COMPLEX SYSTEM MODELING AND SIMULATION
ISSN 2096-9929 04/06 pp 50−66
Volume 4, Number 1, March 2024
DOI: 10.23919/CSMS.2024.0001

A Hybrid Algorithm Based on Comprehensive Search Mechanisms for Job Shop Scheduling Problem

Lin Huang, Shikui Zhao*, and Yingjie Xiong

Abstract: The research on complex workshop scheduling methods has important academic significance and has wide applications in industrial manufacturing. Aiming at the job shop scheduling problem, a hybrid algorithm based on comprehensive search mechanisms (HACSM) is proposed to optimize the maximum completion time. HACSM combines three search methods with different optimization scales, including fireworks algorithm (FW), extended Akers graphical method (LS1+_AKERS_EXT), and tabu search algorithm (TS). FW realizes global search through information interaction and resource allocation, ensuring the diversity of the population. LS1+_AKERS_EXT realizes compound movement with Akers graphical method, so it has advanced global and local search capabilities. In LS1+_AKERS_EXT, the shortest path is the core of the algorithm, which directly affects the encoding and decoding of scheduling. In order to find the shortest path, an effective node expansion method is designed to improve the node expansion efficiency. In the part of centralized search, TS based on the neighborhood structure is used. Finally, the effectiveness and superiority of HACSM are verified by testing the relevant instances in the literature.

Key words: job shop scheduling; fireworks algorithm; tabu search; Akers graphical; hybrid scheduling algorithms

1 Introduction

As a decision-making process for allocating limited resources rationally, production scheduling aims to optimize key objectives, such as the maximum completion time, total processing cost, and total machine load. In the current market environment, using efficient production scheduling technology can not only make more rapid and scientific response production emergencies, but also significantly improve the production efficiency, thereby affecting the competitiveness of enterprises. Job shop scheduling problem (JSP) is the most basic and famous production scheduling problem, which mainly shows two aspects. Firstly, JSP describes the processing sequence and machine constraints of general jobs in the manufacturing process. Secondly, a series of scheduling problems have been extended based on it, such as distributed, flexible, assembly, and flow shop scheduling problems.

After more than half a century of research, scholars have proposed large number of solution algorithms for JSP, which can be categorized into exact and approximation algorithms. Exact algorithms are also called complete algorithms, including branch and bound method and mathematical programming method[1, 2]. Due to its limitations, the exact algorithms were only employed to solve small-scale problems. Approximation algorithms include constructive algorithms[3] and heuristic algorithms. Owing to the unique advantages in solving JSP of heuristic algorithm in solving the problem, scholars were attracted to study it. The achievements were mainly divided into three categories. The first is swarm intelligence algorithms, including genetic, artificial bee colony, ant colony, and

• Lin Huang, Shikui Zhao, and Yingjie Xiong are with the School of Mechanical Engineering, University of Jinan, Jinan 250022, China. E-mail: 670918850@qq.com; me_zhaosk@ujn.edu.cn; 907451779@qq.com.
* To whom correspondence should be addressed.
※ This article was recommended by Associate Editor Xinyu Li.
Manuscript received: 2023-12-07; revised: 2024-02-20; accepted: 2024-03-12

particle swarm[4, 5]. This type of algorithm typically generates multiple individuals. During the optimization process, individuals collaborate with each other and have strong global optimization capabilities. The second is local search algorithm, among which tabu search (TS) algorithm is a typical representative. TS adopts a flexible search method to tabu solutions that have already been searched, avoiding getting stuck in local optima[6−8]. The last is hybrid algorithm[9−11]. This type of algorithm can usually obtain higher quality solutions than individual algorithms, thus gaining widespread attention.

Some hybrid algorithms for solving JSP since 2010 are listed in Table 1. Scholars have tried to mix different types of search strategies, especially those with complementary advantages. For example, Zhang et al.[41] used the simulated annealing formula to store the suboptimal solution in the elite table in the TS search process, thereby implementing a backtracking search function. Gonçalves and Resende[23] extended the Akers graphical and combined it with genetic algorithm and TS to refresh the record of 57 optimal solutions. Peng et al.[27] mixed path reconnection and TS to solve JSP problems and improved the optimal solutions of 49 benchmark examples. Zhao[38] studied a decoding method based on non-delay scheduling. Subsequently, he proposed forward non-delay scheduling conversion algorithms based on head length and backward non-delay scheduling conversion algorithms based on tail length. Xie et al.[39] proposed the N8 neighborhood structure, which combines genetic algorithm with TS-based optimization to further update the optimal solutions of two examples. Yuan et al.[42] proposed a reinforcement learning framework for flexible JSP. The new framework uses lightweight multi-layer perceptrons as state embedding networks to extract state information, which to some extent reduces the computational complexity of the algorithm. Gui et al.[43] investigated the necessary and sufficient conditions for feasible solutions to JSP local search problems. Zhang et al.[44] proposed a multi-agent manufacturing system based on reinforcement learning for dynamic JSP. The system efficiently completes personalized orders through self-organization and self-learning strategies. He et al.[45] proposed a discrete multi-objective fireworks algorithm (FW) for multi-objective flow shop scheduling problems. Pang et al.[46] proposed an improved FW for the permutation flow shop scheduling problem and the hybrid flow shop scheduling problem. They introduced a nonlinear radius in FW to avoid resource waste and designed a mixed mutation operator to improve search ability.

Table 1 Statistics of JSP hybrid algorithm since 2010.

Algorithm	Author	Year	Reference
DDE	Pan et al.	2010	[12]
IABC	Yao et al.	2010	[13]
HPGA	Yusof et al.	2011	[14]
HSS	Sels et al.	2011	[15]
MA	Gao et al.	2011	[16]
AISTS	Zuo et al.	2012	[17]
HEA	Qing-dao-er-ji and Wang	2012	[5]
DESG	Wisittipanich and Kachitvichyanukul	2012	[18]
TSPR	Nasiri and Kianfar	2012	[9]
GES/TS	Nasiri and Kianfar	2012	[10]
ABC	Banharnsakun et al.	2012	[19]
DE-TS	Ponsich and Coello	2013	[20]
ABC	Zhang et al.	2013	[21]
GA/PR	Spanos et al.	2014	[22]
BRKGA	Gonçalves and Resende	2014	[23]
HBBO	Wang and Duan	2014	[24]
GATS	Meeran and Morshed	2014	[25]
CEBFO	Zhao et al.	2014	[26]
TS/PR	Peng et al.	2015	[27]
HPV	Gao et al.	2015	[8]
aLSGA	Asadzadeh	2015	[28]
HIMGA	Kurdi	2015	[29]
NS-HDE/EDA	Zhao et al.	2015	[30]
HPSOSA	Toader	2015	[31]
HEA	Cheng et al.	2016	[32]
GALS	Zhao	2016	[33]
IEBO	Nagata and Ono	2018	[34]
MAGATS	Peng	2019	[35]
SSODM	Zhou et al.	2021	[36]
WOA-LFDE	Liu et al.	2020	[37]
PRTS	Zhao	2021	[38]
MCDE/TS	Mahmud et al.	2021	[11]
HA	Xie et al.	2022	[39]
OGM/TS	Huang et al.	2023	[40]

The design of hybrid algorithms generally requires achieving a balance between global search and centralized search. If the algorithm only has strong centralized search ability or global search ability, it is easy to fall into local optima or divergence. Currently, TS has been proven to have strong local search capabilities, but there are many options available in the global search stage. We used FW and LS1+_AKERS_EXT in the global search process. The two algorithms complement each other's advantages,

thereby enhancing the global search capability. In this paper, we design a hybrid algorithm based on comprehensive search mechanisms (HACSM) to solve JSP. HACSM combines three algorithms, including FW, LS1+_AKERS_EXT, and TS. FW achieves global search through fireworks explosions and Gaussian mutation, thereby ensuring population diversity. TS combines the designed N7 neighborhood structure to conduct centralized search to achieve the purpose of enhanced search[47]. Although LS1+_AKERS_EXT moves multiple jobs each time (means compound move), it has global and local search capabilities with the help of Akers graphical method. In LS1+_AKERS_EXT, the optimization method of scheduling is closely related to the path of Akers graph. Therefore, high requirements need to be placed on path length and calculation time[23]. Because LS1+_AKERS_EXT is a new optimization method and the path search is time-consuming, few scholars have studied it. Aiming at solving the shortest path problem, this paper designs an effective path search algorithm. The algorithm first layers the obstacles in the Akers graphical according to the machine. When a node is expanded, the new node expansion method can avoid some invalid child nodes, thereby achieving fast path search.

The remaining parts are as follows. Section 2 provides an example of JSP. Section 3 introduces FW, including encoding, decoding, fireworks explosion, explosion spark, and Gaussian variation spark. Section 4 introduces LS1+_AKERS_EXT, including Akers graphical method for two-job, LS1+_AKERS_EXT, and path planning algorithm. Section 5 introduces the hybrid algorithm. Section 6 gives the test results and analysis of the algorithm. Section 7 is the conclusion of the paper.

2 Problem Description

JSP can be defined as follows: n jobs are machined on m machines, and each job consists of m consecutive operations. The objective function of scheduling is to minimize the time required to process all jobs, denoted as $C_{\max}$. The objective function of JSP is shown in Eq. (1). Formula (2) provides a JSP mathematical model, where o, P_k, S_{P_k}, and E_h represent the total number of operations, processing time, start time of the previous operation, and the relationship between adjacent processes on the machine, respectively. In Formula (2), the first constraint indicates that the start time of all operations is greater than or equal to 0. The second constraint stipulates the precedence relations among the operations of the same job. The third constraint indicates that each machine can only process one job at a time.

$$C_{\max} = \min\left(\max_{i=1}^{n} C_i\right) \tag{1}$$

$$\begin{aligned} &S_k \geqslant 0,\ k = 1,2,...,o; \\ &S_k - S_{P_k} \geqslant P_{P_k},\ k = 1,2,...,o; \\ &S_i - S_j \geqslant P_{i-1} \text{ or } S_j - S_i \geqslant P_{i-1},\ (i,j) \in E_h, h \in m \end{aligned} \tag{2}$$

In order to describe the JSP, a 4×4 JSP example is constructed in this section (as shown in Table 2). The following examples are all around it. Figure 1 shows the scheduling Gantt chart with $C_{\max}$=21.

3 Fireworks Algorithm

In FW, each firework is considered as a solution. Among all solutions, the solution with good fitness has a small search radius and generates multiple neighborhood solutions. The solution for poor fitness values is exactly the opposite. The solution for poor fitness is exactly the opposite. FW balances global and local search by allocating resources and exchanging information based on each firework's fitness value. Gaussian variation is used to ensure population diversity during the search process. The steps are as follows:

Step 1: Randomly generate X solutions (fireworks) and calculate their fitness values.

Step 2: Calculate the search radius and number of neighborhood solutions for each solution according to the fitness value, and generate neighborhood solutions.

Table 2 Problem data for four-job, four-machine example.

Sequence order No.	J_1		J_2		J_3		J_4	
	Machine	Processing time	Machine	Processing time	Machine	Processing time	Machine	Processing time
1	M_2	3	M_1	2	M_3	5	M_1	2
2	M_3	2	M_2	3	M_4	2	M_4	4
3	M_4	3	M_4	4	M_1	1	M_3	2
4	M_1	4	M_3	2	M_2	4	M_2	3

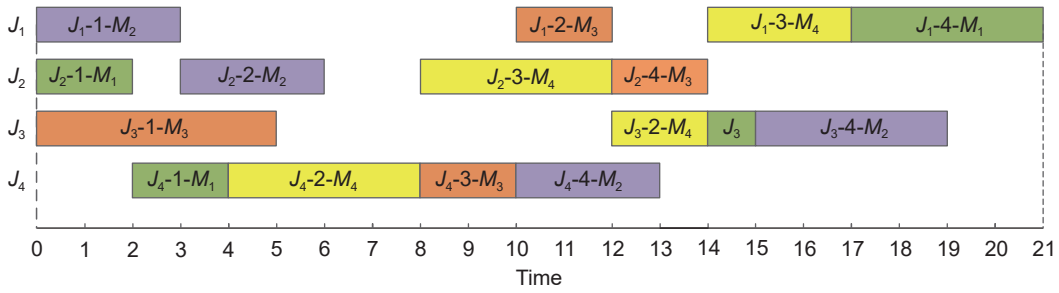


Fig. 1 Schedule Gantt chart (C_{max}=21).

Step 3: Generate mutation solution using Gaussian formula based on mutation probability.

Step 4: Check if the termination conditions are met. If so, output the optimal value and stop the calculation. Otherwise, execute Step 5.

Step 5: Based on the selection rules, identify a new population and return to Step 2.

3.1 Fireworks explosion

The population size of FW is denoted as X, where the i-th solutions are expressed as X_i and the fitness value is f_i. The neighborhood solutions are generated within the explosion radius according to the fitness value of each solution. Equations (3) and (4) represent the search radius of X_i and the number of neighborhood solutions, respectively. In order to ensure that the search radius of each solution and the number of neighborhood solutions generated are integers, Eqs. (3) and (4) shall be rounded, respectively.

$$R_i = R_{\max} \times \frac{f_i - f_{\min} + \varepsilon}{\sum_{i=1}^{X} (f_i - f_{\min}) + \varepsilon} \tag{3}$$

$$S_i = S_{\max} \times \frac{f_{\max} - f_i + \varepsilon}{\sum_{i=1}^{X} (f_{\max} - f_i) + \varepsilon} \tag{4}$$

where $R_{\max}$ is the maximum search radius, $S_{\max}$ is the maximum number of neighborhood solutions, $f_{\min}$ and $f_{\max}$ are the minimum and maximum fitness values, respectively, and ε is a small quantity, which is used to avoid the denominator being zero.

In order to allocate resources reasonably, solutions with good fitness (poor fitness) should not generate too many (too few) neighborhood solutions. Therefore, the number of neighborhood solutions for X_i is limited by Eq. (5), where round is the rounding function. Generally, a = 0.1 and b = 0.3. At the same time, in order to avoid the explosion radius with good fitness value being 0, the radius of X_i is limited by Eq. (6).

$$S_i = \begin{cases} \text{round}(a \times S_{\max}), S_i \leqslant a \times S_{\max}; \\ \text{round}(b \times S_{\max}), S_i \geqslant b \times S_{\max}; \\ S_i, \text{else} \end{cases} \tag{5}$$

$$R_i = \begin{cases} 1, R_i = 0; \\ R_i, \text{else} \end{cases} \tag{6}$$

3.2 Explosive spark and Gaussian variation spark

According to the search radius and the number of neighborhood solutions calculated, X_i generates neighborhood solutions within the radius R_i. The process is following: Z positions are selected randomly and each position $z \in \{1, 2, \ldots, Z\}$ is offset according to Eq. (7) to obtain a new position N_z.

$$N_z = z + \text{randi}[-R_i, R_i] \tag{7}$$

where randi[$-R_i$, R_i] represents the uniformly distributed integer in [$-R_i$, R_i].

To ensure diversity, Gaussian mutation is added to the algorithm, as shown in Eq. (8), namely, all solutions carry out additional Gaussian mutation with a probability of 0.1.

$$N_z = \text{round}(z \times G(1,1)) \tag{8}$$

where G(1, 1) represents the Gaussian distribution function with mean value and variance of 1.

If the new position generated by Eq. (7) or Eq. (8) exceeds the moving range, it needs to be returned to the search space according to the mapping rules. The mapping rules are as shown in Eq. (9).

$$N_z = N_z^{\min} + \text{mod}(|N_z|, N_z^{\max} - N_z^{\min}) \tag{9}$$

where $N_z^{\max}$ and $N_z^{\min}$ are the upper and lower bounds of the value range of the z-th position of solutions, respectively, and "mod" is the remainder function.

4 Search Algorithm Based on Akers Diagram

4.1 Akers graphical method

Akers[48] first combined graphical methods with two-

job JSP problems. Taking the jobs J_1 and J_2 in Table 2 as examples, Fig. 2 shows the two-job Akers graphical method. The vertical axis and horizontal axis of the coordinate system in Fig. 2 are composed of jobs J_2 and J_1, respectively. The obstacle diagram is composed of obstacle blocks and paths, where M_1 represents the obstacle blocks generated by machine M_1, the red path is the shortest path. Below Fig. 2 are the corresponding schedules obtained by path decoding. The rectangular block in Fig. 2 is called an obstacle block, which represents a machine conflict. It is generated by overlapping areas projected horizontally and vertically by processes of the same machine on two axes. The Akers diagram transforms the scheduling problem into solving the shortest path problem.

Gonçalves and Resende[23] expanded the number of jobs on basis of the Akers two-job graphical method and proposed LS1+_AKERS_EXT. The neighborhood solutions were obtained by removing and adding some jobs in the scheduling. The steps for LS1+_AKERS_EXT are as follows:

Step 1: Input the scheduling data CurSch, and remove some jobs from CurSch. Put them into the job set Js{}, and move the remaining operations to the left to generate scheduling PartSch.

Step 2: Place PartSch below the horizontal axis of the Akers diagram. Take out a job from Js{} and put it on the left side of the vertical axis. If the horizontal axis operations and the vertical operations use the same machine, a rectangular obstacle is generated in the Akers diagram.

Step 3: According to the path rules, find the shortest path L from the Akers diagram.

Step 4: The path L is encoded according to the order in which it passes through the operations, and obtain the new scheduling NewSch after decoding.

Step 5: If Js{} is empty, output NewSch and the program ends. Otherwise, PartSch←NewSch, return to Step 2.

Based on the above steps, take the scheduling in Fig. 1 as an example, we remove jobs J_2 and J_4 from the scheduling and move the operations of the remaining jobs to the left. Figure 3 shows the process of adding J_2 back in the remaining operations. The vertical axis of the coordinate system in Fig. 3 is composed of job J_2, and the horizontal axis is composed of the new schedule obtained by left shift of remaining operations in Fig. 1 after the jobs J_2 and J_4 are removed from Fig. 1. The corresponding schedule after decoding the shortest path is shown at the bottom of Fig. 3. Then, Fig. 4 shows the process of adding J_4 back. The vertical axis of the coordinate system in Fig. 4 is

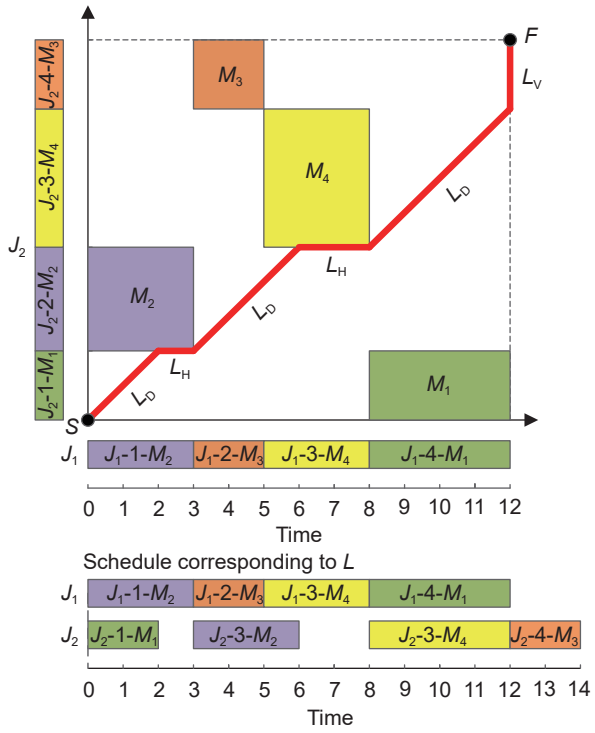


Fig. 2 Two-job Akers graphical method.

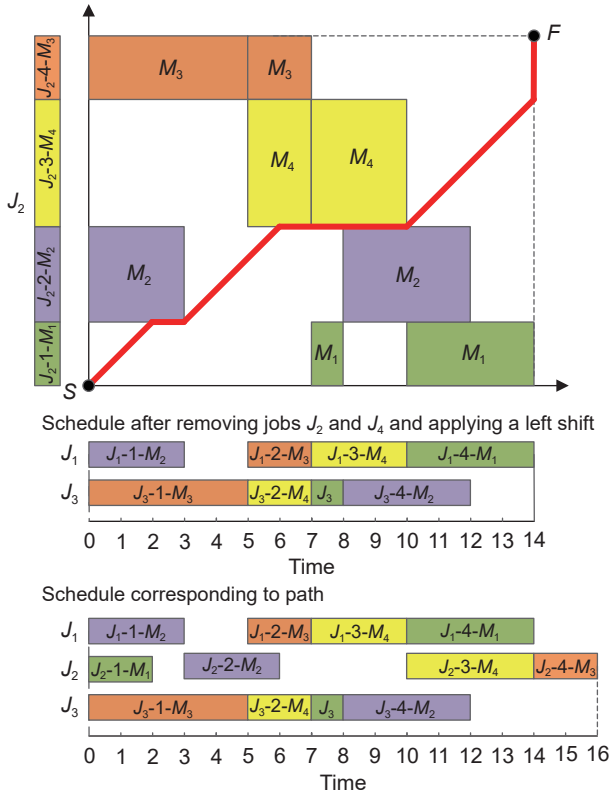


Fig. 3 Schedule after adding back job J_2.

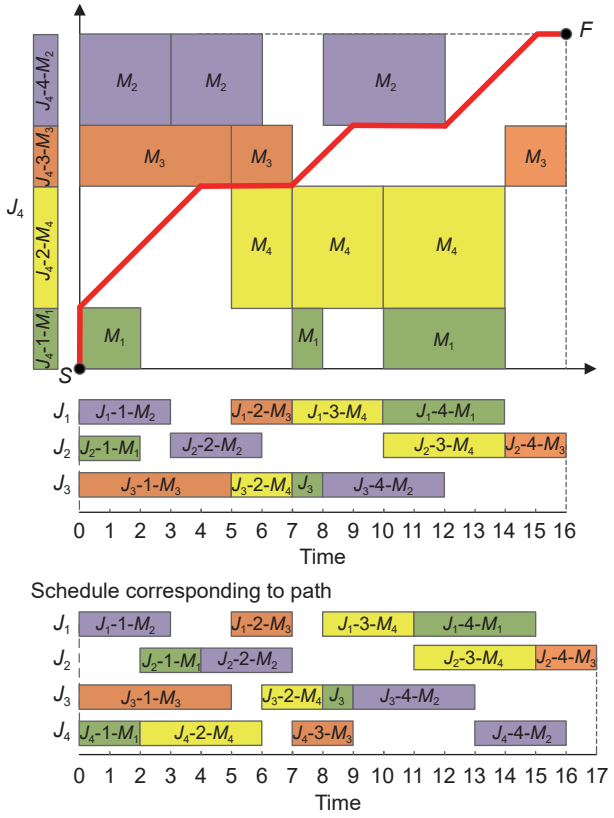


Fig. 4 Schedule after adding back job J_4.

composed of job J_4, the horizontal axis is composed of the scheduling below Fig. 3, and the scheduling corresponding to the decoding of the shortest path is shown below Fig. 4. It is worth noting that because the processing scheduling corresponding to the multi-job Akers diagram path will have operation delay, the semi-active decoding is carried out according to the sequence of the process of the path, the left shift of the scheduling is omitted, and finally the $C_{\max}$ is reduced from 21 to 17 of the original scheduling.

In the process of optimizing the scheduling, if a large number of jobs are deleted, the time cost of adding the job will increase, and the essence of the original encoding cannot be retained. If the number of jobs removed is small, some important neighborhood solutions may be omitted, so that the desired optimization effect will not be obtained. Referring to LS1+_AKERS_EXT method of removing two jobs at a time, the job movement mode has been redesigned. The algorithm ensures that all jobs are removed at least once at the least cost, and produces a small number of neighborhood solutions. The implementation is divided into the following steps:

Step 1: All jobs in scheduling CurSch are randomly grouped in pairs. Gs{} is the set of all groups. If the total number of jobs is odd, one job is selected randomly to combine with the remaining job.

Step 2: Take a set of jobs from Gs{} and remove them from CurSch.

Step 3: Use the twice obstacle diagram to add the removed jobs back and generate a new solution NewSch. Store NewSch in the solution set $S_{ET}\{\}$.

Step 4: If Gs{} is empty, $S_{ET}\{\}$ is output and the algorithm ends; otherwise, return to Step 2.

4.2 Path search algorithm for Akers diagram

A path search algorithm is proposed by Brucker[49] for two-job Akers graph. This algorithm utilizes the interval of obstacles in the vertical direction to improve searching efficiency (as shown in Fig. 5a). With the proposal of LS1+_AKERS_EXT, the number of obstacles in the horizontal direction is greatly increased, leading to a high cost of the path search algorithm proposed by Brucker[49]. After conducting calculations, Gonçalves and Resende[23] found that the time cost of LS1+_AKERS_EXT in the hybrid algorithm (BRKGA) exceeds 90%. Additionally, they pointed out that if a path contains horizontally left-facing segments, that path is invalid (as shown in Fig. 5b). *u* in Fig. 5 presents the parent node, NW and SE are the northwest and Southeast corners of the obstacle, respectively, and *F* is the end point. In Fig. 5a, NW and SE are two child nodes obtained after *u* expands, and the red solid line and dotted line are the paths to the child nodes. The grey path in Fig. 5b

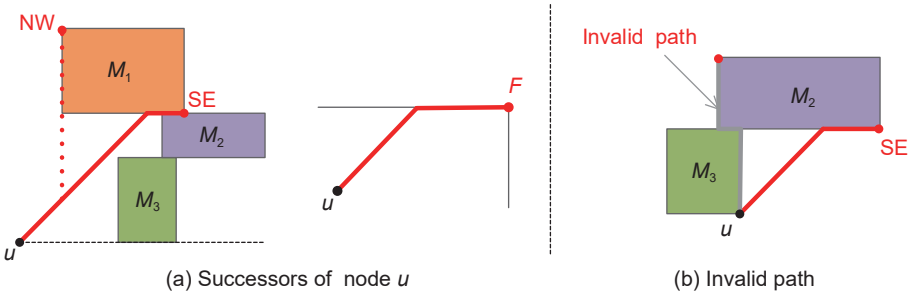


Fig. 5 Traditional node extension method.

produces a horizontal left section, which is invalid.

In order to develop more effective algorithm, the algorithm proposed by Brucker[49] is analyzed. During the nodes extend progress, Brucker's algorithm ends current iteration when reaches an obstacle. However, there are a large number of obstacles in the graph of LS1+_AKERS_EXT algorithm. That increases the number of iterative searches. To solve this problem, we developed a new node expansion method. The new node expansion method screens subsequent nodes based on the distance between obstacles. If the distance between obstacles increases, subsequent nodes will be generated. As shown in Fig. 6, there are three obstacles M_3-1, M_3-2, and M_3-3. In Fig. 6a, a traditional node expansion method was used for search, which expanded a total of three times. In Fig. 6b, the new node expansion method used the size of the gap between obstacles to guide the search, and only expanded once. Compared to the traditional method, the new node expansion method reduced the number of expansions. In order to filter out effective paths from the graph, the Brucker's node extension method needs to search three times, while the new node extension method only needs to search once. Due to the new node expansion method utilizing the distance relationship between obstacles to avoid some invalid nodes, the efficiency of node expansion has been improved. Taking Fig. 6a as an example, compared to child node v_2, the child node v_1 is superior to v_2 in both the subsequent search range and the current optimal

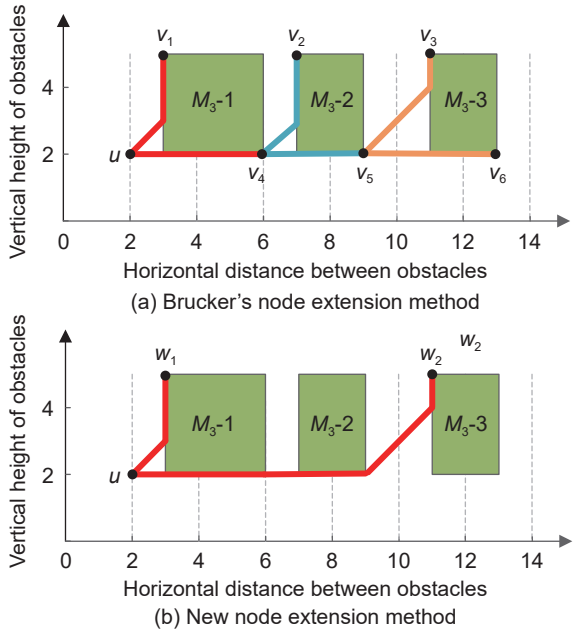


Fig. 6 Comparison of two node expansion methods.

distance. Therefore, v_2 is referred to as an invalid child node. Obviously, in the subsequent node expansion, efforts should be made to avoid searching for invalid child nodes. Finally, the new node expansion method is integrated into the A* algorithm[50] to construct a complete path search algorithm.

The design of the path search algorithm includes a distance evaluation formula. In the algorithm proposed in this paper, only the delay distance of the path is calculated, and the diagonal distance is not calculated. When a node is expanded, select the node with the best evaluation value from the list to be expanded each time. The expansion process iterates back and forth until the node to be expanded reaches its endpoint, and the algorithm stops and backtracks to the optimal path. Table 3 presents the path search data.

5 Hybrid Algorithm

5.1 Algorithm framework

Figure 7 shows the flow chart of HACSM proposed in this paper. HACSM is mainly divided into three optimization stages: global search, local search, and population update strategy. The global search adopts FW and LS1+_AKERS_EXT, respectively, achieving operation-level and job-level search. The centralized search part is the main optimization phase to the solution. The population update strategy ensures the diversity of the population in each iterative search.

5.2 Encoding and decoding

HACSM uses operation-based coding method. It is characterized in that any arrangement and combination of strings can represent feasible scheduling. Taking Fig. 1 as an example, the encoding is [1 2 4 3 4 2 2 3 4 1 4 3 2 1 3 1]. Each job sequence number appears four times in the encoding. The number repeatedly for the k-th time indicates the k-th operation of this job. The semi-active decoding method is adopted, namely scanning coding from left to right, and processing each

Table 3 Node search dataset.

Number of iterations	Parent node	Child node	Evaluation value
1	(0, 0)	(0, 2)	9
		(4, 2)	7
2	(4, 2)	(5, 6)	11
		(16, 6)	14
3	(0, 2)	(4, 6)	9
4	(4, 6)	(5, 8)	13
		(9, 8)	9
5	(9, 8)	(16, 11)	9

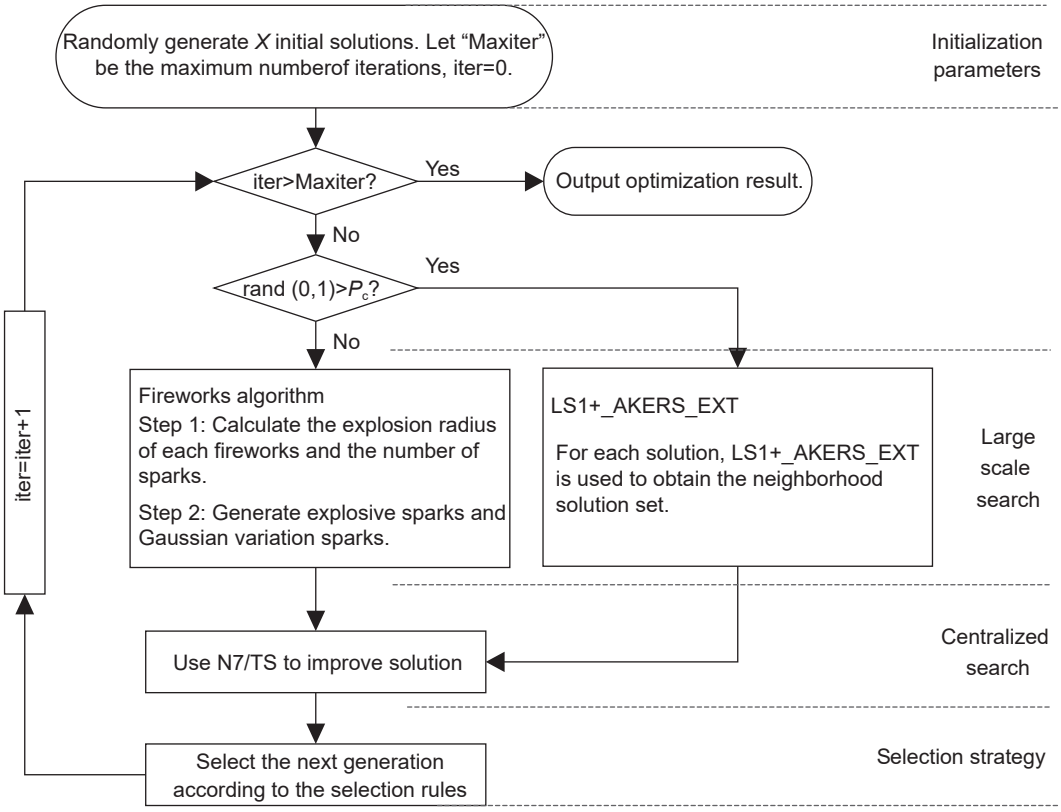


Fig. 7 Flow chart of HACSM.

operation according to the earliest start time, as shown in Fig. 8.

5.3 Global search design

The global search section adopts FW and LS1+_AKERS_EXT. In FW, each solution allocates resources based on the evaluation value, thereby determining the number of neighborhood solutions and the scope of operations search. In LS1+_AKERS_EXT, the optimization of job-level is realized by moving and adding jobs. The above two algorithms are large-scale optimization method of scheduling problem. Parameter P_c controls the weight of FW and LS1+_AKERS_EXT. When P_c takes 1, HACSM becomes a hybrid algorithm of FW and TS. When P_c takes 0, HACSM becomes a hybrid algorithm of LS1+_AKERS_EXT and TS. When P_c is between 0 and 1, HACSM becomes a hybrid algorithm with comprehensive search mechanisms.

5.4 Tabu search design

For the solutions obtained using the global search strategy, the next step is to use TS based on the N7 neighborhood structure (N7/TS) for centralized search[47] (as shown in Fig. 9). For the candidate solutions generated by the neighborhood structure, the approximate evaluation method is used for evaluation[51]. The aspiration criterion solution with better evaluation value is preferred, followed by the non tabu solution with better evaluation value. The function of the tabu list is to avoid repeating searches, and we adopt the partial feature mechanism of the tabu solution[47]. The length of the tabu list is usually related to the size of the problem. The setting of N7/TS output solution is as follows: If the solution is improved during the iteration of the algorithm, the optimal improved solution will be output. Otherwise, an excellent solution is selected from the current

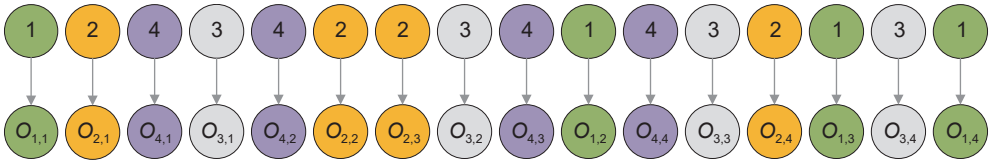


Fig. 8 Coding method.

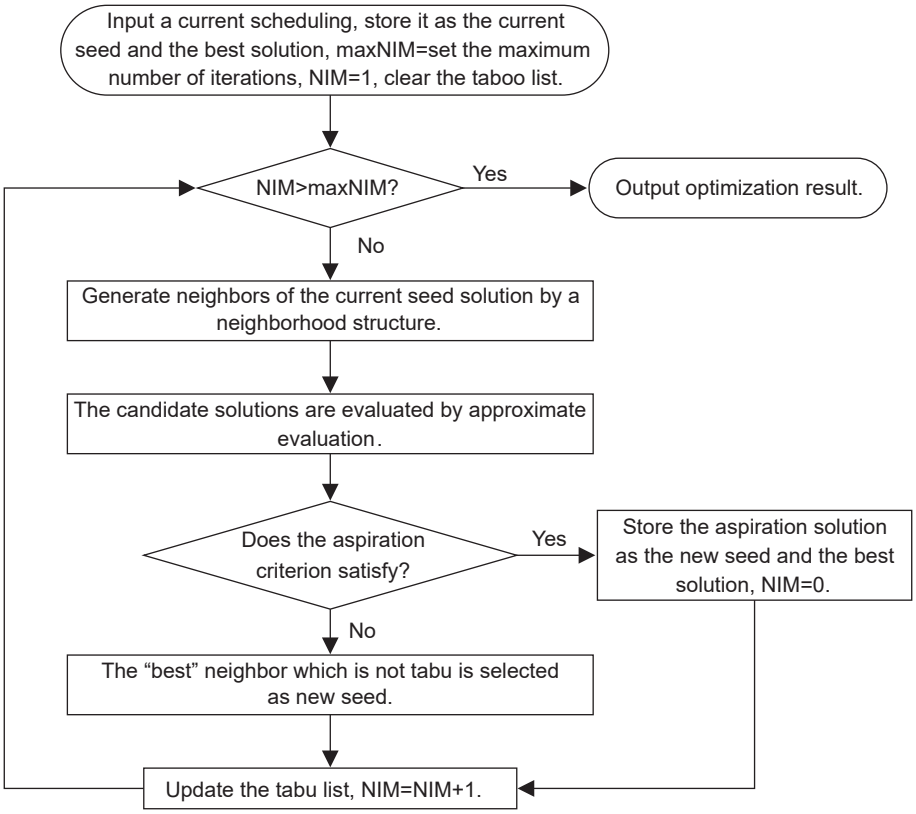


Fig. 9 Flow chart of N7/TS.

neighborhood to output.

5.5 Population iteration and termination criteria

After each iteration, the new population will be selected from the neighborhood solutions of both global and local searches. Firstly, in the global search phase, it is necessary to select an algorithm from FW and LS1+_AKERS_EXT. If FW is chosen, it generates neighborhood solutions through fireworks explosions and Gaussian variation. Otherwise, LS1+_AKERS_EXT conducts independent searches for each individual, and the number of neighborhood solutions is related to the number of jobs. Secondly, N7/TS is used for centralized search on the neighborhood solutions obtained from the global search process. The selection strategy is to select the optimal individual from all neighborhood solutions, and the remaining $X - 1$ individuals are selected through a roulette wheel strategy.

There are three commonly used termination criteria for algorithms: (1) given the maximum number of iteration steps; (2) given the optimal solution or estimated optimal solution of the example; and (3) the maximum running time. HACSM uses termination criterion (1).

5.6 Algorithm complexity analysis

From the optimization process of HACSM, it can be seen that each iteration consists of three parts: global search, local search, and population update. For global search, use FW with P_c probability and LS1+_AKERS_EXT with $1 - P_c$ probability. The computational complexity of FW is $O(1.1 \times P_c \times S_{\max})$, where 1.1 is the neighborhood solution coefficient generated by fireworks explosion and Gaussian variation, and $S_{\max}$ represents the total number of neighborhood solutions. The complexity of LS1+_AKERS_EXT calculation is $O((1 - P_c) \times R \times N \times X)$, where $O(\cdot)$ represents the computational complexity of a single path search for the obstacle map, N represents the number of jobs in the example, and X represents the population size. For local search, the computational complexity is O(maxNIM + xNIM), where maxNIM is the number of consecutive times the optimal solution is not updated, and xNIM is the additional number of searches after updating the optimal solution. The computational complexity of population update is $O(X)$. The final total computational complexity for each generation is $O[((1.1 \times P_c \times S_{\max}) + (1 - P_c) \times R \times N \times X) \times (\text{maxNIM} + \text{xNIM}) + X]$, which can be simplified as

$O[(S_{max} + r \times N \times X) \times \text{maxNIM}]$, it can be seen that the computational complexity of HACSM mainly depends on population size, path search, number of jobs, and local search times.

6 Experimental Test Result and Analysis

In this section, we tested the performance of HACSM. The maximum number of iterations is 3000. The main frequency of the tested computer CPU was 3.2 GHz, and the memory was 16 GB. MATLAB (2020b) programming was used for implementation.

6.1 Benchmark instance and algorithm

We tested HACSM using 60 JSP benchmark instances, including FT06, 10, 20 (3 instances), LA01−40 (40 instances), ABZ07−09 (3 instances), YN01−04 (4 instances), and ORB01−10 (10 instances), with problem sizes ranging from 6×6 to 20×20.

We will compare the test results with the best algorithms in the literature, namely:

- HA (Xie et al.[39], 2022).
- PRTS (Zhao[38], 2021).
- BeFABC (Sharma et al.[7], 2018).
- TS/PR (Peng et al.[27], 2015).
- HIMGA (Kurdi[29], 2015).
- BRKGA (Gonçalves and Resende[23], 2014).
- TS/SA (Zhang et al.[41], 2008).

6.2 Selection of parameters

The HACSM algorithm consists of 7 basic parameters: algorithm selection P_c, population size X, the maximum explosion radius R_{max}, neighborhood solution quantity S_{max}, number of operations moves Z, tabu table length L, and the maximum number of unimproved iterations maxNIM. The values of L and maxNIM are referenced in Ref. [17]. The values of the remaining five parameters are determined using parameter testing. Table 4 lists the reasonable range of values for five parameters. An LA40 example with significant difficulty in solving was selected from the LA dataset to adjust the parameters. Each example runs independently 10 times, with a maximum running time of 5 min each time. Table 5 lists 27 different parameter combinations. Figure 10 shows the trend of the impact of these 5 parameters. From Fig. 10, it can be seen that the optimal parameter combination is: P_c=0.5, X=5, R_{max}=30, S_{max}=25, and Z=0.6.

Table 4 Parameter value range.

Parameter	Value		
	Level 1	Level 2	Level 3
P_c	0.3	0.5	0.7
X	5	10	15
R_{max}	20	30	40
S_{max}	15	20	25
Z	0.04	0.06	0.08

Table 5 Full factor test table and response values.

Experimental combination No.	Parameter					Response value
	P_c	X	R_{max}	S_{max}	Z	
1	0.3	5	20	20	0.04	1228.8
2	0.3	5	20	20	0.06	1228.5
3	0.3	5	20	20	0.08	1229.2
4	0.3	10	30	25	0.04	1228.4
5	0.3	10	30	25	0.06	1228.7
6	0.3	10	30	25	0.08	1229.6
7	0.3	15	40	30	0.04	1229.1
8	0.3	15	40	30	0.06	1230.4
9	0.3	15	40	30	0.08	1230.3
10	0.5	5	30	30	0.04	1226.2
11	0.5	5	30	30	0.06	1225.6
12	0.5	5	30	30	0.08	1225.9
13	0.5	10	40	20	0.04	1227.5
14	0.5	10	40	20	0.06	1226.6
15	0.5	10	40	20	0.08	1227.7
16	0.5	15	20	25	0.04	1226.3
17	0.5	15	20	25	0.06	1227.4
18	0.5	15	20	25	0.08	1226.8
19	0.7	5	40	25	0.04	1227.7
20	0.7	5	40	25	0.06	1227.1
21	0.7	5	40	25	0.08	1227.4
22	0.7	10	20	30	0.04	1227.9
23	0.7	10	20	30	0.06	1228.4
24	0.7	10	20	30	0.08	1228.1
25	0.7	15	30	20	0.04	1229.1
26	0.7	15	30	20	0.06	1227.5
27	0.7	15	30	20	0.08	1228.9

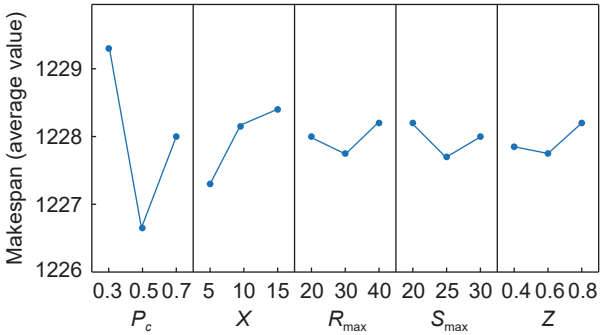


Fig. 10 Influence of key parameter P_c on the performance of HACSM.

6.3 Evaluation of the hybrid strategy

This section selects four different scale examples to test the various components of the algorithm. These four test cases are FT10, LA29, LA40, and YN04, with problem sizes of 10 × 10, 20 × 10, 15 × 15, and 20 × 15. Next, we will analyze the FW, LS1+_AKERS_EXT, TS, and HACSM algorithms on these four examples. Table 6 presents the comparison results of all algorithms after 10 runs. Figure 11 shows the box plots of the optimal solutions obtained from 10 runs of four algorithms. From the above data, it can be seen that FW has a fast convergence speed, but the solution quality is poor. As the number of jobs increases, the solution time increases and the solution quality is better than FW. TS outperforms the first two in terms of solving quality. HACSM is a hybrid of three algorithms, with solution quality and stability far exceeding the components, while verifying the effectiveness of the hybrid strategy.

6.4 Algorithm test and analysis

The test results of HACSM and the other four algorithms on the LA benchmark instances are compared in Table 7. The algorithm performance is analyzed from the perspective of average relative deviation MRE. The MRE obtained by HACSM is 0.002, while the values of HIMGA, BRKGA, TSPR, and HA are 0.006, 0.002, 0.002, and 0.002, respectively. Therefore, in terms of obtaining the optimal solution, HACSM is better than HIMGA, equal to BRKGA, TSPR, and HA. Especially, optimal values obtained by HACSM are the same as those obtained by BRKGA, TSPR, and HA, which prove the effectiveness of HACSM. Except for instance LA29, the lower bound (LB) value of the remaining instances has been obtained. The Gantt chart of the scheduling result of the instance LA29 obtained by HACSM is shown in Fig. 12.

The test results of HACSM and other three

Table 6 Test results of four algorithms on different scale examples.

Problem	FW			LS1+_AKERS_EXT			TS			HACSM		
	C_{max}	AVG	CPU	C_{max}	AVG	CPU	C_{max}	AVG	CPU	C_{max}	AVG	CPU
FT10	978	985.4	3.2	950	963.2	10.6	930	933.6	76.4	930	931.6	56.1
LA29	1232	1243.2	25.6	1190	1200.1	54.1	1164	1170.3	189.7	1152	1156.4	167.4
LA40	1290	1305.5	27.8	1245	1254.7	43.2	1225	1230	256.4	1222	1225.7	217.5
YN04	745	757.6	40.2	712	719.2	80.6	684	689.6	294.1	679	681.2	115.12

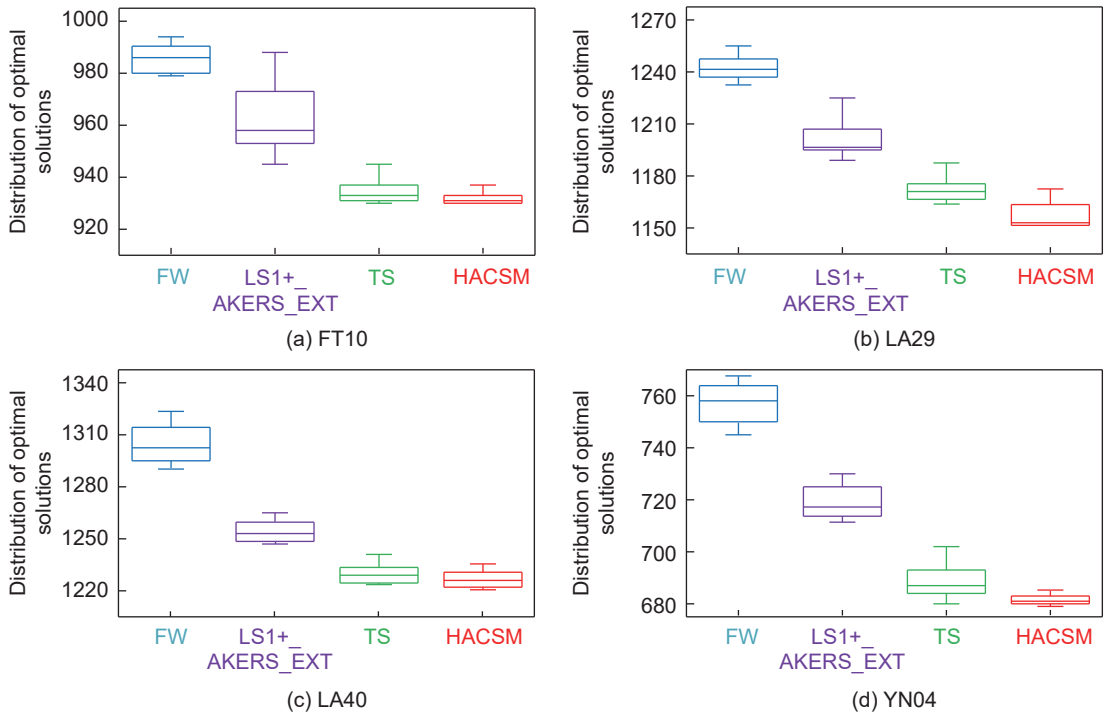


Fig. 11 Box plots of best fitness for FT10, LA29, LA40, and YN04 instances.

Table 7 Test results of LA instances compared with other algorithms.

Problem	$n\times m$	LB	HACSM				HIMGA		BRKGA		TSPR		HA	
			AVG	CPU	C_{max}	RE	C_{max}	RE	C_{max}	RE	C_{max}	RE	C_{max}	RE
LA01	10×5	666	666	0.14	**666**	0	**666**	0	**666**	0	**666**	0	**666**	0
LA02	10×5	655	655	0.63	**655**	0	**655**	0	**655**	0	**655**	0	**655**	0
LA03	10×5	597	597	0.45	**597**	0	**597**	0	**597**	0	**597**	0	**597**	0
LA04	10×5	590	590	1.33	**590**	0	**590**	0	**590**	0	**590**	0	**590**	0
LA05	10×5	593	593	1.68	**593**	0	**593**	0	**593**	0	**593**	0	**593**	0
LA06	15×5	926	926	0.69	**926**	0	**926**	0	**926**	0	**926**	0	**926**	0
LA07	15×5	890	890	0.88	**890**	**0**	**890**	**0**	**890**	**0**	**890**	**0**	**890**	**0**
LA08	15×5	863	863	0.44	**863**	**0**	**863**	**0**	**863**	**0**	**863**	**0**	**863**	**0**
LA09	15×5	951	951	1.34	**951**	**0**	**951**	**0**	**951**	**0**	**951**	**0**	**951**	**0**
LA10	15×5	958	958	1.13	**958**	**0**	**958**	**0**	**958**	**0**	**958**	**0**	**958**	**0**
LA11	20×5	1222	1222	0.85	**1222**	**0**	**1222**	**0**	**1222**	**0**	**1222**	**0**	**1222**	**0**
LA12	20×5	1039	1039	0.74	**1039**	**0**	**1039**	**0**	**1039**	**0**	**1039**	**0**	**1039**	**0**
LA13	20×5	1150	1150	0.86	**1150**	**0**	**1150**	**0**	**1150**	**0**	**1150**	**0**	**1150**	**0**
LA14	20×5	1292	1292	0.66	**1292**	**0**	**1292**	**0**	**1292**	**0**	**1292**	**0**	**1292**	**0**
LA15	20×5	1207	1207	1.16	**1207**	**0**	**1207**	**0**	**1207**	**0**	**1207**	**0**	**1207**	**0**
LA16	10×10	945	945	27.10	**945**	**0**	**945**	**0**	**945**	**0**	**945**	**0**	**945**	**0**
LA17	10×10	784	784	1.79	**784**	**0**	**784**	**0**	**784**	**0**	**784**	**0**	**784**	**0**
LA18	10×10	848	848	4.66	**848**	**0**	**848**	**0**	**848**	**0**	**848**	**0**	**848**	**0**
LA19	10×10	842	842	23.30	**842**	**0**	**842**	**0**	**842**	**0**	**842**	**0**	**842**	**0**
LA20	10×10	902	902.5	44.13	**902**	**0**	**902**	**0**	**902**	**0**	**902**	**0**	**902**	**0**
LA21	15×10	1046	1046	56.61	**1046**	**0**	**1046**	**0**	**1046**	**0**	**1046**	**0**	**1046**	**0**
LA22	15×10	927	927	38.60	**927**	**0**	**927**	**0**	**927**	**0**	**927**	**0**	**927**	**0**
LA23	15×10	1032	1032	9.63	**1032**	**0**	**1032**	**0**	**1032**	**0**	**1032**	**0**	**1032**	**0**
LA24	15×10	935	936.3	77.80	**935**	**0**	**935**	**0**	**935**	**0**	**935**	**0**	**935**	**0**
LA25	15×10	977	977	23.50	**977**	**0**	**977**	**0**	**977**	**0**	**977**	**0**	**977**	**0**
LA26	20×10	1218	1218	24.69	**1218**	**0**	**1218**	**0**	**1218**	**0**	**1218**	**0**	**1218**	**0**
LA27	20×10	1235	1236.4	68.10	**1235**	**0**	**1235**	**0**	**1235**	**0**	**1235**	**0**	**1235**	**0**
LA28	20×10	1216	1216	47.12	**1216**	**0**	**1216**	**0**	**1216**	**0**	**1216**	**0**	**1216**	**0**
LA29	20×10	1152	1156.4	167.40	**1153**	**0.09**	**1153**	**0.09**	**1153**	**0.09**	**1156**	**0.34**	**1153**	**0.09**
LA30	20×10	1355	1355	8.63	**1355**	**0**	**1355**	**0**	**1355**	**0**	**1355**	**0**	**1355**	**0**
LA31	30×10	1784	1784	6.31	**1784**	**0**	**1784**	**0**	**1784**	**0**	**1784**	**0**	**1784**	**0**
LA32	30×10	1850	1850	7.47	**1850**	**0**	**1850**	**0**	**1850**	**0**	**1850**	**0**	**1850**	**0**
LA33	30×10	1719	1719	5.39	**1719**	**0**	**1719**	**0**	**1719**	**0**	**1719**	**0**	**1719**	**0**
LA34	30×10	1721	1721	6.25	**1721**	**0**	**1721**	**0**	**1721**	**0**	**1721**	**0**	**1721**	**0**
LA35	30×10	1888	1888	7.48	**1888**	**0**	**1888**	**0**	**1888**	**0**	**1888**	**0**	**1888**	**0**
LA36	15×15	1268	1269.2	97.30	**1268**	**0**	**1268**	**0**	**1268**	**0**	**1268**	**0**	**1268**	**0**
LA37	15×15	1397	1398.5	86.50	**1397**	**0**	**1397**	**0**	**1397**	**0**	**1397**	**0**	**1397**	**0**
LA38	15×15	1196	1197.6	94.10	**1196**	**0**	**1196**	**0**	**1196**	**0**	**1196**	**0**	**1196**	**0**
LA39	15×15	1233	1235.2	116.40	**1233**	**0**	**1233**	**0**	**1233**	**0**	**1233**	**0**	**1233**	**0**
LA40	15×15	1222	1225.7	217.50	**1222**	**0**	**1224**	**0.16**	**1222**	**0**	**1224**	**0.16**	**1222**	**0**
MRE	–	–	–	–	–	0.002	–	0.006	–	0.002	–	0.012	–	0.002

algorithms on FT and ORB benchmark instances are compared in Table 8. The MRE of all algorithms is 0, which shows that HACSM achieves the same solution results as GES, BRKGA, and TSPR.

Table 9 shows the test results of HACSM and other three algorithms on ABZ and YN benchmark instances. The instances in Table 9 are those with great difficulty. So far, except for example ABZ06, the lower bound

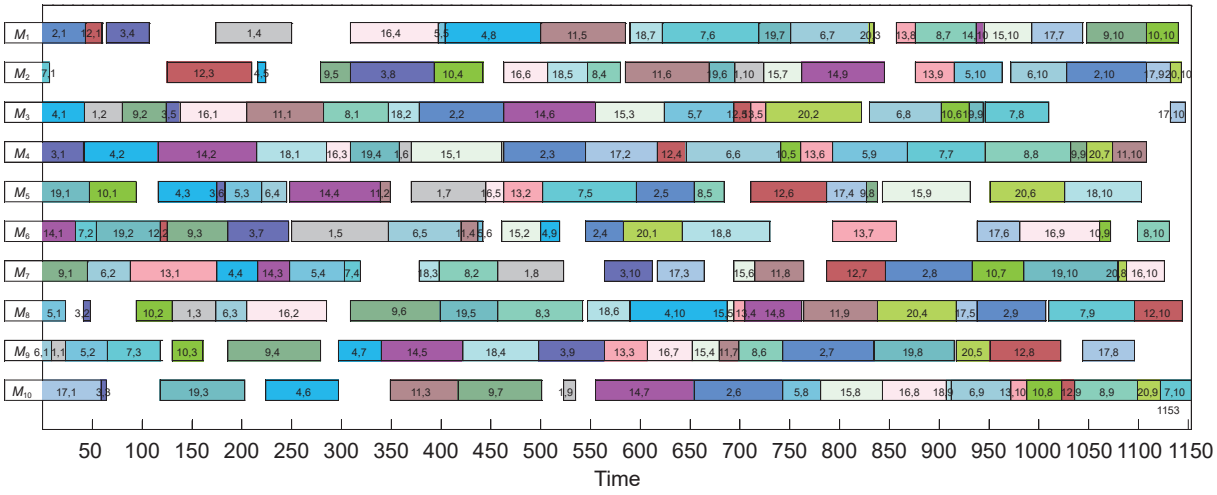


Fig. 12 Gantt charts of LA29 instances obtained by HACSM (C_{max}=1153).

Table 8 Test results of FT and ORB instances compared with other algorithms.

Problem	$n\times m$	LB	HACSM				GES		BRKGA		TSPR	
			AVG	CPU	C_{max}	RE	C_{max}	RE	C_{max}	RE	C_{max}	RE
FT06	6×6	55	55.0	0.10	**55**	0	**55**	0	**55**	0	**55**	0
FT10	10×10	930	931.3	56.51	**930**	0	**930**	0	**930**	0	**930**	0
FT20	20×5	1165	1165.0	18.33	**1165**	0	**1165**	0	**1165**	0	**1165**	0
ORB01	10×10	1059	1059.0	26.30	**1059**	0	**1059**	0	**1059**	0	**1059**	0
ORB02	10×10	888	888.0	46.51	**888**	0	**888**	0	**888**	0	**888**	0
ORB03	10×10	1005	1007.5	76.54	**1005**	0	**1005**	0	**1005**	0	**1005**	0
ORB04	10×10	1005	1005.6	68.66	**1005**	0	**1005**	0	**1005**	0	**1005**	0
ORB05	10×10	887	887.3	45.11	**887**	0	**887**	0	**887**	0	**887**	0
ORB06	10×10	1010	1010.0	16.70	**1010**	0	**1010**	0	**1010**	0	**1010**	0
ORB07	10×10	397	397.0	14.91	**397**	0	**397**	0	**397**	0	**397**	0
ORB08	10×10	899	901.2	45.50	**899**	0	**899**	0	**899**	0	**899**	0
ORB09	10×10	934	934.0	13.20	**934**	0	**934**	0	**934**	0	**934**	0
ORB10	10×10	944	944.0	3.77	**944**	0	**944**	0	**944**	0	**944**	0
MRE	–	–	–	–	–	0	–	0	–	0	–	0

Table 9 Test results of ABZ and YN instances compared with other algorithms.

Problem	$n\times m$	UB (LB)	HACSM				HIMGA		TS/SA		BeFABC	
			AVG	CPU	C_{max}	RE	C_{max}	RE	C_{max}	RE	C_{max}	RE
ABZ07	20×15	656 (656)	661.5	463.60	**658**	0.30	**662**	0.91	**658**	0.30	**659**	0.46
ABZ08	20×15	648 (645)	670.3	161.55	**669**	3.72	**676**	4.81	**669**	3.72	**670**	3.88
ABZ09	20×15	678 (661)	681.2	115.12	**678**	2.72	**688**	4.08	**678**	2.57	**682**	3.18
YN01	20×20	884 (826)	888.7	214.54	**884**	7.02	**893**	8.11	**884**	7.02	**890**	7.75
YN02	20×20	904 (861)	908.1	246.15	**907**	5.34	**913**	6.04	**907**	5.34	**911**	5.81
YN03	20×20	892 (827)	895.6	317.68	**892**	7.86	**900**	8.83	**892**	7.86	**896**	8.34
YN04	20×20	967 (918)	910.3	321.66	**969**	5.56	**977**	6.43	**969**	5.56	**971**	5.77
MRE	–	–	–	–	–	3.83	–	4.68	–	3.83	–	4.20

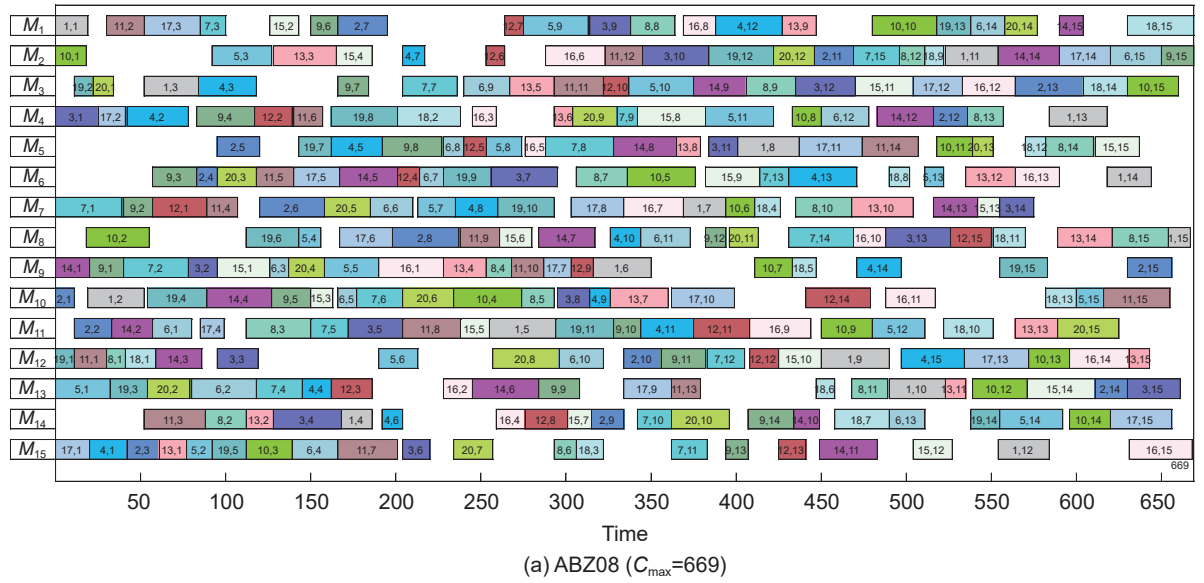
Note: UB is upper bound.

values of others had not been obtained, and algorithm still needs to be improved. It can be seen from Table 9 that the MRE value of HACSM is 3.83, while HIMGA, TS/SA, and BeFABC are 4.68, 3.83, and 4.20, respectively. Therefore, the performance of HACSM to find the optimal solution is better than HIMGA and

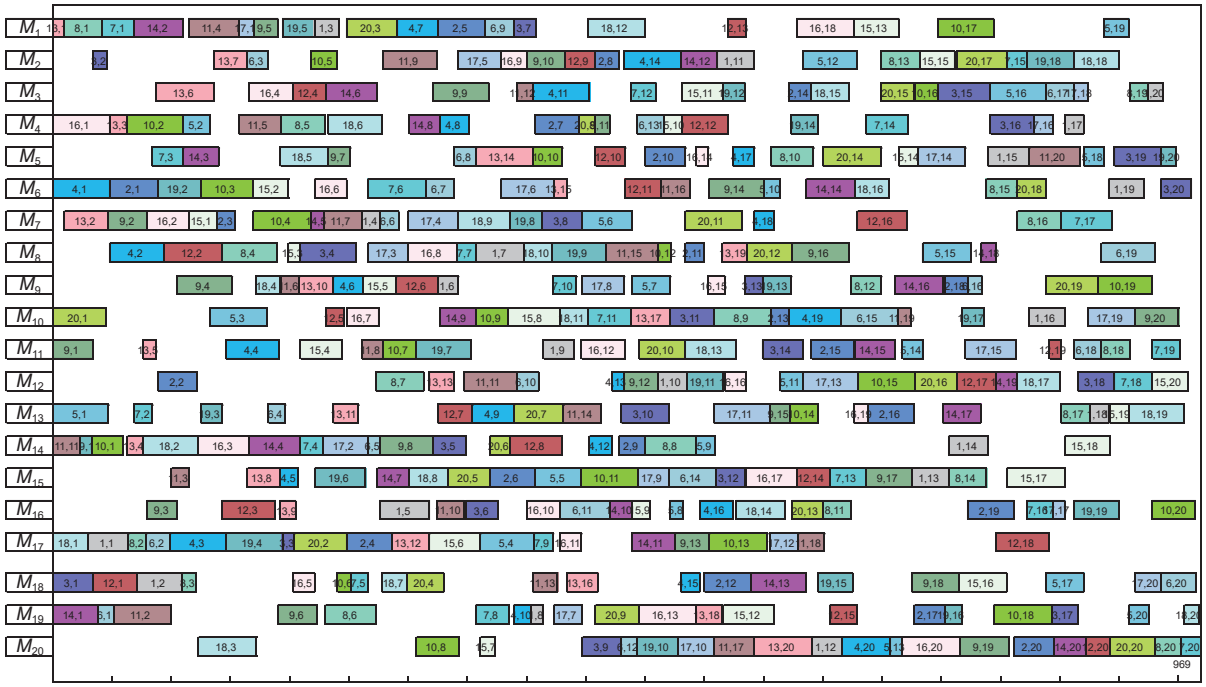
BeFABC, equal to TS/SA, which verifies the effectiveness of HACSM. Figure 13 shows the Gantt chart of ABZ08 and YN04 instances scheduling results obtained by HACSM.

7 Conclusion

This paper proposes a hybrid algorithm HACSM to solve the maximum completion time of minimizing JSP. HACSM was tested using 60 benchmark instances and compared with 7 comparison algorithms, and the test results verified its effectiveness. The HACSM algorithm obtained 52 optimal solutions out of 60 benchmark examples, with an optimal solution rate of 87% and a total relative deviation value of 1.3. When the TS does not improve the optimal solution under a certain number of iterations, the code is adjusted by using FW and LS1+_AKERS_EXT. At the same time, the three algorithms cooperate with each other, which



(a) ABZ08 (C_{max}=669)



(b) YN04 (C_{max}=969)

Fig. 13 Gantt charts of ABZ08 and YN04 instances obtained by HACSM.

can effectively avoid falling into local optimization.

The fireworks algorithm and other algorithms are mixed to solve JSP, and some parameters of the algorithm are given. In LS1+_AKERS_EXT, we designed a new method to find the shortest path in Akers graph. When searching the path, the parent node expands in layers, and multiple child nodes can be expanded at the same time, which improves the efficiency of node expansion. The HACSM algorithm can be used in discrete manufacturing environments with multiple varieties and small batches. For example, enterprises with such needs can use it to develop workshop scheduling algorithms.

In the future research, the algorithm proposed in this paper can be regarded as an effective algorithm framework. In the links of global search and local centralized search, other algorithms with excellent performance can be employed, such as variable neighborhood search algorithm, simulated annealing algorithm, ant colony algorithm, and so on. In addition, we also need to adjust some details of the algorithm. For example, in the process of solution optimization, algorithm in this paper only aims at a single individual optimization and ignores the interaction between different individuals.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (NSFC) (Nos. 52275490 and 51775240).

References

[1] P. Brucker, B. Jurisch, and B. Sievers, A branch and bound algorithm for the job-shop scheduling problem, *Discrete Appl. Math.*, vol. 49, no. 1, pp. 107–127, 1994.

[2] W. Y. Ku and J. C. Beck, Mixed Integer Programming models for job shop scheduling: A computational analysis, *Comput. Oper. Res.*, vol. 73, pp. 165–173, 2016.

[3] S. Dauzere-Peres and J. B. Lasserre, A modified shifting bottleneck procedure for job-shop scheduling, *Int. J. Prod. Res.*, vol. 31, no. 4, pp. 923–932, 1993.

[4] F. D. Croce, R. Tadei, and G. Volta, A genetic algorithm for the job shop problem, *Comput. Oper. Res.*, vol. 22, no. 1, pp. 15–24, 1995.

[5] R. Qing-dao-er-ji and Y. Wang, A new hybrid genetic algorithm for job shop scheduling problem, *Comput. Oper. Res.*, vol. 39, no. 10, pp. 2291–2299, 2012.

[6] L. Asadzadeh, A parallel artificial bee colony algorithm for the job shop scheduling problem with a dynamic migration strategy, *Comput. Ind. Eng.*, vol. 102, no. C, pp. 359–367, 2016.

[7] N. Sharma, H. Sharma, and A. Sharma, Beer froth artificial bee colony algorithm for job-shop scheduling problem, *Appl. Soft Comput.*, vol. 68, no. C, pp. 507–524, 2018.

[8] L. Gao, X. Li, X. Wen, C. Lu, and F. Wen, A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem, *Comput. Ind. Eng.*, vol. 88, pp. 417–429, 2015.

[9] M. M. Nasiri and F. Kianfar, A guided tabu search/path relinking algorithm for the job shop problem, *Int. J. Adv. Manuf. Technol.*, vol. 58, no. 9, pp. 1105–1113, 2012.

[10] M. M. Nasiri and F. Kianfar, A GES/TS algorithm for the job shop scheduling, *Comput. Ind. Eng.*, vol. 62, no. 4, pp. 946–952, 2012.

[11] S. Mahmud, A. Abbasi, R. K. Chakrabortty, and M. J. Ryan, Multi-operator communication based differential evolution with sequential Tabu Search approach for job shop scheduling problems, *Appl. Soft Comput.*, vol. 108, p. 107470, 2021.

[12] Q. Pan, L. Wang, L. Gao, and H. Sang, Differential evolution algorithm based on blocks on critical path for job shop scheduling problems, *J. Mech. Eng.*, vol. 46, no. 22, pp. 182–188, 2010.

[13] B. Z. Yao, C. Y. Yang, J. J. Hu, G. D. Yin, and B. Yu, An improved artificial bee colony algorithm for job shop problem, *Appl. Mech. Mater.*, vols. 26–28, pp. 657–660, 2010.

[14] R. Yusof, M. Khalid, G. T. Hui, S. M. Yusof, and M. F. Othman, Solving job shop scheduling problem using a hybrid parallel micro genetic algorithm, *Appl. Soft Comput.*, vol. 11, no. 8, pp. 5782–5792, 2011.

[15] V. Sels, K. Craeymeersch, and M. Vanhoucke, A hybrid single and dual population search procedure for the job shop scheduling problem, *Eur. J. Oper. Res.*, vol. 215, no. 3, pp. 512–523, 2011.

[16] L. Gao, G. Zhang, L. Zhang, and X. Li, An efficient memetic algorithm for solving the job shop scheduling problem, *Comput. Ind. Eng.*, vol. 60, no. 4, pp. 699–705, 2011.

[17] X. Zuo, C. Wang, and W. Tan, Two heads are better than one: An AIS- and TS-based hybrid strategy for job shop scheduling problems, *Int. J. Adv. Manuf. Technol.*, vol. 63, no. 1, pp. 155–168, 2012.

[18] W. Wisittipanich and V. Kachitvichyanukul, Two enhanced differential evolution algorithms for job shop scheduling problems, *Int. J. Prod. Res.*, vol. 50, no. 10, pp. 2757–2773, 2012.

[19] A. Banharnsakun, B. Sirinaovakul, and T. Achalakul, Job shop scheduling with the best-so-far ABC, *Eng. Appl. Artif. Intell.*, vol. 25, no. 3, pp. 583–593, 2012.

[20] A. Ponsich and C. A. C. Coello, A hybrid differential evolution—Tabu search algorithm for the solution of job-shop scheduling problems, *Appl. Soft Comput.*, vol. 13, no. 1, pp. 462–474, 2013.

[21] R. Zhang, S. Song, and C. Wu, A hybrid artificial bee colony algorithm for the job shop scheduling problem, *Int.*

J. Prod. Econ., vol. 141, no. 1, pp. 167–178, 2013.

[22] A. C. Spanos, S. T. Ponis, I. P. Tatsiopoulos, I. T. Christou, and E. Rokou, A new hybrid parallel genetic algorithm for the job-shop scheduling problem, *Int. Trans. Oper. Res.*, vol. 21, no. 3, pp. 479–499, 2014.

[23] J. F. Gonçalves and M. G. C. Resende, An extended Akers graphical method with a biased random-key genetic algorithm for job-shop scheduling, *Int. Trans. Oper. Res.*, vol. 21, no. 2, pp. 215–246, 2014.

[24] X. Wang and H. Duan, A hybrid biogeography-based optimization algorithm for job shop scheduling problem, *Comput. Ind. Eng.*, vol. 73, pp. 96–114, 2014.

[25] S. Meeran and M. S. Morshed, Evaluation of a hybrid genetic tabu search framework on job shop scheduling benchmark problems, *Int. J. Prod. Res.*, vol. 52, no. 19, pp. 5780–5798, 2014.

[26] F. Zhao, X. Jiang, C. Zhang, and J. Wang, A chemotaxis-enhanced bacterial foraging algorithm and its application in job shop scheduling problem, *Int. J. Comput. Integr. Manuf.*, vol. 28, no. 10, pp. 1106–1121, 2014.

[27] B. Peng, Z. Lü, and T. C. E. Cheng, A tabu search/path relinking algorithm to solve the job shop scheduling problem, *Comput. Oper. Res.*, vol. 53, pp. 154–164, 2015.

[28] L. Asadzadeh, A local search genetic algorithm for the job shop scheduling problem with intelligent agents, *Comput. Ind. Eng.*, vol. 85, no. C, pp. 376–383, 2015.

[29] M. Kurdi, A new hybrid island model genetic algorithm for job shop scheduling problem, *Comput. Ind. Eng.*, vol. 88, no. C, pp. 273–283, 2015.

[30] F. Zhao, Z. Shao, J. Wang, and C. Zhang, A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems, *Int. J. Prod. Res.*, vol. 54, no. 4, pp. 1039–1060, 2015.

[31] F. A. Toader, A hybrid algorithm for job shop scheduling problem, *Stud. Inform. Contr.*, vol. 24, no. 2, pp. 171–180, 2015.

[32] T. C. E. Cheng, B. Peng, and Z. Lü, A hybrid evolutionary algorithm to solve the job shop scheduling problem, *Ann. Oper. Res.*, vol. 242, no. 2, pp. 223–237, 2016.

[33] S. Zhao, A hybrid algorithm with a new neighborhood structure for the job shop scheduling problem, *J. Mech. Eng.*, vol. 52, no. 9, pp. 141–150, 2016.

[34] Y. Nagata and I. Ono, A guided local search with iterative ejections of bottleneck operations for the job shop scheduling problem, *Comput. Oper. Res.*, vol. 90, no. C, pp. 60–71, 2018.

[35] C. Peng, G. Wu, T. W. Liao, and H. Wang, Research on multi-agent genetic algorithm based on tabu search for the job shop scheduling problem, *PLoS One*, vol. 14, no. 9, p. e0223182, 2019.

[36] G. Zhou, Y. Zhou, and R. Zhao, Hybrid social spider optimization algorithm with differential mutation operator for the job-shop scheduling problem, *J. Ind. Manag. Optim.*, vol. 17, no. 2, pp. 533–548, 2021.

[37] M. Liu, X. Yao, and Y. Li, Hybrid whale optimization algorithm enhanced with Lévy flight and differential evolution for job shop scheduling problems, *Appl. Soft Comput.*, vol. 87, p. 105954, 2020.

[38] S. K. Zhao, Research on path relinking based on non-delay scheduling and backtracking tabu search algorithm of job shop scheduling problem, *J. Mech. Eng.*, vol. 57, no. 14, pp. 291–303, 2021.

[39] J. Xie, X. Li, L. Gao, and L. Gui, A hybrid algorithm with a new neighborhood structure for job shop scheduling problems, *Comput. Ind. Eng.*, vol. 169, p. 108205, 2022.

[40] L. Huang, S. K. Zhao, and S. Huang, Hybrid algorithm based on obstacle graph model and tabu search for job shop scheduling problem, *J. Mech. Eng.*, vol. 59, no. 16, pp. 435–444, 2023.

[41] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan, A very fast TS/SA algorithm for the job shop scheduling problem, *Comput. Oper. Res.*, vol. 35, no. 1, pp. 282–294, 2008.

[42] E. Yuan, L. Wang, S. Cheng, S. Song, W. Fan, and Y. Li, Solving flexible job shop scheduling problems via deep reinforcement learning, *Expert Syst. Appl.*, vol. 245, p. 123019, 2024.

[43] L. Gui, X. Li, L. Gao, and C. Wang, Necessary and sufficient conditions for feasible neighbourhood solutions in the local search of the job-shop scheduling problem, *Chin. J. Mech. Eng.*, vol. 36, no. 1, p. 87, 2023.

[44] Y. Zhang, H. Zhu, D. Tang, T. Zhou, and Y. Gui, Dynamic job shop scheduling based on deep reinforcement learning for multi-agent manufacturing systems, *Robot. Comput. Integr. Manuf.*, vol. 78, p. 102412, 2022.

[45] L. He, W. Li, Y. Zhang, and Y. Cao, A discrete multi-objective fireworks algorithm for flowshop scheduling with sequence-dependent setup times, *Swarm and Evolutionary Computation*, vol. 51, p. 100575, 2019.

[46] X. Pang, H. Xue, M. L. Tseng, M. K. Lim, and K. Liu, Hybrid flow shop scheduling problems using improved fireworks algorithm for permutation, *Appl. Sci.*, vol. 10, no. 3, p. 1174, 2020.

[47] C. Zhang, P. Li, Z. Guan, and Y. Rao, A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem, *Comput. Oper. Res.*, vol. 34, no. 11, pp. 3229–3242, 2007.

[48] S. B. Akers, A graphical approach to production scheduling problems, *Oper. Res.*, vol. 4, no. 2, pp. 244–245, 1956.

[49] P. Brucker, An efficient algorithm for the job-shop problem with two jobs, *Computing*, vol. 40, no. 4, pp. 353–359, 1988.

[50] P. E. Hart, N. J. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[51] E. Balas and A. Vazacopoulos, Guided local search with shifting bottleneck for job shop scheduling, *Manag. Sci.*, vol. 44, no. 2, pp. 262–275, 1998.

Lin Huang received the MS degree from University of Jinan, China in 2023. He is currently pursuing the PhD degree at Nanjing University of Aeronautics and Astronautics, China. His research focuses on workshop production systems and intelligent optimization algorithms.

Yingjie Xiong is currently pursuing the master degree at University of Jinan, China. His current research focuses on workshop production scheduling and intelligent optimization algorithm.

Shikui Zhao received the PhD degree in mechanical engineering from Zhejiang University, China in 2013. He is an associate professor at the School of Mechanical Engineering, University of Jinan, China. His current research focuses on workshop production scheduling and intelligent optimization algorithm.