobs [36]. Fig. 2 is a detailed explanation of $D_{(i,i+1)}$. $D_{(i,i+1)}$ is only related to the order of the two jobs, and has nothing to do with the positions of the two jobs in the job sequence. According to the description in [37], the calculation method of $D_{(i,i+1)}$ is as follows.

$$D_{(i,i+1)} = \begin{cases} \max\limits_{k=1,2,\ldots,m} \{\sum\limits_{h=k}^{m}(t_{i+1,h} - t_{i,h}) + t_{i,k}\}, & i = 1, 2, \ldots, n-1 \\ \sum\limits_{j=1}^{m} t_{\pi(1),j}, & i = 0 \end{cases} \tag{26}$$

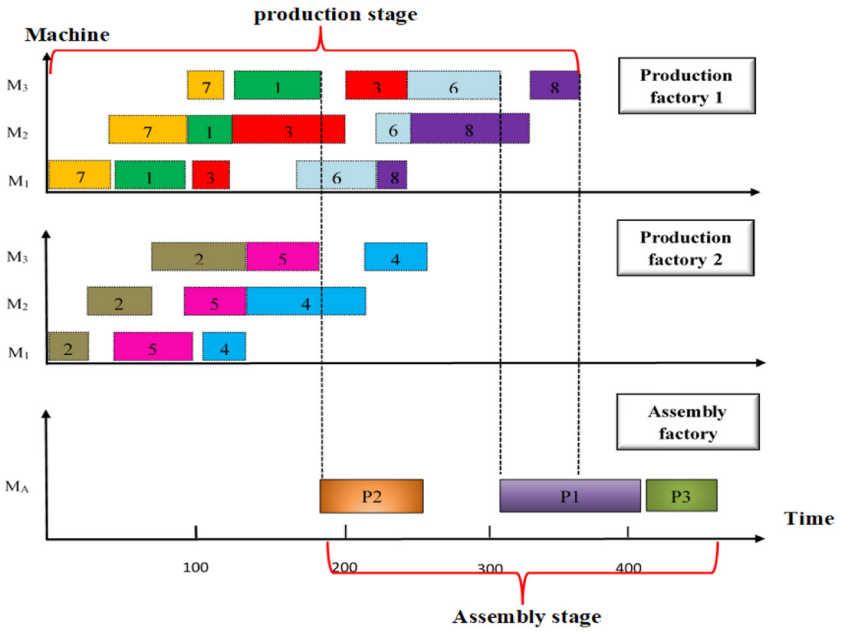


Fig. 3. Gantt chart of an example of calculating product assembly time.

Table 2
Processing time of jobs and products.

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
$C_{i,m}$	180	140	240	260	180	310	120	370
P_h	Product1 {2,5,6}			Product2 {1,7}		Product3 {3,4,8}		
t_h	100			75		50		

where, i is the index of the job, j is the index of the machine, and $\pi(1)$ is the first job in a job sequence. Therefore, the completion time of the job in each factory is as follows.

$$C^f_{i,m} = \sum_{i=\pi^f(1)}^{\pi^f(j_n)} D_{(i-1,i)} \tag{27}$$

Then, calculate the release time R_h of each product in the production factory. Assume that product h is composed of a sequence of jobs $\{J_1, J_2, \ldots, J_{N_h}\}$, that is, $P_h = \{J_1, J_2, \ldots, J_{N_h}\}$. Then the release time of product h is

$$R_h = \max\left(C_{i,m}\right) i \in \{J_1, J_2, \ldots, J_{N_h}\} \tag{28}$$

In other words, the release time of product h is the completion time of the last processed job among the jobs that make up product h. Suppose $\pi_P = \{\pi_1, \pi_2, \ldots, \pi_s\}$ is the product sequence to be assembled in the assembly factory. S_h is the start assembly time of product h, and C_h is the assembly completion time of product h. Then

$$S_{\pi_1} = R_1 \tag{29}$$

$$C_{\pi_1} = S_{\pi_1} + t_{\pi_1} \tag{30}$$

$$S_{\pi_h} = \max\left\{C_{\pi_{h-1}}, R_h\right\} h = 2, 3, \ldots, s \tag{31}$$

$$C_{\pi_h} = S_{\pi_h} + t_{\pi_h} h = 2, 3, \ldots, s \tag{32}$$

$$C_{max}(\pi_P) = C_{\pi_s} \tag{33}$$

The target of DANWFSP is to find an optimal product sequence π^*_P that minimizes $C_{max}(\pi_P)$. In order to explain the definition of DANWFSP more clearly, an example with right jobs, three machines, two production factories, and three products is utilized to illustrate the calculation process of $C_{max}(\pi_P)$. Fig. 3 shows this example in a Gantt chart. Table 2 respectively records the completion time of eight jobs, the assembly time of three products, and the sequence of jobs that make up each product.

From Table 2, product P_1 is composed of jobs J_2, J_5 and J_6. However, J_6 is the last of the three jobs to be completed. Therefore, the release time of product P_1 is $R_1 = C_{5,m} = 310$. Similarly, the release time of product P_2 and P_3 is $R_2 = C_{1,m} = 180$ and $R_3 = C_{8,m} = 370$, respectively. Then, a feasible scheduling product sequence in the assembly factory is obtained as $\pi_P = \{P_2, P_1, P_3\}$. Since P_2 is the first of the three products to be released, P_2 is first assembled in the assembly factory. The start assembly time of P_2 is $S_2 = R_2 = 180$. The assembly completion time of P_2 is $C_2 = S_2 + t_2 = 180 + 75 = 255$. Since product P_1 has not been released when product P_2 is assembled, the start assembly time of product P_1 is $S_1 = \max\{C_2, R_1\} = 310$. The assembly completion time of P_1 is $C_1 = S_1 + t_1 = 310 + 100 = 410$. Similarly, the start assembly time and assembly completion time of product P_3 are $S_3 = 410$ and $C_3 = 460$, respectively. Finally, the maximum assembly completion time of product sequence π_P is obtained as $C_{max}(\pi_P) = 460$.

4. The proposed backtracking search algorithm

4.1. Backtracking search algorithm

Backtracking search algorithm (BSA) is an evolutionary algorithm based on double population iteration. BSA utilizes both the current population and the historical population to generate candidate solutions. In the iterative process of the algorithm, the historical population is used not only to guide the evolution of the current population, but also to maintain the diversity of the current population. More details about BSA can be found in [23].

4.2. Solution representation

When meta-heuristic algorithms are utilized to solve optimization problems, the encoding and decoding of candidate solutions is an important issue. The original BSA was originally utilized to solve continuous optimization problems. Therefore, the real number encoding mechanism in the original BSA is

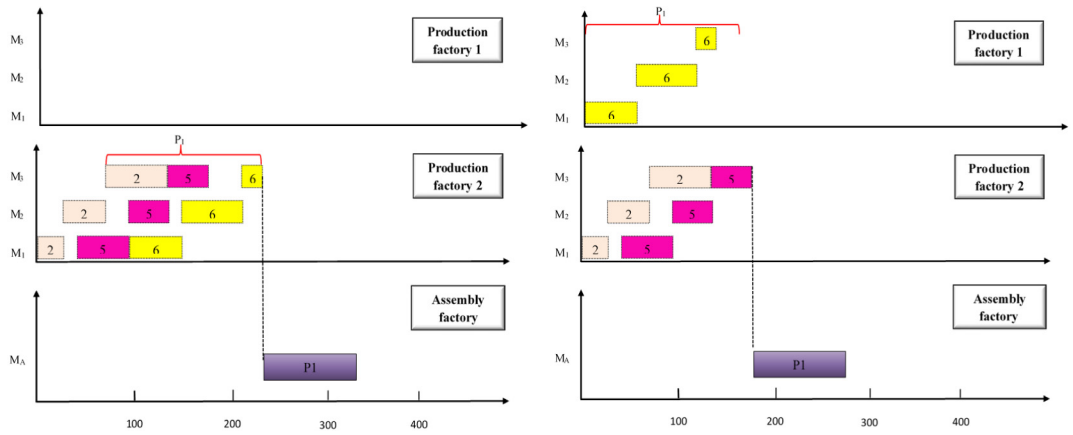


Fig. 4. An instance where the number of products is less than the number of factories.

not suitable for the DANWFSP problem. In this paper, a two-level coding method is utilized to express candidate solutions. Let $\pi_P = \{P_1, P_2, \ldots, P_s\}$ be a product sequence. Each product $P_h = (J_1, J_2, \ldots, J_{N_h})$ is composed of several jobs. The sequence of jobs assigned to each production factory is denoted by $\pi^f = \left\{\pi^f(1), \pi^f(2), \ldots, \pi^f(f_n)\right\}$. For example, $\pi^1 = \{4, 5, 1, 2\}$ indicates that the jobs assigned to factory 1 are J_4, J_5, J_1, and J_2 respectively. $\pi_P = \{3, 2, 1\}$ indicates that a processing sequence of products in the assembly factory is P_3, P_2, and P_1. $P_3 = (4, 6, 7)$ indicates that product 3 is composed of jobs J_4, J_6, and J_7. The mutation operator and crossover operator in the original BSA are not applicable to DANWFSP. Therefore, A block-shifting operator based on optimal job block knowledge is proposed to generate candidate solutions. However, historical populations are preserved because the historical information is essential to the BSA.

4.3. Constructive heuristics

Certain studies have shown that heuristic methods are easier to produce high-quality initial solutions [20]. Therefore, the constructive heuristics is usually utilized to construct the initial solution of the algorithm. The constructive heuristics in the scheduling problem analyzes the characteristics of the problem to be solved and utilized certain rules to construct an approximate optimal solution in a short time. There are three main tasks of DANWFSP, which are the job assignment problem of each production factory, the job sequencing problem of each production factory, and the product sequencing problem of the assembly factory. This paper proposes three constructive heuristics to solve the problem of job allocation in each production factory. The three constructive heuristics respectively utilize the shortest processing time principle (SPT) [38], Framinan and Leisten (FL) heuristic [39], and NEH heuristic [40] to construct a complete job sequence. Then, the principle of minimizing product assembly time (NR_a) is utilized to assign this job sequence to each production factory. Next, the assignment principle of jobs and three constructive heuristics are introduced respectively.

4.3.1. Job assignment rule

The problem of job allocation first appeared in the distributed permutation flow shop scheduling problem (DPFSP) [21]. In DPFSP, two job assignment principles are proposed. The first is to assign job J to the factory with the lowest makespan that does not include this job (NR_1). The second is to assign job J to the factory with the lowest makespan that includes this job (NR_2). These two job assignment principles consider the minimum processing time of the job and cannot effectively solve the problem of minimum product assembly time. However, due to product constraints, DANWFSP generally considers two job assignment principles. One is to assign each job that makes up a product to different factories. The other is to assign each job that makes up a product to the same factory. However, the job sequence that composes this product is not necessarily the optimal sequence. Exchanging jobs of different products or assigning jobs of the same product to different factories may make the required indicators better. Fig. 4 is an example of assigning jobs to each production factory using two job assignment principles when the number of products is less than the number of factories.

From Fig. 4 that when job J_6 is assigned to factory f_1, the release time of product P_1 is advanced. Fig. 5 is an example of assigning jobs to each production factory using two job assignment principles when the number of products is greater than the number of factories. It can be seen from Fig. 5 that when job J_6 is assigned to factory f_1 and jobs J_3 and J_4 are assigned to factory f_2, the assembly time for products P_1, P_2, and P_3 is smaller. The three products are also more compact.

To sum up, it is more conducive to the assembly time of the products that the jobs that make up the same product are assigned to different factories, while also ensuring the compactness of each product [18]. This paper adopts the assignment principle (*NRa*) proposed in [21]. This assignment principle assigns job J to the factory where job J is inserted to make the product assembly time smaller, which is more conducive to the optimization index of this paper. Fig. 6 is an explanation of the assignment principle proposed in [21]. In this example, product P_2 is composed of jobs J_4 and J_5. When job J_4 is assigned to two production factories, job 4 needs to be inserted into any possible position of the two production factories. Then, determine the best position for job J_4. The pseudocode for *NRa* is explained in Algorithm 1. More details about *NRa* are explained in [21].

4.3.2. Heuristic 1

The assembly completion time of the product is determined by the processing time of the jobs constituting the product and the assembly time of the product. Heuristic 1 (Heu1_NRa) applies the SPT rule to the sum τ_h of the total processing time P^h and the assembly time t of product t_h to construct a product sequence π_P. After that, the FL heuristic is utilized to optimize the job sequence P_h that composes the product h. Then, the optimized job sequence P_h of each product is decoded into the corresponding complete job sequence π_J according to the previously constructed product sequence π_P. Finally, utilize the job assignment principle *NRa* to assign all jobs in π_J to each production factory. The purpose of Heu1_NRa is to utilize a simple rule to construct a product

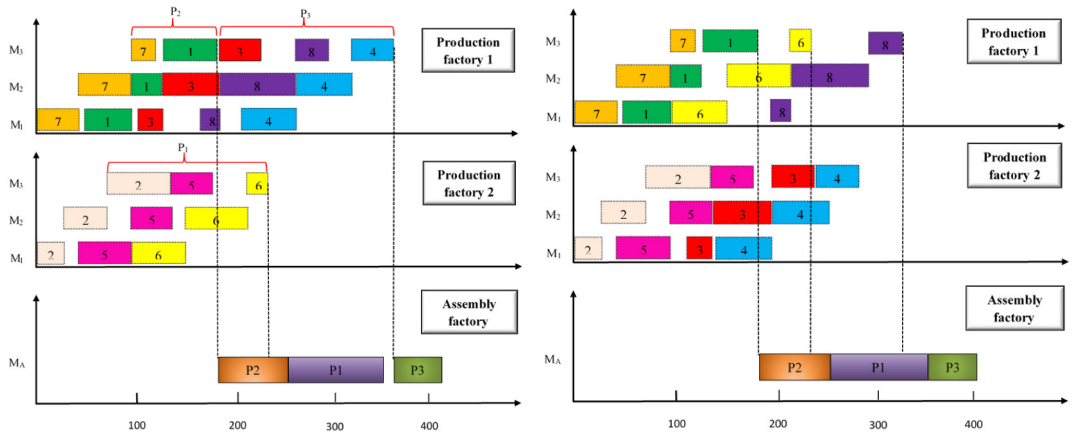


Fig. 5. An instance where the number of products is greater than that of factories.

Algorithm 1:$NR_a(\pi_J)$

1.**for** $i = 1$ **to** F
2. $\pi^i\{i\} = \pi_J(i)$
3.**end for**
4.**for** $i = F + 1$ **to** n
5. $Min = Max, ins = \pi_J(i)$
6. **for** $j = 1$ **to** F
7. Calculate the jobs in π^i and the total number f_n of jobs in π^i.
8. **for** $k = 1$ **to** $f_n + 1$
9. The job ins is inserted into $f_n + 1$ possible positions in factory j, and the product assembly completion time $C_{max}(\pi)$ is calculated according to π.
10. **if** $C_{max}(\pi) < Min$ **then**
11. $Min = C_{max}(\pi), best_{fac} = j, best_{pos} = k$
12. **end if**
13. **end for**
14. **end for**
15. ins is repeated the above process until ins is empty.
16.**end for**
17.**return** π

Table 3
Processing time of jobs and products.

Job	J_1	J_2	J_3	J_4	J_5
M_1	5	10	7	7	4
M_2	5	5	7	5	8
P_h	P_1 (J_1, J_3, J_5)			P_2 (J_2, J_4)	
t_h	5			7	

sequence to reduce the time spent in constructing a product sequence. The pseudocode for *Heu*1_*NRa* is explained in Algorithm 2. The steps of Heu1_NRa are as follows.

Step1: The SPT rule is applied to the sum τ_h of P^h and t_h to construct an initial product sequence π_P.

Step2: The FL heuristic is utilized to optimize the job sequence P_h that composes each product h.

Step3: The optimized P_h is Integrated into a complete job sequence π_J according to the order of the products in π_P constructed in step 1.

Step4: The *NRa* rule is utilized to assign all jobs in π_J to each production factory.

To better explain the principle of Heu1_NRa, an example with 5 jobs, 2 production factories, 2 products, and 2 machines in each production factory is utilized to explain the steps of Heu1_NRa. The processing time of each job on the two machines, the assembly time of each product, and the jobs that make up each product are all recorded in Table 3.

Step1: Calculate the sum τ_1 of the total processing time P^1 of all the jobs composing P_1 and the assembly time t_1 of P_1 is $\tau_1 = P^1 + t_1 = 10 + 14 + 12 + 5 = 41$. Similarly, calculate $\tau_2 = 34$. Then, the SPT rule is utilized to get an initial product sequence as $\pi_P = \{P_2, P_1\}$.

Step2: The FL heuristic is utilized to optimize the sequence of jobs that compose P_1 and P_2. The optimized sequence P_1 is assumed as $P_1 = (5, 3, 1)$, and the maximum completion time of this sequence is $C_{max}(P_1) = 24$. Similarly. The sequence of P_2 after optimization is $P_2 = (4, 2)$, $C_{max}(P_2) = 22$.

Step3: P_1 and P_2 are integrated into a complete job sequence $\pi_J = \{4, 2, 5, 3, 1\}$ according to the order of the products in π_P constructed in step 1.

Step4: All the jobs in π_J are assigned to each production factory according to the *NRa* rule. The jobs J_4, J_2, and J_5 have been assigned to two factories according to the *NRa* rule. Next, the job J_3 is assigned to the production factory. J_3 is a job that composes P_1.

Algorithm 2: Heu1_NRa

```
for i = 1 to s
    Calculate the total processing time τ_i of product i, and construct an initial product
    sequence π_P according to the SPT criterion.
    The FL heuristic is utilized to improve the job sequence P_i that composes product i.
end for
P_i is integrated into π_j according to the order of products in π_P.
for i = 1 to n
    for j = 1 to F
        The job i in π_j is assigned to the production factory j according to the NR_a rule,
        and the job sequence π^j in the factory j is obtained.
    end for
end for
return π
```

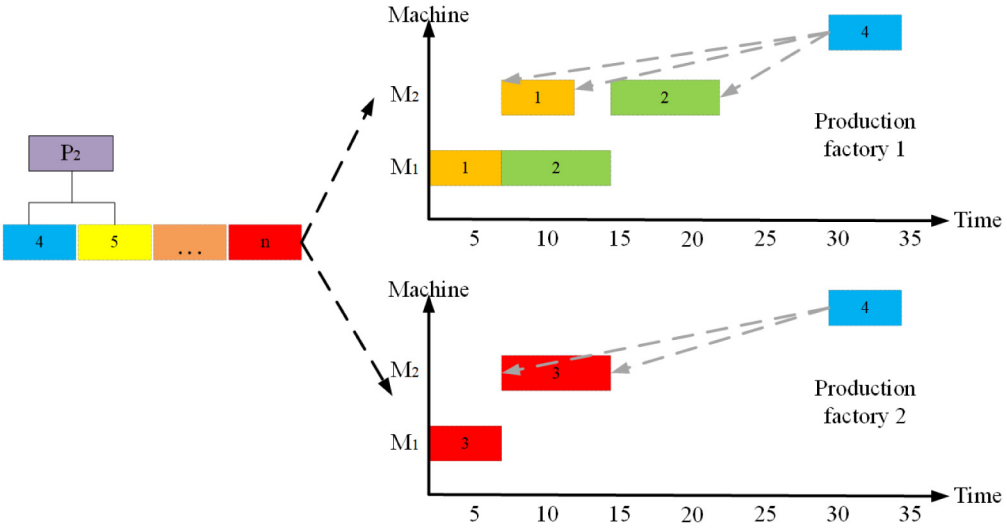


Fig. 6. Diagram of Assignment Principle *NRa*.

Now consider J_3 as the last completed job of P_1, and J_3 is inserted into any possible position of f_1 and f_2. Then, find the position that minimizes the time required to complete the assembly of P_2 and P_1 as the best insertion position for J_3. In Fig. 7, J_3 is inserted into five different positions of f_1 and f_2. In (a), the product sequence is $\pi_P = \{P_1, P_2\}$, and the assembly completion time of π_P is $C_{max}(\pi_P) = 26$. In (b), the product sequence is $\pi_P = \{P_2, P_1\}$, and the assembly completion time of π_P is $C_{max}(\pi_P) = 31$. In (c), the product sequence is $\pi_P = \{P_1, P_2\}$, and the assembly completion time of π_P is $C_{max}(\pi_P) = 36$. In (d), the product sequence is $\pi_P = \{P_1, P_2\}$, and the assembly completion time of π_P is $C_{max}(\pi_P) = 34$. In (e), the product sequence is $\pi_P = \{P_2, P_1\}$, and the assembly completion time of π_P is $C_{max}(\pi_P) = 33$. Therefore, J_3 is inserted into the first position of factory f_1.

Step5: The job J_1 is regarded as the last completed job of the product P_1. Repeat step 4 to find the best insertion position of J_1. This step continues until all the jobs in π_J have been allocated.

4.3.3. Heuristic 2

When Heu1_NRa constructs the initial product sequence π_P, it simply utilizes the total processing time P^h of all the jobs that make up the product h. The order of the jobs that make up product h is not taken into consideration. It is possible to obtain a better π_P by first optimizing the job sequence that composes each product, and then constructing the initial product sequence π_P. Heuristic 2 (Heu2_NRa) first arrange the jobs that make up each product into a job sequence P_h in descending order of the total processing time P^{J_i} of each job. Then, calculate the release time R_h for each product. Next, R_h and the assembly time t_h of each product are utilized to calculate the earliest assembly completion time C'_h of each product. The C'_h of each product is utilized to construct an initial product sequence π_P in descending order. Finally, π_P is decoded as π_J, and all the jobs in π_J are assigned to each production factory according to the *NRa* rule. The pseudocode for *Heu2_NRa* is explained in Algorithm 3. The steps of Heu2_NRa are as follows.

Step1: The jobs composing each product are arranged in descending order into a job sequence P_h of according to the P^{J_i}.

Step2: The R_h and t_h of each product are utilized to calculate the earliest assembly completion time of this product $C'_h = R_h + t_h$.

Step3: The C'_h of each product is utilized to construct an initial product sequence π_P in descending order.

Step4: π_P is decoded into π_J, and all jobs in π_J are assigned to various production factories in accordance with the *NRa* rule.

The examples in Table 3 are utilized to explain the above steps of heuristic 2.

Step1: The total processing time of J_1, J_2, J_3, J_4, J_5 are $P^{J_1} = 10$, $P^{J_2} = 15$, $P^{J_3} = 14$, $P^{J_4} = 12$, $P^{J_5} = 12$ respectively. Therefore, the

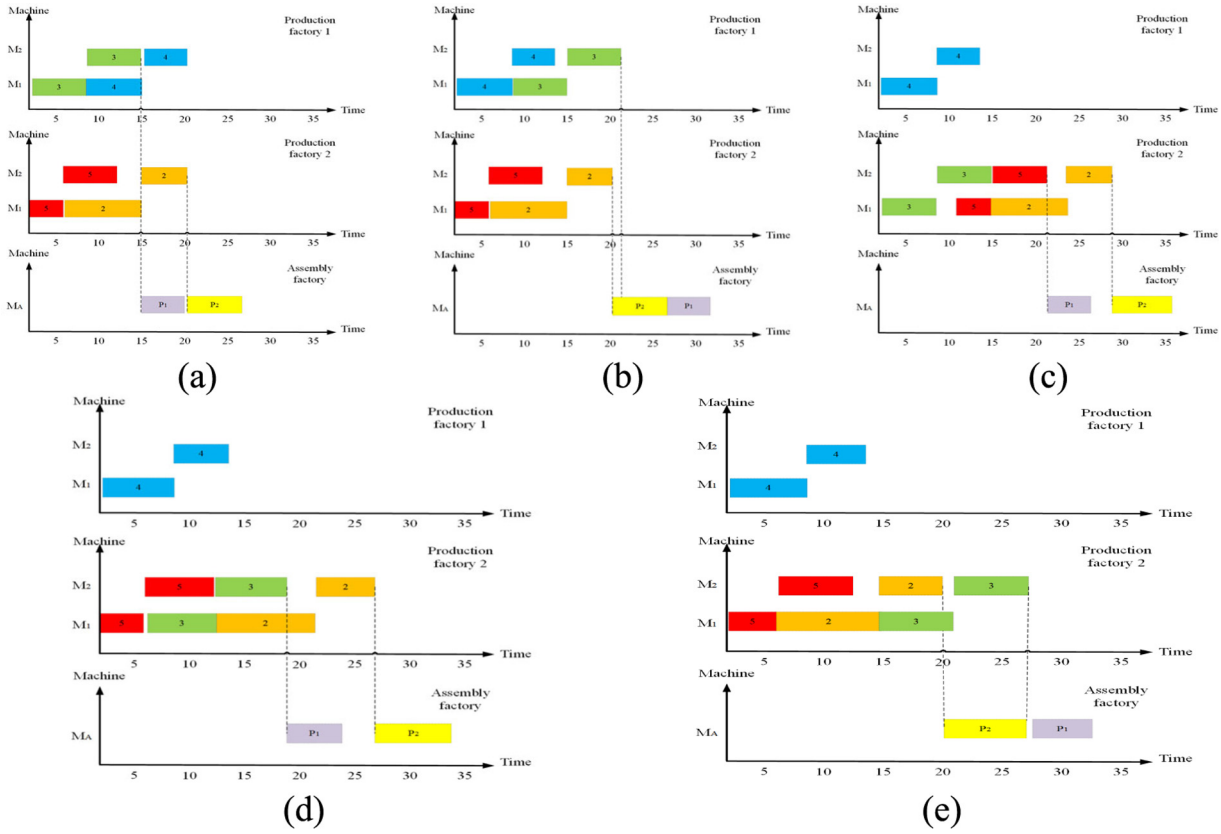


Fig. 7. The Gantt chart of the process that job J_3 is allocated.

job sequence for constructing P_1 is $P_1 = (3, 5, 1)$, and the release time of P_1 is $R_1 = 27$. The job sequence of P_2 is $P_2 = (2, 4)$, and the release time of P_2 is $R_2 = 22$.

Step2: The earliest assembly completion time of P_1 is calculated as $C'_1 = R_1 + t_1 = 27 + 5 = 32$. The earliest assembly completion time of P_2 is calculated as $C'_2 = R_2 + t_2 = 22 + 7 = 29$.

Step3: According to C'_1 and C'_2, an initial product sequence is obtained as $\pi_P = \{P_1, P_2\}$.

Step4: π_P is decoded into $\pi_J = \{3, 5, 1, 2, 4\}$.

Step5: According to the *NRa* rule, all jobs in π_J are assigned to the production factories f_1 and f_2. Assume that J_3, J_5, J_1, and J_2 have been assigned to production factories f_1 and f_2. Next, J_4 will be assigned to production factories f_1 or f_2. Fig. 8 is a Gantt chart that J_4 is assigned according to the *NRa* rule. It can be seen from Fig. 8 that (f) is the optimal situation where J_4 is allocated. Therefore, J_4 is assigned to the third position of f_2, the product sequence is $\pi_P = \{P_1, P_2\}$, and the assembly completion time of π_P is $C_{max}(\pi_P) = 31$.

4.3.4. Heuristic 3

When Heu2_NRa constructs the job sequence π_P that composes each product, it is only arranged in descending order according to the total processing time P^{J_i} of each job. However, this job sequence π_P is likely further optimized utilizing NEH heuristics. This step is considered in heuristic 3 (Heu3_NRa). Therefore, the steps of Heu3_NRa are as follows.

Step1: The jobs that make up each product are constructed as a job sequence according to the ENH heuristic.

Step2: The R_h and t_h of each product are utilized to calculate the earliest assembly completion time of this product $C'_h = R_h + t_h$.

Step3: The C'_h of each product is utilized to construct an initial product sequence π_P in descending order.

Step4: π_P is decoded into π_J, and all jobs in π_J are assigned to various production factories in accordance with the *NRa* rule.

The calculation steps of Heu3_NRa are similar to those of Heu2_NRa. Therefore, there is no explanation.

4.4. Variable neighborhood descent algorithms

Variable neighborhood search (VNS) is a simple and effective local search method. VNS utilizes neighborhood structures composed of different actions to search alternately, and achieves a balance between centrality and evacuation. Variable neighborhood descent (VND), as a simple variant of VNS [19], has been widely applied to flow shop problems [19] and distributed flow shop problems [19]. VND starts from an initial solution S and searches the neighborhood structure $N_i(i = 1, 2, \ldots, m)$ until it falls into the locally optimal solution S'. If S' is superior to S, the first neighborhood is returned. Otherwise, the second neighborhood is searched. The search continues until a locally optimal solution S' is found in all neighborhood structures. In this paper, a method based on two neighborhood structures is proposed. One is a swapping structure (Swap_Job) based on jobs between factories, and the other is an insertion structure (Insert_Job)

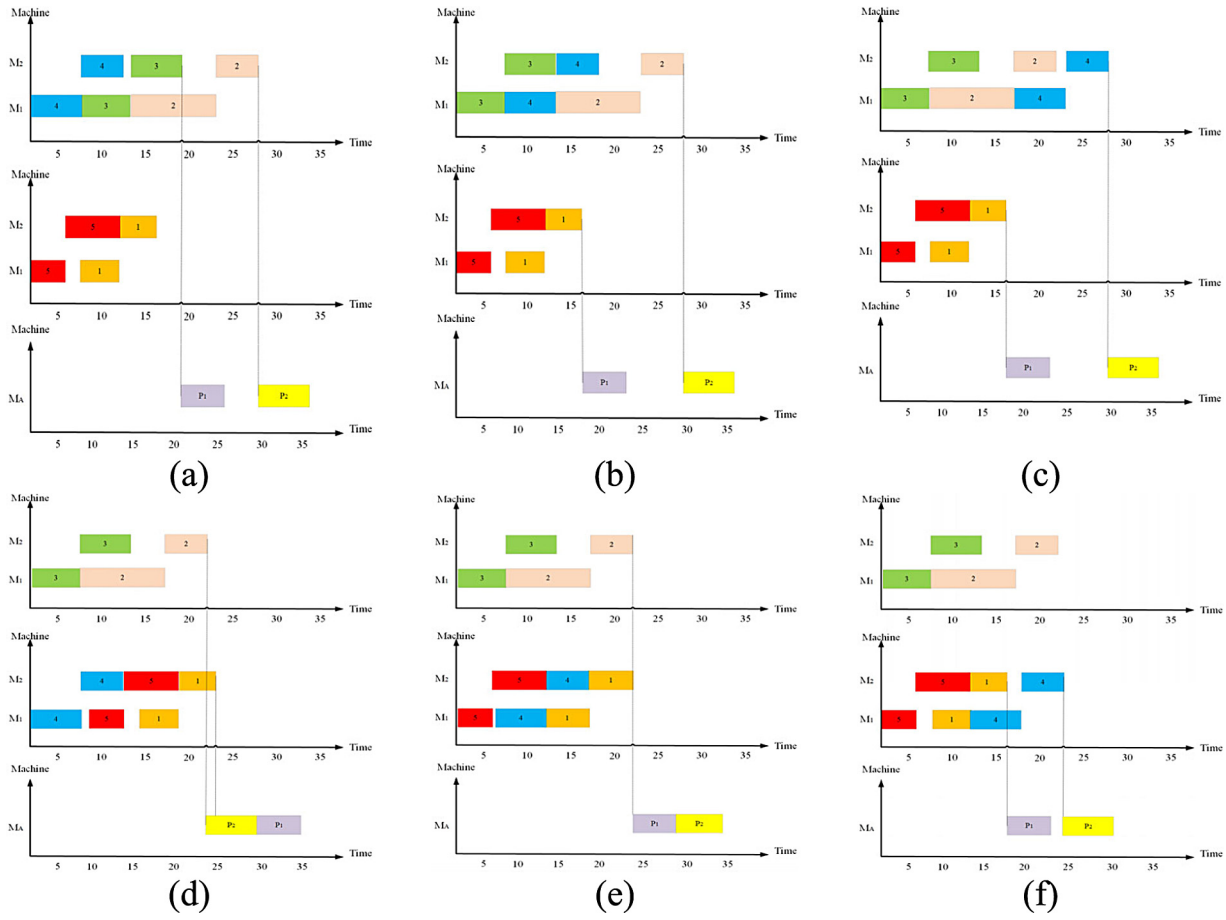


Fig. 8. The Gantt chart of the process that job J_4 is allocated.

Algorithm 3: Heu2_NRa

for $i = 1$ **to** n
  Calculate the total processing time P^{J_i} of job i.
end for
for $i = 1$ **to** s
  The jobs composing product i are arranged in descending order into a job sequence P_i of according to the P^{J_i}.
  The R_i and t_i of product i are utilized to calculate the earliest assembly completion time C_i' of this product.
end for
The C_h' of each product is utilized to construct an initial product sequence π_P in descending order.
π_P is decoded into π_J.
for $i = 1$ **to** n
  for $j = 1$ **to** F
    The job i in π_j is assigned to the production factory j according to the $\boldsymbol{NR_a}$ rule, and the job sequence π^j in the factory j is obtained.
  end for
end for
return π

based on jobs between factories. The pseudocode for VND is explained in Algorithm 6. The next two neighborhood structures are introduced respectively.

4.4.1. Neighborhood structure based on swapping

The swapping neighborhood structure based on jobs between factories swaps all jobs in the largest factory with maximum completion time C_{max} to all jobs in other factories to find scheduling sequences with less assembly completion time. C_{max} is the maximum completion time for the job to be processed. Fig. 9 is a schematic of Swap_Job. The pseudocode for Swap_Job is explained in Algorithm 4.

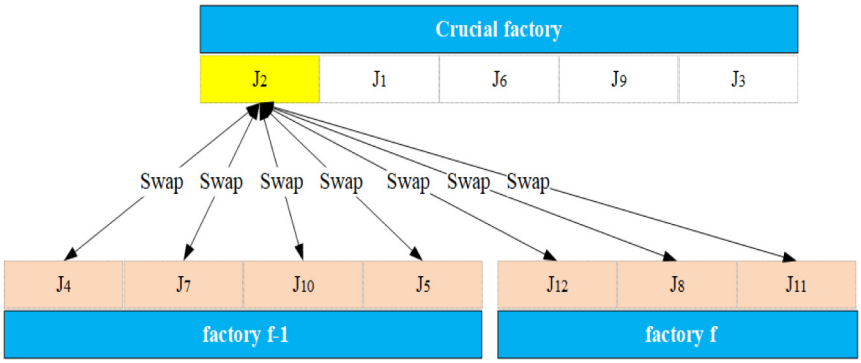


Fig. 9. The schematic of Swap_Job.

Algorithm 4: Swap_Job(π)

1.$Boole = true$
2.**While** $Boole$ **do**
3. Crucial factory K is identified and $Boole = false$.
4. **for** $i = 1$ **to** K_n
5. Job i is swapped with all jobs in other factories to obtain the sequence π' of jobs that minimizes the assembly completion time of the product under this operation.
6. **if** $C_{max}(\pi') < C_{max}(\pi)$ **then**
7. $\pi = \pi', Boole = true$
8. break
9. **end if**
10. **end for**
11.**end while**
12.**return** π

Algorithm 5: Insert_Job(π)

1.$Boole = true$
2.**While** $Boole$ **do**
3. Crucial factory K is identified and $Boole = false$.
4. **for** $i = 1$ **to** K_n
5. Job i is inserted into all possible positions in other factories to obtain the sequence π' of jobs that minimizes the assembly completion time of the product under this operation.
6. **if** $C_{max}(\pi') < C_{max}(\pi)$ **then**
7. $\pi = \pi', Boole = true$
8. break
9. **end if**
10. **end for**
11.**end while**
12.**return** π

4.4.2. Neighborhood structure based on insertion

The insertion neighborhood structure based on jobs between factories, all jobs in the maximum factory with completion time C_{max} are inserted into all possible positions of other factories to find scheduling sequences with less assembly completion time. C_{max} is the maximum completion time for the job to be processed. Fig. 10 is a schematic of Insert_Job. The pseudocode for Insert_Job is explained in Algorithm 5.

4.5. The block-shifting based on optimal block knowledge

4.5.1. The optimal block knowledge

In the No-wait flow shop scheduling problem (NWFSP), the same job has no waiting time between two adjacent processes [36]. Therefore, all the processes of the same job are able to be spliced into a whole to calculate the completion time difference $D_{(i,i+1)}$ between two adjacent jobs. The smaller the value of $D_{(i,i+1)}$ is, the better the job block composed of these two jobs is. There is such an optimal job block in each job sequence. When operating on candidate solutions, such optimal job block should not be destroyed and should be retained. Such job blocks are referred to as the optimal knowledge implicit in a job sequence. It is an effective method to guide the operation of the candidate solution with the optimal knowledge. Fig. 11 illustrates the knowledge of the optimal job block. In the example in Fig. 11, the sequence of jobs is $\{J_1, J_4, J_2, J_5, \}$. The completion time difference between J_1 and J_4 is $D_{(1,4)} = 26$. The completion time difference between J_4 and J_2 is $D_{(4,2)} = 26$. The completion time difference between J_2 and J_5 is $D_{(2,5)} = 23$. Therefore, the job block composed of J_2 and J_5 is considered to be the optimal job block in the sequence

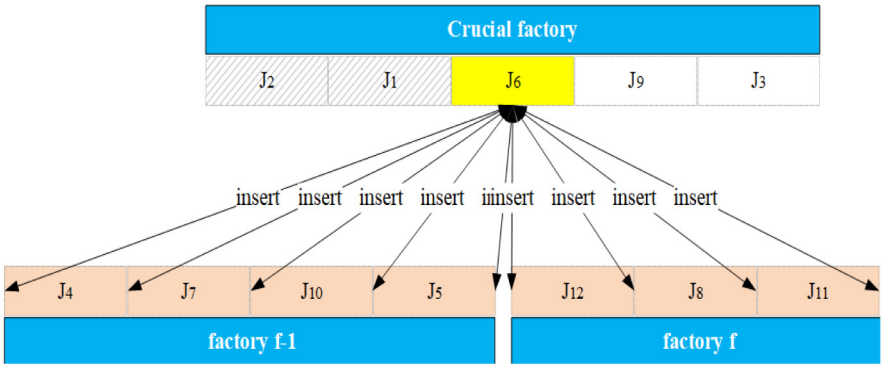


Fig. 10. The schematic of Insert_Job.

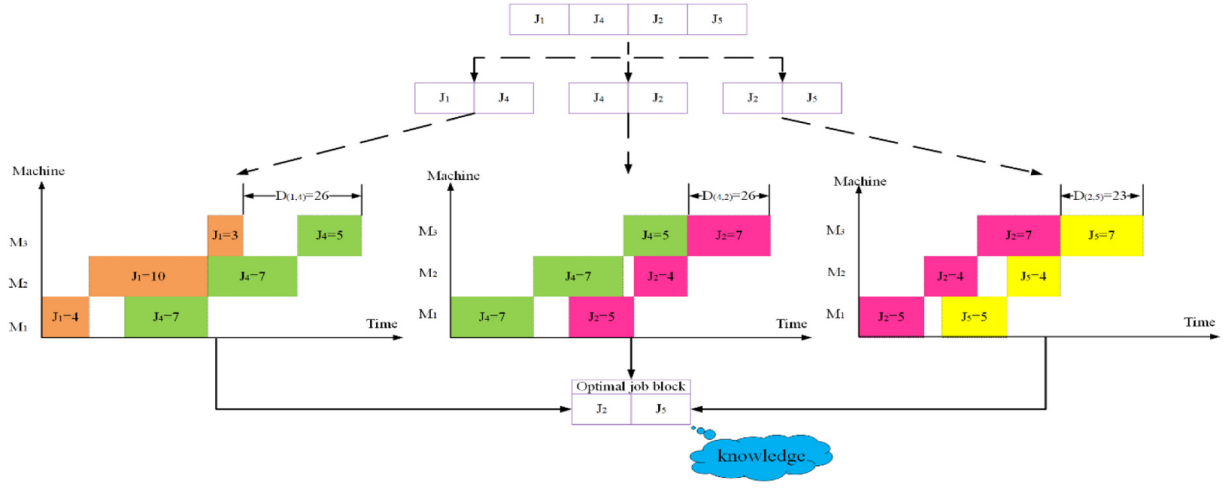


Fig. 11. An example of optimal block knowledge.

$\{J_1, J_4, J_2, J_5, \}$. When $\{J_1, J_4, J_2, J_5, \}$ are operated, J_2 and J_5 should be guaranteed not to be separated.

4.5.2. The block-shifting

Block-shifting is an extension of pairs of swap operations and insert operations. Studies have shown that block-shifting is able to explore a larger neighborhood structure than paired swap operations and insert operations. Therefore, block-shifting is able to effectively overcome the disadvantage that paired swap operations and insert operations tend to make candidate solutions fall into local optimality [41]. To satisfy both the application conditions of the optimal job block and the recommendation of the job block size Z in [41]. In the block-shifting proposed in this paper, the size of the job block Z satisfies the condition $Z \in [2, \frac{f_n}{2}]$. f_n is the total number of jobs assigned to factory f. The use of knowledge about optimal job blocks is strictly controlled by a feedback mechanism. Before each iteration, if the candidate solution is identical to the candidate solution of the previous generation, then Z consecutive jobs are randomly selected for block-shifting operation. Otherwise, select the optimal job block composed of Z continuous job blocks for block-shifting operation. In other words, the degree of similarity between the candidate solutions serves as a feedback indicator to control the use of knowledge about the optimal job block. Fig. 12 is an example of block-shifting operation for the candidate solution $\{J_1, J_4, J_2, J_5, \}$ with and without the knowledge of the optimal job block, respectively. In the above example, the job block composed of J_2 and J_5 is the optimal job block of the sequence $\{J_1, J_4, J_2, J_5, \}$. J_4, J_2 are randomly selected block of jobs. As can be seen from the figure, the block-shifting operation using the optimal job block finds a better C_{max}.

4.6. The BSA with optimal job block knowledge

BSA based on the knowledge of the optimal job block (BKBSA) has five main steps, including initialization, selection-I, mutation, local search, and selection-II. Fig. 13 is a flow chart of BKBSA. The pseudocode for BKBSA is explained in Algorithm 7.

4.6.1. Initialization

In DANWFSP, there is one current population and one historical population for each production factory. N is the size of the population. The best of the three heuristic constructs described in Section 3.3 is used to construct a candidate solution for the current population. The remaining $N - 1$ candidate solutions consist of random permutations of the jobs assigned to the population. The candidate solutions in the historical population are all composed of the random arrangement of jobs in the population.

4.6.2. Selection-I

The selection-I of BKBSA still takes the operator in the original BSA. In the selection-I, the historical population *oldP* is updated with a random probability by the current population P. The details are as follows.

$$oldP = \begin{cases} P, & a < b, a, b \sim U(0,1) \\ oldP, & otherwise \end{cases} \tag{34}$$

where, a and b are two random Numbers that are uniformly distributed.

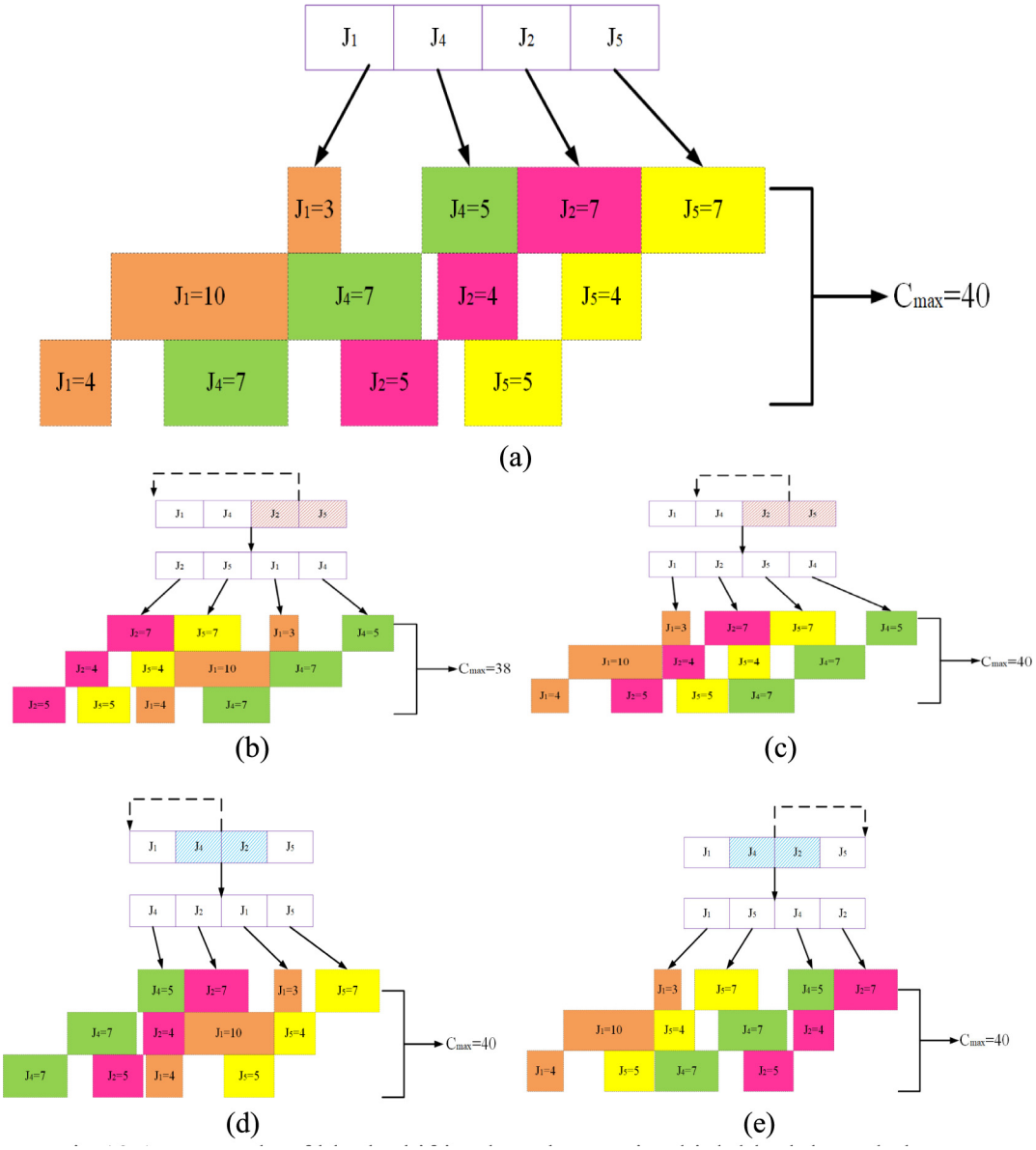


Fig. 12. An example of block-shifting based on optimal job block knowledge.

Algorithm 6: VND(π)

1. $r = 1$
2. $r_{max} = 2$
3. **While** $r \le r_{max}$ **do**
4. **switch** (r)
5. **case 1**: Optimize π with Insert_Job and output the optimized π'.
6. **case 2**: Optimize π' with Swap_Job and output the optimized π''.
7. **end switch**
6. **if** $C_{max}(\pi'') < C_{max}(\pi)$ **then**
7. $\pi = \pi'', r = 1$
8. **else**
9. $r = r + 1$
10. **end if**
11. **end while**
12. **return** π

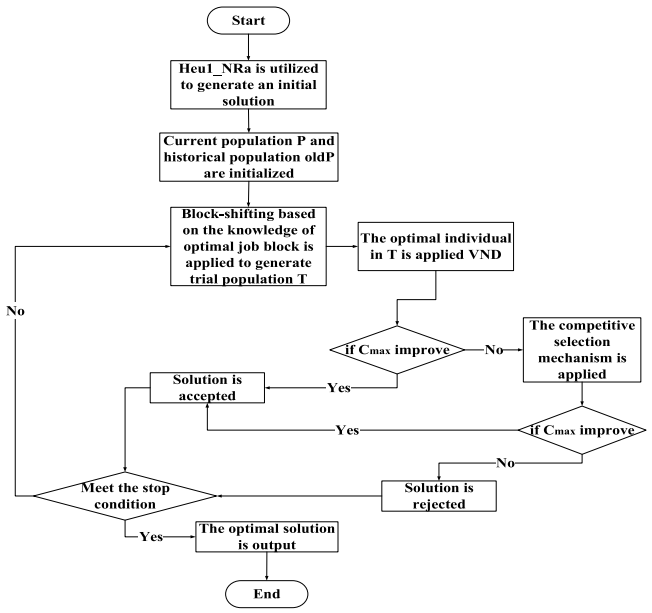


Fig. 13. The flow chart of BKBSA.

4.6.3. Mutation

In the mutation operation of BKBSA, the block-shifting based on the knowledge of the optimal job block proposed in Section 3.5 is used to generate the trial population T. In the mutation operation, each candidate solution in the current population P has to go through the block-shifting operation. During block-shifting, the optimal job block from each candidate solution in P is inserted at all possible positions of the candidate solution. Then the optimal candidate solution is selected as the candidate solution in T.

4.6.4. Local search

After the mutation operation, the variable neighborhood descending algorithm VND proposed in Section 3.4, was applied to the trial population T to improve the optimal candidate solution in T. In each iteration, the above actions are performed.

4.6.5. Selection-II

In the selection-II of HKBSA, the selection-II of the original BSA was preserved. Moreover, a competitive selection mechanism is proposed. Competition-based selection mechanism is used in the selection-II process for individuals who fail to make it into the next generation. Assuming that m individuals failed to enter the next generation in the selection-II, m historical individuals with optimal fitness values are selected from *oldP* to compete with the previous m individuals. Then, the better m individuals are selected as the candidate solutions of the next generation population P.

5. Simulation experiment and discussion

In this paper, the large test suite and small test suite of Benchmark suite proposed in [1] is used to test the effectiveness of the BKBSA. The test suite has a total of 1710 test instances, consisting of different number of jobs, machines, factories, and products. The evaluation criterion adopted in this paper is the average relative percentage deviation (ARPD). ARPD is calculated as follows.

$$ARPD = \frac{1}{Run}\sum_{r=1}^{Run}\frac{C^r - C^*}{C^*} \times 100 \tag{35}$$

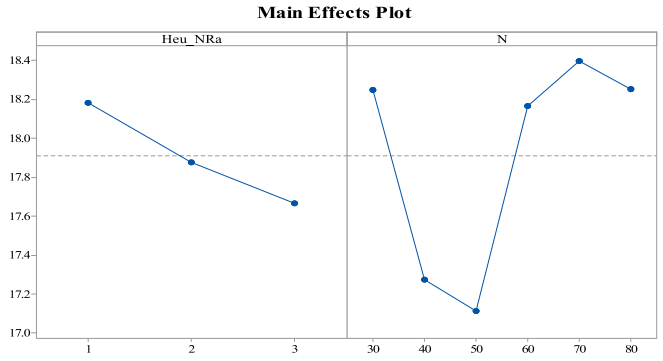


Fig. 14. Main effect plot for BKBSA.

where, Run is the number of times the experiment runs independently, and C^r is the solution obtained on an instance when an algorithm reaches the stop criterion in the *rth* experiment. C^* is the optimal solution obtained by all algorithms in this paper on an instance until the stopping criterion is satisfied. The stopping criterion for the experiment is the maximum CPU time $T_{max} = \rho \cdot mn, \rho \in \{10, 20, 30\}$ in milliseconds, consistent with that in [19]. In order to ensure the same experimental environment, the code of all algorithms in this paper is written in MATLAB 2016 and run on PC with 3.4 GHz, Intel (R), Core (TM) i7-6700 CPU, 8GB of ARM and 64-bit OS. Minitab 17 and SPSS are used to process the experimental data.

5.1. Parameter setting

In this section, design of experiments (DOE) and analysis of Variance (ANOVA) [42] are used to calibrate the parameters in the BKBSA. There are two kinds of parameters in BKBSA to be calibrated. The first is the constructional Heuristic *Heu_NRa*, and the second is the size of population N. *Heu_NRa* has three levels *Heu*1_*NRa*, *Heu*2_*NRa*, and *Heu*3_*NRa*. According to the original BSA and [34,35], the value of N is also set to six levels, namely 30, 40, 50, 60, 70 and 80. Therefore, a total of 9 parameter combinations should be tested. The 810 large-scale instances are 81 combinations of $n = \{100, 200, 500\}$, $m = \{5, 10, 20\}$, $s = \{30, 40, 50\}$, $F = \{4, 6, 8\}$. Each combination is divided into 10 different instances. The 10 instances differ only in the processing time of the job. For each combination, one of 10 instances is randomly selected, and a total of 81 instances are selected as the test suite of this experiment. Each parameter combination experiment is run independently for 5 times. Table 4 is the ANOVA analysis results of parameter calibration experiments.

From the main effect plot in Fig. 14, when the population size is 30, 60, 70, 80, the ARPD value obtained by the algorithm is larger. Certain factories may be assigned fewer jobs in the job allocation phase. If the population size is less than the full arrangement of the job sequence, some candidate solutions will not be represented, which will lead to the algorithm unable to find the optimal solution. If the population is larger than the full arrangement of the job sequence, some candidate solutions will be repeatedly represented, which will increase the computational complexity of the algorithm and reduce the efficiency of the algorithm. From Table 4, both N and *Heu_NRa* have significant influences on BKBSA, especially *Heu_NRa*. Since p-value of N and *Heu_NRa* are less than 0.05. Therefore, $N = 50$ and *Heu*3_*NRa* is the best parameter combination of the BKBSA.

Algorithm 7: BKBSA

1.Heu3_NRa is utilized to generate an initial solution.
2.Current population P and historical population $oldP$ are initialized.
3.**While** $t \leq T_{max}$ **do**
4. Block-shifting based on the knowledge of optimal job block is applied to generate trial population T.
5. The optimal individual in T is applied VND.
6. **if** $C_{max}(T) < C_{max}(P)$ **then**
7. $P = T$
8. **else**
9. The competitive selection mechanism is utilized to select P.
10. **end if**
11.**end while**
12.Output the optimal solution found so far.

Table 4
ANOVA results for parameters of BKBSA.

Source	Sum	d.f.	Mean	F-ratio	*p*-value
Main effects	Sq.		Sq.		
N	0.8116	2	0.40581	6.94	**0.0129**
Heu_NRa	4.7214	5	0.94428	16.15	**0.0002**
Error	0.5846	10	0.05847		
ToTal	6.1177	17			

5.2. Model validation

In this section, the CPLEX-12.5 optimizer is utilized to evaluate the MILP of DANWFSP to verify the correctness of the MILP. Certain small-scale instances and large-scale instances were used to verify the accuracy of MILP. If there is a little difference between the optimal solutions obtained by CPLEX and BKBSA on these instances, it shows that the MILP established in this paper is correct.

The completion time difference D between two adjacent jobs is only related to the relative order of the two jobs in NWFSP. Therefore, NWFSP is transformed into an asymmetric traveling salesman problem (ATSP) to simplify the difficulty of solving the problem. The completion time difference D between two adjacent parts can be calculated in advance. The transformation relationship between NWFSP and the traveling salesman problem (TSP) is shown in Fig. 15. In the figure, d represents the distance between two cities. $D(0,1)$ represents the completion time difference between the virtual job J_0 and job J_1. The processing time of the virtual job on all machines is 0.

The results of the comparison between CPLEX and BKBSA on certain small instances are recorded in Table 5. From the table, the optimal solutions obtained by CPLEX and BKBSA on small-scale instances are almost the same. However, in certain instances, the production sequence π_P and job sequence π_J are different from each other. ARPD comparison results of CPLEX and BKBSA in large-scale instances are recorded in Table 6. In the large-scale test set, only the instances with $n = 100$ were selected because of the high problem complexity of large-scale instances and the long CPU time. From the table, the ARPD of CPLEX is greater than that of BKBSA. CPLEX only operates on the initial solution generated by the construction heuristic, and does not adopt the VND operation, which makes the candidate solution have a certain disadvantage in the expansion of the neighborhood structure. However, there is no significant difference between the optimal solution obtained by CPLEX and BKBSA. Therefore, the MILP established in this paper is accurate.

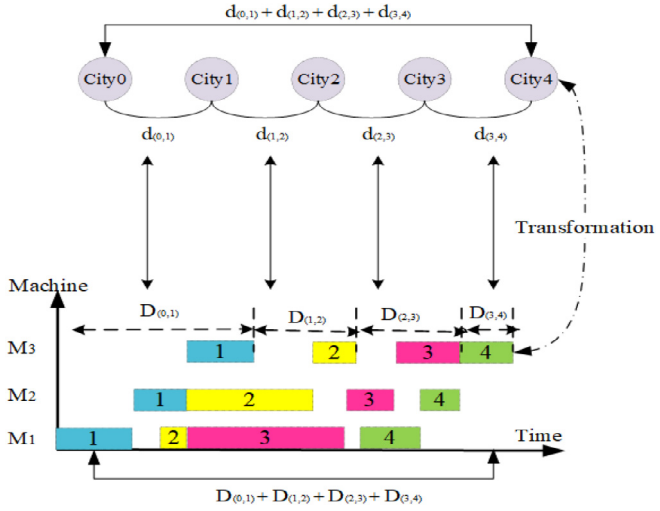


Fig. 15. Schematic diagram of model transformation between NWFSP and TSP.

5.3. Effectiveness analysis of the strategies in BKBSA

In this section, the effectiveness of the three proposed constructive heuristics, the block-shifting based on the knowledge of the optimal job blocks and VND in BKBSA are discussed separately. The selection of test suite in this section is consistent with the selection of test suite in Section 5.1. Each experiment is run independently for 5 times.

5.3.1. Effectiveness analysis of constructive heuristics

In this section, the three constructive heuristics proposed in this paper are compared with the other eight constructional heuristics to verify their effectiveness. Eight heuristics are presented in [1,19], and [21] respectively. Table 15 records the ARPD of eleven constructive heuristics. From the statistical results, the ARPD of *Heu2_NRa* and *Heu3_NRa* is significantly smaller than that of the other nine constructivist heuristics. Since the assembly completion time of the product is considered in the assignment of the job stage of *Heu3_NRa*, the job will be allocated to the factory with the minimum assembly completion time of the product. This conclusion is also reflected in the interval plot of Fig. 16. *H2NRa* is slightly better than *Heu1_NRa* in certain instances.

5.3.2. Effectiveness analysis of proposed block-shifting

This section discusses the effectiveness of block-shifting based on knowledge of optimal job blocks in BKBSA. The BKBSA without

Table 5
Solution results of CPLEX and BKBSA in small-scale instances.

Instances	CPLEX			BKBSA		
$n \times m \times f \times s$	$C_{max}(\pi_P)$	π_P	π_J	$C_{max}(\pi_P)$	π_P	π_J
$8 \times 2 \times 2 \times 2 \times 1$	468	{2, 1}	{7, 3, 8, 2, 6, 5, 1, 4}	468	{2, 1}	{7, 3, 8, 2, 6, 5, 1, 4}
$8 \times 2 \times 2 \times 3 \times 2$	525	{1, 2, 3}	{1, 2, 3, 5, 8, 7, 4, 6}	525	{1, 2, 3}	{1, 2, 4, 3, 5, 8, 7, 6}
$8 \times 2 \times 2 \times 4 \times 4$	509	{1, 2, 3, 4}	{8, 7, 2, 3, 1, 4, 6, 5}	509	{1, 2, 4, 3}	{8, 7, 3, 6, 1, 4, 5, 2}
$8 \times 3 \times 2 \times 3 \times 3$	584	{3, 1, 2}	{6, 1, 4, 8, 5, 3, 72}	584	{3, 1, 2}	{6, 1, 4, 2, 8, 5, 3, 7}
$8 \times 3 \times 3 \times 4 \times 2$	605	{4, 1, 2, 3}	{5, 7, 4, 1, 3, 6, 8, 2}	605	{4, 1, 23}	{5, 7, 4, 1, 3, 6, 8, 2}
$8 \times 3 \times 4 \times 3 \times 3$	679	{1, 2, 3}	{2, 5, 6, 7, 4, 3, 1, 8}	679	{1, 2, 3}	{2, 5, 6, 7, 4, 3, 1, 8}
$8 \times 3 \times 4 \times 4 \times 1$	417	{4, 3, 1, 2}	{7, 3, 2, 4, 5, 6, 1, 8}	417	{4, 3, 1, 2}	{7, 3, 2, 4, 5, 6, 1, 8}
$8 \times 4 \times 2 \times 3 \times 5$	599	{2, 1, 3}	{3, 1, 2, 5, 6, 4, 7, 8}	599	{2, 3, 1}	{3, 1, 6, 2, 5, 4, 8, 7}
$8 \times 5 \times 2 \times 4 \times 2$	666	{2, 4, 3, 1}	{1, 3, 6, 8, 2, 7, 5, 4}	654	{2, 4, 1, 3}	{2, 6, 8, 4, 7, 5, 1, 3}
$8 \times 5 \times 3 \times 3 \times 1$	667	{1, 3, 2}	{7, 4, 1, 8, 5, 3, 2, 6}	667	{1, 3, 2}	{3, 2, 6, 8, 5, 7, 4, 1}

Table 6
Comparison of ARPD between CPLEX and BKBSA in large-scale instances.

Instances	CPLEX	BKBSA	Instances	CPLEX	BKBSA
$n \times m \times f \times s$	*ARPD*	*ARPD*	$n \times m \times f \times s$	*ARPD*	*ARPD*
$100 \times 5 \times 4 \times 30$	0.454	0.168	$100 \times 5 \times 8 \times 50$	0.239	0.092
$100 \times 5 \times 4 \times 40$	0.359	0.108	$100 \times 10 \times 4 \times 30$	0.884	0.147
$100 \times 5 \times 4 \times 50$	0.363	0.103	$100 \times 10 \times 4 \times 40$	0.665	0.123
$100 \times 5 \times 6 \times 30$	0.605	0.151	$100 \times 10 \times 4 \times 50$	0.605	0.136
$100 \times 5 \times 6 \times 40$	0.478	0.128	$100 \times 10 \times 6 \times 30$	0.916	0.286
$100 \times 5 \times 6 \times 50$	0.258	0.075	$100 \times 10 \times 6 \times 40$	0.512	0.137
$100 \times 5 \times 8 \times 30$	0.678	0.175	$100 \times 10 \times 6 \times 50$	0.430	0.080
$100 \times 5 \times 8 \times 40$	0.184	0.128	$100 \times 10 \times 8 \times 30$	0.875	0.191

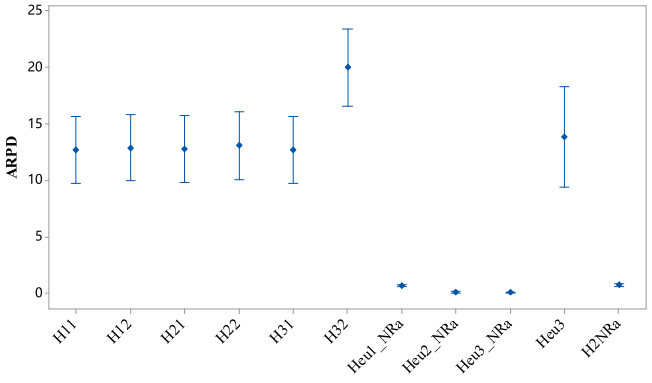


Fig. 16. The interval plot of 11 constructive heuristic.

Table 7
The ARPD of the BKBSA1 and BKBSA2.

		BKBSA1	BKBSA2
	100	0.364	**0.016**
n	200	0.167	**0.094**
	500	0.138	**0.056**
	4	0.302	**0.016**
f	6	0.238	**0.020**
	8	0.197	**0.024**
	30	0.271	**0.039**
s	40	0.227	**0.035**
	50	0.227	**0.036**
	5	0.214	**0.025**
m	10	0.286	**0.042**
	20	0.211	**0.026**
Mean		0.237	**0.036**

block-shifting is denoted as BKBSA1, and the BKBSA with block-shifting is denoted as BKBSA2. To ensure the fairness of the experiment, BKBSA1 and BKBSA2 adopt the same initial solution. Fig. 17 and Table 7 are the comparison result of BKBSA1 and BKBSA 2. From the results, the ARPD of BKBSA2 is better than that of BKBSA1. BKBSA2 adopts the block moving operation based on the optimal job block, which enables the parent candidate solution to inherit its own optimal subsequence to its offspring completely. This optimization strategy will make the candidate solution spend less computation time when being evaluated, thus increasing the time for the candidate solution to expand more neighborhood structures. Therefore, the proposed block-shifting is effective.

5.3.3. Effectiveness analysis of proposed VND

This section discusses the effectiveness of VND in BKBSA. On the basis of BKBSA2, the VND is deleted and denoted as BKBSA3. BKBSA2 and BKBSA3 adopt the same initial solution. Fig. 18 and Table 8 show the comparison result between BKBSA2 and BKBSA3. BKBSA2 adopted VND as a local search strategy after each iteration of the population. VND includes the swapping and insertion structure based on the crucial factory, which makes the neighborhood structure in the crucial factory change greatly, so as to produce better candidate solutions. Therefore, the impact of VND on BKBSA is significant.

5.4. Comparison with other algorithms

In this section, the BKBSA is compared with eight other comparison algorithms to verify the effectiveness of BKBSA. The eight comparison algorithms are the optimal competitiveness algorithms of DAPFSP in recent years, which are BR-ILS [20], IG [19], HDIWO, HDIWOp, TDIWO [2], ILS [21], ILS-NR3 [22] and gIGA [43]. For these eight comparison algorithms, the parameters in the literature are strictly observed to ensure the maximum similarity of the algorithm. The only change is that all algorithms

Table 8
The ARPD of the BKBSA2 and BKBSA3.

		BKBSA2	BKBSA3
	100	**0.045**	0.263
n	200	**0.016**	0.190
	500	**0.019**	0.145
	4	**0.023**	0.202
f	6	**0.029**	0.203
	8	**0.032**	0.180
	30	**0.023**	0.220
s	40	**0.024**	0.234
	50	**0.015**	0.140
	5	**0.028**	0.210
m	10	**0.015**	0.184
	20	**0.027**	0.186
Mean		**0.025**	0.197

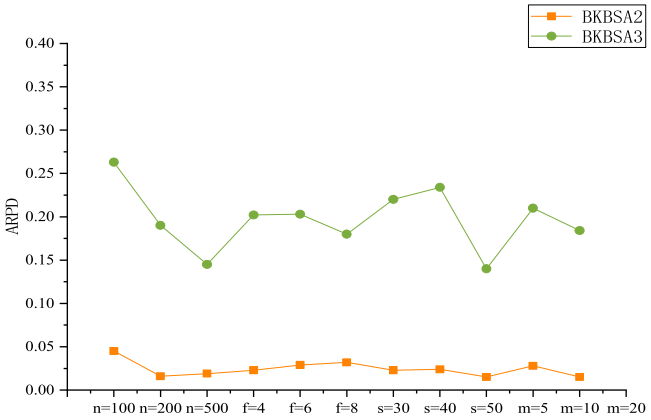


Fig. 18. The series plots of BKBSA2 and BKBSA3.

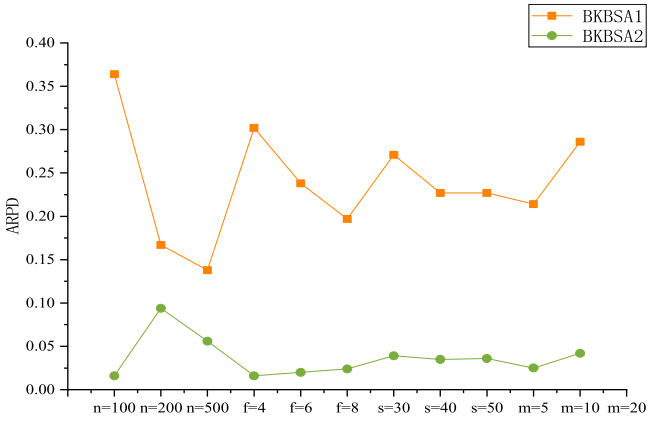


Fig. 17. The series plots of BKBSA1 and BKBSA2.

adopt the stop criterion proposed in this paper. Each algorithm is run independently 5 times. Tables 9–11 record the ARPD values of nine algorithms when $\rho = 10$, $\rho = 20$ and $\rho = 30$ on large-scale instances respectively.

The ARPD values of HDIWO, HDIWOp, TDIWO, ILS-NR3 and BKBSA are significantly better than those of the other four algorithms under the three stop criteria. When $\rho = 10$, the ARPD of BKBSA at $f = 8$ is 0, which indicates that the optimal solution obtained by BKBSA in this case is smaller than other comparison algorithms. When $\rho = 20, 30$, the ARPD of BKBSA at $n = 100$ and $f = 8$ is 0, which indicates that the optimal solution obtained by BKBSA in these two cases is smaller than other comparison algorithms. From Tables 9–11, the ARPD of HDIWO, HDIWOp, TDIWO, ILS-NR3 and BKBSA has little difference. Therefore, the results of the above five algorithms in different instances are recorded in Fig. 20 to compare the differences between these algorithms. The curves representing ILS-NR3 and BKBSA in the figure are basically lower than those of the other three algorithms, indicating that the performance of ILS-NR3 and BKBSA is better than that of the other three algorithms.

When the number of jobs is 100 and 200, there is little difference in ARPD between BKBSA and ILS-NR3. When the number of jobs is 500, the ARPD of ILS-NR3 is better than that of BKBSA. The heuristic used in ILS-NR3 inserts the jobs into adjacent positions of different jobs belonging to the same product during the construction of the initial solution, which makes the jobs of the same product more compact and has an earlier release time. In the local search stage, the local search method adopted in ILS-NR3 is also operated at the adjacent positions of different jobs of the same product, which reduces the complexity of the problem for large-scale problems. However, BKBSA is still superior to ILS-NR3 on certain instances. Overall, BKBSA is comparable to ILS-NR3 in performance.

Tables 12–14 show the ARPD of the nine algorithms in the small-scale test set. According to the table, the ARPD of BKBSA is almost superior to other comparison algorithms except ILS-NR3 in the three stop criteria. BKBSA has a historical population, which provides better diversity for the population in the evolutionary process so that the algorithm can obtain better candidate solutions in a limited time. The comparison results of HDIWO,

Table 9
The ARPD of the nine algorithms with $\rho = 10$ on large-scale instances.

		BKBSA	BRILS	gIGA	HDIWO	HDIWOp	IG	ILS	ILS-NR3	TDIWO
	100	0.001	1.074	0.491	0.002	0.002	0.858	0.743	**0.000**	0.003
n	200	**0.002**	3.828	2.881	0.003	0.006	7.339	0.897	0.002	0.006
	500	0.013	15.873	8.978	0.017	0.027	8.666	0.778	**0.006**	0.024
	4	0.013	10.376	6.076	0.013	0.019	10.730	0.897	**0.006**	0.019
f	6	0.002	6.055	3.968	0.007	0.011	4.589	0.777	**0.001**	0.010
	8	**0.000**	4.345	2.306	0.004	0.006	1.543	0.745	0.001	0.005
	30	0.011	8.261	5.287	0.013	0.020	3.658	1.064	**0.004**	0.018
s	40	0.002	6.797	3.782	0.007	0.012	6.168	0.751	**0.001**	0.013
	50	**0.002**	5.717	3.282	0.003	0.003	7.036	0.604	0.003	0.003
	5	0.009	5.050	1.107	0.007	0.009	1.527	0.586	**0.001**	0.009
m	10	**0.002**	6.648	3.307	0.006	0.010	4.901	0.797	0.002	0.010
	20	0.005	9.077	7.936	0.010	0.016	10.434	1.036	**0.004**	0.014
Mean		0.005	6.925	4.117	0.008	0.012	5.621	0.806	0.003	0.011

Table 10
The ARPD of the nine algorithms with $\rho = 20$ on large-scale instances.

		BKBSA	BRILS	gIGA	HDIWO	HDIWOp	IG	ILS	ILS-NR3	TDIWO
	100	**0.000**	0.967	0.183	0.001	0.001	0.740	0.742	0.000	0.001
n	200	**0.001**	3.084	2.138	0.003	0.005	6.517	0.896	0.001	0.005
	500	0.012	14.184	7.907	0.013	0.023	8.383	0.778	**0.003**	0.021
	4	0.011	9.185	4.751	0.008	0.015	10.166	0.894	**0.003**	0.013
f	6	0.003	5.333	3.135	0.006	0.010	4.111	0.777	**0.001**	0.009
	8	**0.000**	3.717	2.343	0.003	0.005	1.363	0.745	0.000	0.004
	30	0.009	7.188	4.935	0.010	0.015	3.226	1.061	**0.001**	0.014
s	40	0.002	5.982	2.647	0.005	0.011	5.670	0.750	**0.001**	0.009
	50	0.003	5.065	2.647	0.001	0.004	6.745	0.604	**0.002**	0.003
	5	0.009	4.434	0.948	0.005	0.008	1.377	0.586	**0.001**	0.007
m	10	0.002	5.711	2.837	0.005	0.010	4.587	0.797	**0.001**	0.008
	20	**0.003**	8.090	6.444	0.007	0.012	9.676	1.034	0.003	0.011
Mean		0.005	6.078	3.410	0.006	0.010	5.213	0.805	0.002	0.009

Table 11
The ARPD of the nine algorithms with $\rho = 30$ on large-scale instances.

		BKBSA	BRILS	gIGA	HDIWO	HDIWOp	IG	ILS	ILS-NR3	TDIWO
	100	**0.000**	0.910	0.373	0.001	0.001	0.766	0.742	0.000	0.001
n	200	**0.001**	2.656	2.228	0.003	0.005	6.340	0.896	0.001	0.005
	500	0.012	13.111	6.798	0.013	0.023	8.213	0.778	**0.003**	0.021
	4	0.011	8.454	4.233	0.008	0.015	10.051	0.894	**0.003**	0.013
f	6	0.003	4.868	3.265	0.006	0.010	4.032	0.777	**0.001**	0.009
	8	**0.000**	3.354	1.901	0.003	0.005	1.236	0.745	0.000	0.004
	30	0.009	6.494	4.347	0.010	0.015	3.166	1.061	**0.001**	0.014
s	40	0.002	5.486	2.517	0.005	0.011	5.546	0.750	**0.001**	0.009
	50	0.003	4.696	2.536	0.001	0.004	6.606	0.604	**0.002**	0.003
	5	0.009	3.971	0.816	0.005	0.008	1.226	0.586	**0.001**	0.007
m	10	0.002	5.160	2.767	0.005	0.010	4.586	0.797	**0.001**	0.008
	20	**0.003**	7.546	5.817	0.007	0.012	9.506	1.034	0.003	0.011
Mean		0.005	5.559	3.133	0.006	0.010	5.106	0.805	0.002	0.009

Table 12
The ARPD of the nine algorithms with $\rho = 10$ on small-scale instances.

		BKBSA	BRILS	gIGA	HDIWO	HDIWOp	IG	ILS	ILS-NR3	TDIWO
	8	0.293	2.852	4.928	0.334	0.334	4.742	2.116	**0.052**	0.336
	12	**0.046**	3.022	5.022	0.255	0.256	5.198	2.845	0.051	0.274
n	16	**0.027**	4.753	6.472	0.112	0.187	6.721	4.206	0.038	0.191
	20	**0.047**	4.493	6.602	0.098	0.202	6.847	5.232	0.048	0.149
	24	**0.056**	4.950	7.231	0.106	0.230	7.138	4.584	0.101	0.222
	2	0.174	6.214	4.553	0.229	0.326	4.681	5.211	**0.123**	0.297
f	3	0.115	3.730	6.208	0.144	0.195	6.258	3.674	**0.035**	0.193
	4	0.039	2.098	7.392	0.170	0.205	7.449	2.504	**0.016**	0.213
	2	0.115	4.658	11.976	0.209	0.309	12.083	4.472	**0.070**	0.264
s	3	**0.039**	4.062	3.897	0.201	0.247	3.972	3.577	0.044	0.244
	4	0.117	3.322	2.279	0.133	0.169	2.333	3.341	**0.060**	0.196
	2	**0.024**	3.614	6.011	0.095	0.146	6.271	2.614	0.029	0.142
m	3	0.108	3.942	5.925	0.247	0.324	5.806	4.010	**0.098**	0.319
	4	0.150	4.505	6.340	0.225	0.269	6.354	4.366	**0.053**	0.264
	5	0.156	3.995	5.928	0.157	0.228	6.087	4.196	**0.052**	0.212
Mean		0.100	4.014	6.051	0.181	0.242	6.129	3.797	0.058	0.234

HDIWOp, TDIWO, ILS-NR3 and BKBSA on different small-scale test sets are recorded in Fig. 20. The horizontal axis in the figure represents the test sets under different groups, and the vertical axis represents ARPD. It can be clearly seen from the figure that the curve representing BKBSA is always below the other three curves except ILS-NR3, which also indicates that the ARPD value of BKBSA is significantly lower than that of other three comparison algorithms. Although in certain instances the ARPD of ILS-NR3 is better than that of BKBSA, overall, the performance of BKBSA is comparable to that of ILS.

Fig. 19 is box plot of nine algorithms using different stop criteria in the same test set. When the number of jobs is 200 and $\rho = 10, 30$, there is no outliers in the box plot of BKBSA, which indicates that the ARPD obtained by BKBSA on the cases with $n =$

Table 13
The ARPD of the nine algorithms with $\rho = 20$ on small-scale instances.

		BKBSA	BRILS	gIGA	HDIWO	HDIWOp	IG	ILS	ILS-NR3	TDIWO
	8	0.303	2.563	4.575	0.333	0.333	4.661	2.115	**0.052**	0.333
	12	**0.026**	2.910	4.983	0.235	0.239	5.115	2.824	0.026	0.272
n	16	**0.037**	4.193	7.192	0.126	0.189	6.100	4.217	0.037	0.223
	20	**0.041**	4.285	6.474	0.096	0.158	6.201	5.225	0.041	0.142
	24	**0.028**	4.637	6.746	0.091	0.217	6.813	4.585	0.057	0.175
	2	0.171	5.429	4.501	0.229	0.316	4.402	5.208	**0.092**	0.290
f	3	0.118	3.484	5.918	0.143	0.177	5.676	3.677	**0.034**	0.184
	4	0.031	2.238	7.563	0.156	0.189	7.256	2.495	**0.002**	0.212
	2	0.113	4.187	11.976	0.201	0.286	11.633	4.469	**0.057**	0.262
s	3	**0.021**	3.775	3.899	0.199	0.234	3.661	3.572	0.021	0.255
	4	0.115	3.191	2.107	0.128	0.162	2.040	3.338	**0.050**	0.169
	2	**0.028**	3.317	6.045	0.095	0.133	6.093	2.619	0.028	0.135
m	3	**0.070**	3.618	5.797	0.244	0.312	5.340	4.009	0.074	0.302
	4	0.075	4.222	6.111	0.217	0.268	5.927	4.353	**0.042**	0.279
	5	0.022	3.712	6.022	0.149	0.195	5.752	4.192	**0.026**	0.199
Mean		0.080	3.717	5.994	0.176	0.227	5.778	3.793	0.043	0.229

Table 14
The ARPD of the nine algorithms with $\rho = 30$ on small-scale instance.

		BKBSA	BRILS	gIGA	HDIWO	HDIWOp	IG	ILS	ILS-NR3	TDIWO
	8	0.251	2.620	4.468	0.333	0.333	4.508	2.115	**0.052**	0.333
	12	**0.034**	2.763	4.886	0.243	0.247	4.803	2.832	0.034	0.280
n	16	**0.039**	3.785	6.221	0.128	0.190	5.928	4.219	0.039	0.224
	20	**0.055**	4.160	6.237	0.110	0.172	6.290	5.240	0.055	0.156
	24	**0.074**	4.023	6.558	0.108	0.234	6.504	4.603	0.074	0.192
	2	0.183	5.301	3.883	0.241	0.328	4.085	5.220	**0.104**	0.302
f	3	0.129	3.219	5.953	0.155	0.188	5.741	3.689	**0.045**	0.195
	4	0.032	1.891	7.185	0.157	0.190	6.995	2.496	**0.004**	0.214
	2	0.127	3.780	11.510	0.215	0.300	11.397	4.484	**0.071**	0.276
s	3	0.095	3.668	3.609	0.203	0.238	3.492	3.576	**0.025**	0.259
	4	0.122	2.963	1.903	0.135	0.169	1.930	3.345	**0.056**	0.176
	2	**0.032**	3.193	5.614	0.099	0.137	5.657	2.623	0.032	0.139
m	3	0.123	3.460	5.431	0.261	0.329	5.419	4.026	**0.091**	0.319
	4	**0.039**	3.828	5.869	0.229	0.279	5.631	4.365	0.054	0.290
	5	**0.017**	3.399	5.780	0.150	0.196	5.718	4.193	0.027	0.200
Mean		0.090	3.470	5.674	0.184	0.235	5.607	3.802	0.051	0.237

Table 15
The ARPD of the 11 constructive heuristics.

		H11	H12	H21	H22	H31	H32	Heu1_NRa	Heu2_NRa	Heu3_NRa	Heu3	H2NRa
	100	5.571	6.092	5.821	5.946	5.712	9.439	0.685	**0.032**	**0.017**	33.445	0.697
n	200	12.437	12.711	12.517	12.923	12.546	18.971	0.629	**0.025**	**0.021**	19.079	0.856
	500	19.910	20.013	20.044	20.325	19.933	28.655	0.754	**0.264**	**0.178**	9.328	0.548
	4	18.584	18.724	18.709	18.982	18.585	18.959	0.687	**0.296**	**0.205**	32.587	0.625
f	6	11.286	11.651	11.483	11.760	11.394	19.010	0.729	**0.022**	**0.011**	18.854	0.738
	8	8.228	8.443	8.190	8.453	8.212	19.096	0.652	**0.002**	**0.001**	10.411	0.739
	30	14.875	15.133	14.944	15.331	14.818	22.313	0.873	**0.194**	**0.149**	9.850	0.897
s	40	12.607	12.882	12.725	13.072	12.772	18.844	0.642	**0.069**	**0.040**	19.492	0.653
	50	10.616	10.802	10.713	10.791	10.602	15.907	0.552	**0.057**	**0.027**	32.511	0.552
	5	9.690	9.706	9.729	10.040	9.758	14.599	0.506	**0.028**	**0.017**	6.569	0.552
m	10	12.326	12.633	12.559	12.646	12.335	18.471	0.697	**0.076**	**0.053**	17.167	0.708
	20	16.083	16.478	16.094	16.508	16.098	23.995	0.865	**0.216**	**0.146**	38.117	0.841
Mean		12.699	12.939	12.794	13.065	12.730	19.021	0.689	**0.107**	**0.072**	20.617	0.700

200 is better than that obtained by other comparison algorithms. The box plot representing the five algorithms: HDIWO, HDIWOp, TDIWO, ILS-NR3 and BKBSA are superior to the other algorithms under different instances. However, the fluctuation of box plots of BKBSA and ILS-NR3 are the least. From the comparison results of box graph, it can be concluded that the ARPD of BKBSA in the same test case is generally smaller than that of other comparison

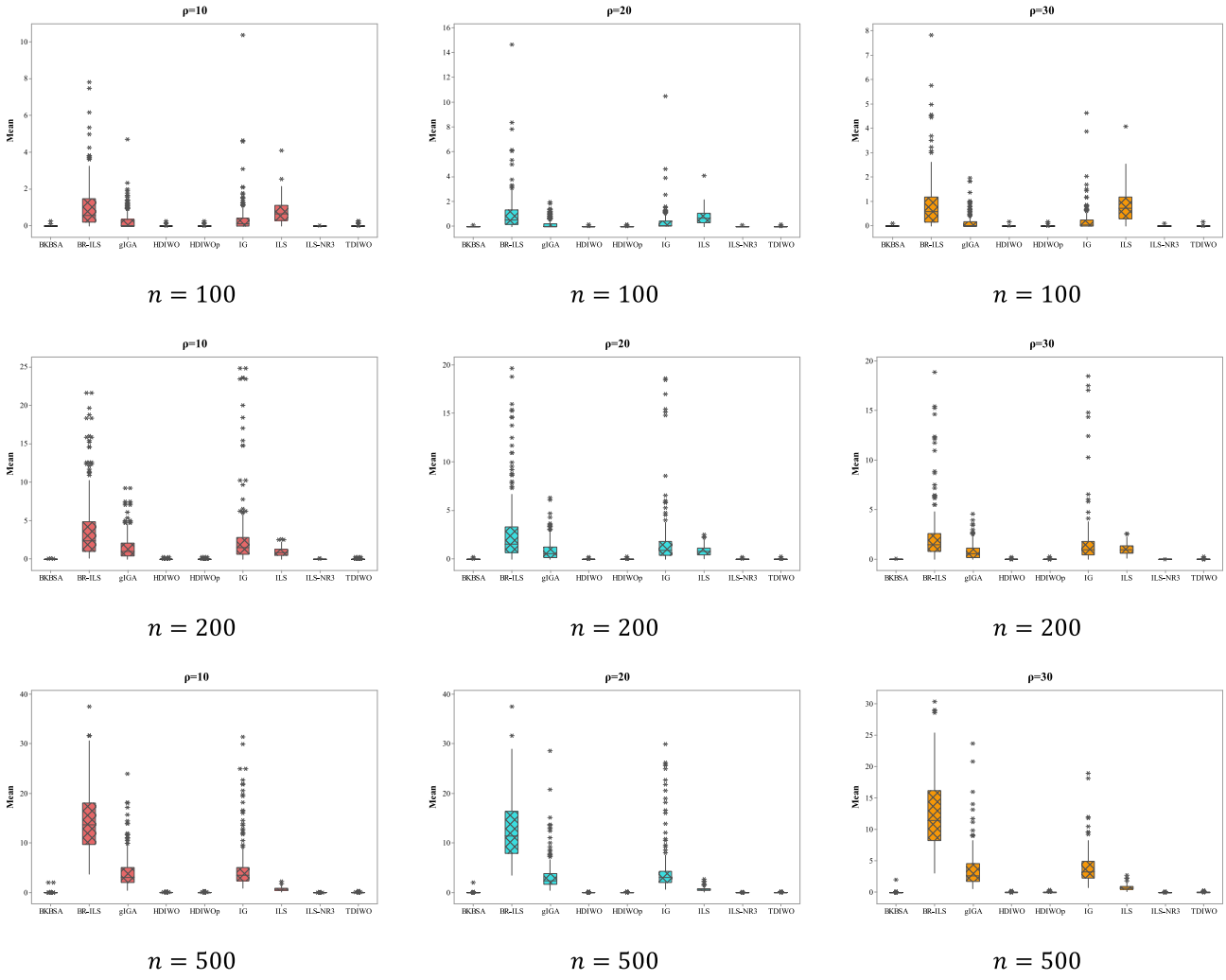


Fig. 19. Box plot of nine algorithms in different stop criteria.

algorithms except ILS-NR3. The above results also show that the BKBSA algorithm has excellent stability.

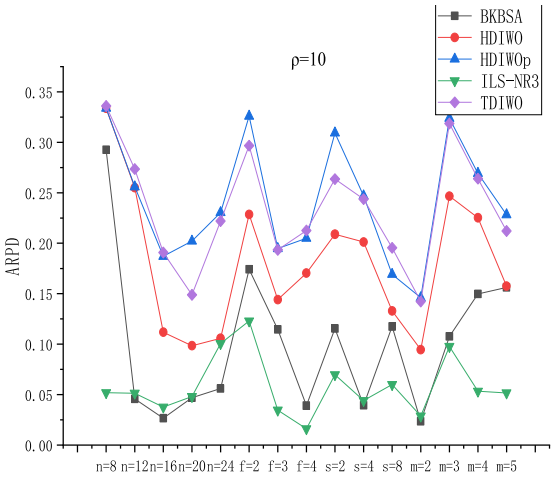
Interval plots of the nine algorithms in different groups are shown in Fig. 21. The interval graph is used to compare whether the experimental results of the seven algorithms have significant differences at a certain confidence interval. It can be seen that the interval graphs of BKBSA and those of IG, BRILS, gIGA and ILS do not overlap in all groups, which also indicates that BKBSA is significantly different from these four algorithms at a confidence interval of 95%. Although there is no significant advantage between BKBSA and HDIWO, HDIWOP, TDIWO and ILS-NR3 in the experimental results, the walking degree of the interval graph of BKBSA is lower than that of the other four comparison algorithms in certain instances. The reason for the above results is that the historical population of BKBSA can cause the algorithm to perform poorly on some large test sets.

In general, BKBSA has advantages in algorithm stability and solving ability compared with other comparison algorithms. There are three reasons for this advantage: construction heuristic generates potential initial solutions for BKBSA, knowledge-based block shifting makes candidate solutions have better diversity and dominant subsequence, and VND further improves the quality of the current optimal candidate solutions.

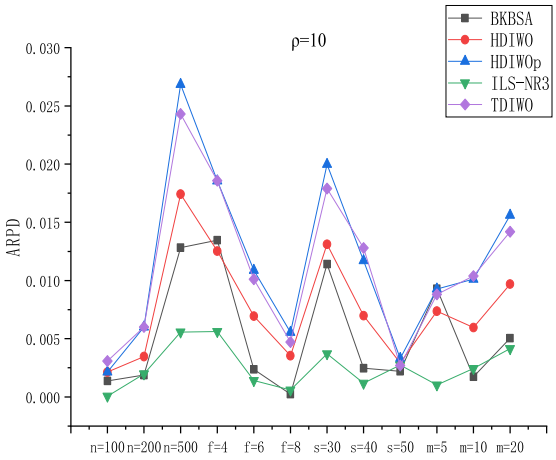
6. Conclusion and future work

In this paper, a backtracking search algorithm (BKBSA) based on job block knowledge is proposed to solve DANWFSP with the maximum assembly completion time $C_{max}(\pi_P)$ as the optimization indicator. In BKBSA, three constructive heuristics are proposed to generate the initial solution, block-shifting based on job block knowledge is proposed as the mutation strategy of BKBSA, and a variable neighborhood descending algorithm (VND) is proposed as a local search method to further improve the optimal candidate solution of BKBSA. The DOE experiment and ANOVA analysis are utilized to calibrate the parameters of BKBSA. The Friedman test and Wilcoxon symbolic Rank test are utilized to analyze the effectiveness of BKBSA and three other comparison algorithms on 810 large instances. The statistical analysis results show that BKBSA is a competitive algorithm to solve DANWFSP. In addition, this paper analyzes the influence of three strategies of BKBSA on algorithm performance. Experimental results show that all three strategies are effective. However, BKBSA also has some disadvantages. For example, although the historical population is introduced in the process of population evolution to ensure diversity, it also slows down the convergence speed of the algorithm, especially in some large-scale instances.

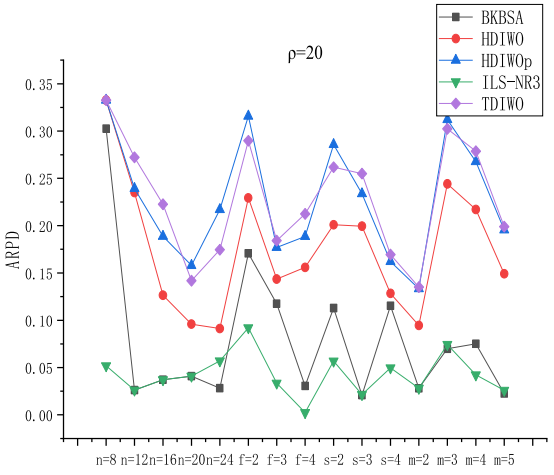
This paper discusses the assembly scheduling problem in an assembly factory. However, in the real world, it is more common to assemble products in multiple assembly factories, which is our future study.



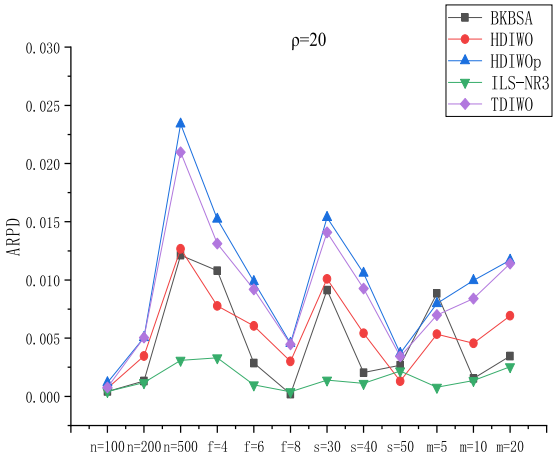
Comparison of five algorithms in the small test set with $\rho = 10$



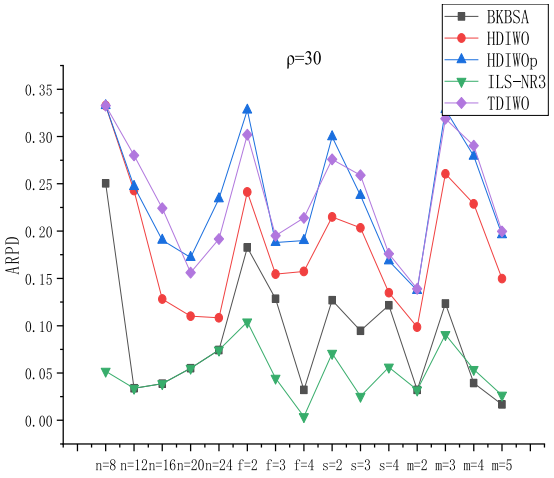
Comparison of five algorithms in the large test set with $\rho = 10$



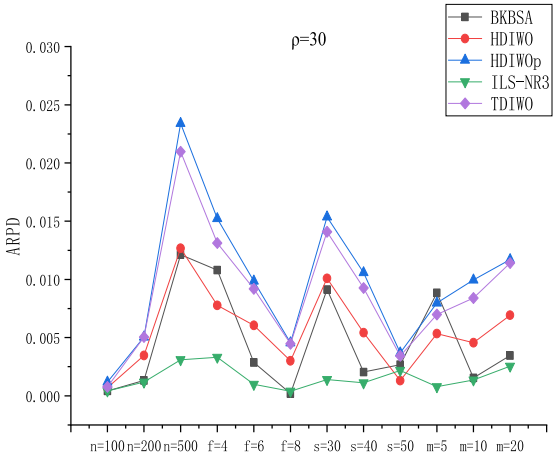
Comparison of five algorithms in the small test set with $\rho = 20$



Comparison of five algorithms in the large test set with $\rho = 20$



Comparison of five algorithms in the small test set with $\rho = 30$



Comparison of five algorithms in the large test set with $\rho = 30$

Fig. 20. Comparison of seven algorithms in the different instances.

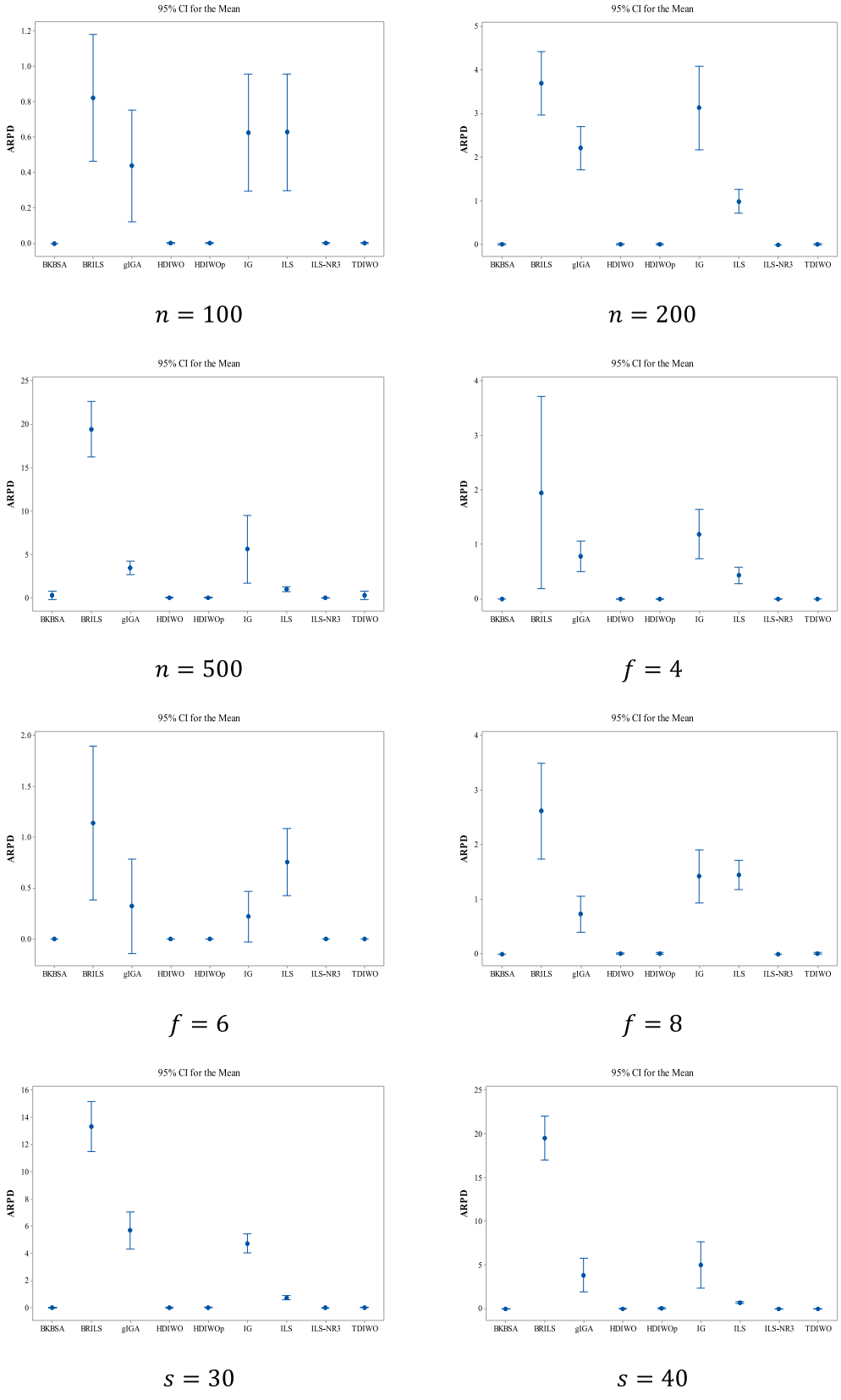


Fig. 21. Interval plot of nine algorithms in the large test set.

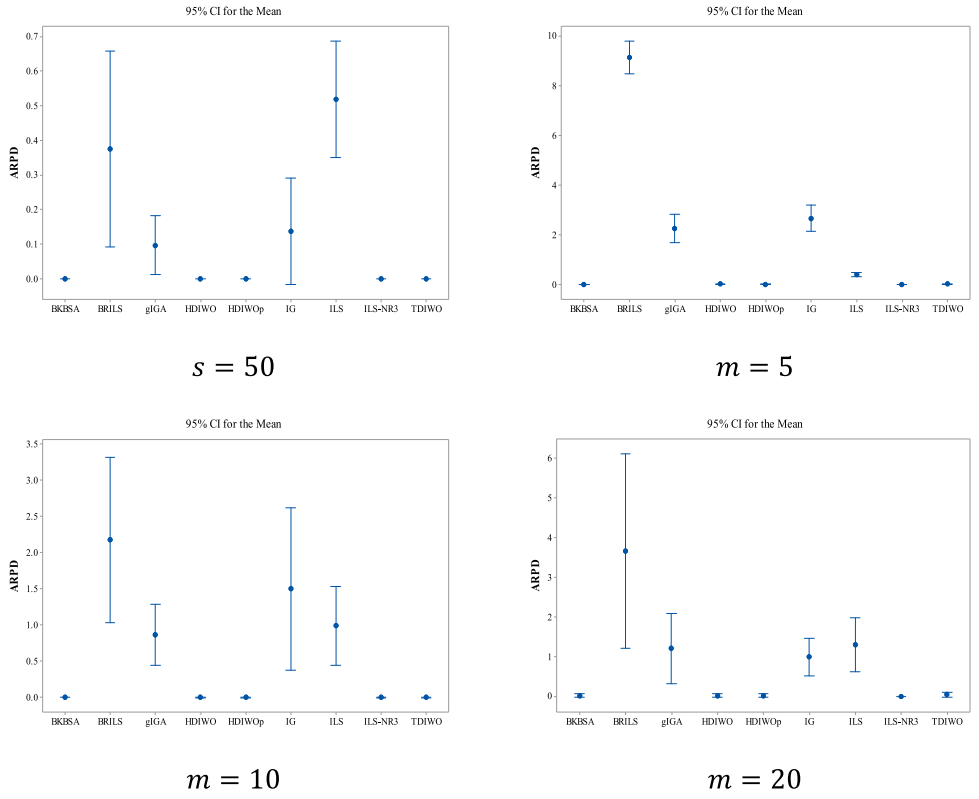


Fig. 21. (*continued*).

CRediT authorship contribution statement

Fuqing Zhao: Funding acquisition, Investigation, Project administration, Supervision. **Jinlong Zhao:** Investigation, Software, Writing – original draft, Experiments of the algorithms. **Ling Wang:** Methodology, Resources. **Jianxin Tang:** Conceptualization, Formal analysis, Writing – review & editing, Visualization.

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.

Acknowledgment

This work was financially supported by the National Key Research and Development Plan under grant number 2020YFB1713600 and the National Natural Science Foundation of China under grant numbers 62063021. It was also supported by the Lanzhou Science Bureau project (2018-rc-98), Public Welfare Project of Zhejiang Natural Science Foundation (LGJ19E050001), respectively.

References

[1] S. Hatami, R. Ruiz, C. Andres-Romano, The distributed assembly permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (2013) 5292–5308.
[2] H.Y. Sang, Q.K. Pan, J.Q. Li, P. Wang, Y.Y. Han, K.Z. Gao, P. Duan, Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion, Swarm Evol. Comput. 44 (2019) 64–73.
[3] M. Adams-Bigelow, Chapter 36, in: First Results from the 2003 Comparative Performance Assessment Study (CPAS), John Wiley & Sons Inc., 2007.
[4] B.H. Abed-Alguni, N.A. Alawad, Distributed Grey Wolf Optimizer for scheduling of workflow applications in cloud environments, Appl. Soft Comput. (2021).
[5] N.A. Alawad, B. Abed-Alguni, Discrete Island-based Cuckoo Search with Highly Disruptive Polynomial Mutation and Opposition-based Learning Strategy for Scheduling of Workflow Applications in Cloud Environments, Arab. J. Sci. Eng., 1–21.
[6] 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 (2014) 1269–1283.
[7] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, European J. Oper. Res. 239 (2014) 323–334.
[8] R. Companys, I. Ribas, Efficient Constructive Algorithms for the distributed blocking flow shop scheduling problem, 2015.
[9] K.-C. Yine, S.-W. Lin, C.-Y. Cheng, C.-D. He, Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems, Comput. Ind. Eng. 110 (2017) 413–423.
[10] S.-W. Lin, K.-C. Ying, Minimizing makespan for solving the distributed no-wait flowshop scheduling problem, Comput. Ind. Eng. 99 (2016) 202–209.
[11] 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.
[12] J.-f. Chen, L. Wang, Z.-p. Peng, A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling, Swarm Evol. Comput. 50 (2019).
[13] J.-J. Wang, L. Wang, A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop, IEEE Trans. Syst. Man Cybern. 50 (2020) 1805–1819.
[14] W. Shao, Z. Shao, D. Pi, Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem, Knowl.-Based Syst. 194 (2020).
[15] Z. Shao, D. Pi, W. Shao, Hybrid enhanced discrete fruit fly optimization algorithm for scheduling blocking flow-shop in distributed environment, Expert Syst. Appl. 145 (2020).
[16] G. Wang, X. Li, L. Gao, P. Li, A multi-objective whale swarm algorithm for energy-efficient distributed permutation flow shop scheduling problem with sequence dependent setup times, IFAC-PapersOnLine, 52, 2019, pp. 235–240.
[17] J. Lin, S. Zhang, An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem, Comput. Ind. Eng. 97 (2016) 128–136.

[18] S.-Y. Wang, L. Wang, An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst. Man Cybern. 46 (2016) 139–149.
[19] Q.-K. Pan, L. Gao, X.-Y. Li, F.M. Jose, Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem, Appl. Soft Comput. 81 (2019).
[20] D. Ferone, S. Hatami, E.M. Gonzalez-Neira, A.A. Juan, P. Festa, A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem, Int. Trans. Oper. Res. 27 (2020) 1368–1391.
[21] W. Shao, D. Pi, Z. Shao, Local search methods for a distributed assembly no-idle flow shop scheduling problem, Ieee Syst. J. 13 (2019) 1945–1956.
[22] Z. Shao, W. Shao, D. Pi, Effective constructive heuristic and metaheuristic for the distributed assembly blocking flow-shop scheduling problem, Appl. Intell. (2020).
[23] P. Civicioglu, Backtracking search optimization algorithm for numerical optimization problems, Appl. Math. Comput. 219 (2013) 8121–8144.
[24] X. Xu, Z. Hu, Q. Su, Z. Xiong, M. Liu, Multi-objective learning backtracking search algorithm for economic emission dispatch problem, Soft Comput. (2020).
[25] X. Yan, Y. Han, X. Gu, An improved discrete backtracking searching algorithm for fuzzy multiproduct multistage scheduling problem, Neurocomputing 398 (2020) 153–165.
[26] Y.-F. Jin, Z.-Y. Yin, Enhancement of backtracking search algorithm for identifying soil parameters, Int. J. Numer. Anal. Methods Geomech. 44 (2020) 1239–1261.
[27] L. Wang, L. Peng, S. Wang, S. Liu, Advanced backtracking search optimization algorithm for a new joint replenishment problem under trade credit with grouping constraint, Appl. Soft Comput. 86 (2020).
[28] Z. Wang, Y.-R. Zeng, S. Wang, L. Wang, Optimizing echo state network with backtracking search optimization algorithm for time series forecasting, Eng. Appl. Artif. Intell. 81 (2019) 117–132.
[29] J. Zhou, H. Ye, X. Ji, W. Deng, An improved backtracking search algorithm for casting heat treatment charge plan problem, J. Intell. Manuf. 30 (2019) 1335–1350.
[30] M.A. Ahandani, A.R. Ghiasi, H. Kharrati, Parameter identification of chaotic systems using a shuffled backtracking search optimization algorithm, Soft Comput. 22 (2018) 8317–8339.
[31] L. Chen, N. Sun, C. Zhou, J. Zhou, Y. Zhou, J. Zhang, Q. Zhou, Flood forecasting based on an improved extreme learning machine model combined with the backtracking search optimization algorithm, Water 10 (2018).
[32] K. Yu, J.J. Liang, B.Y. Qu, Z. Cheng, H. Wang, Multiple learning backtracking search algorithm for estimating parameters of photovoltaic models, Appl. Energy 226 (2018) 408–422.
[33] S. Pare, A.K. Bhandari, A. Kumar, V. Bajaj, Backtracking search algorithm for color image multilevel thresholding, Signal Imag. Video Process. 12 (2018) 385–392.
[34] C. Lu, L. Gao, X. Li, Q. Pan, Q. Wang, Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm, J. Cleaner Prod. 144 (2017) 228–238.
[35] J. Lin, Z.-J. Wang, X. Li, A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem, Swarm Evol. Comput. 36 (2017) 124–135.
[36] F. Zhao, X. He, L. Wang, A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem, IEEE Trans. Cybern. (2020).
[37] J.-Y. Ding, S. Song, J.N.D. Gupta, R. Zhang, R. Chiong, C. Wu, An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem, Appl. Soft Comput. 30 (2015) 604–613.
[38] F.R. Jacobs, Manufacturing Planning and Control for Supply Chain Manageme, 2005.
[39] J.M. Framinan, R. Leisten, An efficient constructive heuristic for flowtime minimisation in permutation flow shops, Omega 31 (2009) 311–317.
[40] J.I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega (1983).
[41] J.-Y. Ding, S. Song, R. Zhang, S. Zhou, C. Wu, A Novel Block-shifting Simulated Annealing Algorithm for the No-wait Flowshop Scheduling Problem, Ieee, 2015.
[42] F. Zhao, L. Zhao, L. Wang, H. Song, An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion, Expert Syst. Appl. 160 (2020).
[43] Y.Y. Huang, et al., An improved iterated greedy algorithm for the distributed assembly permutation flowshop scheduling problem, Comput. Ind. Eng. 152 (3) (2021).