IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 27, 2021, accepted October 5, 2021, date of publication October 8, 2021, date of current version October 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3118986

Feature-Extraction-Based Iterated Algorithms to Solve the Unrelated Parallel Machine Problem With Periodic Maintenance Activities

JIHONG PANG[1,2], YA-CHIH TSAI[3], AND FUH-DER CHOU[1]
[1]College of Mechanical and Electrical Engineering, Wenzhou University, Wenzhou, Zhejiang 325035, China
[2]College of Business, Shaoxing University, Shaoxing, Zhejiang 312000, China
[3]Department of Hotel Management, Vanung University, Jung-Li, Taoyuan City 32061, Taiwan

Corresponding author: Fuh-Der Chou (fdchou@tpts7.seed.net.tw)

This work was supported in part by the National Natural Science Foundation, China, under Grant 72071149 and Grant 71671130; and in part by the Provincial Natural Science Foundation, Zhejiang, China, under Grant LY20G010014.

ABSTRACT This paper considers an unrelated parallel machine problem with job release times and maintenance activities, in which machines have to periodically undergo maintenance since the status of the machines will be deteriorated by job-induced dirt. The problem is inspired by a wet station for cleaning operations in a semiconductor manufacturing process. The objective is to minimize the makespan. Since the considered problem is proven to be NP-hard, obtaining optimal solutions is almost impossible in a reasonable computational time when the problem becomes large. We develop specific feature-extraction procedures to recognize important information in a job sequence and linkage encoding (LE) procedures to generate new job sequences. The two above procedures are embedded into an iterated algorithm, called a feature-extraction-based iterated algorithm (FEBIA), to obtain optimal or better solutions for the considered problem. To examine the performance of the FEBIA, the FEBIA is compared with two population-based algorithms, the particle swarm optimization (PSO) algorithm and the genetic algorithm (GA), using many test data. The results reveal that the proposed FEBIA perform better than the two population-based algorithms, demonstrating the potential of the FEBIA to solve the unrelated parallel machine problem with periodic maintenance and job release times.

INDEX TERMS Unrelated parallel machine, makespan, flexible maintenance, scheduling.

I. INTRODUCTION

This paper considers job scheduling and maintenance activity optimization problems for unrelated parallel machines with dynamic job release times. In the literature, the majority of joint dispatching job and maintenance activity problems examine single-machine configurations [1]–[3] and assume that all jobs are ready to be processed at the same time. However, unrelated parallel machine configuration problems are commonly encountered in different manufacturing systems, such as semiconductor and electronics manufacturing, since a bank of parallel machines can overcome the impact of a bottleneck in the production line to improve the system throughput. Additionally, a case in which the job release times are the same or equal to zero is not practical for such a dynamic manufacturing situation, especially in a semiconductor manufacturing system.

Semiconductor wafer fabrication consists of hundreds of operations and the scheduling complexity is increased by specific process requirements such as sequence-dependent setups, machine maintenance, and batch-processing. Some researchers have noted that when maintenance planning and production scheduling lack coordination, machines may remain idle due to waiting for engineering personnel to perform maintenance activities, even though jobs are ready to be processed [4], [5]. Low throughput and inefficient system performance consequently occur. This work is inspired by the importance of integrating job scheduling and maintenance simultaneously for a semiconductor manufacturing process, more specifically, cleaning operations. The cleaning operation is one stage of the semiconductor manufacturing process, which cleans the dirt off of wafer surfaces so that the wafers can maintain a good status for the next process.

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

Dirt is a catch-all term for any type of residue, such as the particles, organic material, and metal-salts remaining in a machine during this cleaning operation. The amount of dirt will increase as more (wafer) processing jobs are conducted. Once the amount of dirt exceeds a specific threshold value, it will damage the wafer during processing. Therefore, wet stations (machines) must be stopped and the cleaning agent must be periodically changed to keep the machines in a good status.

Our considered problem can be defined as follows. There are *n* jobs to be processed on m unrelated parallel machines. Each machine M_i is at initial status, i.e., there is no dirt in each machine, and M_i has a positive cleaning time w_i and a dirt threshold T_i. In addition, after cleaning activity, machines become initial. Each job J_j has a positive processing time p_{ij}, a release time r_j, and an amount of dirt d_{ij}, wherein p_{ij} and d_{ij} are determined by the assigned machine M_i. Given a feasible schedule S of jobs, the amount of dirt in each machine must not exceed its dirt threshold. The objective is to find a schedule that minimizes the makespan. According to the standard machine scheduling classification [6], this problem is denoted as $R_m|r_j, d_{ij} \le T_i, fpa|C_{max}$, where *fpa* in the second field means that the maintenance activity is flexible and occurs periodically [7]. Flexible maintenance activity is one type of maintenance where the starting time of the maintenance must be determined during the production scheduling process. In addition, the various kinds of flexible maintenance activities and their practical requirements have been addressed by many researchers in the literature. Mosheiov and Sarif [8] considered a single machine problem where the machine has to complete only one maintenance activity prior to a given deadline, and they proposed a pseudopolynomial dynamic programming (DP) method and a heuristic to minimize the total weighted completion time. Lee and Chen [4] considered the parallel machine problem, and proposed branch and bound algorithms to minimize the total weighted completion time, for the two cases in which machines can be maintained simultaneously and in which multiple machines cannot be maintained simultaneously. Levin *et al.* [9] also considered a parallel-machine problem where all maintenance activities must be simultaneously performed once. They proved the problem to be NP-hard and provided a pseudopolynomial DP algorithm and an SPT-based algorithm to minimize the total completion time. Yoo and Lee [10] considered the same problem as Lee and Chen [4], and proved the problems with the objective of the makespan, the total weighted completion time, the maximum lateness or the total lateness to be NP-hard. Qi *et al.* [11] considered another type of flexible maintenance where the machine must be stopped and maintained after working for a period of time. For the single machine, they proposed three simple heuristics and a branch and bound algorithm to minimize the total completion time. Sbihi and Varnier [12] proposed a heuristic and a branch and bound algorithm to minimize the maximum tardiness. Lee *et al.* [13] considered a parallel machine problem in which the objective was to minimize the total tardiness. A branch and bound algorithm was developed that included the implementation of the lower and upper bounding procedure, which can find the optimal solutions for problems with up to 20 jobs. Costa *et al.* [14] considered a parallel machine problem with tool changes where each machine needs to change to a new tool if the tool service life is reached. The objective was to minimize the total completion time, and they proposed a mixed integer linear programming model and a genetic algorithm (GA) to solve both small- and large-sized problems. Another well-known flexible maintenance activity in which maintenance must be started and finished in a predetermined maintenance interval [u, v] was defined. Chen ([2] and [7]) developed mixed binary integer programming (BIP) models and an efficient heuristic algorithm to minimize the mean flow time and the makespan, respectively, for a single problem with this type of flexible maintenance. Low *et al.* [15] compared the performances of six kinds of heuristics for a single problem with the objective of minimizing makespan and provided the calculation of the error bounds. A more realistic ε-almost periodic maintenance was assumed by Xu *et al.* [5], where the difference in the times of any consecutive maintenance activities of the machine is within ε, and proposed an approximation algorithm to minimize the makespan for the parallel-machine problem. Qamhan *et al.* [16] considered a single-machine problem with time window periodic maintenance where the time between two maintenance activates was a fixed interval (T) and each maintenance activity could start in a time window, i.e., T$\mp$w. Additionally, the time of maintenance activity was not equal. For the problem, they proposed a mixed-integer linear programming model and ant colony optimization (ACO) to minimize the number of tardy jobs. Hidri *et al.* [17] considered a parallel-machine problem with a single robot server, where the availability interval of the machines and the duration of PM activity were deterministic and known in advance. For the problem, they developed lower bound, simulated annealing (SA), tabu search (TS), and GA to find better solutions under the objective of minimizing the makespan.

In addition to those tasks in which flexible maintenance activity is considered, most studies in the literature considered another type of maintenance activity in the scheduling problem. This problem is called a scheduling problem with machine availability constraints, where the starting time and duration of maintenance are assumed to be fixed and known in advance [1], [3], [18]. Comprehensive reviews for this case are provided by Sanlaville and Schmidt [19], Schmidt [20], and Ma *et al* [21]. Some researchers considered both cases of fixed and flexible maintenance activities in their studies. Cui *et al.* [22] considered a nonpermutation flow shop problem where two cases of nonavailability intervals are involved. In the first case, the maintenance activity is periodically fixed and known in advance, while in the second case, the machine has to be maintained after working for a period of time. Since the problem is NP-hard in the strong sense, a hybrid incremental genetic algorithm was proposed. It is worth noting that the maintenance activities in the above studies are related to a time constraint.

Unlike the abovementioned studies where machine maintenance is related to a time constraint, i.e., given fixed time intervals or a maximum continuous working time, Bock *et al.* [23] addressed a new model where maintenance activity is subject to job-dependent machine deterioration where the maintenance level drops by a certain job-dependent amount. They considered two types of maintenance activities. One is full maintenance activity, where the maintenance level increases to the maximum maintenance level once maintenance activity is completed, and the other is partial maintenance activity, where the maintenance level increases to an arbitrary maintenance level that is less than the maximum maintenance level. To address this problem, they considered the makespan, the total completion time, the maximum lateness, and the number of tardy jobs and provided complexity analysis for the problems. Girgoriu and Briskorn [24] and Tian *et al.* [25] extended the study of Bock *et al.* [23] to parallel machines. Su and Wang [26] introduced another new model where the machine status deteriorates due to job-induced dirt; that is, the machine needs to be maintained before the accumulation of dirt reaches a threshold value, and the machine status will return to its initial condition once the maintenance activity is finished. The main difference from the study of Bock *et al.* [23] is that the maintenance time is fixed and not dependent on the current maintenance level. To address the problem, they proposed a mixed binary integer programming model and an effective heuristic to minimize the total absolute deviation of job completion times (TADC). Later, Chen *et al.* [27] also considered the same problem that Su and Wang [26] had, where the objective is minimizing the total completion time. Pang *et al.* [28] extended the study of Su and Wang [26] from a static problem with a single objective function to a dynamic problem with a biobjective function, and they proposed a scatter simulated annealing algorithm to obtain nondominated solutions.

In this paper, we extend the study of Su and Wang [26] from a single machine to unrelated parallel machines with job release times, since the literature on parallel machine problems is still not as extensive as that of single-machine scheduling problems. To the best of our knowledge, only three studies address maintenance constraints and job release times simultaneously [28]–[30]; however, the three studies considered single machine problems. Without considering the job release times, when the processing times of all jobs are the same and there is only one machine, the considered problem of $R_m \left| r_j, d_{ij} \leq T_i, fpa \right| C_{max}$ could be reduced to the one-dimensional bin-packing problem, which aims to pack n items of size d_j into a minimum number of bins of capacity T, which it has been proven to be NP-hard [31]. Since this problem can be reduced to a bin-packing problem, it is also NP-hard, and hence, we focus on developing a feature-extraction-based iterated algorithm (FEBIA) to efficiently obtain high-quality solutions. The proposed FEBIA is inspired by observations from many evolutionary search algorithms. In the FEBIA, some information of an individual schedule in the past generation could be further extracted to generate good solutions instead of straightly ignoring the information hidden in the individual schedule. Numerical experiments are conducted to demonstrate the performance of the FEBIA by comparing it with a basic particle swarm optimization (PSO) algorithm and a simple genetic algorithm (GA). The computational results show that the proposed FEBIA gives promising and better results for the problem under study. The rest of the paper is organized as follows. Section II presents a mathematical model for formally describing the considered problem. Section III presents four feature-extraction matrices to recognize important features of job sequences. Our proposed FEBIA is addressed in Section IV. Section V presents the experimental results. Section VI offers the conclusions and future research directions.

II. MIXED INTEGER PROGRAMMING MODEL

In this section, a mixed integer programming (MIP) model is constructed to describe the characteristics of the considered problems in which a set of jobs $J = \{J_1, J_2, \ldots, J_n\}$ is to be scheduled on m unrelated parallel machines. Preemption is not allowed, and machines are not available all the time because they must be stopped to be cleaned periodically to prevent amount of dirt in the machine from exceeding the threshold. The parameters and decision variables are listed as follows. Our objective is to minimize the makespan, i.e., C_{max}.

A. PARAMETERS

n: number of jobs

m: number of machines

i: index of machines, where $i = 1, \ldots, m$

j: index of jobs, where $j = 1, \ldots, n$

k: index of the processing sequence, where $k = 1, \ldots, n$

M_i: machine i

J_j: job j

r_j: the release time for job j

p_{ij}: the processing time of job j on machine i

d_{ij}: the remaining dirt for job j when processed on machine i

T_i: the maximum dirt allowance for machine i

w_i : the cleaning time for machine i

BM: a very large positive integer; $BM = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$

B. DECISION VARIABLES

X_{ijk} : 1 if job j is processed on machine i at position k, and 0 otherwise

Y_{ik} : 1 if the maintenance is implemented immediately after position k where machine i finished a job, and 0 otherwise.

ST_{ik}: the job processing start time for machine i at position k

PT_{ik}: the processing time for machine i at position k

CT_{ik}: the completion time for machine i at position k

Q_{ik}: the accumulated dirt in machine i after position k where machine i finished a job

C_j: the completion time of job j.

$C_{max} = \max C_j$.

C. MODEL

Objective function

$$Min.\ C_{max} \tag{1}$$

$$\text{Subject to } \sum_{i=1}^{m}\sum_{k=1}^{n} X_{ijk} = 1 \quad \forall j = 1,\ldots,n \tag{2}$$

$$\sum_{j=1}^{n} X_{ijk} \leq 1 \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 1,\ldots,n \end{cases} \tag{3}$$

$$ST_{ik} \geq \sum_{j=1}^{n}\left(r_j \times X_{ijk}\right) \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 1,\ldots,n \end{cases} \tag{4}$$

$$PT_{ik} = \sum_{j=1}^{n}\left(p_{ij} \times X_{ijk}\right) \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 1,\ldots,n \end{cases} \tag{5}$$

$$ST_{ik} \geq CT_{i,k-1} + \left(w_i \times Y_{i,k-1}\right) \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 2,\ldots,n \end{cases} \tag{6}$$

$$CT_{ik} \geq ST_{i,k} + PT_{ik} \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 1,\ldots,n \end{cases} \tag{7}$$

$$Q_{i1} = \sum_{j=1}^{n}\left(d_{ij} \times X_{ij1}\right) \quad \forall i = 1,\ldots,m \tag{8}$$

$$Q_{i,k-1} + \sum_{j=1}^{n}\left(d_{ij} \times X_{ijk}\right) \leq Q_{ik} + \left(BM \times Y_{i,k-1}\right) \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 2,\ldots,n \end{cases} \tag{9}$$

$$\sum_{j=1}^{n}\left(d_{ij} \times X_{ijk}\right) \leq Q_{ik} + \left[BM \times \left(1 - Y_{i,k-1}\right)\right] \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 2,\ldots,n \end{cases} \tag{10}$$

$$Q_{ik} \leq T_i \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 1,\ldots,n \end{cases} \tag{11}$$

$$\sum_{j=1}^{n} X_{ijk} \leq \sum_{j=1}^{n} X_{ij,k-1} \quad \forall \begin{cases} i = 1,\ldots,m \\ k = 2,\ldots,n \end{cases} \tag{12}$$

$$C_j \geq CT_{ik} + BM \times \left(X_{ijk} - 1\right) \quad \forall \begin{cases} i = 1,\ldots,m \\ j = 1,\ldots,n \\ k = 1,\ldots,n \end{cases} \tag{13}$$

$$C_{max} \geq C_j \quad \forall j = 1,\ldots,n \tag{14}$$

The objective function (1) minimizes the makespan. Constraint (2) ensures that each job is assigned to only one position on a machine. Constraint (3) ensures that not more than one job is assigned to any position in the sequence for any machine. Constraint (4) defines the available time for processing jobs at the k*th* position on machine i. Constraint (5) specifies the job processing time at the k*th* position on machine i. Constraint (6) ensures that the start time at the k*th* position on machine i should be greater than or equal to the completion time of the previous position. Constraint (7) prevents a job from being processed before its ready time. Constraint (8) defines the total remaining dirt at the first position on machine i. Constraints (9) and (10) define the total remaining dirt left at the k*th* position on machine i, excluding the first position. Constraint (11) ensures that the total dirt is not greater than the maximum dirt allowance for each machine. Constraint (12) makes it so that at least one job has to be processed between maintenance activities. Constraint (13) defines the completion time of job j. Constraint (14) defines the makespan.

III. FEATURE-EXTRACTION MATRICES

In this paper, we develop feature-extraction matrices and construct ten procedures to find heuristic solutions based on these feature-extraction matrices. The purpose of the feature-extraction matrices is to find the job-position, job-job, job-machine, and machine-job relations from each individual solution and use this information to guide job allocations and generate better schedule solutions. To construct these matrices, we first convert each solution into a corresponding feature value to identify its solution quality among the population and then fill the feature value in the matrices. The conversion steps from the objective solutions to the feature values are described as follows.

A. FEATURE VALUE

Step 1. Decode each individual π with $(n + m - 1)$ elements to obtain a makespan value (C_{max}^{π}), which is described in Section IV. To enlarge the disparities among the solutions of individuals, we square C_{max}^{π} and obtain $A_\pi = \left(C_{max}^{\pi}\right)^2$.

Step 2. Calculate the mean value $\bar{A}$ and standard deviation σ_A for all $A_\pi s$ in this population according to the following equations:

$$\bar{A} = \sum_{\pi=1}^{L} A_\pi \Big/ L$$

$\sigma_A = \sqrt{\sum_{\pi=1}^{L}\left(A_\pi - \bar{A}\right)^2 \Big/ (L-1)}$, where L is the number of individuals in the population.

Step 3. Normalize A_π, i.e., $Z_\pi = \left(\bar{A} - A_\pi\right)\big/\sigma_A$.

Step 4. Obtain a feature value for an individual by cubing Z_π, that is, $E_i = (Z_\pi)^3$.

The steps to obtain the feature value for an individual in the population has two functions: (1) Our objective is to minimize the makespan. We use this conversion to maximize the feature values. In other words, a higher feature value is preferred. (2) The conversion steps can make the better/worse individuals more distinguished in the population. Because the feature

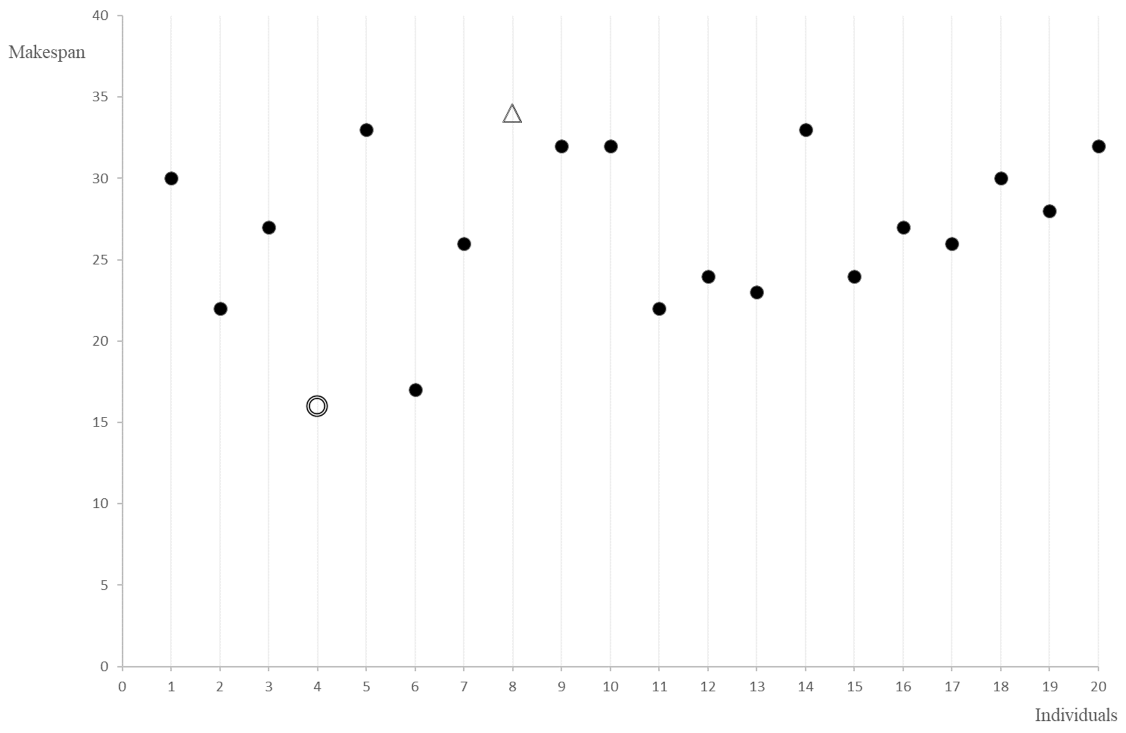


FIGURE 1. The makespan values of the twenty individuals for 5-job example of Table 1.

values of all individuals are normalized and then enlarged, the best solution is above zero, and the best and worse solution is significant. An example of five jobs in Table 1 is used to generate twenty individuals randomly. Among the twenty individuals, in Fig. 1 and Fig. 2, the fourth individual marked by a double circle means that the solution is the best where the makespan and feature value are 16 and 7.03089, respectively. Additionally, it can be seen that the best and worse individuals seem to be much more distinguished in Fig. 2 where the worse feature value is far below zero. Moreover, for other insignificant individuals, their values seem to be slightly close to zero, as shown in Fig. 2.

Once we obtained the feature values for all individuals in the population, we next used the feature values to construct four matrices, including job-position, job-job, job-machine, and machine-job matrices. The first two matrices were first developed by Chou [32] and embedded into a genetic algorithm to successfully solve the single machine total weighted tardiness scheduling problem. We extended the idea of Chou [32] and constructed job-position, job-job, job-machine, and machine-job matrices using the feature values. The construction steps for the matrices are described as follows.

B. JOB-POSITION $(n + m - 1, n + m - 1)$ MATRIX

The job-position matrix is used to provide the semaphores of the positions for the jobs in a job-sequence based on the global feature values. A higher semaphore value of job i at position j suggests that this kind of designation could be found in better solutions. The construction of the job-position matrix is described as follows.

TABLE 1. An example with five jobs and two machines.

J_j	r_j	p_{1j}	p_{2j}	d_{1j}	d_{2j}
J_1	0	3	4	5	4
J_2	3	7	8	3	3
J_3	5	4	5	4	3
J_4	2	3	4	6	5
J_5	6	5	6	6	4
$(T_1, T_2) = (10,8)$					
$(w_1, w_2) = (3,4)$					

Step 1. We initialize two two-dimensional arrays (JP1 and JP2) with $n + m - 1$ rows and $n + m - 1$ columns, where each element in the two arrays is equal to zero.

Step 2. For each selected individual in the population, if job i exists at position j, the values of JP1(i, j) is added by its feature, and the value of JP2(i, j) is added by 1.

Step 3. We repeat Step 2 until all individuals of the population have been selected and we obtain JP1 and JP2 for this population. To demonstrate the working process, a 5-job example of Table 1 is used. Suppose there are twenty sequences (individuals) whose feature values are obtained. Then, we accumulate the information from these twenty individuals to generate JP1 and JP2 arrays as shown in Fig. 3.

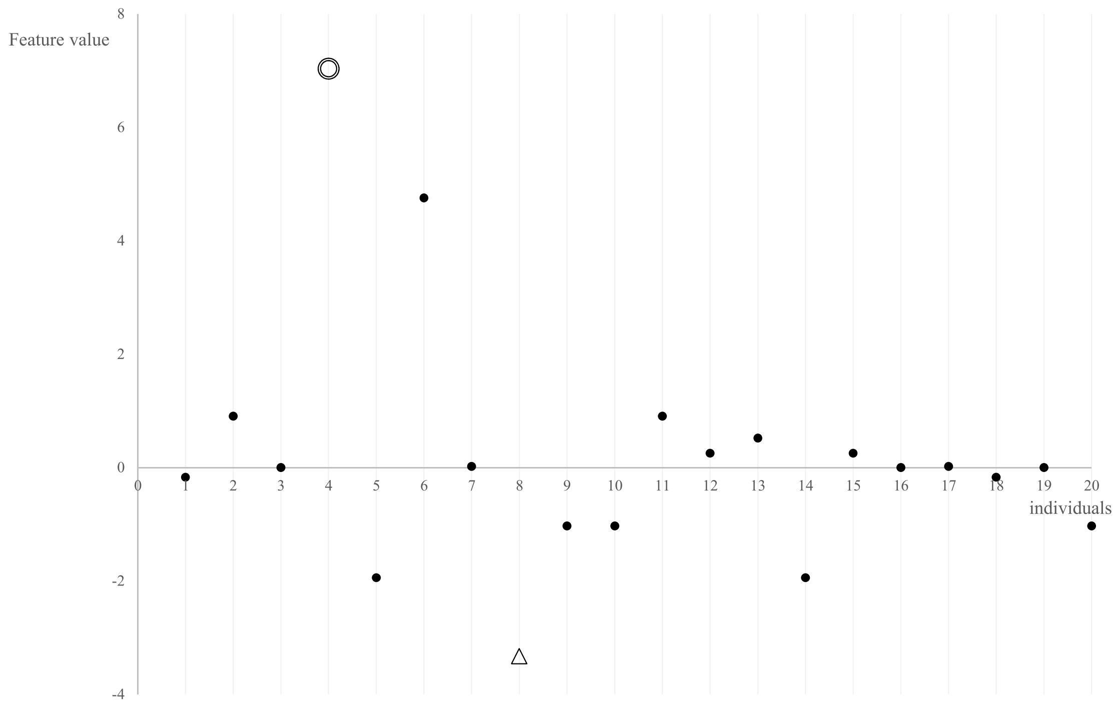
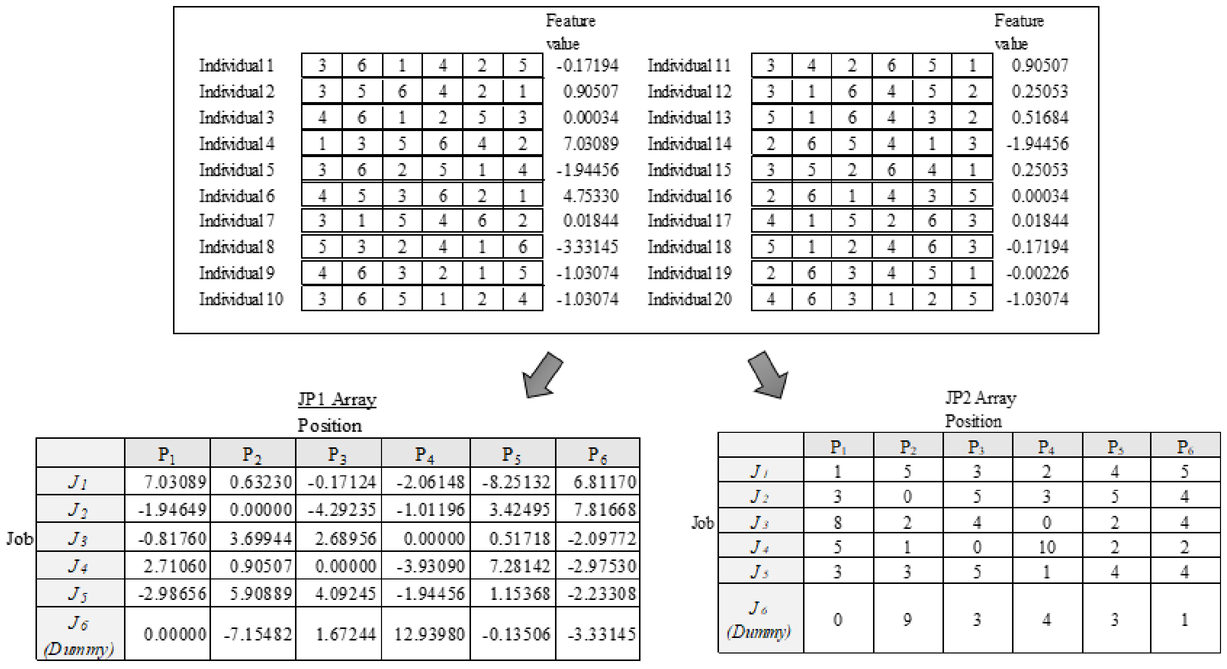


FIGURE 2. The feature values of the same twenty individuals for 5-job example of Table 1.



							Feature value								Feature value
Individual 1	3	6	1	4	2	5	-0.17194	Individual 11	3	4	2	6	5	1	0.90507
Individual 2	3	5	6	4	2	1	0.90507	Individual 12	3	1	6	4	5	2	0.25053
Individual 3	4	6	1	2	5	3	0.00034	Individual 13	5	1	6	4	3	2	0.51684
Individual 4	1	3	5	6	4	2	7.03089	Individual 14	2	6	5	4	1	3	-1.94456
Individual 5	3	6	2	5	1	4	-1.94456	Individual 15	3	5	2	6	4	1	0.25053
Individual 6	4	5	3	6	2	1	4.75330	Individual 16	2	6	1	4	3	5	0.00034
Individual 7	3	1	5	4	6	2	0.01844	Individual 17	4	1	5	2	6	3	0.01844
Individual 8	5	3	2	4	1	6	-3.33145	Individual 18	5	1	2	4	6	3	-0.17194
Individual 9	4	6	3	2	1	5	-1.03074	Individual 19	2	6	3	4	5	1	-0.00226
Individual 10	3	6	5	1	2	4	-1.03074	Individual 20	4	6	3	1	2	5	-1.03074

JP1 Array

Position

		P_1	P_2	P_3	P_4	P_5	P_6
Job	J_1	7.03089	0.63230	-0.17124	-2.06148	-8.25132	6.81170
	J_2	-1.94649	0.00000	-4.29235	-1.01196	3.42495	7.81668
	J_3	-0.81760	3.69944	2.68956	0.00000	0.51718	-2.09772
	J_4	2.71060	0.90507	0.00000	-3.93090	7.28142	-2.97530
	J_5	-2.98656	5.90889	4.09245	-1.94456	1.15368	-2.23308
	J_6 *(Dummy)*	0.00000	-7.15482	1.67244	12.93980	-0.13506	-3.33145

JP2 Array

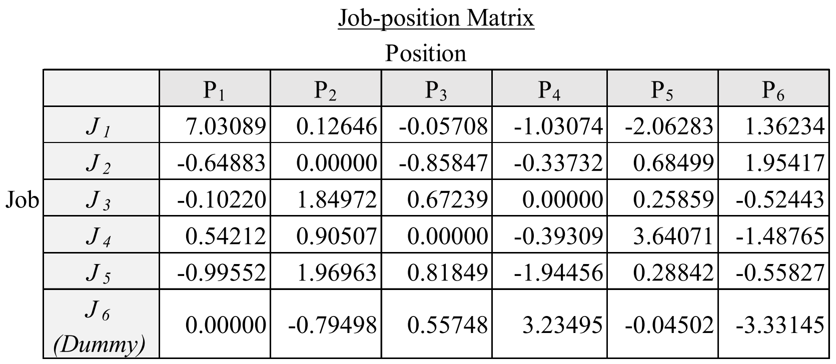
Position

		P_1	P_2	P_3	P_4	P_5	P_6
Job	J_1	1	5	3	2	4	5
	J_2	3	0	5	3	5	4
	J_3	8	2	4	0	2	4
	J_4	5	1	0	10	2	2
	J_5	3	3	5	1	4	4
	J_6 *(Dummy)*	0	9	3	4	3	1

FIGURE 3. Collect information from individuals and generate JP1 and JP2 arrays.

Step 4. The job-position matrix is obtained by $\frac{JP1}{JP2}$. According to Fig. 3, the job-position matrix is obtained as shown in Fig. 4.

It is worth noting that if an element in the JP2 matrix is equal to zero, the division is neglected, and the value is straightly replaced with zero.

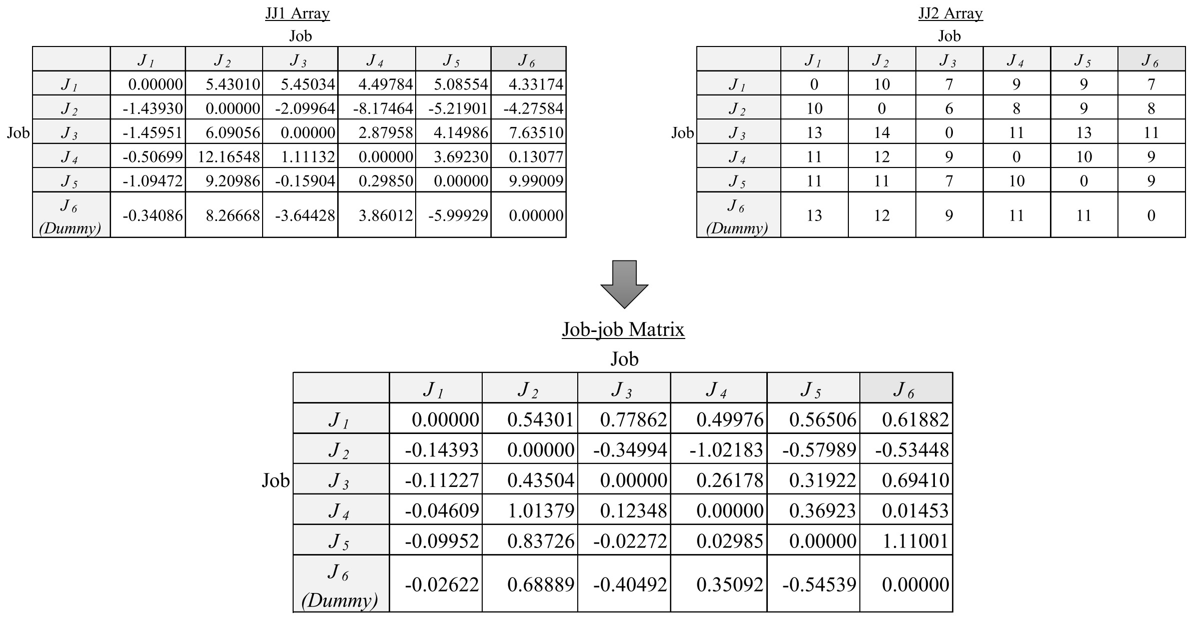


Job-position Matrix

Position

Job		P$_1$	P$_2$	P$_3$	P$_4$	P$_5$	P$_6$
Job	*J$_1$*	7.03089	0.12646	-0.05708	-1.03074	-2.06283	1.36234
	J$_2$	-0.64883	0.00000	-0.85847	-0.33732	0.68499	1.95417
	J$_3$	-0.10220	1.84972	0.67239	0.00000	0.25859	-0.52443
	J$_4$	0.54212	0.90507	0.00000	-0.39309	3.64071	-1.48765
	J$_5$	-0.99552	1.96963	0.81849	-1.94456	0.28842	-0.55827
	J$_6$ (Dummy)	0.00000	-0.79498	0.55748	3.23495	-0.04502	-3.33145

FIGURE 4. **The job-position matrix obtained based on the twenty individuals.**



JJ1 Array

Job

		J$_1$	*J$_2$*	*J$_3$*	*J$_4$*	*J$_5$*	*J$_6$*
Job	*J$_1$*	0.00000	5.43010	5.45034	4.49784	5.08554	4.33174
	J$_2$	-1.43930	0.00000	-2.09964	-8.17464	-5.21901	-4.27584
	J$_3$	-1.45951	6.09056	0.00000	2.87958	4.14986	7.63510
	J$_4$	-0.50699	12.16548	1.11132	0.00000	3.69230	0.13077
	J$_5$	-1.09472	9.20986	-0.15904	0.29850	0.00000	9.99009
	J$_6$ (Dummy)	-0.34086	8.26668	-3.64428	3.86012	-5.99929	0.00000

JJ2 Array

Job

		J$_1$	*J$_2$*	*J$_3$*	*J$_4$*	*J$_5$*	*J$_6$*
Job	*J$_1$*	0	10	7	9	9	7
	J$_2$	10	0	6	8	9	8
	J$_3$	13	14	0	11	13	11
	J$_4$	11	12	9	0	10	9
	J$_5$	11	11	7	10	0	9
	J$_6$ (Dummy)	13	12	9	11	11	0

Job-job Matrix

Job

		J$_1$	*J$_2$*	*J$_3$*	*J$_4$*	*J$_5$*	*J$_6$*
Job	*J$_1$*	0.00000	0.54301	0.77862	0.49976	0.56506	0.61882
	J$_2$	-0.14393	0.00000	-0.34994	-1.02183	-0.57989	-0.53448
	J$_3$	-0.11227	0.43504	0.00000	0.26178	0.31922	0.69410
	J$_4$	-0.04609	1.01379	0.12348	0.00000	0.36923	0.01453
	J$_5$	-0.09952	0.83726	-0.02272	0.02985	0.00000	1.11001
	J$_6$ (Dummy)	-0.02622	0.68889	-0.40492	0.35092	-0.54539	0.00000

FIGURE 5. **The job-job matrix is obtained based on the twenty individuals.**

C. JOB-JOB ($n + m - 1$, $n + m - 1$) MATRIX

The precedence information between the jobs in a job sequence is also an important factor in determining an optimal schedule [33]. Thus, we use the job-job matrix to provide the signals for any pair of jobs in the job-sequence based on the global feature values. The generation of the job-job matrix is similar to the steps of the job-position matrix and described in the following.

Step 1. We initialize two two-dimensional arrays (JJ1 and JJ2) with $n+m-1$ rows and $n+m-1$ columns, where each element in these matrices is equal to zero.

Step 2. For each selected individual in the population, if job i is before job j, the values of JJ1 (i, j) and JJ1 (j, i) is added by its feature, and the element of JJ2(i, j) and JJ2(j, i) are added by 1.

Step 3. Repeat Step 2 until all individuals of the population have been selected and obtain the JJ1 and JJ2 for this population.

Step 4. The job-job matrix is obtained by $\frac{JJ1}{JJ2}$.

Using these steps for the job-job matrix, for the same population, the JJ1, JJ2, and job-job matrices are obtained and shown in Fig. 5.

For unrelated parallel-machine scheduling problems, there are two interrelated subproblems: first, one needs to determine which jobs are to be allocated to which machines; and, second, one needs to determine the sequence of the jobs allocated to each machine. The above job-position and job-job matrices mainly provide information about the job sequence, whereas the information of the second subproblem is provided by the following job-machine and machine-job matrices. The processes are demonstrated in the following

D. JOB-MACHINE (n, m) MATRIX

The job-machine matrix aims to find the information on which jobs are assigned to which machine. The steps of building the job-machine matrix are also similar to those of the job-position matrix and are described below.

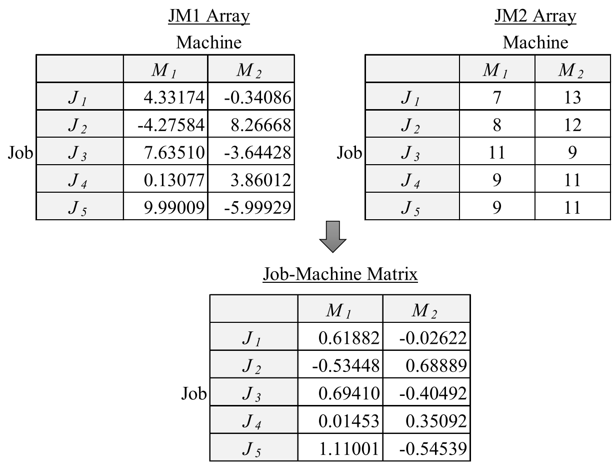
Step 1. We initialize two two-dimensional arrays (JM1 and JM2) with n rows and m columns, where each element in these matrices is equal to zero.

Step 2. We construct the JM1 and JM2 arrays, which is similar to Step 2 of the job-position matrix. If job i is assigned at machine j, the values of JM1(i, j) is added by its feature value, and the value of JM2(i, j) is added by 1.

Step 3. Repeat Step 2 until all individuals of the population have been selected and obtain JM1 and JM2 for this population.

Step 4. The job-machine matrix is obtained by $\frac{JM1}{JM2}$.

For the 5-job example of Table 1, the job-machine matrix is obtained as shown in Fig. 6.



JM1 Array

Job	Machine M_1	Machine M_2
J_1	4.33174	-0.34086
J_2	-4.27584	8.26668
J_3	7.63510	-3.64428
J_4	0.13077	3.86012
J_5	9.99009	-5.99929

JM2 Array

Job	Machine M_1	Machine M_2
J_1	7	13
J_2	8	12
J_3	11	9
J_4	9	11
J_5	9	11

Job-Machine Matrix

Job	M_1	M_2
J_1	0.61882	-0.02622
J_2	-0.53448	0.68889
J_3	0.69410	-0.40492
J_4	0.01453	0.35092
J_5	1.11001	-0.54539

FIGURE 6. The job-machine matrix is obtained based on the twenty individuals.

E. MACHINE-JOB $(n + 1, m)$ MATRIX

The machine-job matrix is applied to forecast the number of jobs processed in each machine. The steps are described below.

Step 1. We initialize two two-dimensional arrays (MJ1 and MJ2) with $n + 1$ rows and m columns, where row number starts with zero, and each element in these matrices is equal to zero.

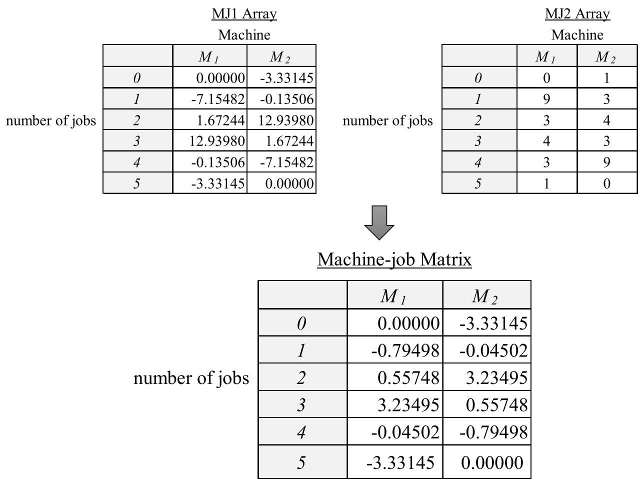
Step 2. For each selected individual in the population, if the number of jobs processed on machine j is i, the values of MJ1 (i, j) is added by its feature value, and the value of MJ2(i, j) is added by 1.

Step 3. We repeat Step 2 until all individuals of the population have been selected and obtain the MJ1 and MJ2 for this population.

Step 4. The Machine-job matrix is obtained by $\frac{MJ1}{MJ2}$.

Based on the above steps, the machine-job matrix is obtained for the same population with 20 instances and is displayed in Fig. 7.

Following the construction steps of each matrix, the computational complexity of each matrix is O $(L \cdot (n + m))$.



MJ1 Array

number of jobs	Machine M_1	Machine M_2
0	0.00000	-3.33145
1	-7.15482	-0.13506
2	1.67244	12.93980
3	12.93980	1.67244
4	-0.13506	-7.15482
5	-3.33145	0.00000

MJ2 Array

number of jobs	Machine M_1	Machine M_2
0	0	1
1	9	3
2	3	4
3	4	3
4	3	9
5	1	0

Machine-job Matrix

number of jobs	M_1	M_2
0	0.00000	-3.33145
1	-0.79498	-0.04502
2	0.55748	3.23495
3	3.23495	0.55748
4	-0.04502	-0.79498
5	-3.33145	0.00000

FIGURE 7. The machine-job matrix is obtained based on the twenty individuals.

IV. FEATURE-EXTRACTION-BASED ITERATED ALGORITHM (FEBIA)

In this paper, we propose a population-based stochastic search algorithm, called a feature-extraction-based iterated algorithm (FEBIA), where a group of solutions is maintained in each iteration using linking encoding (LE) procedures based on the mentioned feature-extraction matrices to solve $R_m|r_j, d_{ij} \leq T_i, fpa|C_{max}$ problems.

Similar to many metaheuristic algorithms such as genetic algorithms (GAs) [34], particle swarm optimization (PSO) algorithms [35], and simulated annealing (SA) algorithms [36], the individuals of the group in the FEBIA are maintained by certain operators, and each individual is represented by an encoding scheme and then decoded to the corresponding objective value. The encoding scheme makes a solution recognizable, whereas the decoding obtains a feasible solution for algorithms. Before demonstrating the proposed FEBIA, the encoding and decoding scheme for the $R_m|r_j, d_{ij} \leq T_i, fpa|C_{max}$ problem is described as follows.

A. ENCODING/DECODING SCHEME

Regarding the encoding scheme, Borovska [37] noted that permutation coding is the best method for ordering problems, and to date, many meta-heuristic algorithms have used permutation coding to solve parallel machine scheduling problems in the literature (Vallada Regalado and Ruiz Garcia [38], Borovska [37]). Thus, we also used permutation coding similar to that of Cheng and Gen [39], where an $(n + m - 1)$ array is used to represent an individual. For 5-job instance with two machine in Table 1, the corresponding solution encoding is represented as 1-3-5-6-2-4 as shown in Fig. 10, in which the dummy job (job 6) is used to separate machines, indicating jobs 1, 3, and 5 to machine 1, and the rest of the jobs 2 and 4 are to allocated machine 2.

For our considered problem, there are three decisions to be made: (1) allocating jobs to each machine; (2) the sequence of jobs to be processed on each machine; and (3) the timing point

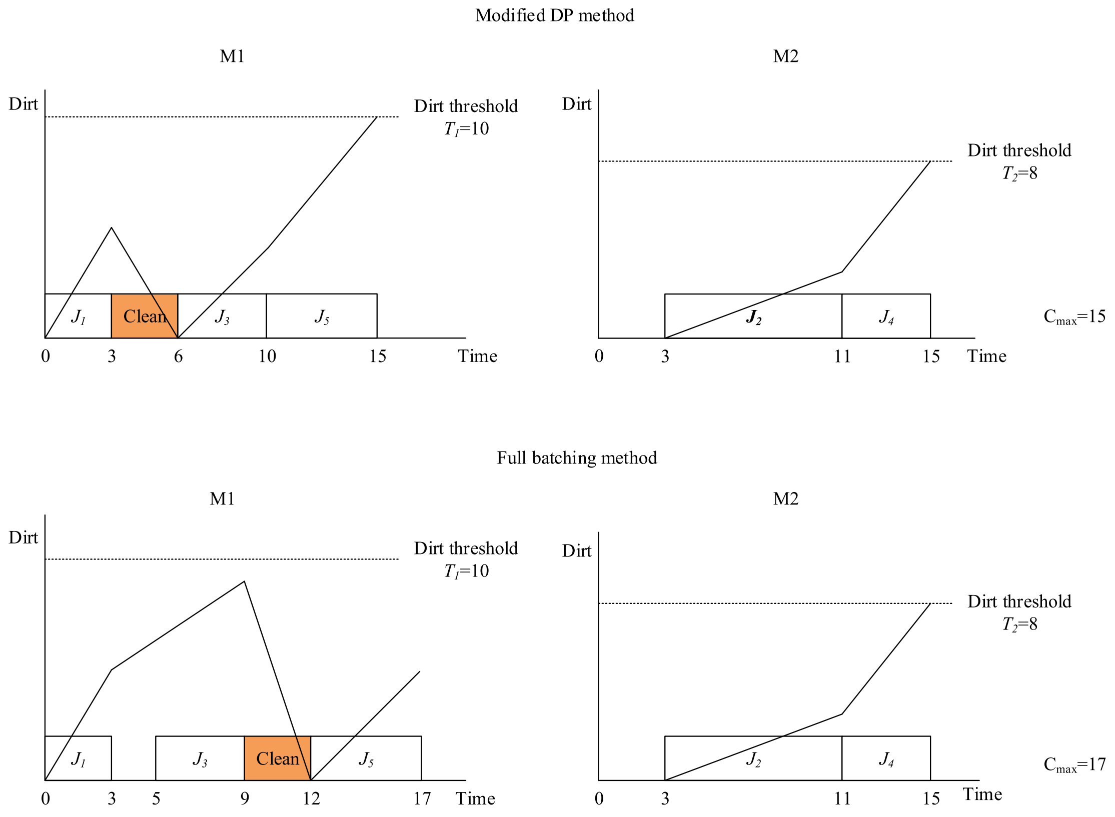


FIGURE 8. Comparison between the modified DP and the full batching methods.

of cleaning activity for each machine under the limit of dirt left in each machine. The proposed permutation coding could solve the first two decisions as shown in Fig. 10. Regarding the third decision, we modify the dynamic programming (DP) method provided by Pang *et al.* [28] to solve this issue. Additionally, we use this instance of Table 1 for comparison with the full batching method commonly used in the bin-packing problem [40]. The comparison is shown in Fig. 8. In the third decision, the amount of dirt in each machine not exceeding the threshold value is a hard constraint. That is, the modified DP method is used to minimize the makespan; in the meantime, the obtained solution has to satisfy the constraint of dirt left in each machine. Thus, a certain relationship between the solution and dirt left in the final state was not verified in this paper. Fig. 8 shows that using the two batching methods, the amount of dirt in the final state of the machine is different, and using the modified DP method, the final solution (i.e., makespan) is shorter, but there is more dirt left in the machine. However, another example in Fig. 9 shows that there is different dirt left in the machine at the final state even when the makespan is the same. In this paper, we adopt the modified DP method to obtain complete feasible solution. For the sake of brevity, for the procedures of the DP method, the readers can refer to the study of Pang *et al.* [28].

B. LINKAGE ENCODING (LE) PROCEDURES

The LE procedure aims to generate ten individuals based on the four feature-extraction matrices mentioned above for the next iteration of the FEBIA. The ten different LE procedures described in the following.

1) LE01 PROCEDURE

The LE01 procedure employs an assignment heuristic method to obtain a permutation encoding based on the job-position matrix. The steps of LE01 are given in the following.

Step 1. Initialize a permutation encoding with $(n + m - 1)$ items.
Step 2. Find the largest feature value (v^*_{ij}) among the job-position matrix and place job i into position j in the permutation encoding.
Step 3. Replace all values (v_{ij}) in row i and column j of the job-position matrix with null.
Step 4. Repeat step 2 until the values (v_{ij}) in the job-position matrix are equal to null.

2) LE02 PROCEDURE

The LE02 procedure employs the roulette wheel technique to obtain a permutation encoding based on the job-position matrix. The steps of LE02 are given below.

Step 1. Initialize a permutation encoding with $(n + m - 1)$ items.
Step 2. Calculate the range value (R_j) for each column of the job-position matrix, i.e., max (v_{*j})-min (v_{*j}). Select the column j' with the largest $R_{j'}$.

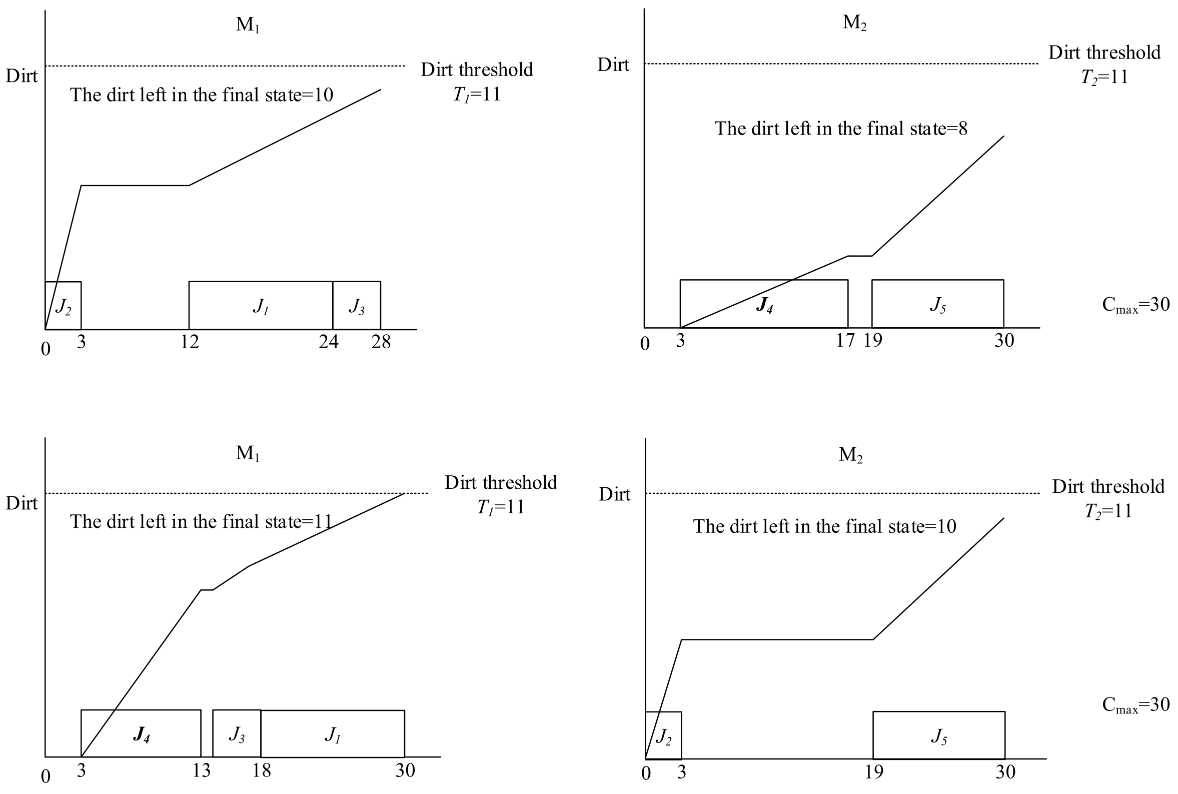


FIGURE 9. The same makespan with different dirt left in the final state.

1	3	5	6 (Dummy job)	2	4

FIGURE 10. Permutation coding.

Step 3. Calculate the probability $prob_i$ for each row i of column j' based on its corresponding $v_{ij'}$, where $i = 1, \ldots, (n+m-1)$. If $v_{ij'} = null$, then $prob_i = 0$.
Step 4. Generate a random number r from the range (0, 1].
Step 5. Calculate the cumulative probability q_i for each i of column j', where $i = 1, \ldots, (n+m-1)$.
Step 6. Select job l such that $q_{l-1} < r \leq q_l$.
Step 7. Place the selected job l into position j' in the permutation encoding.
Step 8. Replace all values $v_{lj'}$ in row l and column j' of the job-position matrix with null.
Step 9. Repeat step 2 until the values of v_{ij} in the job-position matrix are equal to null.

3) LE03 PROCEDURE

LE03 is similar to LE02. The difference is that the calculations for the roulette wheel are based on the rows of the job-position matrix. The steps of LE03 are given below.

Step 1. Initialize a permutation encoding with $(n+m-1)$ items.
Step 2. Calculate the range value (R_i) for each row of the job-position matrix, i.e., max (v_{i*})-min (v_{i*}). Select the row i' having the largest $R_{i'}$
Step 3. Calculate the probability $prob_j$ for each column j of row i' based on its corresponding $v_{i'j}$, where $j = 1, \ldots, (n+m-1)$. If $v_{i'j} = null$, then $prob_j = 0$.
Step 4. Generate a random number r from the range (0, 1]
Step 5. Calculate the cumulative probability q_j for each j of row i', where $j = 1, \ldots, (n+m-1)$.
Step 6. Select the position h such that $q_{h-1} < r \leq q_h$. The selected position is denoted as h.
Step 7. Place job i' into the selected position h in the permutation encoding.
Step 8. Replace all values of $v_{i'h}$ in row i' and column h of the job-position matrix with null.
Step 9. Repeat step 2 until the values of v_{ij} in the job-position matrix are equal to null.

For the three procedures of LE01, LE02, and LE03, the computational complexity is $O((n+m)^2)$, where n is the number of jobs and m is the number of machines.

The following four procedures (LE04, LE05, LE06, and LE07) are developed based on the job-job matrix.

4) LE04 PROCEDURE

The LE04 procedure tries to find the higher priority jobs in the permutation encoding. In the job-job matrix, the value of a pair (i, j) indicates the priority that job i is before job j in the permutation encoding from the past population. Therefore, we develop the LE04 method using the information of the job-job matrix. The steps of LE04 are given below.

Step 1. Sum the values for each row of the job-job matrix, i.e., $Sum_i = \sum_{j=1}^{n+m-1} u_{ij}$.

Step 2. Obtain a permutation encoding by sorting the jobs in descending order of their Sum_i.

5) LE05 PROCEDURE

LE05 is also developed to obtain a permutation encoding based on the job-job matrix. The steps of LE05 are given below.

Step 1. Sum the values for each column of the job-job matrix, i.e., $Sum_j = \sum_{i=1}^{n+m-1} u_{ij}$.

Step 2. Obtain a permutation encoding by sorting the jobs in increasing order of their Sum_j.

6) LE06 PROCEDURE

LE04 and LE05 only use a simple index (Sum_i or Sum_j, respectively) to generate a permutation encoding. In LE06, both the Sum_i and Sum_j indexes are considered. The steps of LE06 are given below.

Step 1. Sum the values for row i and column j of the job-job matrix, i.e., $RSum_i = \sum_{j=1}^{n+m-1} u_{ij}$ and $CSum_j = \sum_{i=1}^{n+m-1} u_{ij}$.

Step 2. Calculate the priority (PV_i) for job i using the formula ($CSum_i - RSum_i$).

Step 3. Obtain a permutation encoding by sorting the jobs in increasing order of their PV_i.

For the three procedures of LE04, LE05, and LE06, the computational complexity is $O((n+m)\, log(n+m))$.

7) LE07 PROCEDURE

In the job-job matrix, a positive value for a pair (i, j) implies that job i is before job j; in contrary, job i is after job j if the value is negative. Therefore, we can count the successive number of jobs for job i according to positive values in the job-job matrix. The steps of LE07 are given below.

Step 1. Set $k = 1$.

Step 2. Based on the job-job matrix, count the number (CN_i) of rows where u_{ij} is greater than zero, and sum the u_{ij} values for the rows where $u_{ij} > 0$, i.e., Sum_i.

Step 3. Choose the row i' with the largest CN_i. If there is a tie, choose the row with the largest Sum_i; otherwise, choose arbitrarily. Place job i' in position k.

Step 4. Replace the values of $u_{i'j}$ with null for $j = 1, \ldots, (n+m-1)$. $k = (k+1)$.

Step 5. If $k = (n+m-1)$, place the unassigned job into the last position. Then, stop the procedure. Otherwise, return to step 2.

8) LE08 PROCEDURE

The feature-extraction procedures mentioned above only use a single matrix such as the job-position or the job-job matrix. We integrated the information from the job-position and job-job matrices to develop LE08 and the steps of LE08 are given below.

Step 1. Calculate the mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$ based on the values in the job-position matrix. Set a threshold value (ω) equal to $\hat{\mu} + 3\hat{\sigma}$.

Step 2. For each column j, find the value ($v_{i'j'}$) of the job-position matrix that is greater than the threshold value (ω). If there is a tie, choose the largest $v_{i'j'}$ and place job i' in position j' in the permutation encoding.

Step 3. If the permutation encoding is null, randomly generate a permutation encoding. Otherwise, go to step 4.

Step 4. Find the position j' first occupied by a job in the permutation encoding. If $j' = 1$, go to step 5 (forward filling). Otherwise, go to step 9 (first backward filling and then forward filling).

Step 5. Update the job-job matrix by replacing the values of the occupied rows and columns with null.

Step 6. Sum the values for each row of the job-job matrix, i.e., $Sum_i = \sum_{j=1}^{n+m-1} u_{ij}$.

Step 7. Select the job i' with the largest $Sum_{i'}$ and put it into the next unoccupied position in the permutation encoding. If there is a tie, choose randomly.

Step 8. Return to step 5 until all successive positions are occupied, and then stop.

Step 9. Update the job-job matrix by replacing the values of the occupied rows and columns with null.

Step 10. Sum the values for each row of the job-job matrix, i.e., Sum_i.

Step 11. Select job i with the largest Sum_i and put it into the previous unoccupied position in the permutation encoding. If there is a tie, choose randomly.

Step 12. Return to step 9 until all previous positions are occupied.

Step 13. If one of the successive positions is not occupied, return to step 5; otherwise, stop.

C. LE09 PROCEDURE

LE09 integrates the information of the job-position, job-machine and machine-job matrices to generate a new permutation encoding. First, the Machine-Job matrix is used to find the number of jobs processed in each machine. Next, we applied the job-machine matrix to find which jobs were processed on which machine. Finally, we use the job-position matrix to determine the job sequence in each machine. The detail steps of LE09 are listed below.

Step 1. $j = 1$, and $k = 0$

Step 2. According to the Machine-Job matrix, find the largest value ($\alpha^*_{i'j}$) in column j, i.e., machine j. The corresponding row number (i') is the number of jobs (NM_j) processed on machine j, where $k = \left(k + i'\right)$.

Step 3. Delete column j from the Machine-Job matrix and replace the values of α_{ij} for $i > (n-k)$ with null for the Machine-Job matrix. $j = (j+1)$

Step 4. If $j \leq m$, return to step 2; otherwise, go to step 5.
Step 5. Set $j = 1$.
Step 6. Choose the number of jobs (NM_j) with the largest values of $\beta^*_{i'j}$ in the job-machine matrix and put the chosen jobs into the set for machine j (MJ_j).
Step 7. Delete the chosen row from the job-machine matrix. $j = (j + 1)$.
Step 8. If $j \leq m$, return to Step 6; otherwise, go to step 9.
Step 9. Set $j = 1$.
Step 10. Construct a sub job-position matrix from the current job-position matrix based on the chosen jobs in the set of machine j (MJ_j).
Step 11. Apply the assignment heuristic method to determine the job sequence based on the sub job-position matrix and put the jobs into the permutation encoding accordingly.
Step 12. If $j < m$, add the number (i.e., $n + j$) at the next position in the permutation encoding, $j = (j+1)$ and return to step 10. Otherwise, stop the procedure.

D. LE10 PROCEDURE

LE10 integrates the information of the job-job, job-machine and machine-job matrices to generate a new permutation encoding. LE10 is similar to LE09. The difference is the steps of determining the job sequence; thus, steps 1 to 8 of LE10 are the same as steps 1 to 8 of LE09.

Steps 1 to 8. Use steps1 to 8 of LE09.
Step 9. Set $j = 1$.
Step 10. Construct a sub job-job matrix from the current job-job matrix based on the chosen jobs in the set of machine j (MJ_j).
Step 11. Sum the values for each row of the sub job-job matrix, i.e., Sum_i.
Step 12. Obtain a permutation encoding by sorting the jobs in decreasing order of their Sum_i. $j = (j + 1)$.
Step 13. If $j < m$, add the number (i.e., $n + j$) at the next position in the permutation encoding, $j = (j + 1)$ and return to step 10. Otherwise, stop the procedure.

For the four procedures of LE07, LE08, LE09, and LE10, the computational complexity is $O((n + m)^2)$.

Regarding the mentioned 5-job instance, the four feature-extraction matrices are given earlier in Figs. 4-7, therefore, ten permutation lists could be obtained by the LE procedures, and then the solutions are obtained by the modified DP methods, as illustrated in Table 2. All new individuals obtained by LE procedures are put into the next iteration. Notably, the optimal solution of the 5-job instance is 15.

In Table 2, it can be seen that using LE procedures to generate ten new individuals in each iteration is likely to promote the FEBIA to search for good individuals because the optimal or better solutions are included in those obtained by LE procedures. On the other hand, Table 2 also shows that LE procedures will impede the diversity of the population such that it is trapped in local optima. To overcome this drawback, we generate other individuals randomly in each iteration of the FEBIA to diverse the population. That is, L individuals of the population in the FEBIA are randomly generated in the initial iteration, and in a subsequent iteration, the population includes that of the best individual of the current population (i.e., adopting elite strategy), ten individuals are generated by the LE procedures, and the remaining individuals, i.e., L-11, are randomly generated. In the FEBIA, all new individuals with $(n + m - 1)$ elements are manipulated for the next iteration based on permutation encoding; thus, their corresponding solution schedules are feasible by the DP method. Additionally, in each iteration, each matrix is updated by the following equation (seen in Fig. 11) for the next iteration. The process continues until the stopping criterion is met. Fig. 11 shows the framework of the proposed FEBIA.

TABLE 2. The ten solutions obtained by LE procedures for the 5-job instance.

Procedures	Matrix used	Permutation lists	Makespan
LE01	Job-position matrix	1–5–3–6–4–2	15
LE02	Job-position matrix	1–3–5–6–4–2	16
LE03	Job-position matrix	2–5–3–6–4–1	22
LE04	Job-job matrix	1–5–3–4–6–2	21
LE05	Job-job matrix	1–4–3–5–6–2	21
LE06	Job-job matrix	1–5–3–4–6–2	21
LE07	Job-job matrix	1–3–5–6–4–2	16
LE08	Job-position and job-job matrix	1–5–3–6–4–2	15
LE09	Job-position, job-machine and machine-job matrices	1–5–3–6–4–2	15
LE10	Job-job, job-machine and machine-job matrices	1–3–5–6–4–2	16

It is worth noting that the FEBIA almost does not need to tune parameters and the simple framework is also easy to implement for other problems.

V. COMPUTATIONAL EXPERIMENT

To examine the efficiency of our proposed algorithm, we test our FEBIA, the basic PSO and simple GA on different sets of randomly generated instances.

The experiment is conducted for fifteen different problem sizes, that is, n = 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 400, and 500. The release time r_j, processing time p_{ij} and dirt d_{ij} of the problems are generated using discrete uniform distributions over [0, 20], [3, 15], and [1, 9], respectively. For machines, the dirt threshold values are generated in the ranges of [10, 12], [10, 15], and [15, 25]. Finally, the maintenance times for machines were generated in the ranges of [3, 5], [5, 10], and [10, 20]. Ten instances are generated for each combination; thus, there are a total of 90 instances for each problem size.

TABLE 3. The parameter settings of the GA method in the preliminary experiment.

	Parameter settings				
Reproduction(r_{rep})	5%	10%	15%	20%	30%
Crossover(r_c)	95%	90%	85%	80%	70%
Mutation(r_m)	0.1				

TABLE 4. The parameter settings for PSO, GA, and FEBIA.

	PSO	GA	FEBIA
Population size (P_s)	$(2 \times n \times m)$		
Parameter values	$c_1 = c_2 = 2$ $\lvert V_{max}\rvert = \lvert X_{max}\rvert = 5$	$(r_{rep}, r_c, r_m) = (0.1, 0.9, 0.1)$ $(r_{rep}, r_c, r_m) = (0.15, 0.85, 0.1)$	None
Stopping condition (TL)	$(0.005 \times n \times m)$		

To obtain the benchmark solutions, the MIP model is solved using IBM ILOG CPLEX Optimization studio version 12.7.1 on a PC with an Intel Xeon E-2124 3.4 G-Hz CPU with 32 GB of DRAM, and the time limit of the MIP is set to 1800 seconds. For the basic PSO algorithm [35], that is, $v_i(t+1) = v_i(t) + c_1 r_1 \left(pb_i - x_i(t)\right) + c_2 r_2 \left(pb_g - x_i(t)\right)$, $x_i(t+1) = x_i(t) + v_i(t+1)$, where c_1 and c_2 are learning factors, usually $c_1 = c_2 = 2$ [35], [41], and to control the excessive roaming of particles outside the search space, the values of x_i(t) and v_i(t) are limited, i.e., v_i (t) $\leq \lvert V_{max}\rvert = 5$, and x_i (t) $\leq \lvert X_{max}\rvert = 5$ in this paper. Regarding a simple GA to generate new population, $r_{rep} \times P_s$ individuals with the lowest makespan values from previous generation are automatically copied to the next generation, where r_{rep} and P_s are the reproduction rate and population size. Two-point crossover is used to generate the rest of individuals for the next generation. Swap mutation is performed on offspring with a probability r_{mut} to make the offspring more diversified. For the three algorithms (PSO, GA, and FEBIA), five replications are performed for each instance. These algorithms are coded in C++ and run on the same personal computer mentioned above.

For the GA, we first conduct a preliminary experiment. The parameter values are set in Table 3. For each instance, the minimum, average, and maximum makespan among 5 replications are obtained, and then the mean values for the minimum, average, and maximum makespan are

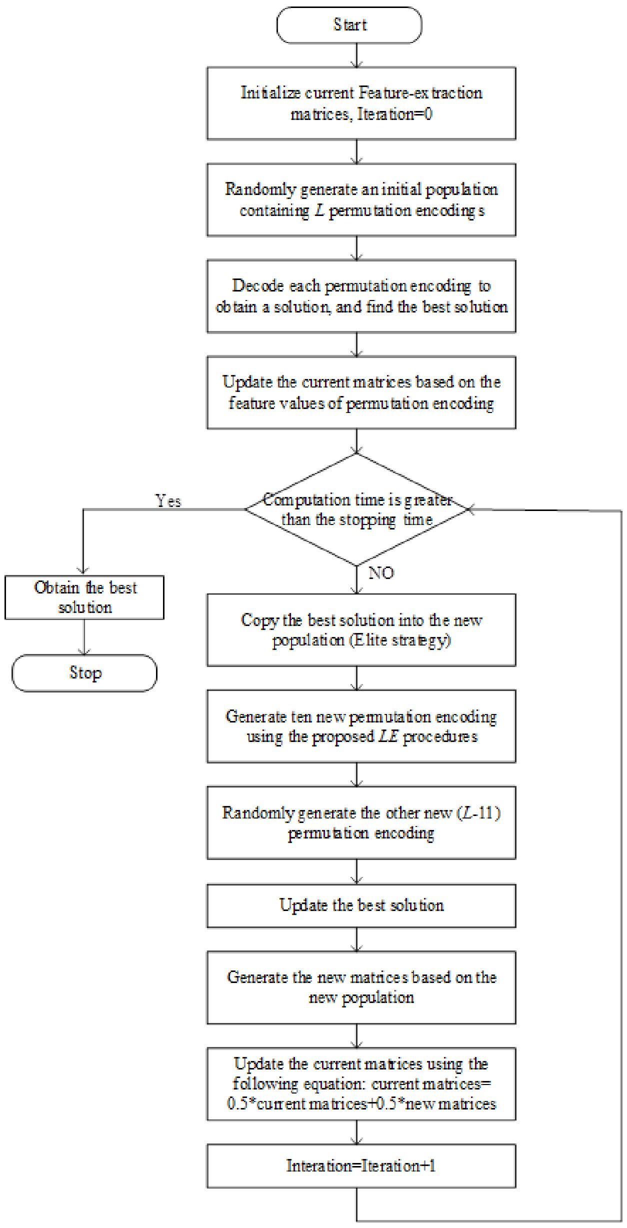

Start
Initialize current Feature-extraction matrices, Iteration=0
Randomly generate an initial population containing L permutation encodings
Decode each permutation encoding to obtain a solution, and find the best solution
Update the current matrices based on the feature values of permutation encoding
Computation time is greater than the stopping time
Yes
NO
Obtain the best solution
Stop
Copy the best solution into the new population (Elite strategy)
Generate ten new permutation encoding using the proposed LE procedures
Randomly generate the other new (L-11) permutation encoding
Update the best solution
Generate the new matrices based on the new population
Update the current matrices using the following equation: current matrices= 0.5*current matrices+0.5*new matrices
Interation=Iteration+1

FIGURE 11. The framework of the proposed FEBIA.

TABLE 5. The results obtained by PSO, GA and the FEBIA compared with the MIP model.

Number of jobs	*PSO*			GA			FEBIA		
	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}
5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
6	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
10	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

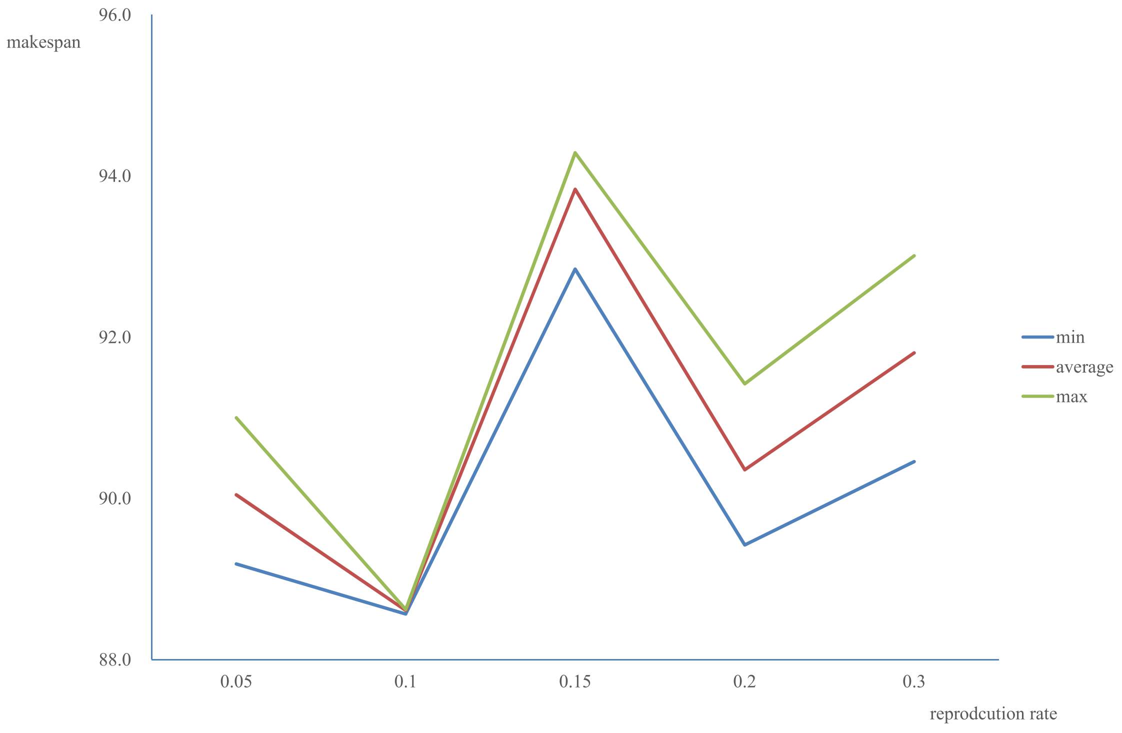


FIGURE 12. The results under different reproduction rate for $n = 20$.

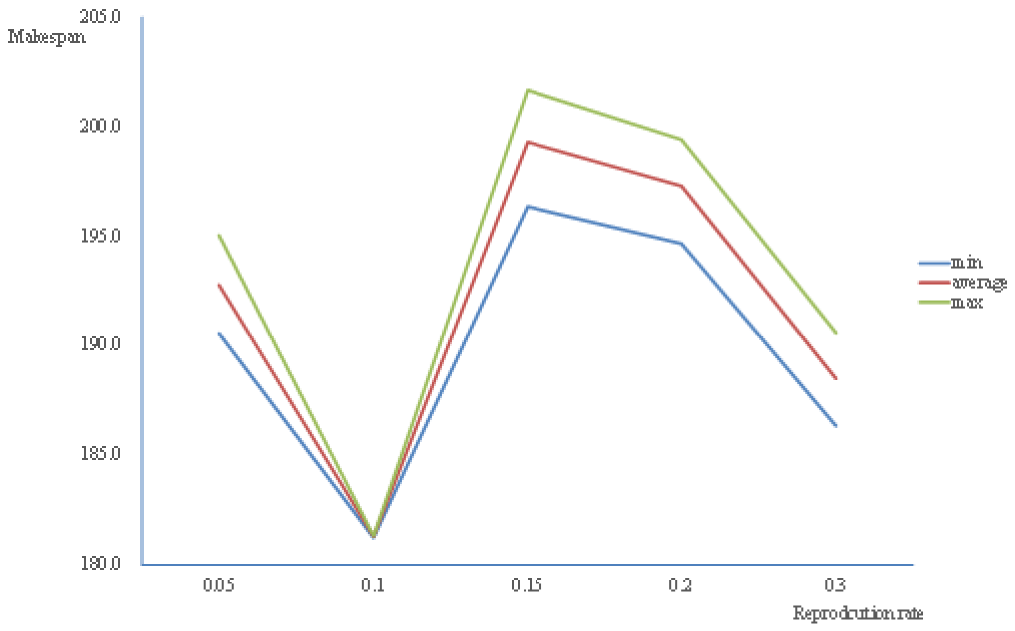


FIGURE 13. The results under different reproduction rate for $n = 40$.

calculated among ninety instances for each problem. The comparison results under different parameter values shown in Figs. 12-16. These results indicate that when the number of jobs is less than 100, the parameter setting of ($r_{rep} = 0.1, r_c = 0.9, r_{mut} = 0.1$) is better. However, when the number of jobs is equal to 100, the parameter setting of ($r_{rep} = 0.15, r_c = 0.85, r_{mut} = 0.1$) seems to improve. Table 4 shows our parameter settings for PSO, GA, and FEBIA in the computational experiment, where for a fair comparison, the number of individuals (P_s) in the population and stopping condition (TL) are the same setting depending on the problem sizes.

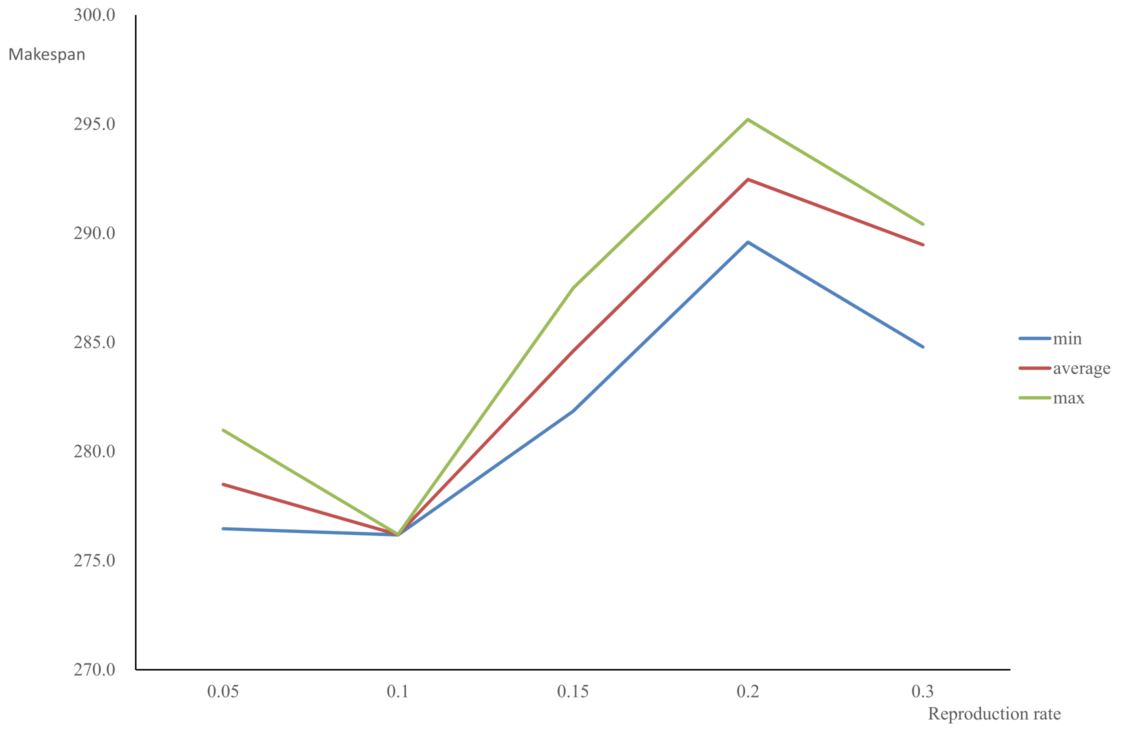


FIGURE 14. **The results under different reproduction rate for n = 60.**

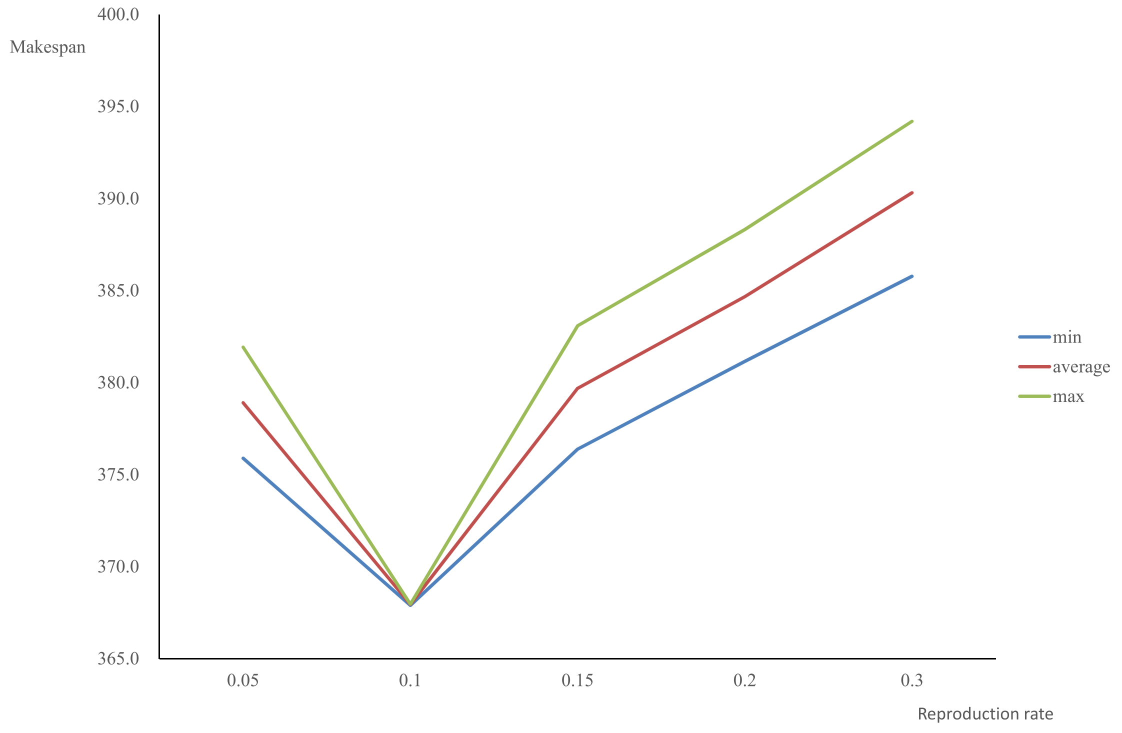


FIGURE 15. **The results under different reproduction rate for *n* = 80.**

In this paper, the average relative percentage deviation ($ARPD$) is adopted as a performance index, and the formula is as follows:

$$ARPD = \sum\nolimits_1^R \left(\frac{(C_{max}(Alg) - Best) \times 100}{Best}\right) \Big/ R,$$

where $C_{max}(Alg)$ denotes the makespan generated by the PSO, GA, and FEBIA algorithm in each run, $Best$ is the best solution obtained by the algorithm or the optimal solution by the MIP model, and R is the number of replications for each instance, i.e., R is equal to five in this paper. In addition,

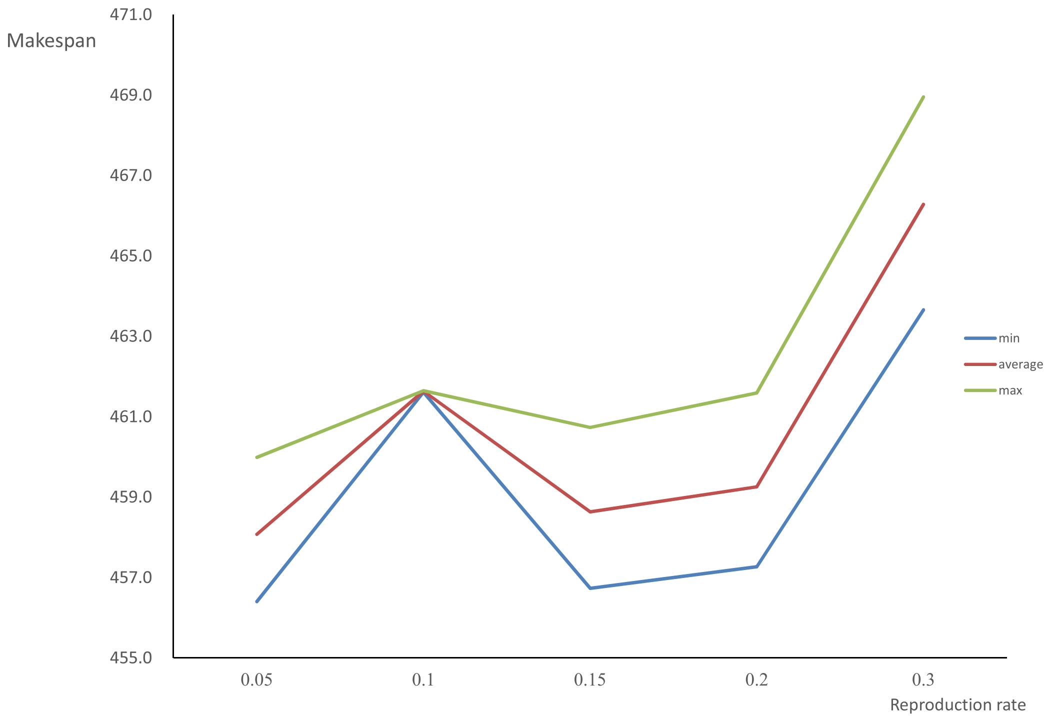


FIGURE 16. The results under different reproduction rate for $n = 100$.

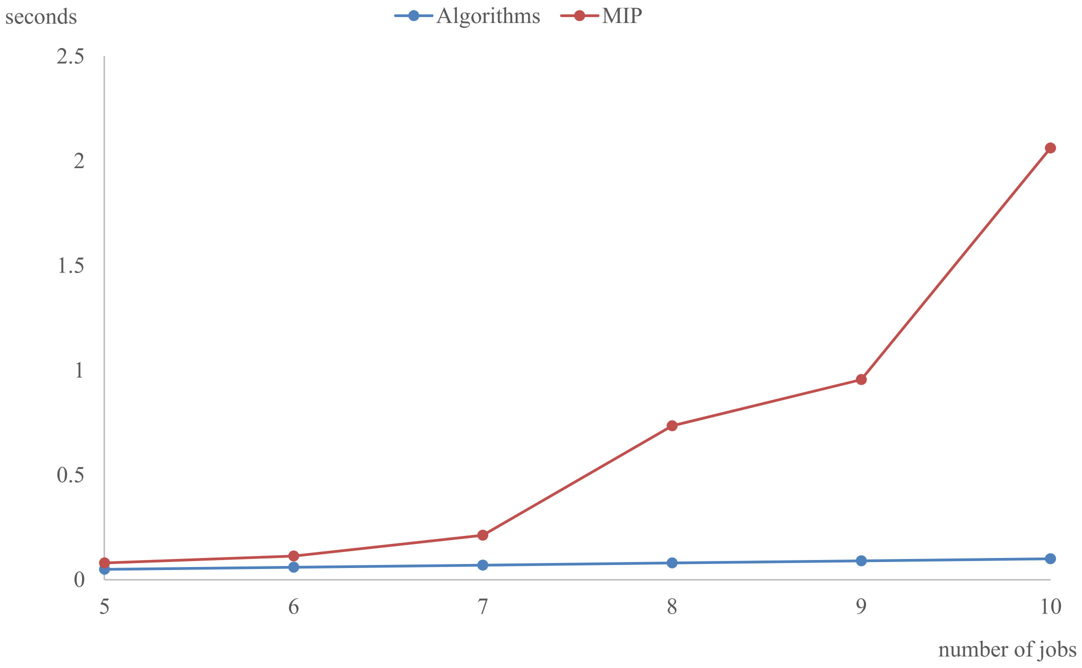


FIGURE 17. Average computational time required by MIP and algorithms.

Δ_{min}, Δ_{avg}, and Δ_{max} denotes the minimum, mean and maximum of ARPD values among 90 instances for each problem. For small-sized problems, all three algorithms (PSO, GA, and FEBIA) could consistently obtain optimal solutions in a short time compared to the MIP model shown in Table 5 and Fig. 17.

When the number of jobs is greater than or equal to 20, the solutions obtained by the MIP model in the time limit of 1800 seconds are not certain to be optimal. Table 6 shows the Δ_{min}, Δ_{avg}, and Δ_{max} values obtained for each method. In Table 6, it is obvious that the FEBIA outperforms other algorithms in terms of solution quality and stability. The proposed FEBIA

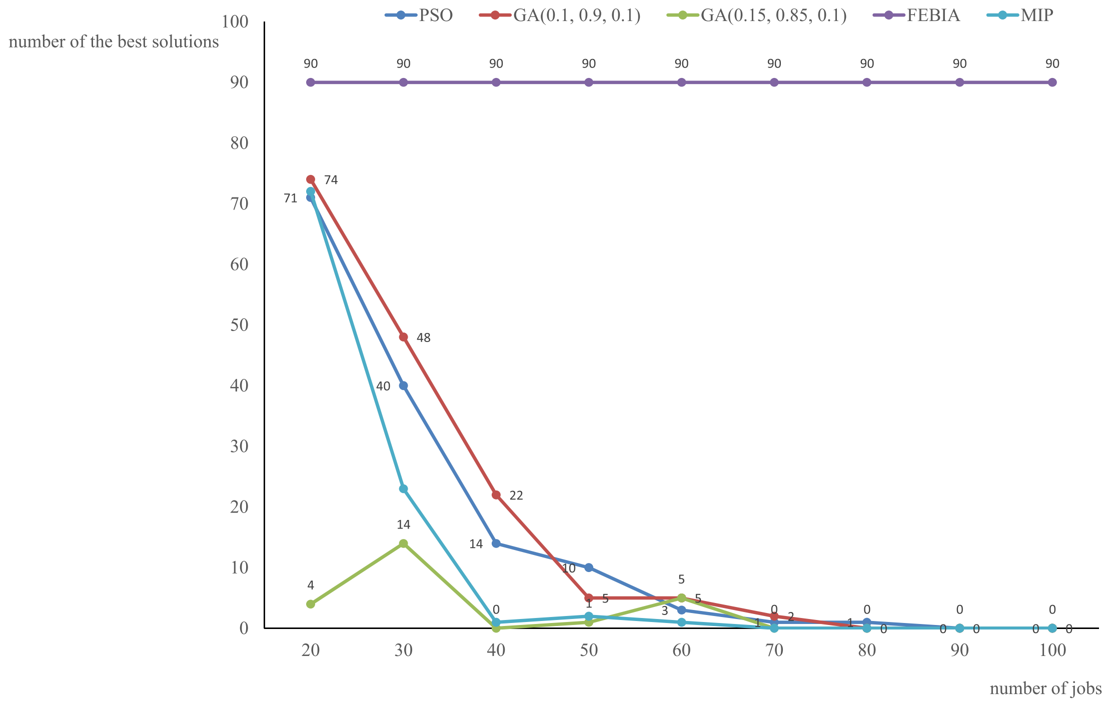


FIGURE 18. The number of the best solutions found by different algorithms.

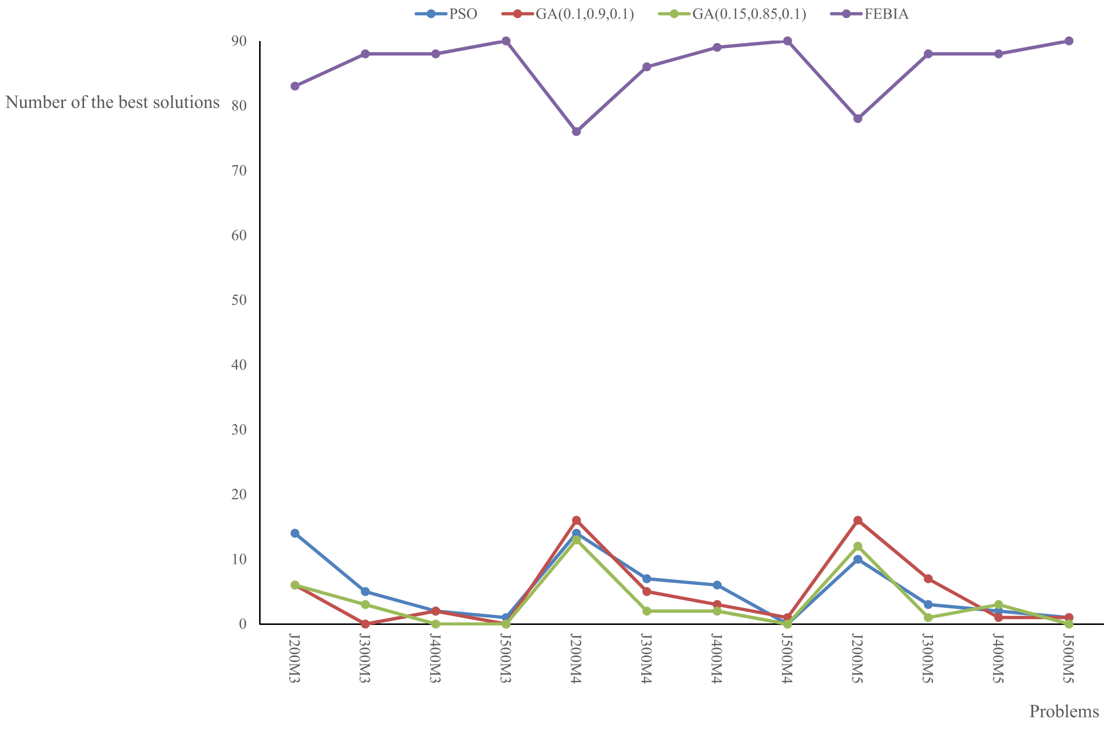


FIGURE 19. The number of the best solutions found by different algorithms for large-sized problems.

found all of the best solutions among 810 instances, while the number of the best solutions obtained by the PSO, GA (0.1, 0.9, 0.1), GA (0.15, 0.85, 0.1) and MIP incredibly decreases as problem size increases.

Table 7 provides the comparison results for large-sized problems, the results reveal that FEBIA performed very better in terms of solution quality and stability than the other compared algorithms. From Tables 6 and 7, the

TABLE 6. The results obtained by MIP, PSO, GA and the FEBIA for medium-sized problems with 2 machines.

Number of jobs	MIP			*PSO*			GA(0.1, 0.9, 0.1)			GA(0.15, 0.85, 0.1)			FEBIA		
	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}
20	0.00	0.27	2.80	0.00	0.34	3.74	0.00	0.27	3.74	1.67	6.15	11.17	0.00	0.00	0.00
30	0.00	1.72	6.93	0.00	0.71	3.57	0.00	0.63	3.21	0.49	5.87	13.06	0.00	0.00	0.00
40	0.00	2.47	5.73	0.00	1.11	3.21	0.00	1.01	3.28	1.26	11.18	16.21	0.00	0.00	0.00
50	0.00	3.21	8.70	0.00	1.28	3.18	0.00	1.31	3.60	0.83	5.45	10.48	0.00	0.00	0.00
60	0.00	3.87	10.92	0.00	1.45	3.38	0.00	1.42	3.38	0.41	4.61	12.91	0.00	0.00	0.00
70	0.40	4.96	79.01	0.00	1.54	3.19	0.00	1.67	4.26	0.86	6.70	17.71	0.00	0.00	0.00
80	0.70	5.24	71.61	0.00	1.62	3.32	0.29	1.60	4.09	0.70	5.03	13.59	0.00	0.00	0.00
90	1.16	6.16	110.48	0.53	1.72	3.37	0.31	1.88	3.98	0.83	2.57	10.53	0.00	0.00	0.00
100	1.39	17.53	166.41	0.55	1.72	3.61	0.52	1.91	3.87	0.66	1.32	3.25	0.00	0.00	0.00

TABLE 7. Comparison of PSO, GA and the FEBIA for large-sized problems.

Number of jobs		*PSO*			GA(0.1, 0.9, 0.1)			GA(0.15, 0.85, 0.1)			FEBIA			Time* (seconds)
		Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{min}	Δ_{avg}	Δ_{max}	
200	3	0.378	0.923	2.013	0.403	0.976	2.248	0.306	0.999	2.450	0.000	0.398	1.426	3.00
300		0.167	0.942	1.899	0.206	0.969	2.504	0.191	1.006	2.849	0.000	0.335	1.214	4.50
400		0.303	1.041	3.026	0.403	1.163	5.094	0.437	1.170	3.778	0.061	0.309	0.992	6.00
500		0.280	1.131	4.475	0.282	1.224	5.131	0.337	1.386	8.017	0.018	0.256	0.790	7.50
200	4	0.446	1.237	2.727	0.312	1.216	2.412	0.464	1.193	2.751	0.065	0.646	1.794	4.00
300		0.350	1.117	3.121	0.337	1.043	2.144	0.337	1.188	3.234	0.090	0.545	2.460	6.00
400		0.337	1.064	2.199	0.383	1.131	3.337	0.337	1.188	3.234	0.032	0.487	1.751	8.00
500		0.425	1.138	2.390	0.524	1.149	2.076	0.575	1.472	3.686	0.061	0.352	1.441	10.00
200	5	0.515	1.761	5.930	0.642	1.762	3.315	0.502	1.823	4.394	0.199	1.005	2.656	5.00
300		0.490	1.672	3.399	0.293	1.667	4.343	0.663	1.765	3.578	0.057	0.716	2.420	7.50
400		0.419	1.608	3.021	0.798	1.694	3.063	0.674	1.746	3.560	0.168	0.695	1.972	10.00
500		0.640	1.632	3.188	0.743	1.719	3.098	0.698	2.505	5.126	0.111	0.597	2.099	12.50

Remark: The symbol "*" indicates that all algorithms were stopped by the time limit of $(0.005 \times n \times m)$ seconds.

GA(0.15, 0.85, 0.1) seemed more sensitive to problems than the GA(0.1, 0.9, 0.1). Fig. 19 shows that the number of the best solution found by each algorithm. The proposed FEBIA apparently performs well and the number of better solutions found by the FEBIA slightly increases as the number of jobs increases with the same number of machines.

To further show how fast the algorithms converge to a better solution, we use an instance with 100 jobs and two machine to record the best solution found by each algorithm every 0.005 seconds. As revealed in Fig. 20, the proposed FEBIA converges fast to a better solution. This result confirms that the proposed feature-extraction matrices and LE procedures could make an iterated algorithm to find better quality solutions and that the convergence of the solution can be faster. On the whole, the experimental results provide evidence that the proposed FEBIA has significant performance advantages in terms of solution quality and stability. This achievement is mainly due to the inclusion of LE procedures based on feature-extraction matrices in the FEBIA together with randomness when generating new solutions for the next iteration. Moreover, the proposed FEBIA does not require excessive parameter tuning, which is an apparent advantage

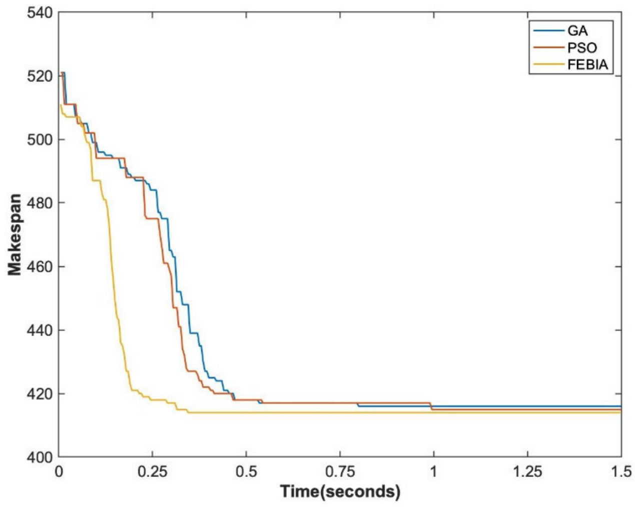


FIGURE 20. The convergence of the GA, PSO and the FEBIA.

of the FEBIA in decreasing the influence on the solution due to different parameter settings.

VI. CONCLUSION

In this paper, we introduce the unrelated parallel machine scheduling problem with periodic maintenance activities and a dynamic job release time, which is denoted as $R_m|r_j, d_{ij} \leq T_i, fpa|C_{max}$. This problem is NP-hard and is of considerable practical interest since it can be used to model many industrial applications, such as for the motivation of this article, which was semiconductor manufacturing cleaning operations. Because the considered problem is NP-hard, algorithms for obtaining an optimal solution in polynomial time are unlikely to exist. Thus, this paper develops an effective and efficient novel scheduling algorithm, namely, the FEBIA, for the $R_m|r_j, d_{ij} \leq T_i, fpa|C_{max}$ problem. The FEBIA is similar to population-based algorithms. Each individual in the population is represented by permutation encoding that considers the job allocation and job sequence simultaneously, and the corresponding solution, i.e., the makespan, is obtained via a dynamic programming algorithm. In the FEBIA, the two novel parts are the feature-extraction matrices that are used to recognize the feature values from the previously found solutions and the LE procedures that are used to find better solutions based on the matrices. Based on the results of a comprehensive experiment, the FEBIA outperformed MIP, PSO and the GA, especially for small to medium problem sizes. Additionally, the solutions of the FEBIA are better than those obtained by PSO and the GA for large problem sizes with the same computation time. This study concludes that the combination of feature-extraction matrices and LE procedures gives the FEBIA encouraging performance. Moreover, it is worth noting that the advantages of the FEBIA include its simple framework, fewer parameters to tune and quick convergence, all of which make it a promising and competitive scheduling algorithm.

In summary, the proposed FEBIA is successful at solving the $R_m|r_j, d_{ij} \leq T_i, fpa|C_{max}$ problem. The problem is a general real-world case that has not been investigated by many researchers in the literature. Our research will continue in the following directions. (1) We will consider combining the feature-extraction matrices and LE procedure with other meta heuristic algorithms such as the GA or SA. (2) In the current FEBIA, no improvement mechanism is involved, so local search procedures will be included in our algorithm. (3) Other shop environments such as open shops, flow shops, job shop problems can be studied in future research.

REFERENCES

[1] I. Adiri, J. Bruno, E. Frostig, and A. H. G. R. Kan, "Single machine flow-time scheduling with a single breakdown," *Acta Inform.*, vol. 26, no. 7, pp. 679–696, 1989.

[2] J. S. Chen, "Single machine scheduling with flexible and periodic maintenance," *J. Oper. Res. Soc.*, vol. 57, pp. 703–710, 2006.

[3] C. Y. Lee and S. D. Liman, "Single machine flow-time scheduling with scheduled maintenance," *Acta Inform.*, vol. 29, no. 4, pp. 375–382, Apr. 1992.

[4] C.-Y. Lee and Z.-L. Chen, "Scheduling jobs and maintenance activities on parallel machines," *Nav. Res. Logistics*, vol. 47, no. 2, pp. 145–165, Mar. 2000.

[5] D. Xu, K. Sun, and H. Li, "Parallel machine scheduling with almost periodic maintenance and non-preemptive jobs to minimize makespan," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1344–1349, Apr. 2008.

[6] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[7] J.-S. Chen, "Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan," *Eur. J. Oper. Res.*, vol. 190, pp. 90–102, Oct. 2008.

[8] G. Mosheiov and A. Sarig, "Scheduling a maintenance activity to minimize total weighted completion-time," *Comput. Math. Appl.*, vol. 57, pp. 619–623, Feb. 2009.

[9] A. Levin, G. Mosheiov, and A. Sarig, "Scheduling a maintenance activity on parallel identical machines," *Nav. Res. Logistics*, vol. 56, no. 1, pp. 33–41, Feb. 2009.

[10] J. Yoo and I. S. Lee, "Parallel machine scheduling with maintenance activities," *Comput. Ind. Eng.*, vol. 101, pp. 361–371, Nov. 2016.

[11] X. Qi, T. Chen, and F. Tu, "Scheduling the maintenance on a single machine," *J. Oper. Res. Soc.*, vol. 50, pp. 1071–1078, Oct. 1999.

[12] M. Sbihi and C. Varnier, "Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness," *Comput. Ind. Eng.*, vol. 55, no. 4, pp. 830–840, Nov. 2008.

[13] J.-Y. Lee, Y.-D. Kim, and T.-E. Lee, "Minimizing total tardiness on parallel machines subject to flexible maintenance," *Int. J. Ind. Eng.*, vol. 25, no. 4, pp. 472–489, 2018.

[14] A. Costa, F. A. Cappadonna, and S. Fichera, "Minimizing the total completion time on a parallel machine system with tool changes," *Comput. Ind. Eng.*, vol. 91, pp. 290–301, Jan. 2016.

[15] C. Low, M. Ji, C.-J. Hsu, and C.-T. Su, "Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance," *Appl. Math. Model.*, vol. 34, pp. 334–342, Feb. 2010.

[16] A. A. Qamhan, A. Ahmed, I. M. Al-Harkan, A. Badwelan, A. M. Al-Samhan, and L. Hidri, "An exact method and ant colony optimization for single machine scheduling problem with time window periodic maintenance," *IEEE Access*, vol. 8, pp. 44836–44845, 2020.

[17] L. Hidri, K. Alqahtani, A. Gazdar, and A. Badwelan, "Integrated scheduling of tasks and preventive maintenance periods in a parallel machine environment with single robot server," *IEEE Access*, vol. 9, pp. 74454–74470, 2021.

[18] E. Moradi, S. M. T. Fatemi Ghomi, and M. Zandieh, "An efficient architecture for scheduling flexible job-shop with machine availability constraints," *Int. J. Adv. Manuf. Technol.*, vol. 51, nos. 1–4, pp. 325–339, Nov. 2010.

[19] E. Sanlaville and G. Schmidt, "Machine scheduling with availability constraints," *Acta Inform.*, vol. 35, no. 9, pp. 795–811, 1998.

[20] G. Schmidt, "Scheduling with limited machine availability," *Eur. J. Oper. Res.*, vol. 121, pp. 1–15, Feb. 2000.

[21] Y. Ma, C. Chu, and C. Zuo, "A survey of scheduling with deterministic machine availability constraints," *Comput. Ind. Eng.*, vol. 58, pp. 199–211, Mar. 2010.

[22] W. W. Cui, Z. Lu, B. Zhou, C. Li, and X. Han, "A hybrid genetic algorithm for non-permutation flow shop scheduling problems with unavailability constraints," *Int. J. Comput. Integr. Manuf.*, vol. 29, no. 9, pp. 944–961, 2016.

[23] S. Bock, D. Briskorn, and A. Horbach, "Scheduling flexible maintenance activities subject to job-dependent machine deterioration," *J. Scheduling*, vol. 15, no. 15, pp. 565–578, Oct. 2012.

[24] L. Grigoriu and D. Briskorn, "Scheduling jobs and maintenance activities subject to job-dependent machine deteriorations," *J. Scheduling*, vol. 20, no. 2, pp. 183–197, Apr. 2017.

[25] H. Tian, S. Yu, and W. Luo, "Parallel machine scheduling on jobs and partial maintenance activities due to job-dependent machine deteriorations," *Amer. J. Math. Manage. Sci.*, vol. 38, no. 3, pp. 250–260, Jul. 2019.

[26] L.-H. Su and H.-M. Wang, "Minimizing total absolute deviation of job completion times on a single machine with cleaning activities," *Comput. Ind. Eng.*, vol. 103, pp. 242–249, Jan. 2017.

[27] Y. Chen, L.-H. Su, Y.-C. Tsai, S. Huang, and F.-D. Chou, "Scheduling jobs on a single machine with dirt cleaning consideration to minimize total completion time," *IEEE Access*, vol. 7, pp. 22290–22300, 2019.

[28] J. Pang, H. Zhou, Y.-C. Tsai, and F.-D. Chou, "A scatter simulated annealing algorithm for the bi-objective scheduling problem for the wet station of semiconductor manufacturing," *Comput. Ind. Eng.*, vol. 123, pp. 54–66, Sep. 2018.

[29] W. W. Cui and Z. Lu, "Minimizing the makespan on a single machine with flexible maintenances and jobs' release dates," *Comput. Ind. Eng.*, vol. 80, pp. 11–22, Apr. 2017.

[30] B. Detienne, "A mixed integer linear programming approach to minimize the number of late jobs with and without machine availability constraints," *Eur. J. Oper. Res.*, vol. 235, no. 3, pp. 540–552, Jun. 2014.

[31] S. Melouk, P. Damodaran, and P.-Y. Chang, "Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing," *Int. J. Prod. Econ.*, vol. 87, no. 2, pp. 141–147, Jan. 2004.

[32] F.-D. Chou, "An experienced learning genetic algorithm to solve the single machine total weighted tardiness scheduling problem," *Expert Syst. Appl.*, vol. 36, no. 2, pp. 3857–3865, Mar. 2009.

[33] V. Srinivasan, "A hybrid algorithm for the one machine sequencing problem to minimize total tardiness," *Naval Res. Logistics Quart.*, vol. 18, no. 3, pp. 317–327, 1971.

[34] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1992.

[35] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, Nov./Dec. 1995, pp. 1942–1948.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[37] P. Borovska, "Solving the travelling salesman problem in parallel by genetic algorithm on multicomputer cluster," in *Proc. Int. Conf. Comput. Syst. Technol. (CompSysTech)*, Veliko Tarnovo, Bulgaria: Univ. Veliko Tarnovo, Jun. 2006, pp. 1–6.

[38] E. Vallada and R. Ruiz, "A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times," *Eur. J. Oper. Res.*, vol. 211, pp. 612–622, Jun. 2011.

[39] R. Cheng and M. Gen, "Parallel machine scheduling problems using memetic algorithms," *Comput. Ind. Eng.*, vol. 33, nos. 3–4, pp. 761–764, Dec. 1997.

[40] E. G. Coffman, M. R. Garey, and D. S. Johnson, "Approximation algorithms for bin packing: A survey," in *Approximation Algorithms for NP-Hard Problems*. D. S. Hochbaum, Ed. Boston, MA, USA: PWS Publishing Company, 1997, pp. 46–93.

[41] Z. Lian, X. Gu, and B. Jiao, "A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan," *Appl. Math. Comput.*, vol. 175, no. 1, pp. 773–785, 2006.

JIHONG PANG born in 1978. He received the Ph.D. degree from Chongqing University, China, in 2011. He is currently an Associate Professor with the College of Business, Shaoxing University, China. His research interests include quality engineering and intelligent manufacturing systems.

YA-CHIH TSAI received the M.S. and Ph.D. degrees from the Department of Industrial Engineering and Management, University of Yuan-Ze. She is currently an Associate Professor with the Department of Hotel Management, Vanung University. Her research interests include production scheduling and semiconductor manufacturing management.

FUH-DER CHOU received the M.S. degree in industrial engineering from Chung Yuan Christian University, in 1988, and the Ph.D. degree in industrial engineering and management from the National Chiao Tung University, in 1997. He is currently a Professor with the College of Mechanical and Electronic Engineering, Wenzhou University, China. His research interests include production scheduling, semiconductor manufacturing management, and group technology.