ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20230602000692
Original scientific paper

An Iterated Greedy Algorithm for a Parallel Machine Scheduling Problem with Re-entrant and Group Processing Features

Shuaipeng YUAN*, Bailin WANG, Tike LI

Abstract: This research paper addresses a novel parallel machine scheduling problem with re-entrant and group processing features, specifically motivated by the hot milling process in the modern steel manufacturing industry. The objective is to minimize the makespan. As no existing literature exists on this problem, the paper begins by analyzing the key characteristics of the problem. Subsequently, a mixed integer linear programming model is formulated. To tackle the problem, an improved iterated greedy algorithm (IGA) is proposed. The IGA incorporates a problem-specific heuristic to construct the initial solution. Additionally, it incorporates an effective destruction and reconstruction procedure. Furthermore, an acceptance rule is developed to prevent the IGA from getting stuck in local optima. The proposed approach is evaluated through computational experiments. The results demonstrate that the proposed IGA outperforms three state-of-the-art meta-heuristics, highlighting its high effectiveness. Overall, this research contributes to the understanding and solution of the parallel machine scheduling problem with re-entrant and group processing features in the context of the hot milling process. The proposed algorithm provides insights for practical applications in the steel manufacturing industry.

Keywords: group processing; iterated greedy algorithm; parallel machine scheduling; programming model; re-entrant

1 INTRODUCTION

Scheduling is one of the core links of manufacturing systems [1-3]. The main motivation of this study is a real-life parallel machine scheduling problem with re-entrant and group processing features, which is arisen from the hot rolling process of steel plate products in the modern steel manufacturing industry. Hot rolling is a key procedure in the steel production process. However, as shown in Fig. 1, unlike the conventional processing processes, the rolling process of steel slab products has the following characteristics: (1) for each slab, two operations, namely rough and finishing rolling operations, need to be performed in sequence on the same rolling mill; (2) due to the specific process requirements, a certain waiting time is required between the two operations to conform the rolling temperature requirement and avoid uncontrollable deformation of a slab (e.g., wave roll and sickle bend); (3) a slab can temporarily release the rolling mill during the waiting period, but after a certain waiting time, it must immediately re-enter the mill for subsequent processing. To make full use of the waiting time between the rolling operations and thus improve the utilization rate of the mill, decision-makers usually schedule two slabs for group rolling. Therefore, for two adjacent slabs on the same rolling mill, if the group rolling conditions are met, they can be group rolling shown in Fig. 2a. That is, the former slab, slab *i*, first preempts the rolling mill for rough rolling until its completion and then temporarily releases the mill and enters the back roller (buffer) area to wait. During the waiting period, the latter slab, slab *j*, enters the rolling mill at the right moment to complete its rough rolling. After the waiting time specified for slab *i* is reached, it re-enters the rolling mill to complete the finishing rolling. Finally, slab *j* is sent back to the mill for finishing rolling. However, if two adjacent slabs do not meet the conditions for group rolling, they can only be rolled in a one-by-one manner, denoted as non-group rolling, as shown in Fig. 2b. This will result in long idle time on the rolling mill, which decreases productivity and increases energy consumption.

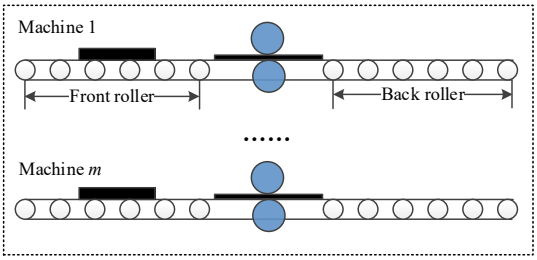

Figure 1 Hot rolling process

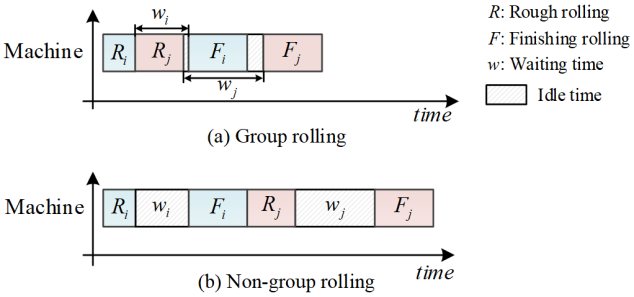

Figure 2 Schematic diagram for group rolling and non-group rolling

Combing the scheduling theory, the above-mentioned rolling process of slabs can be regarded as a complex parallel machine scheduling problem with re-entrant and group processing features. The re-entrant feature means that each slab (job) needs to visit the same machine (rolling mill) repeatedly to perform two operations, and there is a fixed waiting time between the two operations. The group processing feature indicates that for any two adjacent jobs processed on a machine, group processing is allowed if the given conditions are met. This problem needs to determine the assignment of jobs to machines, the sequence of jobs assigned to each machine, and the processing manner of adjacent jobs to maximize production efficiency. The specific problem is discussed in Section 3.

A number of studies have considered the hot rolling scheduling problem in the steel manufacturing field, but the existing research has been mainly focused on the basic rolling process constraints, such as heating temperature, roll change, and wide-to-narrow constraints [4]. In addition, one of the basic assumptions in the previous

studies has been that each slab is not allowed to leave a mill before the entire operation is completed. Thus, re-entrant and group processing features have not been considered, which deviates from the actual production of the enterprises. To the best of the authors' knowledge, there has been no published literature that addresses this problem. Designing an effective scheduling method for this problem is crucial to guiding workshop management of the modern iron and steel industry. Therefore, we are motivated to investigate such a problem and develop an effective solution method. The objective is to minimize the makespan.

The main contributions of this paper can be summarized as follows. First, some important characteristics of the studied problem are analyzed, which lays a solid foundation for the design of algorithm. Second, a mixed integer linear programming model is designed to formulate the considered problem. Third, a simple but efficient meta-heuristic algorithm, the iterated greedy algorithm (IGA), is developed to solve the studied problem. In the IGA, a series of efficient techniques, including a problem-specific initialization strategy, a destruction and reconstruction (DR) procedure and a local search strategy based on tailored variable neighborhood descent procedures, are used to achieve a balanced trade-off between the solution quality and computation time. Finally, the effectiveness of the proposed IGA is demonstrated by extensive numerical comparisons.

The remainder of this paper is structured as follows. In Section 2, the literature review relevant to this study is provided. Section 3 presents the formal descriptions of the problem, the properties of the problem and a mixed integer linear programming model. The developed IGA is then presented in Section 4. Computational results are carried out in Section 5. Finally, Section 6 concludes the paper and suggests areas for future study.

2 RELATED WORK

The related studies on the parallel machine scheduling, re-entrant scheduling, group scheduling, and IGA are briefly reviewed.

Parallel machine scheduling problems have been widely studied due to the ubiquity in practical industrial applications [5]. Bitar et al. [6] presented an unrelated parallel machine scheduling problem with auxiliary resources in a semiconductor plant. A memetic algorithm was proposed with two objective functions including the maximization of the number of produced wafers and the minimization of the weighted completion times. A series of experiments were conducted to determine the best configurations of the proposed algorithm. However, the results were not compared with other works or reference values. Chen et al. [7] investigated an unrelated parallel machine scheduling problem with sequence-dependent setup times and job release times, which was arisen from the ion implantation process of wafer fabrication. The objective was first to maximize the number of processed jobs and then minimize the makespan, and finally minimize the maximum completion times of non-bottleneck machines. A mixed integer programming model was proposed as a solution approach to satisfy the first two objectives. Using the obtained solution, a hybrid tabu search algorithm was further developed to satisfy all the three objectives. For the same problem, Soares et al. [8] designed a biased random-key genetic algorithm hybridized with variable neighborhood descent method. Abu-Marrul et al. [9] dealt with a problem arising from the oil industry, and regarded it as an identical parallel machine scheduling problem where jobs were composed of intersecting sets of operations. Three integer linear programming formulations were designed to solve the problem. On the basis of this research, Abu-Marrul et al. [10] further addressed a batch scheduling problem with identical parallel machines and non-anticipatory family setup times to minimize the total weighted completion time. A greedy randomized adaptive mate-heuristic was developed using a constructive heuristic. Zhang et al. [11] studied a parallel machine scheduling problem with machine health conditions and preventive maintenance, which was derived from the semiconductor manufacturing. Two mixed integer linear programming models and a general variable neighborhood search algorithm were presented. Chung et al. [12] addressed a resource-constrained parallel machine scheduling problem with setup times in microelectronic components manufacturing, where a mathematical model and three effective constructive heuristics were presented. To assess the quality of the proposed methods, a discrete particle swarm optimization algorithm and a variable neighborhood search method were additionally presented.

In the above-mentioned parallel machine scheduling studies, each job visits each machine at most once. Nevertheless, in some manufacturing processes, a job may be processed by the same machine twice or more due to the high cost of processing equipment and the repeated sets of processes. Such a processing environment is called "re-entrant" in the scheduling area [13, 14]. Wang et al. [15] considered a novel surgery scheduling problem in outpatient procedure centers to minimize the average recovery completion time of all patients. The problem was regarded as a no-wait re-entrant hybrid flow shop scheduling problem with fuzzy service times. A new hybrid meta-heuristic, integrating genetic algorithm and variable neighborhood search, was developed to schedule outpatients for surgical services. Wu et al. [16] studied a re-entrant hybrid flow shop scheduling problem with batch processing machines, which was arisen from the production process of the cold-drawn seamless steel pipe in steel manufacturing sector. To minimize the makespan and the energy consumption of the batch processing machines, a mathematical model was formulated at first, and then an improved multi-objective evolutionary algorithm based on decomposition technique was developed. Frihat et al. [17] addressed a realistic re-entrant hybrid job-shop problem with time lags and sequence-dependent setup times, which was derived from the tannery industries. Two different models based on mixed integer programming and constraint programming were proposed. For both models, a problem-oriented optimization technique was proposed to reduce the problem size. Xu et al. [18] considered a re-entrant permutation flowshop scheduling problem to minimize the makespan. A memetic algorithm was developed to solve the problem.

As for the re-entrant scheduling problem with parallel machine environment, there are few related studies. Shin

[19] addressed a re-entrant parallel machine scheduling problem with process quality, due dates, and sequence-dependent setup times from a viewpoint of process stability as well as on-time delivery. A dispatching algorithm called quality and rework with due dates was proposed. Chakhlevitch et al. [20] explored a two stage re-entrant workshop with parallel machines at the first stage and a single machine at the second stage. A heuristic based on a simple strategy of initializing jobs in batches on the primary machines was developed. Computational experiments on a broad range of benchmark problem sets indicated that the algorithm can find optimal or near optimal schedules in different production scenarios.

Another topic related to our problem is the group scheduling. Although a lot of studies can be found in the literature [21], they all assume that the group formation problem (i.e., the assignment of jobs to groups) is known, and mainly focus on other two sub-problems, namely the sequence of groups and the sequence of jobs within each group. However, in this work, the group formation problem is an important decision item, even though there are only two jobs in a group. Thus, there is a great difference between the classical group scheduling problem and the studied problem.

Meta-heuristic algorithm has been proven to be an effective method for solving scheduling problems [22-26]. The IGA is a simple and efficient meta-heuristic algorithm [27]. It always records two solutions including the current solution and the best solution found so far. In each iteration, it starts from an initial solution and then tries to improve the current solution by DR operation. To improve the global search ability, an acceptance rule is usually adopted. Compared with other meta-heuristic algorithms, the IGA has a simpler structure and fewer parameters, and thus is easy to code and implement. Some studies have used the IGA to solve scheduling problems, and good results were achieved [28-30]. In view of this, the idea of proposing an IGA with problem-specific search strategies to solve the studied problem seems promising and thus is one of the main goals of this work.

3 RESEARCH PROBLEM

3.1 Problem Description

The problem considered in this work can be described as follows. A set of n jobs $J=\{J_1,J_2,...,J_n\}$ need to be processed on a set of m identical parallel machines $M=\{M_1,M_2,...,M_m\}$. Each job J_i is composed by two operations (O_{i1},O_{i2}), respectively with associated processing times p_{i1} and p_{i2}. Two operations within a job have to be assigned sequentially to the same machine, which means that the operation O_{i1} must precede the operation O_{i2}. For each job, there is a certain waiting time, denoted as w_i, between the two operations. During the waiting period, the job can be temporarily released from the machine, but after the waiting time is reached, it must re-enter the machine immediately to complete the second operation. Therefore, the re-entrant properties (i.e., jobs may visit a machine more than once) exit in this problem. In addition, as shown in Section 1, for any two jobs J_i and J_j processed on the same machine, if the conditions for group processing are met, it is allowed to process them in a group. In this case, the processing order is $O_{i1},O_{j1},O_{i2},O_{j2}$; otherwise, they can only be processed one by one, where the processing order is $O_{i1},O_{i2},O_{j1},O_{j2}$. For the convenience of the description below, the former processing manner is defined as group processing, and the latter as non-group processing. Obviously, arranging group processing for two jobs can make full use of the idle times of machines, thereby improving production efficiency. For any two jobs J_i and J_j, the sufficient conditions for group processing are $w_i \geq p_{j1}$ and $p_{i2} \leq w_j$.

To illustrate the scheduling problem more clearly, a simple instance with two machines and five jobs is given in the following. The processing times of operations and waiting time between the operations are shown in Tab. 1. Fig. 3 plots three different scheduling schemes. For scheme 1 shown in Fig. 3a, jobs J_1, J_2, and J_3 are processed on M_1, where jobs J_1 and J_2 are group processing. Jobs J_4 and J_5 are processed on machine M_2 and are non-group processing. The makespan of this scheme is 31. For scheme 2 shown in Fig. 3b, jobs J_1, J_2, and J_3 are still processed on M_1, where jobs J_2 and J_3 are arranged for group processing. On machine M_2, the job sequence is changed to $J_5 \rightarrow J_4$, and they are group processing. The makespan is reduced to 25. Regarding scheme 3 shown in Fig. 3c, the sequence on machine M_1 is changed to $J_3 \rightarrow J_1 \rightarrow J_2$, and jobs J_3 and J_1 are group processing.

Table 1 Job information of the example instance

Job index	p_{i1}	w_i	p_{i2}
1	2	3	4
2	3	5	5
3	2	7	3
4	4	5	6
5	6	7	3

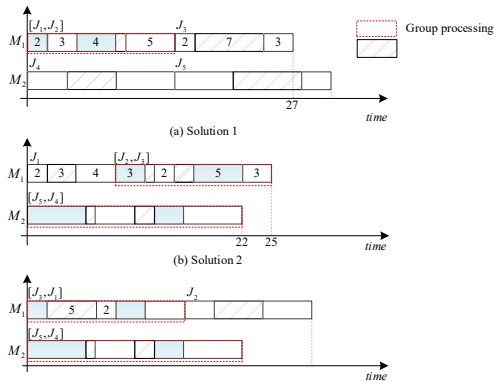

Figure 3 Gantt charts for three scheduling schemes

The scheduling information on machine M_2 is consistent with scheme 2. In this way, the makespan is 29. Therefore, even if the assignment scheme of jobs to machines is the

same, different scheduling sequences and the processing manner can impact the $C_{\max}$ value.

3.2 Mathematical Model

Except for the notations mentioned above, the following indices and notations are used throughout the study.

(1) Indices and notations:

i', i Job index, $i', i = 0,1,2,...,n$.

k Machine index, $k = 1,2,...,m$.

P_i Sum of processing time and waiting time for job J_i, having $P_i = p_{i1} + w_i + p_{i2}$.

M A large positive value.

(2) Decision variables and objective:

λ_{ij} Auxiliary decision binary variable, equal to 1 if J_i and J_j meet the group processing conditions; otherwise, equal to 0.

x_{ijk} Binary variable, equal to 1 if job J_i immediately precedes job J_j on machine k; otherwise, equal to 0.

y_{ij} Binary variable, equal to 1 if jobs J_i and J_j are arranged for group processing; otherwise, equal to 0.

C_{i1}, C_{i2} Completion time for the first, second operation of job J_i, respectively.

$C_{\max}$ Objective of makespan.

A mixed integer linear programming model is formulated as follows.

$$\min\ C_{\max} \tag{1}$$

$$\sum_{i=0, i\neq j}^{n}\sum_{k=1}^{m} x_{ijk} = 1,\ j = 1,2,...,n \tag{2}$$

$$\sum_{i=0, i\neq j}^{n}\sum_{k=1}^{m} x_{jik} = 1,\ j = 1,2,...,n \tag{3}$$

$$\sum_{i=1}^{n}\sum_{k=1}^{m}(x_{ijk} + x_{jik}) \le 1\ \ j = 1,2,...,n \tag{4}$$

$$\sum_{i=1}^{n} x_{0ik} = 1,\ k = 1,2,..,m \tag{5}$$

$$y_{ij} \le \lambda_{ij},\ \ i = 1,2,...,n,\ j = 1,2,...,n \tag{6}$$

$$y_{ij} \le \sum_{k=1}^{m} x_{ijk},\ \ i = 1,2,...,n,\ j = 1,2,...,n \tag{7}$$

$$\sum_{i=1}^{n}(y_{ij} + y_{ji}) \le 1,\ \ j = 1,2,...,n \tag{8}$$

$$C_{j1} \ge p_{j1},\ \ j = 1,2,...,n \tag{9}$$

$$C_{j2} = C_{j1} + w_j + p_{j2},\ \ j = 1,2,...,n \tag{10}$$

$$C_{j1} - p_{j1} \ge C_{i2} - M\cdot(1 - \sum_{k=1}^{m} x_{ijk} + y_{ij}),\ \ i, j = 1,2,...,n \tag{11}$$

$$C_{j1} - p_{j1} \ge C_{i1} - M\cdot(2 - \sum_{k=1}^{m} x_{ijk} - y_{ij}),\ \ i, j = 1,2,...,n \tag{12}$$

$$C_{j2} - p_{j2} \ge C_{i2} - M\cdot(2 - \sum_{k=1}^{m} x_{ijk} - y_{ij}),\ \ i, j = 1,2,...,n \tag{13}$$

$$C_{\max} \ge C_{i2}\ \ \ i = 1,2,...,n \tag{14}$$

$$x_{ijk}, y_{ij} \in \{0,1\}\ i = 0,1,2,...,n,\ j = 0,1,2,...,n \tag{15}$$

$$C_{j1}, C_{j2} \ge 0,\ j = 1,2,...,n \tag{16}$$

Objective (1) is to minimize the makespan. Eq. (2) and Eq. (3) represent that each job should be assigned to exactly one machine, and have exactly one predecessor and one successor (including the dummy job J_0). Eq. (4) ensures that the predecessor and the successor cannot be same. Eq. (5) indicates that on each machine, the dummy job J_0 must be arranged at the first position. Eq. (6) and Eq. (7) represent the preconditions for any two jobs to perform group processing. One is the time relation mentioned in Section 3.1, and the other is the sequence relation, i.e., the two jobs are on the same machine and adjacent. Eq. (8) indicates that a job can be assigned for group processing at most once. Eq. (9) and Eq. (10) restrict the completion time of the first operation and second operation, respectively, for each job. Eq. (11) suggests the temporal relationship between a job and its predecessor when they are non-group processing. That is, a job cannot start processing until its predecessor has completed all the two operations. Eq. (12) and Eq. (13) relate the starting times of a job and the completion times of its predecessor on two operations, respectively, when they are group processing. Specifically, for an arbitrary job J_i, the starting times of the two operations are no less than the completion times of the two operations for its predecessor. Eq. (14) indicates that the makespan is equivalent to the maximum completion time of all jobs. Eq. (15) and Eq. (16) represent the types and value ranges of decision variables.

The above model has been verified by CPLEX solver 12.10. However, due to the complexity and existence of the big-M constraints, only very small scale instances can be solved, resulting in a low practicability.

3.3 Property Analysis

The classical parallel machine scheduling problem $P_m \,||\, C_{\max}$ has been proven to be strongly NP-hard when $m \ge 2$; so, the problem $P_m \,|\, rcrc, group \,|\, C_{\max}$ is also strongly NP-hard and more complex. To improve solution efficiency, theorem 1presents the properties of the problem. To facilitate the description, the following two definitions are first given.

Definition 1. If jobs J_i and J_j are two jobs processed on the same machine and arranged for group processing, they are defined as a block, and one is called partner of another.

Definition 2. If there is a job J_i that does not meet the group processing conditions with all other jobs, i.e., $\sum_{\forall j}(\lambda_{ij}+\lambda_{ji})=0$, J_i is defined as a non-active job; otherwise, it is defined as an active job.

Theorem 1. On each parallel machine, if the jobs assigned to the machine are identified, there exists at least one optimal job sequence, where the processing manner (group or non-group processing) of adjacent jobs satisfies the following greedy rule. Starting from the first position of the sequence, if the job at the current position and its successor meet the conditions for group processing, they can be arranged directly for group processing without considering the processing manner of jobs in the subsequent sequence.

Proof of Theorem 1. It is proven by a contradiction way. Suppose that there exists an optimal schedule containing subsequence $\pi'=\{J_i,J_j,J_k\}$ that does not satisfy the above greedy rule. That is, although job J_j and its predecessor J_i meet the group processing conditions, they are arranged for non-group processing. It suffices to conclude that J_j is processed either independently or together with its successor J_k as a group. ① If J_j is processed independently, it can always be processed with J_i as a group, constructing a more optimal sub-sequence that satisfies the greedy rule, which contradicts the optimality assumption. ② If J_j is processed together with J_k as a group, then J_i must be processed independently. We can always move J_i to the end of the sequence, making J_j and J_k process preferentially, and constructing a schedule satisfying the greedy rule. Since J_i is processed independently before moving, the movement does not increase the $C_{\max}$ value. This completes the proof. □

4 THE PROPOSED IGA

To solve the problem $P_m \mid rcrc, group \mid C_{\max}$, this paper proposes an improved IGA. In the IGA, a simplified encoding-decoding strategy is designed to deal with the three sub-problems, which can effectively control the search regions of optimal solution space. A problem-specific heuristic is used to construct an initial solution with high quality. To guide the search, an innovative DR strategy, a local search strategy and an acceptance rule, are developed. All the steps of the IGA are explained in the following subsections.

4.1 Encoding and Decoding

For the problem $P_m \mid rcrc, group \mid C_{\max}$, three sub-problems, including the assignment of jobs to machines, the sequences of jobs on each machine, and the processing manner of adjacent jobs, should be determined. It can be known from Theorem 1 that for any one instance, there exists at least one optimal schedule where the processing manner of adjacent jobs on each machine satisfies the greedy rule. Therefore, to improve the search efficiency, only other two sub-problems are considered in the encoding phase, and a decoding method based on greedy rule is presented to map a coding scheme to a feasible solution. The specific encoding and decoding strategies are as follows. A permutation-based vector σ ranging from one to $(n+m-1)$ is used to represent the considered problem. In σ, the $m-1$ elements that are greater than n divide the σ into m sub-sequences, denoted as σ_k, $k=1,2,...,m$. In the decoding phase, jobs in σ_k are assigned to machine M_k and processed in the order of their position. Furthermore, the greedy rule presented in Theorem 1 is used to determine whether adjacent jobs are arranged for group processing or not.

4.2 Initialization

To promote the algorithm to evolve to the dominant region at a faster rate, an effective approach is to design a problem-specific heuristic to construct an initial solution with higher quality [31]. Regarding the studied problem, the following heuristic is introduced to construct the initial solution.

Step 1: Initialize related information, including the job set J, the machine set M, and the earliest available time of each parallel machine.

Step 2: Calculate the reduction value s_{ij} for any two jobs J_i and J_j while they are arranged for group processing, and generate a reduction matrix S. If J_i and J_j do not meet the group processing conditions, set s_{ij}=0.

Step 3: Identify the element with the largest reduction value, denoted as $s_{i'j'}$, from S and schedule the corresponding jobs $J_{i'}$ and $J_{j'}$ to the parallel machine with the earliest available time. Simultaneously, update the earliest available time of the selected machine.

Step 4: Remove the selected jobs $J_{i'}$ and $J_{j'}$ from set J, and reset the related elements in S to zero, i.e., set $S(i',:)=0$, $S(j',:)=0$, $S(:,i')=0$, and $S(:,j')=0$.

Step 5: Repeat Step 3 and Step 4 until $J=\varnothing$ or all elements in S are equal to 0. If there are unscheduled jobs, i.e., $J\neq\varnothing$, proceed to Step 6; otherwise, proceed to Step 9.

Step 6: For unscheduled jobs in J, sort them in decreasing order of their total processing time P_i.

Step 7: Extract one job from J at a time according to the sorting results, and assign it to the machine with the earliest available time.

Step 8: Repeat Step 7 until all jobs in J are scheduled, and proceed to Step 9.

Step 9: Construct a complete schedule according to the sub-sequence σ_k on each machine.

Given an instance with 9 jobs and 3 machines (see Tab. 2), the construction process of the initial solution is shown in Fig. 4. It can be known from the matrix S that jobs J_6

and J_3 have the largest reduction value when they are arranged for group processing; so, they are preferably scheduled to machine M_1. Among the remaining jobs, J_5 and J_2 become the largest contributors and thus are scheduled to machine M_2. Next, jobs J_4 and J_8 win and are assigned to machine M_3. Further, jobs J_9 and J_1 are selected and assigned to machine M_1 that has the earliest available time. Afterword, only job J_7 is left, which can only be processed by Non-group processing; so it is assigned to machine M_2 according to the assignment rules. Finally, the initial schedule with an objective value of 38 is obtained. The corresponding scheduling Gantt chart is shown in Fig. 5a.

Table 2 Job information for an example instance

J_i	p_{i1}	w_i	$\cdot p_{i2}$
1	3	2	3
2	6	3	8
3	5	5	4
4	5	2	6
5	3	6	3
6	2	5	5
7	10	2	6
8	1	6	10
9	2	4	2

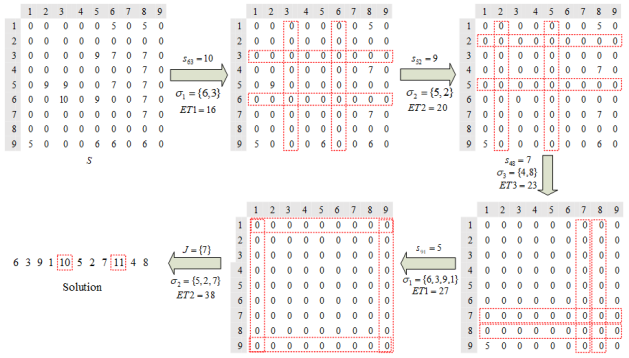

Figure 4 Construction process for the initial solution

4.3 DR Procedure

In the classical DR strategies, jobs are randomly extracted from the current solution and inserted into all possible positions to construct several new solutions [32]. However, as described in Section 4.1, the coding strategy used in this work not only includes the sequence information of jobs but also the information on the processing manner of adjacent jobs. Therefore, using the classical DR strategies can severely destroy the existing blocks in the current solution, which is not conducive to algorithm convergence. In view of this, an improved DR procedure is developed to mine better solutions, where two terms, including mining new block structures and balancing the load of parallel machines, are mainly considered to improve the effectiveness. The specific steps of the implementation process are as follows.

Step1: Randomly select an active job J_i from the job sequence on the bottleneck parallel machine.

Step2: Identify all jobs that can be group processed with J_i and sort them in non-decreasing order of the reduction value; then, generate a list π.

Step 3: Extract a job, denoted as J_j, from list π in turn, and generate a block according to the configuration relationship of the reduction value, that is, if the reduction value $s_{ij} > s_{ji}$, set $block = [J_i, J_j]$; otherwise, set $block = [J_j, J_i]$.

Step 4: Determine whether the generated block already exists in the current solution, and if so, proceed to Step 8; otherwise, proceed to Step 5.

Step 5: Remove job J_i from the current solution and determine whether there is a partner of job J_i in the current solution; if so, move it to the end of the sub-sequence corresponding to the weakest machine.

Step 6: Remove job J_j from the current solution and judge whether there is a partner for job J_j, and if so, move the partner to the end of the sub-sequence corresponding to the weakest machine.

Step 7: Insert *block* into the front of the sequence on the weakest machine and generate a new complete solution.

Step 8: Repeat Steps 4 - 7 to obtain $\min(|\pi|, new\_solutions)$ solutions.

Step 9: Select the best solution from the generated solutions as a new solution.

It should be noted that in the reconstruction phase, the number of new solutions constructed in each iteration is denoted by $new\_solutions$, and its value is related to the solution efficiency. The optimal value of $new\_solutions$ is determined experimentally. In addition, to ensure load balancing of parallel machines, the completion time of each machine is dynamically updated in each move, and the weakest machine is always selected for every insert operation.

Taking the initial solution obtained in Fig. 4 as an example, Fig. 5 illustrates the process of the DR procedure.

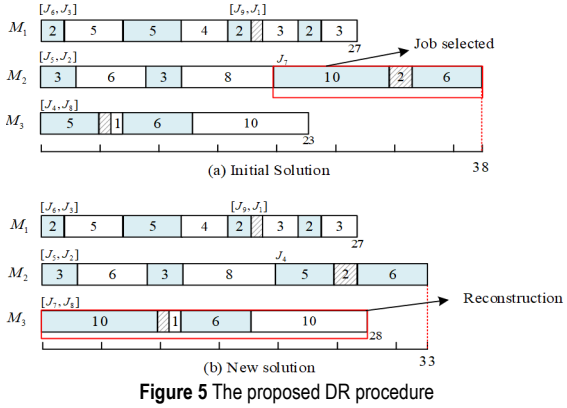

Figure 5 The proposed DR procedure

Assume that job J_7 on machine M_2 is selected, according to the matrix S, only job J_8 can be processed with job J_7 as a group, i.e., $\pi = [8]$. Because job J_7 has no partner, it is first removed from the current solution, then machine M_2 becomes the weakest one. Job J_8 has

already processed as a group with job J_4 in the current solution; so, job J_4 is moved to machine M_2. By doing so, machine M_3 becomes the weakest one. Thus, the new block $[J_7, J_8]$ is inserted into machine M_3. Finally, a new solution with an objective of 33 is obtained.

4.4 Local Search Strategy

The VND strategy has been commonly used as a local search strategy. Insert and exchange are two widely-used operations in solving the scheduling problem, and their effectiveness has been validated [33]. In view of this, based on the classical insert and exchange operations, two problem-oriented VNDs, including block- and job-insert operations, are designed to explore the search space. It should be noted that whenever an improved neighbour solution is found, the exploration is restarted from the first neighbourhood.

4.4.1 Block-Insert Operation

Because a lot of block information exists in the current solution, a simple block-insert operation is designed. First, sequence on the bottleneck machine is analyzed, and the block with the shortest processing time is removed and inserted into the weakest machine. To avoid insert operation destroying the existing block structures in the current solution, the selected block inserted into the first position of the sequence corresponds to the weakest machine. In particular, if no block exists on the bottleneck machine, skip this operation.

4.4.2 Job-Insert Operation

After completing the block-insert operation, job-insert operation is further performed for jobs that are non-group processing on the bottleneck machine. Namely, if there are independently processed jobs, the one with the shortest processing time is removed and inserted into the weakest machine. Unlike the block-insert operation, the selected job is inserted at the end of the sequence on the weakest machine. It should be noted that throughout the whole local search procedure, if a block needs to be inserted into a new sequence, it is always inserted at the beginning of the sequence, and if the inserted object is a single job, it is inserted at the end of the sequence. The purpose of this is to prevent the move operation from destroying the existing block structures in the current solution.

4.5 Acceptance Rule

In the IGA, an improved acceptance criterion is defined following the idea of the age threshold used in the migratory bird optimization algorithm, that is, age is used to indicate the updating status. During each iteration, if the current solution cannot be improved, its age is increased by one; otherwise, its age is initialized as zero. If the age exceeds the preset threshold φ, the algorithm will accept a new (worse) solution with a probability $e^{-\Delta E/\tau}$, where ΔE is the gap of $C_{\max}$ between the new solution and the current solution, and τ is the temperature parameter, and is set that $\tau = \sum_{i=1}^{n}(p_{i1} + w_i + p_{i2})$.

4.6 Procedure of the IGA

Algorithm 1 presents the pseudo code for the proposed IGA, where the number of new solutions is generated in the reconstruction phase, and is the age threshold in the acceptance rule.

Algorithm 1: IGA

Parameters: $new\_solutions$, φ

Output: Solution X_{best}

$X_{init} \leftarrow$ Generate initial solution

$X_{best}, X_{cur} \leftarrow X_{init}$

$iter \leftarrow 1$ //consecutive iterations that have not been improved

While stop criterion is not meet *do*

$X' \leftarrow DR(X_{cur}, new\_solutions)$

$X' \leftarrow LocalSerach(X')$

if $f(X') < f(X_{best})$ *then* //*f* is the evaluation function

$X_{best} \leftarrow X'$

end

if $f(X') < f(X_{cur})$ *then*

$X_{cur} \leftarrow X'$

$iter = 1$

else

$iter \leftarrow iter + 1$

if $accept(X', X_{cur}, iter, \varphi)$ // Acceptance rule

$X_{cur} \leftarrow X'$

$iter \leftarrow 1$

end

end

end

return X_{best}

5 COMPUTATIONAL EXPERIMENTS

In this section, computational experiments are presented to evaluate the performance of the proposed model and IGA. First, instances with rich features are generated and grouped according to the problem scales. Next, the effectiveness of key strategies in the IGA is verified, and the parameter values are calibrated. Then, the solution performance of the model is tested using small scale instances. Finally, the IGA is compared with three state-of-art algorithms using large scale instances. All of the algorithms are implemented using MATLAB R2015a, and the model is solved by CPLEX 12.10. The code running environment is Intel(R) i5-6200U CPU /16.0 GB.

5.1 Test Data

Since the problem considered in this work is a new scheduling problem, and there have been no existing benchmarks in this field, a number of test instances are generated based on the real situations in a hot rolling workshop of a large steel enterprise in China. Each instance is defined by three parameters as $\{m, n, W\}$, where m represents the number of machines, n is the number of jobs,

and W indicates the pattern used for generating the operation processing times and waiting times between the operations.

The number of machines is classified into two sets: $m \in \{2,3\}$ (small scale), and $m \in \{5,8,10\}$ (large scale). The number of jobs is set to be $n \in \{5,10,20,50\}$ (small scale), and $n \in \{100,150,300,500\}$ (large scale). In the considered problem, the relationship between the processing time and the waiting time can affect the possibility of group processing, thus affecting the difficulty of the problem-solving process. Therefore, for a reliable comparison, three different patterns of parameter W are set as follows. ① $W = 1$ represents that the value range of waiting time is less than that of the processing time, where the processing time is set as $p_{i1}, p_{i2} \in DU[5,50]$, and the waiting time is set as $w_i \in DU[5,25]$. $DU[a,b]$ denotes a discrete uniform distribution ranging from a to b. ② $W = 2$ indicates that the value ranges of waiting and processing times are the same. Here, the processing time is set as $p_{i1}, p_{i2} \in DU[5,50]$, and the waiting time is set as $w_i \in DU[5,50]$. ③ $W = 3$ indicates that the value range of waiting time is larger than that of the processing time, having $p_{i1}, p_{i2} \in DU[5,50]$, and $w_i \in DU[50,75]$.

Combining different values of the three parameters, 24 small-scale categories and 36 large-scale categories are generated. For each category, 10 problem instances are generated, which results in a total of (24 + 36) × 10 = 600 instances.

5.2 Algorithm Calibration

The improvement strategies and parameters used in the IGA are calibrated. The improvement strategies include the initialization strategy, DR strategy, local search strategy, and acceptance rule. The IGA has two parameters, the number of new solutions ($new\_solutions$) in the reconstruction phase and the age threshold in the acceptance rule (φ). Based on the preliminary experiments, the comparison strategies and parameter values are listed below.

- Initialization: two levels (our initial strategy and random initial strategy).
- DR: two levels (our DR strategy and classical DR strategy).
- Insert operation: three levels (our block and job insert operation, classical insert operation, No insert operation).
- Exchange operation: three levels (our block- and job-exchange operation, classical exchange operation, No exchange operation).
- Relocation operation: two levels (Use and no-use job relocation operation).
- $new\_solutions$: four levels (1,3, 5 and 10).
- φ: three levels (10, 30 and $+\infty$), where $+\infty$ means no acceptance rule is used.

The above combinations can yield a total of 2 × 2 × 3 × 3 × 2 × 4 × 3 = 864 different configurations for the IGA. All 864 configurations are evaluated using a full factorial experimental design. One instance is randomly selected from each of the 36 large-scale categories, obtaining 36 instances in total. Each algorithm is evaluated by 5 independent replications for each instance with a termination criterion of a maximum of $50000 + n \times m \times 50$ ms, which is composed by a basic term plus another term increasing with the problem size. The percentage relative deviation (PRD) [34] is used as the response variable to measure different algorithms. PRD is calculated by

$$PRD = \frac{C_{\max}(A) - C^*_{\max}}{C^*_{\max}} \times 100\% \tag{17}$$

where $C_{\max}(A)$ is the makespan generated by algorithm A, and $C^*_{\max}$ is the best makespan obtained by all algorithms. Following the statistical model commonly used in the literature, the experimental results are analyzed by a multi-factor analysis of variance (ANOVA) at a 95% confidence level. The ANOVA results are shown in Tab. 3, where F-value shows the influence of the factor on the algorithm performance, and P-value indicates whether there is a significant difference among the levels in each factor.

Table 3 Results of ANOVA

Source	Sum of Squares	D*F*	Mean Square	*F-value*	*P-value*
Initialization	1322.90	1	1322.87	132.3	<0.001
DR	671.05	1	670.97	70.86	<0.001
Insert operation	559.35	2	279.67	28.23	<0.001
Exchange operation	894.73	2	447.37	44.74	<0.001
Relocation operation	681.86	1	681.86	68.20	<0.001
new solutions	569.32	3	189.77	19.01	<0.001
φ	228.5	2	114.28	15.38	0.035
Error	8430.1	851	9.906		
Total	13078.13	863			

As shown in Tab. 3, *P*-value results of the seven factors are all less than 0.05, indicating significant differences in the performance between different levels of each factor. According to the results, the initialization strategy leads to the largest F-value, showing that it has the most important effect on the algorithm performance among all considered factors. Based on the results, using the proposed initialization strategy can significantly improve the solution performance. The DR strategy achieves the second largest F-value, demonstrating the effectiveness of the proposed DR strategy.

For the three factors in the local search phase, job relocation strategy achieves the largest F-value, followed by the exchange operation and insert operation. It can be seen that the IGA effectiveness can be significantly improved when incorporating the designed job relocation operation.

The two parameters $new\_solutions$ and φ are finally examined. In our results, the IGA produces the best results when $new\_solutions = 5$ and $\varphi = 30$. Based on these results, we set $new\_solutions = 5$ and $\varphi = 30$ for the IGA in the following experiments.

5.3 Optimality Test

Next, the aforementioned 24 small-scale categories containing 240 instances are used to evaluate the effectiveness of the two proposed models, and to observe the deviation of $C_{\max}$ yielded by the IGA from the optimal one. For each instance, the time limit of CPLEX solver is set to 3600 s, and the maximum elapsed CPU time is set to $50000 + n \times m \times 50$ ms for the IGA. Experimental results are shown in Tab. 4, $\overline{C}_{\max}$ represents the average $C_{\max}$ values, Time is the average running time required by the CPLEX solver, and N_{opt} indicates the number of instances that the model and the IGA obtain the optimal $C_{\max}$ in each category.

Table 4 Experimental results for two models and IGA

m	n	W	*Model*			IGA		
			$\overline{C}_{\max}$	N_{opt}	*Time*	$\overline{C}_{\max}$	N_{opt}	*PRD*
2	5	1	171.0	10	0.89	171.0	10	0.000
2	5	2	189.7	10	0.90	189.7	10	0.000
2	5	3	233.2	10	0.92	233.2	10	0.000
2	10	1	346.5	10	342.04	346.6	9	0.035
2	10	2	316.3	10	363.61	316.3	10	0.000
2	10	3	374.6	10	652.90	374.6	10	0.000
2	20	1	605.7	10	14.36	605.8	9	0.016
2	20	2	581.2	10	14.55	584.3	9	0.531
2	20	3	699.9	10	14.86	702.4	8	0.358
2	50	1	1558.2	4	128.95	1558.3	3	0.376
2	50	2	1461.7	5	132.93	1461.7	5	0.780
2	50	3	1714.0	6	132.08	1720.9	3	0.407
3	5	1	132.4	10	1.157	132.4	10	0.000
3	5	2	122.8	10	0.952	122.8	10	0.000
3	5	3	150.9	10	0.886	150.9	10	0.000
3	10	1	214.0	10	344.21	214.2	9	0.088
3	10	2	226.0	10	353.63	226.0	10	0.000
3	10	3	265.5	10	1333.96	265.7	9	0.080
3	20	1	398.6	10	26.40	400.1	9	0.387
3	20	2	426.7	10	30.79	430.7	6	1.217
3	20	3	470.0	10	43.68	471.4	9	0.311
3	50	-	-	0	141.04	1070.7	-	-
3	50	-	-	0	163.64	986.7	-	-
3	50	-	-	0	182.37	1159.7	-	-
Average			/	/	/	/	/	0.218
Total			/	195		/	178	/

It can be seen from Tab. 4 that for the instances with a very small scale, the proposed model and IGA have the same $C_{\max}$ value, which confirms the correctness of the model. With the increase in the problem size, the solution time of model shows a trend of rapid increase, and when $m = 3$ & $n = 50$, the model cannot obtain the optimal solution within the specified time. Overall, a total of 195 out of 240 instances can be solved to be the optimal solution by the model, proving the feasibility of the model. As for the IGA, 178 out of 195 instances (45 instances without optimal solutions are not considered) can be solved with the optimal $C_{\max}$, which shows that for most test instances (approximately 91%), the IGA can obtain the optimal solution in a limited time. For the instances whose optimal solution cannot be obtained by the IGA, the overall PRD value is only 0.218%, indicating that the deviations in the schedules obtained by the IGA from the optimal ones are very small. Consequently, the above results verify that the IGA has a good ability to construct high-quality solutions for small-scale instances.

5.4 Comparison with State-of-Art Algorithms

5.4.1 Adaptation and Aalibration of Atate-of-Art Algorithms

Although many pieces of literature are cited in Section 2, as mentioned above, there are certain differences between the problem studied in this work and those in the literature; so, no comparison algorithms can be directly applied to the problem considered in this paper. In addition, the problem studied in this wok can be regarded as a parallel machine scheduling problem. Therefore, three state-of-art meta-heuristic algorithms proposed in the literature for the parallel machine scheduling problem are selected for the comparison. To ensure fairness of the comparison, certain necessary adaptations in these algorithms are made.

The selected algorithms include one population-evolutionary meta-heuristic algorithm, the hybrid biased random-key genetic algorithm (HBRGA) proposed by [8]; two individual-evolutionary meta-heuristic algorithms, the general variable neighbourhood search (GVNS) algorithm proposed by [11] and the iterated greedy algorithm proposed by [29], denoted as IGA_O. The adaptations made to these algorithms are as follows. First, the objective functions of the three comparison algorithms are uniformly revised to makespan. Second, since the problem studied in this paper requires considering the processing manner of adjacent jobs, the designed decoding method is adopted for those algorithms to ensure the fairness of the comparison, that is, all algorithms only focus on other two sub-problems, namely the assignment of jobs to machines and the sequence of jobs on each machine.

Table 5 Parameter values of the state-of-art algorithms

Algorithm	Parameter	Description
GVNS	None	None
HBRGA	$p_{size} = n$ $p_{elite} = 0.2 \times p_{size}$ $p_{mutation} = 0.1 \times p_{size}$	Population size Elite group size Number of mutant individuals
IGA_O	$\varepsilon = 0.15$ $\lambda = 0.1$	Destruction parameter Solution restore parameter

To determine the best parameter values and thus ensure the fairness of the experiment, a similar procedure as that presented in Section 5.2 is used for each of the algorithms. Specifically, for each algorithm, a full factorial experimental design is adopted using the factors and levels taken from the original paper. The 36 instances presented in Section 5.2 are used as a criterion. The best parameter values of each algorithm are reported in Tab. 5. It should be noted that since some of the local search strategies that cannot be applied to the considered problem were discarded, the corresponding relevant parameters of these strategies are automatically discarded.

5.4.2 Experimental Results

The 360 large-scale instances are used to evaluate the performance of the comparison algorithms and IGA. Ten independent runs are conducted using each algorithm to solve each instance, and all of the algorithms are tested with the same termination criterion of $50000 + n \times m \times 50$ *ms*. The multi-factor ANOVA is used to analyse differences in the average PRD values of the four

algorithms, where the algorithm is regarded as a factor. The LSD intervals at the 95% confidence level are shown in Fig. 6.

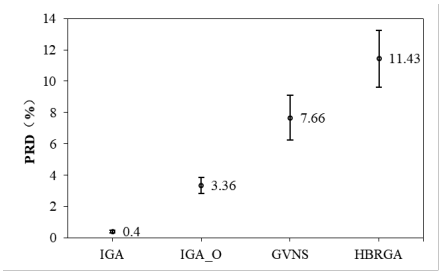

Figure 6 LSD intervals at the 95% confidence level for four algorithms

According to the results in Fig. 6, the IGA obtains the lowest average PRD values on all 36 combinations, and its overall average PRD value is only 0.40%, which is much lower than that of the IGA_O (3.36%), GVNS (7.66%), and HBRGA (11.43%). This demonstrates that the quality of the IGA solution is better than those of the other three comparison algorithms. Regarding the confidence level of the PRD value, the IGA also has the lowest result among the four algorithms, which indicates that the IGA can not only generate better solutions but also perform more steadily with various problem instances. Among the three comparison algorithms, the IGA_O performs the best. The rankings of the GVNS and HBRGA deviate with the problem size, while GVNS is performed slightly better than the HBRGA on the whole. The reason for this result may be that the HBRGA adopts a coding strategy based on a continuous interval, where the processing order of jobs is determined by numerical sorting during decoding. However, this coding strategy is difficult to dig-out high-quality blocks during evolution and has large randomness.

Moreover, a series of statistical analyses of the average PRD values of the four algorithms is performed for different numbers of jobs and machines, and the results are shown in Figs. 7 and Fig. 8, respectively. As shown in Fig. 7, the average PRD values of the IGA_O, GVNS, and HBRGA show an upward trend with the increase in the number of jobs, indicating that the job size has a certain influence on the performance of the algorithm. However, the IGA does not fluctuate significantly and maintains a low level, which is far lower than that of the second-ranked IGA_O. This further confirms the robustness of the proposed IGA. In Fig. 8, it can be seen that as the number of machines increases, the average PRD values of the IGA and IGA_O are relatively stable. The performances of the GVNS and HBRGA are relatively poor and have large fluctuations. In summary, the IGA has a stable performance for various combinations of jobs and machines.

In addition to the job scale and machine numbers, the relationship between the processing time and the waiting time, i.e., parameter W, is also an influential factor of the algorithm performance. The mean plots of the four algorithms for different W values are shown in Fig. 9. It can be seen that for three levels of W, the average PRD values of the IGA, IGA_O, and GVNS all first increase and then decrease. For instance, the average PRD values of the IGA are 0.29%, 0.46%, and 0.43% for $W=1$, $W=2$, and $W=3$, respectively. However, the HBRGA deteriorates as W increases. This shows that the configuration relationship between the processing time and the waiting time has a great impact on the algorithm performance. Indeed, when $W=1$, there are many jobs that do not meet the conditions for group processing; so, the problem is relatively easy to solve, and performance differences between the four algorithms are small. For the case when $W=2$, the combination space of jobs for group processing is large, increasing the difficulty of solving the problem. When $W=3$, although the combination space is very large, the impact of different schemes on $C_{\max}$ is weakened, which reduces the problem-solving difficulty.

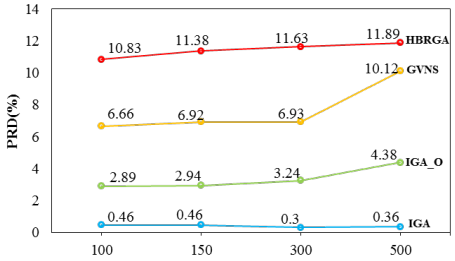

Figure 7 Average PRD values with different number of jobs

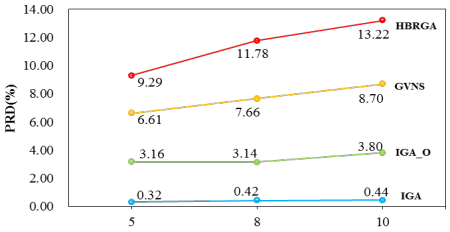

Figure 8 Average PRD values with different number of machines

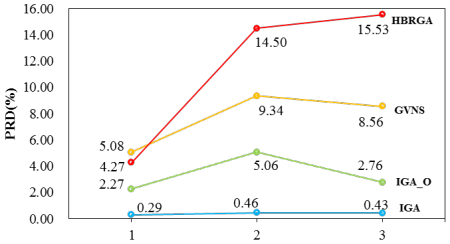

Figure 9 Average PRD values with different *w*

Based on the above comparisons, it can be concluded that the IGA is very effective and superior to the IGA_O, GVNS, and HBRGA at solving the studied problem.

6 CONCLUSIONS

This work studies a new parallel machine scheduling problem with re-entrant and group processing features, which arises from the important production management requirements of hot rolling workshop in modern iron and steel enterprises, but has received little attention in the literature. First, a mixed integerlinear programming model with the makespan criterion is formulated, and then the properties of the studied problem are analyzed. Then, an improved IGA is developed. To improve the performance of the IGA, several new techniques, including an initialization strategy, an enhanced DR procedure, and a tailored local search strategy, are proposed. The

effectiveness of the model and IGA are verified by extensive instances. The results show that the proposed IGA is effective at solving the considered problem, and performs better than the other three mainstream meta-heuristic algorithms.

Although this paper has done some work, there are still the following limitations. ① In terms of problem characteristics, only a single optimization objective is considered, while in actual industrial environments, multiple optimization objectives, such as production efficiency and energy-saving indicators, often need to be considered simultaneously. Besides, this paper only considers the case of two jobs being processed in a group. In actual industrial environments, there may be more jobs being processed in a group, which will be the main direction in the future research. ② The algorithm designed in this work adopts a single threaded implementation method. In the future, distributed parallel algorithms can be developed using multi-threading technology to improve solution performance.

7 REFERENCES

[1] Boudjemline, A., Chaudhry, I. A., Rafique, A. F., Elbadawi, I. A., Aichouni, M., & Boujelbene, M. (2022). Multi-Objective Flexible Job Shop Scheduling Using Genetic Algorithms. *Tehnicki vjesnik-Technical Gazette*, *29*(5), 1706-1713. https://doi.org/10.17559/TV-20211022164333

[2] Shuang, W., Xiaomeng, D., Ting, Z., & Xiaodong, W. (2022). Task Scheduling Based on Grey Wolf Optimizer Algorithm for Smart Meter Embedded Operating System. *Tehnicki vjesnik-Technical Gazette*, *29*(5), 1629-1636. https://doi.org/10.17559/TV-20220518055833

[3] Behmanesh, R. & Rahimi, I. (2021). Improved Ant Colony Optimization for Multi-Resource Job Shop Scheduling: A Special Case of Transportation. *Economic Computation & Economic Cybernetics Studies & Research*, *55*(4), 277-294. https://doi.org/10.24818/18423264/55.4.21.18

[4] Özgür, A., Uygun, Y., & Hütt, M. (2021). A review of planning and scheduling methods for hot rolling mills in steel production. *Computers & Industrial Engineering*, *151*, 106606. https://doi.org/10.1016/j.cie.2020.106606

[5] Liao, C., Lee, C., & Tsai, H. (2016). Scheduling with multi-attribute set-up times on unrelated parallel machines. *International Journal of Production Research*, *54*(16), 4839-4853. https://doi.org/10.1080/00207543.2015.1118574

[6] Bitar, A., Dauzère-Pérès, S., Yugma, C., & Roussel, R. (2016). A memetic algorithm to solve an unrelated parallel machine scheduling problem with auxiliary resources in semiconductor manufacturing. *Journal of Scheduling*, *19*(4), 367-376. https://doi.org/10.1007/s10951-014-0397-6

[7] Chen, C., Fathi, M., Khakifirooz, M., & Wu, K. (2022). Hybrid tabu search algorithm for unrelated parallel machine scheduling in semiconductor fabs with setup times, job release, and expired times. *Computers & Industrial Engineering*, *165*, 107915. https://doi.org/10.1016/j.cie.2021.107915

[8] Soares, L. C. R. & Carvalho, M. A. M. (2022). Application of a hybrid evolutionary algorithm to resource-constrained parallel machine scheduling with setup times. *Computers & Operations Research*, *139*, 105637. https://doi.org/10.1016/j.cor.2021.105637

[9] Abu-Marrul, V., Martinelli, R., Hamacher, S., & Gribkovskaia, I. (2021). Matheuristics for a parallel machine scheduling problem with non-anticipatory family setup times: Application in the offshore oil and gas industry. *Computers & Operations Research*, *128*, 105162. https://doi.org/10.1016/j.cor.2020.105162

[10] Abu-Marrul, V., Martinelli, R., & Hamacher, S. (2021). Scheduling pipe laying support vessels with non-anticipatory family setup times and intersections between sets of operations. *International Journal of Production Research*, *59*(22), 6833-6847. https://doi.org/10.1080/00207543.2020.1828637

[11] Zhang, X. & Chen, L. (2022). A general variable neighborhood search algorithm for a parallel-machine scheduling problem considering machine health conditions and preventive maintenance. *Computers & Operations Research*, *143*, 105738. https://doi.org/10.1016/j.cor.2022.105738

[12] Chung, T., Gupta, J. N. D., Zhao, H., & Werner, F. (2019). Minimizing the makespan on two identical parallel machines with mold constraints. *Computers & Operations Research*, *105*, 141-155. https://doi.org/10.1016/j.cor.2019.01.005

[13] Rau, H. & Cho, K. (2009). Genetic algorithm modeling for the inspection allocation in reentrant production systems. *Expert Systems with Applications*, *36*(8), 11287-11295. https://doi.org/10.1016/j.eswa.2009.03.020

[14] Wang, M. Y., Sethi, S. P., & van de Velde, S. L. (1997). Minimizing Makespan in a Class of Reentrant Shops. *Operations Research*, *45*(5), 702-712.

[15] Wang, K., Qin, H., Huang, Y., Luo, M., & Zhou, L. (2021). Surgery scheduling in outpatient procedure centre with re-entrant patient flow and fuzzy service times. *Omega*, *102*, 102350. https://doi.org/10.1016/j.omega.2020.102350

[16] Wu, X. & Cao, Z. (2022). An improved multi-objective evolutionary algorithm based on decomposition for solving re-entrant hybrid flow shop scheduling problem with batch processing machines. *Computers & Industrial Engineering*, *169*, 108236. https://doi.org/10.1016/j.cie.2022.108236

[17] Frihat, M., B. Hadj-Alouane, A., & Sadfi, C. (2022). Optimization of the integrated problem of employee timetabling and job shop scheduling. *Computers & Operations Research*, *137*, 105332. https://doi.org/10.1016/j.cor.2021.105332

[18] Xu, J., Yin, Y., Cheng, T. C. E., Wu, C., & Gu, S. (2014). A memetic algorithm for the re-entrant permutation flowshop scheduling problem to minimize the makespan. *Applied Soft Computing*, *24*, 277-283. https://doi.org/10.1016/j.asoc.2014.07.002

[19] Shin, H. J. (2015). A dispatching algorithm considering process quality and due dates: an application for re-entrant production lines. *The International Journal of Advanced Manufacturing Technology*, *77*(1-4), 249-259. https://doi.org/10.1007/s00170-014-6436-9

[20] Chakhlevitch, K. & Glass, C. A. (2009). Scheduling reentrant jobs on parallel machines with a remote server. *Computers & Operations Research*, *36*(9), 2580-2589. http://doi.org/10.1016/j.cor.2008.11.007

[21] Costa, A., Cappadonna, F. V., & Fichera, S. (2020). Minimizing makespan in a Flow Shop Sequence Dependent Group Scheduling problem with blocking constraint. *Engineering Applications of Artificial Intelligence*, *89*, 103413. http://doi.org/10.1016/j.engappai.2019.103413

[22] Divyashree, H. B., Puttamadappa, C., & Prasad, K. S. (2022). Multi Objective Energy based Hybrid Optimization Algorithm for Clustering and Routing in WSN. *Journal of System and Management Sciences*, *12*(1), 80-497.

[23] Singh, G., Prakash, S., & Kumar, S. (2021). Minimizing Makespan Time in Cloud Computing using Heuristic Elasticity based Dynamic Task Scheduling Algorithms. *Journal of System and Management Sciences*, *11*(2), 29-47. https://doi.org/10.33168/JSMS.2021.0203

[24] Huang, Y., Huang, S., & Jin, C. (2021), 3D-Container Loading Problem with a Distribution Plan Based on Hybrid Quantum Genetic Algorithm. *Economic Computation And Economic Cybernetics Studies And Research*, *55*(4), 117-

132. https://doi.org/10.24818/18423264/55.4.21.08
[25] Sarkar, T., Salauddin, M., Hazra, S., Choudhury, T., & Chakraborty, R. (2021). Comparative Approach of Artificial Neural Network and Thin Layer Modelling for Drying Kinetics and Optimization of Rehydration Ratio for Bael (Aegle marmelos (L) correa) Powder Production. *Economic Computation And Economic Cybernetics Studies And Research*, *55*(1), 167-184. https://doi.org/10.24818/18423264/55.1.21.11
[26] Wang, Y. J., Wang, N. D., Cheng, S. M., Zhang, X. C., Liu, H. Y., Shi, J. L., Ma, Q. Y., & Zhou, M. J. (2021). Optimization of disassembly line balancing using an improved multi-objective Genetic Algorithm. *Advances in Production Engineering & Management*, *16*(2), 240-252. https://doi.org/10.14743/apem2021.2.397
[27] Ruiz, R. & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*(3), 2033-2049. https://doi.org/10.1016/j.ejor.2005.12.009
[28] Arroyo, J. E. C., Leung, J. Y. T., & Tavares, R. G. (2019). An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times. *Engineering Applications of Artificial Intelligence*, *77*, 239-254. https://doi.org/10.1016/j.engappai.2018.10.012
[29] Mecler, D., Abu-Marrul, V., Martinelli, R., & Hoff, A. (2022). Iterated greedy algorithms for a complex parallel machine scheduling problem. *European Journal of Operational Research*, *300*(2), 545-560. https://doi.org/10.1016/j.ejor.2021.08.005
[30] Rodriguez, F. J., Lozano, M., Blum, C., & García-Martínez, C. (2013). An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers & Operations Research*, *40*(7), 1829-1841. http://doi.org/10.1016/j.cor.2013.01.018
[31] Pan, Q. (2016). An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling. *European Journal of Operational Research*, *250*(3), 702-714. http://doi.org/10.1016/j.ejor.2015.10.007
[32] Arroyo, J. E. C. & Leung, J. Y. T. (2017). An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Computers & Industrial Engineering*, *105*, 84-100. http://doi.org/10.1016/j.cie.2016.12.038
[33] Rerkjirattikal, P., Wanwarn, T., Starita, S., Huynh, V. N., Supnithi, T., & Olapiriyakul, S. (2020). Heuristics for noise-safe job-rotation problems considering learning-forgetting and boredom-induced job dissatisfaction effects. *Environmental Engineering & Management Journal*, *19*(8), 1325-1337.
[34] Ren, J. F., Ye, C. M., & Li, Y. (2021). A new solution to distributed permutation flow shop scheduling problem based on NASH Q-Learning. *Advances in Production Engineering & Management*, *16*(3), 269-284. https://doi.org/10.14743/apem2021.3.399.

Contact information:

Shuaipeng YUAN, Lecturer
(Corresponding author)
School of Economics and Management,
University of Science & Technology Beijing,
Beijing 100083 P. R. China
E-mail: 17801002601@163.com

Bailin WANG, Professor
School of Economics and Management,
University of Science & Technology Beijing,
Beijing 100083 P. R. China
E-mail: wangbl@ustb.edu.cn

Tieke LI, Professor
School of Economics and Management,
University of Science & Technology Beijing,
Beijing 100083 P. R. China
E-mail: tieke@ustb.edu.cn