Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.049480

ARTICLE

A Cooperated Imperialist Competitive Algorithm for Unrelated Parallel Batch Machine Scheduling Problem

Deming Lei* **and Heen Li**

College of Automation, Wuhan University of Technology, Wuhan, 430070, China

*Corresponding Author: Deming Lei. Email: deminglei11@163.com

Received: 09 January 2024 Accepted: 29 March 2024 Published: 15 May 2024

ABSTRACT

This study focuses on the scheduling problem of unrelated parallel batch processing machines (BPM) with release times, a scenario derived from the moulding process in a foundry. In this process, a batch is initially formed, placed in a sandbox, and then the sandbox is positioned on a BPM for moulding. The complexity of the scheduling problem increases due to the consideration of BPM capacity and sandbox volume. To minimize the makespan, a new cooperated imperialist competitive algorithm (CICA) is introduced. In CICA, the number of empires is not a parameter, and four empires are maintained throughout the search process. Two types of assimilations are achieved: The strongest and weakest empires cooperate in their assimilation, while the remaining two empires, having a close normalization total cost, combine in their assimilation. A new form of imperialist competition is proposed to prevent insufficient competition, and the unique features of the problem are effectively utilized. Computational experiments are conducted across several instances, and a significant amount of experimental results show that the new strategies of CICA are effective, indicating promising advantages for the considered BPM scheduling problems.

KEYWORDS

Release time; assimilation; imperialist competitive algorithm; batch processing machines scheduling

1 Introduction

Unlike traditional scheduling, batch processing machines (BPM) scheduling consists of at least one BPM, and on BPM all jobs in a batch are processed simultaneously after the batch is formed. BPM scheduling problems widely exist in many real-life industries such as casting, chemical engineering, semiconductors, transportation, etc. BPM can be divided into parallel BPM and serial BPM, the processing time of a batch on the former is defined as the maximum processing time of all jobs in the batch, and the processing time of a batch on the latter is defined as the sum of processing time of all jobs in the batch. BPM scheduling problems can be divided into single BPM scheduling, parallel machine scheduling, and hybrid flow shop scheduling with BPM, etc., which have received extensive attention.

There are some works about single BPM scheduling problem, in which all batches are processed by a BPM. Uzsoy [1] first studied the problem with job size, proved that it is NP-hard, and proposed a heuristic algorithm to minimize makespan and total completion time. Lee [2] presented a polynomial

algorithm and pseudo-polynomial-time algorithm to solve the problem with dynamic job arrivals. Melouk et al. [3] solved the problem with job sizes by a simulated annealing algorithm. Zhou et al. [4] considered the problem with unequal release times and job sizes and proposed a self-adaptive differential evolution algorithm with adaptively chosen mutation operators based on their historical performances to minimize makespan. Yu et al. [5] proposed a branch-and-price algorithm to solve additive manufacturing problems with the minimization of makespan. Zhang et al. [6] presented two heuristics to minimize total weighted earliness and tardiness penalties of jobs.

The parallel BPM scheduling problem has attracted much attention. Regarding the uniform parallel BPM scheduling problem, some results are obtained. Xu et al. [7] presented a Pareto-based ant colony system to simultaneously minimize makespan and maximum tardiness. Jiang et al. [8] presented a hybrid algorithm with discrete particle swarm optimization and genetic algorithm (GA), in which a heuristic and a local-search strategy are introduced. Zhang et al. [9] proposed a multi-objective artificial bee colony (ABC) to solve the problem of machine eligibility in fabric dyeing process. Jia et al. [10] proposed a fuzzy ant colony optimization (ACO) for the problem with fuzzy processing time. Liu et al. [11] designed a GA and a fast heuristic for the problem in coke production. Jia et al. [12] applied two heuristics and an ACO for the problem with arbitrary capacities. Li et al. [13] developed a discrete bi-objective evolutionary algorithm to minimize maximum lateness and total pollution emission cost. Xin et al. [14] developed an efficient 2-approximation algorithm for the problem with different BPM capacities and arbitrary job sizes.

A number of works have been obtained on unrelated parallel BPM scheduling problems. Arroyo et al. [15] proposed an effective iterated greedy for the problem with non-identical capacities and unequal ready times. Lu et al. [16] presented a hybrid ABC with tabu search to minimize the makespan of the problem with maintenance and deteriorating jobs. Zhou et al. [17] provided a random key GA for the problem with different capacities and arbitrary job sizes. Zhang et al. [18] proposed an improved evolutionary algorithm by combining a GA with a heuristic placement strategy for the problem in additive manufacturing. Kong et al. [19] applied a shuffled frog-leaping algorithm with variable neighborhood search for the problem with nonlinear processing time. Song et al. [20] developed a self-adaptive multi-objective differential evolution algorithm to minimize total energy consumption and makespan. Zhang et al. [21] formulated a mixed-integer programming (MIP) model and presented an improved particle swarm optimization algorithm for the problem with production and delivery in cloud manufacturing. Fallahi et al. [22] studied the problem in production systems under carbon reduction policies and used non-dominated sorting genetic algorithm-II and multi-objective gray wolf optimizer to simultaneously minimize makespan and total cost. Xiao et al. [23] proposed a tabu-based adaptive large neighborhood search algorithm for the problem with job sizes, ready times, and the minimization of makespan. Zhang et al. [24] provided a MIP model and a modified elite ant system with local search (MEASL) to minimize the total completion time of the problem with release times, and job sizes. Jiang et al. [25] applied an iterated greedy (IG) algorithm for the problem of release times, job sizes, and incompatible job families. Ji et al. [26] developed a hybrid large neighborhood search (HLNS) with a tabu strategy and local search to solve the problem with job sizes, ready times, and incompatible family. Ou et al. [27] presented an efficient polynomial time approximation scheme with near-linear-time complexity to solve the problem with dynamic arrivals job and rejection.

As stated above, the existing works on single BPM scheduling and parallel BPM scheduling are mainly about job sizes, incompatible family, and BPM capacity constraints, that is, the sum of the weight of all jobs in a batch cannot exceed the prefixed capacity, parallel BPM scheduling problems are handled in actual production processes like coke production, fabric dyeing, and additive

manufacturing, which are close to the real situation of manufacturing process and their optimization results have high application values; however, parallel BPM scheduling problems with more constraints are not studied fully. For example, in the moulding process of a foundry [28,29], a batch is first allocated into a sandbox and then the sandbox is added to the assignment machine of the batch, for each batch, has to meet sandbox volume constraint and machine capacity constraint, and complexity of the problem increases. This is a challenge in moulding process in a foundry. On the other hand, release times of all jobs are mostly different in unrelated parallel BPM scheduling problems [24,25], release times have a positive impact on the performance of the schedule and are a frequently considered real-life condition, which also exist in moulding process, so it is necessary to deal with parallel BPM scheduling problem with sandbox volume constraint and release times.

The imperialist competitive algorithm (ICA) is an intelligent optimization algorithm inspired by social and political behavior [30]. Each intelligent optimization algorithm has its own advantages [31–34]. ICA has many notable features such as good neighborhood search ability, effective global search property, good convergence rate, and flexible structure [35]. As the main method of production scheduling [36–40], ICA has been successfully applied to solve parallel machine scheduling problem (PMSP) [41–45]. Lei et al. [37] proposed an ICA with a novel assimilation strategy to deal with low-carbon PMSP. Yadani et al. [46] developed a hybrid ICA for PMSP with two-agent and tool change activities. The good searchability and advantages of ICA in solving PMSP are tested and proven. The parallel BPM scheduling problem is the extended PMSP and the same coding can be used in these two problems. The successful applications of ICA to PMSP reveal that ICA has potential advantages in solving parallel BPM scheduling, so ICA is used.

In this study, an unrelated parallel BPM scheduling problem with release times is considered, which is refined from moulding process of a foundry, and a cooperated imperialist competitive algorithm (CICA) is presented to minimize makespan. To produce high-quality solutions, the number of empires is not used as a parameter and four empires always exist throughout the search process, cooperation between the strongest empire and the weakest empire is applied in the assimilation of these two empires, and a combination of the two remaining empires is used in the assimilation of these two empires. A new imperialist competition is given. Extensive experiments are conducted and the computational results demonstrate that new strategies of CICA are effective and that CICA has promising advantages on considered problems.

The remainder of the paper is organized as follows. The problem description is described in Section 2. Section 3 shows the proposed CICA for unrelated parallel BPM scheduling problems with the release times stage. Numerical test experiments on CICA are reported in Section 4. Conclusions and some topics of future research are given in the final section.

2 Problem Description

Unrelated parallel BPM scheduling problem in moulding process of a foundry is described as follows. n jobs $J_1, J_2, \ldots, J_n$ are processed by m parallel BPM $M_1, M_2, \ldots, M_m$, r_i indicates release times of J_i, p_{ik} denotes processing time of J_i on machine M_k. All jobs are divided into l families according to materials. Each mater-ial corresponds to a job family and jobs of different families cannot be grouped into a batch. $f_i \in \{1, 2, \ldots, l\}$ indicates the family of job J_i, J_i has weight w_i and volume v_i. There are sufficient sandbox and all sandboxes have the same volume V, all machines are given the same capacity W. Each batch is placed in a sandbox and then sandbox is placed in BPM for moulding, so two constraints are required to be satisfied for each batch: Total volume of all jobs in the batch cannot exceed V and total weight of all jobs in the batch cannot exceed W. Processing time of a batch

is maximum processing time of all jobs in the batch. Release time of a batch is maximum release time of job in the batch.

There are some constraints on jobs and machines:

Each BPM can only handle one batch at a time.

No job can be processed in different batches on more than one BPM.

Operations cannot be interrupted.

The problem has three sub-problems: (1) batch formation for deciding which jobs used to form each batch, (2) BPM assignment for choosing a BPM for each batch, (3) batch scheduling for determining processing sequence of all batches on each machine. There are strong coupled relations among them. Batch formation decides directly the optimization content of other two sub-problems and optima solution of the problem can be obtained by comprehensive coordination of three sub-problems.

The goal of problem is to minimize makespan under the condition that all constraints are met.

$$C_{\max} = \max_{i=1,2,\ldots,n} C_i \tag{1}$$

where C_i is completion time of J_i, $C_{\max}$ indicates maximum completion time of all jobs.

Table 1 shows an example with 12 jobs, 3 families and 3 parallel BPMs, $W = 10$, $V = 10$. A schedule of the example is shown in Fig. 1.

Table 1: Processing data of the example

J_i	w_i	v_i	f_i	r_i	p_{i1}	p_{i2}	p_{i3}
J_1	4	5	1	2	2	3	4
J_2	2	1	2	4	3	6	4
J_3	2	2	1	5	1	3	2
J_4	3	4	2	3	3	2	4
J_5	2	5	1	7	5	3	5
J_6	8	10	3	1	2	4	3
J_7	9	1	3	3	4	2	3
J_8	5	5	1	1	6	5	2
J_9	1	7	3	4	6	7	4
J_{10}	5	4	2	3	5	3	3
J_{11}	9	5	1	1	3	2	7
J_{12}	4	7	2	0	3	6	5

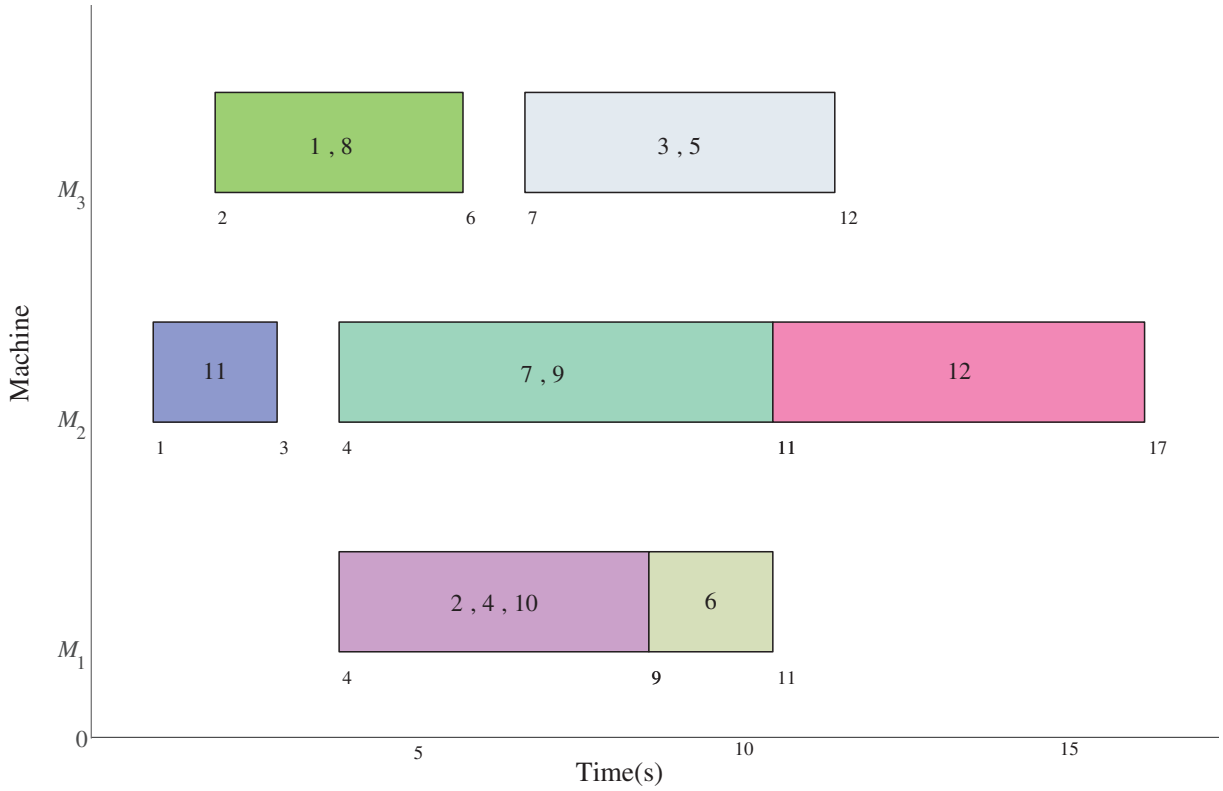

Figure 1: A schedule of the example

3 CICA for Parallel BPM Scheduling

In the existing ICAs [30,35], there are N_{im} initial empires and N_{im} is used as parameter. In this study, CICA is proposed, in which N_{im} is not a parameter and set to be 4 in the whole search process, then assimila-tion based on cooperation between two empires and assimilation with empire combination are implemented respectively and a new imperialist competition is given. The detailed steps of CICA are shown below.

3.1 Initialization and Formation of Four Empires

Lei et al. [47] presented a two-string representation to describe solution of unrelated PMSP, which can be directly used to describe solution of unrelated parallel BPM scheduling problem. For the problem with n jobs and m BPM, its solution is represented as a scheduling string $[\pi_1, \pi_2, \ldots, \pi_n]$, and a machine assignment string $[\theta_1, \theta_2, \ldots, \theta_n]$, where $\pi_i \in \{1, 2, \ldots, n\}$, $\theta_i \in \{1, 2, \ldots, m\}$.

There are some differences of the above method from coding method [47]: Scheduling string is a perm-utation of all jobs, machine assignment string is used for all batches, the number η of batches is determined by batch formation. Usually, $\eta < n$, so only the first η elements of machine assignment string are used in decoding.

The decoding procedure is described below:

Step 1: Let $k = 1$, $b\text{-}th = 1$, $WB_{b\text{-}th} = 0$, $VB_{b\text{-}th} = 0$, let $B_{b\text{-}th}$ be empty. Each job J_i is assigned a $mark_i$, if job is not in a batch, $mark_i = 0$, otherwise $mark_i = 1$.

Step 2: Repeat the following steps until each job is assigned into a batch.

1) choose first job π_b with $mark_{\pi_b} = 0$ and the first machine M_{θ_k} from machine assignment string, then $B_{b\text{-}th} = B_{b\text{-}th} \cup \{J_{\pi_b}\}$, $mark_{\pi_b} = 1$, $WB_{b\text{-}th} = w_{\pi_b} + WB_{b\text{-}th}$, $VB_{b\text{-}th} = v_{\pi_b} + VB_{b\text{-}th}$.

2) repeat the following steps until no job π_c can be chosen or $WB_{b\text{-}th}+w_{\pi_c} > W$ or $VB_{b\text{-}th}+v_{\pi_c} > V$: Choose a job π_c with $mark_{\pi_c} = 0$ and $f_{\pi_c} = f_{\pi_b}$ from scheduling string, if $WB_{b\text{-}th}+w_{\pi_c} < W$, $VB_{b\text{-}th}+v_{\pi_c} < V$, then $B_{b\text{-}th} = B_{b\text{-}th} \cup \{J_{\pi_c}\}$, $mark_{\pi_c} = 1$, $WB_{b\text{-}th} = w_{\pi_c} + WB_{b\text{-}th}$, $VB_{b\text{-}th} = v_{\pi_c} + VB_{b\text{-}th}$.

3) Obtain batch $B_{b\text{-}th}$, $b\text{-}th = b\text{-}th + 1$, let $B_{b\text{-}th}$ be empty, $WB_{b\text{-}th} = 0$, $VB_{b\text{-}th} = 0$, $k = k + 1$.

Step 3: η batches are finally formed and processed sequentially in terms of $B_1, B_2, \ldots, B_\eta$.

Where $WB_{b\text{-}th}$ and $VB_{b\text{-}th}$ are total weight of all jobs and total volume of all jobs in batch $B_{b\text{-}th}$.

For the example in Table 1, a solution is scheduling string [8, 2, 6, 4, 11, 1, 10, 9, 5, 12, 7, 3] and machine as-signment string [3, 1, 1, 2, 2, 3, 2, 3, 1, 2, 1, 3]. The corresponding schedule is shown in Fig. 1. Job J_8 is chosen first, $B_1 = \{J_8\}$, $WB_1 = 5$, $VB_1 = 5$, then find a job J_1 with the same family with J_8, $WB_1 + w_1 = 9$, $VB_1 + v_1 = 10$, $B_1 = \{J_8, J_1\}$. When job J_3 is considered, sandbox volume constraint is violated, so $B_1 = \{J_8, J_1\}$. Other batches are obtained in the same way. $B_2 = \{J_2, J_4, J_{10}\}$, $B_3 = \{J_6\}$, $B_4 = \{J_{11}\}$, $B_5 = \{J_9, J_7\}$, $B_6 = \{J_5, J_3\}$. On M_1, processing sequence of B_1, B_6. The processing sequence on M_2 and M_3 are B_4, B_5, B_7 and B_2,B_3, respectively. $C_{\max}$ is 17.

An initial population P with N solutions are randomly generated, then four initial empires are construc-ted based on initial population P.

Four initial empires are constructed in the following way:

Step 1: Calculate cost $c_i = C_{\max}^{x_i}$ of each solution $x_i \in P$, sort all solutions in P in ascending order of cost.

Step 2: Choose four solutions with lowest cost as imperialists, which are IM_1, IM_2, IM_3, IM_4, and calc-ulate normalized cost $\bar{c}_k$ of IM_k, pow_k and $NC_k = round\,(N_{col} \times pow_k)$.

Step 3: Randomly allocate NC_k colonies for each imperialist k.

Where $round\,(x)$ is a function that gives the nearest integer of x, $N_{col} = N - 4$ denotes total number of co-lonies. $pow_k = \bar{c}_k / \sum_{i=1}^{4} c_i$ is power of imperialist k, NC_k indicates the number of colonies in empire k.

Normalized cost $\bar{c}_k$ of IM_k is usually defined as $\bar{c}_k = \max_l \{c_l\} - c_k$, so at least one imperialist has $\bar{c}_k = 0$, $pow_k = 0$ and no allocated colonies and the corresponding empire will have no colony, as a result, assimilation and revolution cannot be done in the empire, $\bar{c}_k$ is defined to avoid the above case:

$$\bar{c}_k = 2 \times \max_l \{c_l\} - c_k \tag{2}$$

$\overline{TC}_k$ of empire k is defined and four empires are sorted in the descending order of $\overline{TC}_k$, suppose $\overline{TC}_1 \geq \overline{TC}_2 \geq \overline{TC}_3 \geq \overline{TC}_4$, the strongest empire is empire 1, empire 4 is the weakest empire, empires 2,3 have close normalization total cost.

$$\overline{TC}_k = \bar{c}_k + \xi \sum_{\lambda \in T_k} \frac{\bar{c}_\lambda}{NC_k}, k = 1, 2, 3, 4. \tag{3}$$

where T_k is set of colonies possessed by imperialist k and ξ is real number, $\xi = 0.1$.

3.2 Assimilation and Revolution

Unlike the previous ICAs [30,35], CICA has new implementation way for assimilation. In CICA, assimilations of empires 1,4 are executed together based on their cooperation because there has greater difference on their normalization total cost, empires 2,3 have close normalization total cost, when their assimilations are done, they are first merged and assimilation is conducted for the merged empire. N_{im} is fixed to be 4 to achieve the above two kinds of assimilations; moreover, imperialist competition will be simplified.

Cooperation-based assimilation of empires 1,4 is shown as follows:

Step 1: Sort all colonies in empire 1 in the ascending order of c_i, decide the first α colonies including the best colony $\lambda^* \in T_1$, determine the best α colonies of empire 4 using the same approach and include them into a set Λ, $\tau = 1$.

Step 2: Repeat the following steps until $\tau > \alpha$: Choose the τ-*th* colony $\lambda \in T_1$ and the τ-*th* colony $\overline{\lambda} \in T_4$ execute global search between $\lambda, \overline{\lambda}$, a new solution z is obtained, if $C^z_{\max}$ less than one of $C^{\lambda}_{\max}, C^{\overline{\lambda}}_{\max}$, suppose $C^z_{\max} < C^{\lambda}_{\max}$, then update the set θ with λ and let z substitute for λ; if $C^z_{\max} < C^{\lambda}_{\max}$ and $C^z_{\max} < C^{\overline{\lambda}}_{\max}$, then update the set θ with the worse one of $\lambda, \overline{\lambda}$ and let z substitute for the worse one of $\lambda, \overline{\lambda}$.

Step 3: For each colony $\lambda \in \Lambda$, execute global search between λ and $\lambda^* \in T_1$, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then update the set θ with λ and let z substitute for λ. If $C^z_{\max} \geq C^{\lambda}_{\max}$, then conduct global search between λ and IM_1, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then update the set θ with λ and replace λ with z.

Step 4: Choose the best colony in Λ, if it better than IM_4, then exchange it with IM_4.

Step 5: For each colony $\lambda \in T_1 \backslash \{\lambda^*\}$, conduct global search between λ and λ^*, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then update θ with λ and replace λ with z; else, execute global search between λ and IM_1, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then update the set θ with λ and replace λ with z.

Step 6: For each colony $\lambda \in T_4 \backslash \Lambda$, choose a $\overline{\lambda} \in \Lambda \cup \{IM_4\}$, by using roulette selection, execute global search between λ and $\overline{\lambda}$, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then update the set θ with λ and let z su-bstitute for λ.

Where selection probability of each solution x_i is $\left(\sum_{y \in \Lambda \cup \{IM_4\}} C^y_{\max} - C^{x_i}_{\max}\right) / \sum_{y \in \Lambda \cup \{IM_4\}} C^y_{\max}$ for roulette selection.

After the best α colonies of empires 1, 4 are decided respectively, then cooperation is conducted by global search on α pairs of the decided best colonies$\lambda \in T_1, \overline{\lambda} \in T_4$ in step 2, and the usage of the best colony $\lambda^* \in T_1$ or IM_1 for improving solution quality of the best α colonies of empire 4 in step 3.

Empires 2,3 have close $\overline{TC}_2, \overline{TC}_3$ and there are high similarity between them, to avoid the waste of com-puting resource on the worst solutions, empires 2,3 are first combined into a new empire D, assimilation is done on all colonies of D except Q the chosen worst solutions in D.

Combination-based assimilation in empires 2,3 is described below:

Step 1: Obtain temporary empire D by combining empires 2,3 together, imperialists of empire D are defined as IM_2 and IM_3, choose Q colonies with the biggest cost, let Θ be a set of these colonies.

Step 2: For each colony $\lambda \in D \backslash \Theta$, select one imperialist from IM_2 and IM_3 by roulette selection, suppose IM_2 is selected, execute global search between λ and IM_2, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then if λ is one of Q colonies of D with the lowest cost, then random select colony $\overline{\lambda} \in \Theta$,

update $\theta$with $\overline{\lambda}$, replace λ with z; if λ is not one of the best Q colonies in D, then conduct multiple neighborhood search on colon λ, random select colony $\overline{\lambda} \in \Theta$, update θ with $\overline{\lambda}$, replace λ with z.

Step 3: Assign all solutions of D into their original empire.

where roulette selection is done, $\{IM_2, IM_3\}$ substitutes for $\Lambda \cup \{IM_4\}$.

The set θ is used to retain historical optimization data in the search process. The process of updating θ with solution x is shown as follows: If the number of solutions in θ is than its maximum size I, then the worst solution is eliminated and x is added to θ if x is better than the worst solution in θ, otherwise, x is added to θ.

Global search is described below: For solutions x and y, if $rand \leq 0.5$, order crossover [48] is executed for scheduling strings of x, y, otherwise, two-point crossover [40] is performed between machine assign-ent string of x and y. Where *rand* is random number following uniform distribution on [0,1].

5 neighborhood structures are applied, $\mathcal{N}_1$ is used to swap two randomly selected π_{k_1} and π_{k_2}. $\mathcal{N}_2$ is swap operator on machine assignment string and applied to swap two randomly selected θ_{k_1} and θ_{k_2}, $k_1, k_2 < \eta$. $\mathcal{N}_3$ is shown as follows: Decide machine M_{k_1} with smallest completion time and M_{k_2} with the biggest comp-letion time, randomly choose $\theta_{i_1} = k_1$ and $\theta_{i_2} = k_2, i_1, i_2 \leq \eta$ and swap $\theta_{i_1}, \theta_{i_2}$. $\mathcal{N}_4$ is executed in the following way: Choose machine M_{k_1} with biggest completion time, randomly select $\theta_i = k_1, i \leq \eta$, randomly choose M_{k_2}, let $\theta_i = k_2$. $\mathcal{N}_5$ is described as follows: Decide all jobs on a machine with biggest completion time in scheduling string, and sort these jobs in ascending order of release times.

Multiple neighborhood search is below: Let $\vartheta = 1$, repeat the following steps until $\vartheta = 5$: a new solution $z \in \mathcal{N}_\vartheta(x)$ is obtained, if $C^z_{\max} < C^x_{\max}$, then update $\theta$with x, replace x with z, $\vartheta = \vartheta + 1$. Where $\mathcal{N}_\vartheta(x)$ is denotes neighborhood solution set of solution x by using $\mathcal{N}_\vartheta$.

When N_{im} is fixed to be 4, empires 1, 4 with bigger differences on $\overline{TC}_k$ can be easily obtained, cooperation between them can be done and global search ability can be enhanced; meanwhile, empires 2,3 will have close normalized total cost and the search in empire 2 is similar with the search in empire 3 and the waste of computing resource will occur on some worst solutions twice. When the above assimilation is done on empires 2,3, the waste of computing resource will diminish greatly, thus, it can be found that the usage of four empires is appropriate, so N_{im} is not parameter in CICA and always equal to 4.

Revolution of empire k is conducted in the following way:

Step 1: Determine the number δ of colonies in empire k according to revolution probability R.

Step 2: Sort all colonies of empire k in ascending order of cost and decide δ colonies with the smallest cost, then for each decided colony λ, perform multiple neighborhood on colony λ, a new solution z is obtained, if $C^z_{\max} < C^\lambda_{\max}$, then replace the worst colony in T_k with λ, replace λ with z; otherwise, if solution z is better than the worst colony of T_k, replace the worst colony of T_k with z.

3.3 Algorithm Description

As stated above, CICA is made up of initialization, formation of four empires, assimilation, revolution and imperialist competition.

Its detailed steps of CICA are shown below:

Step 1: Randomly produce initial population P and construct four initial empires.

Step 2: While the stopping condition is not met, **do**

Sort four empires in the descending order of $\overline{TC}_k$.

Execute assimilation in empires 1,4.

Perform assimilation in empires 2,3.

Execute revolution.

Apply imperialist competition.

End While

Only four empires are used and exist in the whole search process, so a new imperialist competition is given to adapt this new situation.

The new imperialist competition is described as follows:

1) Calculate $\overline{TC}_k$ and $EP_k, k = 1, 2, 3, 4$, construct vector $[EP_1 - rand, EP_2 - rand, EP_3 - rand, EP_4 - rand]$, choose empire g with the biggest $EP_g - rand$ as winning empire. Suppose $g = 1$.

2) Choose I colonies with the lowest cost in empire g, for each chosen colony λ, execute global search between λ and IM_g, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then update θ with λ and replace λ with z, then execute multiple neighborhood search on colony λ.

3) Construct vector $[EP_2 - rand, EP_3 - rand, EP_4 - rand]$, empire g with the biggest $EP_g - rand$ as winning empire, suppose $g = 2$, then choose I colonies with the lowest cost of empire 2 and execute global search between each chosen colony and IM_g as done in step 2).

4) Choose empire g with the biggest $EP_g - rand$ from vector $[EP_3 - rand, EP_4 - rand]$, suppose $g = 3$, then choose I colonies with the lowest cost of empire 3 and execute global search between each chosen colony and IM_g as done in step 2).

5) In empire 4, execute multiple neighborhood search for each solution $x \in \theta$, delete I worst colony from empire 4, add all solutions of θ into empire 4.

Where EP_k denotes power of empire k,

$$EP_k = \left| \frac{\overline{TC}_k}{\sum_{i=1}^{4} \overline{TC}_i} \right| \tag{4}$$

Unlike the existing ICAs [30,35], CICA has four empires and no elimination of empire. In CICA, assimilations of empires 1,4 are handled together and cooperation is used, assimilations of empires 2,3 are conducted on the temporary empire formed by these two empires, a new imperialist competition is also given. These new features can lead to the enhanced global search ability and the avoidance of the waste of computing resource.

4 Computational Experiments

Extensive experiments are conducted to test the performance of CICA for considered parallel BPM scheduling problem. All experiments are implemented by using Microsoft Visual Studio C++2022 and run on 8.0 G RAM 2.4 Hz CPU PC.

4.1 Instances, Comparative Algorithms and Metrics

96 instances are used, each of which depicted by $n \times l \times m$, where $n \in \{10, 20, 40, 60, 100, 140, 180, 220, 260, 300\}$, $l \in \{3, 4\}$, $m \in \{3, 4, 5\}$, $p_{ik} \in [10, 50]$, $v_i \in [1, 10]$, $w_i \in [1, 10]$, $V = 10$, $W = 10$, $r_i \in [0, 25]$. p_{ik}, v_i, w_i and r_i are integer following uniform distribution in the above intervals. These instances consist of small-scale ones, medium-scale ones and large-scale ones and can be show the optimization ability differences of different algorithms.

Zhang et al. [24] proposed a MEASL to minimize makespan for parallel BPM scheduling problem with release time, job size and processing time. Jiang et al. [25] presented IG algorithm for parallel BPM scheduling problem with release time, job sizes, incompatible job families. Ji et al. [26] provided HLNS to solve parallel BPM scheduling problem with release time, job sizes, incompatible job families and makespan minimization.

MEASL, IG and HLNS can be directly applied to solve the considered parallel BPM scheduling probl-em and the computational results show that these three algorithms have promising advantages on solving parallel BPM scheduling, so they are chosen as comparative algorithms.

To show the effect of new strategies of CICA, CICA is compared with ICA [30,35], in ICA, assimilatio-n of empire k, is done below: For each colony $\lambda \in T_k$, execute global search between λ, IM_k, a new solution z is obtained, if $C^z_{\max} < C^{\lambda}_{\max}$, then replace λ with z. When revolution is done, multiple neighborhood search acts on the chosen colony.

Three metrics are used. For each instance, each algorithm randomly runs 10 times and an elite solution with the smallest makespan is obtained in a run, MIN is the best solution found in 10 runs, MAX denotes the worst elite solutions in 10 runs and AVG indicates the average makespan of 10 elite solutions. MIN, AVG, MAX are used to measure convergence, average performance and stability of algorithms. These metrics are often used to evaluate results of single objective problem.

4.2 Parameter Settings

It can be found that CICA can converges well when $0.6 \times n$ s CPU time reaches; moreover, when $0.6 \times n$ s CPU time is applied, MEASL, IG, HLNS and ICA also converge fully within this CPU time, so this time is chosen as stopping condition.

Other parameters of CICA, namely N, α, Q, R and I, are tasted by using Taguchi method [49] on instance $140 \times 3 \times 3$. The levels of each parameter are shown in Table 2. CICA with each combination runs 10 times independently for the chosen instance.

Table 2: Levels of parameters

Parameters	Factor Level		
	1	2	3
N	50	60	70
α	3	5	7
Q	5	6	7
R	0.4	0.5	0.6
I	5	6	7

Fig. 2 shows the result of MIN and S/N ratio, which is defined as $-10 \times \log_{10}\left(\text{MIN}^2\right)$. It can be found in Fig. 2 that CICA with following combination $N = 60$, $\alpha = 5$, $Q = 6$, $R = 0.5$, $I = 6$ can be obtain better results than CICA with other combinations, so above combination is adopted.

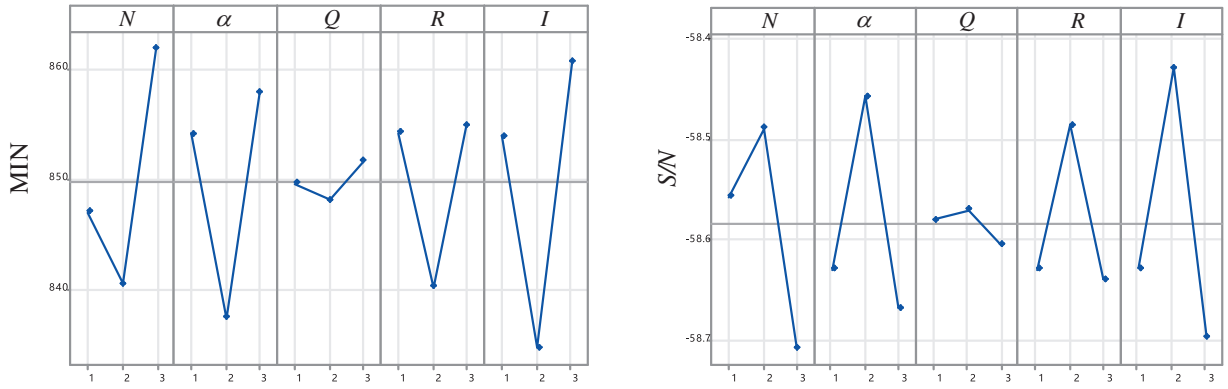

Figure 2: Main effect plot for mean MIN and S/N ratio

Parameters of ICA are shown below. $N = 60$, $R = 0.5$, $N_{im} = 4$ and stopping condition is $0.6 \times ns$ CPU time. These settings are obtained by experiments. All parameters of MEASL, IG and HLNS except stopping condition are obtained directly from [24–26]. The experimental results show that those setting of each comparative algorithm are still effective and comparative algorithms with those setting can produce better results than MEASL, IG and HLNS with other settings.

4.3 Results and Discussions

CICA is compared with MEASL, IG, HLNS and ICA. Each algorithm randomly runs 10 times on each instance. Tables 3–5 describe corresponding results of five algorithms. Figs. 3 and 4 show box plots of all algorithms and convergence curves of instances $100 \times 3 \times 3$ and $220 \times 3 \times 3$. The relative percentage deviation (RPD) between the best performs algorithm and other four algorithms is used in Fig. 3. RPD_{MIN}, RPD_{AVG}, RPD_{MAX} are defined:

$$\text{RPD}_{\text{MIN}} = \frac{\text{MIN} - \text{MIN}^*}{\text{MIN}^*} \times 100\% \tag{5}$$

where MIN^* (MAX^*, AVG^*) is the smallest MIN (MAX, AVG) obtained by all algorithms, when MIN and MIN^* are replaced with MAX(AVG) and $\text{MAX}^*(\text{AVG}^*)$, respectively, RPD_{MAX} (RPD_{AVG}) is obtained in the same way.

Table 3: Computational results of five algorithms on MIN

Instance	CICA	ICA	MEASL	IG	HLNS	Instance	CICA	ICA	MEASL	IG	HLNS
$10 \times 3 \times 3$	**50**	50	51	55	53	$160 \times 3 \times 3$	**980**	1196	1088	1459	1248
$10 \times 3 \times 4$	**48**	56	54	65	62	$160 \times 3 \times 4$	**741**	907	824	1081	948
$10 \times 3 \times 5$	**46**	46	46	47	48	$160 \times 3 \times 5$	**565**	724	620	836	727
$10 \times 4 \times 3$	**51**	56	53	60	63	$160 \times 4 \times 3$	**1018**	1212	1101	1471	1278
$10 \times 4 \times 4$	52	53	51	50	**46**	$160 \times 4 \times 4$	**752**	913	823	1086	940
$10 \times 4 \times 5$	**47**	50	53	59	56	$160 \times 4 \times 5$	**575**	726	618	853	700

(Continued)

Table 3 (continued)

Instance	CICA	ICA	MEASL	IG	HLNS	Instance	CICA	ICA	MEASL	IG	HLNS
20 × 3 × 3	**121**	146	131	158	158	180 × 3 × 3	**1156**	1299	1187	1611	1466
20 × 3 × 4	**95**	102	102	120	117	180 × 3 × 4	**815**	995	864	1141	980
20 × 3 × 5	79	99	81	**72**	94	180 × 3 × 5	**657**	821	707	943	827
20 × 4 × 3	**122**	146	130	160	157	180 × 4 × 3	**1177**	1356	1217	1620	1452
20 × 4 × 4	**100**	110	100	117	121	180 × 4 × 4	**832**	1003	870	1170	1009
20 × 4 × 5	**79**	106	79	95	95	180 × 4 × 5	**675**	856	704	978	848
40 × 3 × 3	**230**	264	241	294	280	200 × 3 × 3	**1235**	1446	1336	1797	1545
40 × 3 × 4	162	197	**159**	237	202	200 × 3 × 4	**906**	1109	975	1328	1135
40 × 3 × 5	**124**	169	126	187	152	200 × 3 × 5	**723**	887	759	1064	872
40 × 4 × 3	**231**	265	238	295	282	200 × 4 × 3	**1305**	1481	1363	1822	1611
40 × 4 × 4	**155**	196	171	235	187	200 × 4 × 4	**925**	1122	992	1346	1150
40 × 4 × 5	135	169	**130**	191	133	200 × 4 × 5	**758**	937	791	1105	915
60 × 3 × 3	**317**	374	394	477	417	220 × 3 × 3	**1402**	1635	1492	1934	1769
60 × 3 × 4	**244**	287	313	370	302	220 × 3 × 4	**995**	1194	1071	1463	1280
60 × 3 × 5	**174**	225	242	274	221	220 × 3 × 5	**787**	917	865	1179	938
60 × 4 × 3	**325**	384	414	475	437	220 × 4 × 3	**1432**	1653	1540	1948	1823
60 × 4 × 4	**237**	291	310	359	315	220 × 4 × 4	**1028**	1218	1087	1482	1273
60 × 4 × 5	**180**	232	253	280	226	220 × 4 × 5	**806**	989	862	1189	1011
80 × 3 × 3	**448**	534	554	657	585	240 × 3 × 3	**1463**	1719	1611	2165	1913
80 × 3 × 4	**338**	427	451	494	438	240 × 3 × 4	**1086**	1273	1175	1587	1416
80 × 3 × 5	**254**	329	346	385	318	240 × 3 × 5	933	951	**903**	1243	926
80 × 4 × 3	**474**	542	590	673	585	240 × 4 × 3	**1508**	1720	1630	2160	1930
80 × 4 × 4	**336**	433	456	501	419	240 × 4 × 4	**1093**	1316	1195	1630	1402
80 × 4 × 5	**243**	336	356	392	319	240 × 4 × 5	**941**	983	946	1268	952
100 × 3 × 3	**543**	626	676	838	689	260 × 3 × 3	**1615**	1843	1729	2381	2094
100 × 3 × 4	**420**	514	457	659	541	260 × 3 × 4	**1205**	1393	1266	1709	1491
100 × 3 × 5	**313**	407	339	488	412	260 × 3 × 5	**949**	1153	1014	1417	1228
100 × 4 × 3	**548**	631	683	840	680	260 × 4 × 3	**1647**	1871	1709	2367	2105
100 × 4 × 4	**423**	530	559	664	545	260 × 4 × 4	**1211**	1440	1319	1767	1563
100 × 4 × 5	**317**	408	330	506	409	260 × 4 × 5	**960**	1162	1034	1417	1233
120 × 3 × 3	**726**	838	754	1033	859	280 × 3 × 3	**1809**	2037	1966	2581	2127
120 × 3 × 4	**517**	647	579	792	681	280 × 3 × 4	**1348**	1552	1405	1969	1511
120 × 3 × 5	**418**	521	436	618	479	280 × 3 × 5	**1006**	1200	1100	1499	1133
120 × 4 × 3	**735**	868	782	1046	876	280 × 4 × 3	**1874**	2105	1967	2675	2155
120 × 4 × 4	**555**	665	591	815	667	280 × 4 × 4	**1367**	1570	1401	1945	1662
120 × 4 × 5	**410**	534	454	625	517	280 × 4 × 5	**1036**	1221	1111	1537	1194
140 × 3 × 3	**820**	923	865	1216	943	300 × 3 × 3	**1988**	2236	2128	2864	2313
140 × 3 × 4	**590**	719	614	917	763	300 × 3 × 4	**1400**	1621	1491	2076	1615
140 × 3 × 5	**462**	585	499	737	612	300 × 3 × 5	**1131**	1312	1210	1652	1250
140 × 4 × 3	**822**	937	886	1226	1063	300 × 4 × 3	**2043**	2306	2210	2843	2454
140 × 4 × 4	**602**	722	631	904	777	300 × 4 × 4	**1442**	1650	1496	2054	1610
140 × 4 × 5	478	595	**472**	721	599	300 × 4 × 5	**1158**	1350	1208	1653	1283

Table 4: Computational results of five algorithms on AVG

Instance	CICA	ICA	MEASL	IG	HLNS	Instance	CICA	ICA	MEASL	IG	HLNS
10 × 3 × 3	**52**	53	59	58	57	160 × 3 × 3	**1036**	1205	1116	1512	1310
10 × 3 × 4	**50**	59	51	58	58	160 × 3 × 4	**767**	917	839	1111	971
10 × 3 × 5	**48**	56	49	57	57	160 × 3 × 5	**592**	738	636	872	747
10 × 4 × 3	**53**	65	56	68	68	160 × 4 × 3	**1052**	1224	1129	1506	1326
10 × 4 × 4	53	57	**52**	56	55	160 × 4 × 4	**780**	932	848	1118	970
10 × 4 × 5	**50**	61	55	69	68	160 × 4 × 5	**585**	743	641	875	733
20 × 3 × 3	**135**	160	138	168	166	180 × 3 × 3	**1181**	1336	1261	1670	1502
20 × 3 × 4	**104**	122	107	129	134	180 × 3 × 4	**837**	1010	898	1210	1035
20 × 3 × 5	91	109	88	**86**	102	180 × 3 × 5	**682**	836	732	991	850
20 × 4 × 3	**133**	152	136	168	163	180 × 4 × 3	**1206**	1364	1275	1675	1506
20 × 4 × 4	109	126	**105**	126	129	180 × 4 × 4	**856**	1021	912	1225	1058
20 × 4 × 5	87	112	**86**	112	103	180 × 4 × 5	**700**	872	743	1008	900
40 × 3 × 3	**248**	279	253	319	292	200 × 3 × 3	**1278**	1463	1374	1836	1616
40 × 3 × 4	173	215	**167**	251	212	200 × 3 × 4	**941**	1129	1013	1365	1183
40 × 3 × 5	**137**	179	137	206	166	200 × 3 × 5	**750**	919	799	1109	926
40 × 4 × 3	251	277	**247**	322	293	200 × 4 × 3	**1318**	1499	1403	1850	1662
40 × 4 × 4	**166**	204	181	251	204	200 × 4 × 4	**963**	1140	1019	1398	1191
40 × 4 × 5	140	177	141	205	**138**	200 × 4 × 5	**780**	954	827	1128	937
60 × 3 × 3	**329**	384	409	502	437	220 × 3 × 3	**1470**	1658	1545	2050	1835
60 × 3 × 4	**258**	297	320	387	320	220 × 3 × 4	**1029**	1227	1112	1502	1320
60 × 3 × 5	**183**	234	253	291	237	220 × 3 × 5	**819**	1003	887	1209	961
60 × 4 × 3	**344**	400	427	503	450	220 × 4 × 3	**1457**	1668	1582	2073	1856
60 × 4 × 4	**256**	303	324	384	329	220 × 4 × 4	**1053**	1241	1130	1519	1313
60 × 4 × 5	**190**	244	262	298	241	220 × 4 × 5	**823**	1024	899	1219	1051
80 × 3 × 3	**472**	546	580	702	602	240 × 3 × 3	**1513**	1727	1639	2197	1961
80 × 3 × 4	**360**	439	466	526	452	240 × 3 × 4	**1117**	1298	1201	1638	1453
80 × 3 × 5	**269**	345	367	409	329	240 × 3 × 5	958	966	**933**	1303	955
80 × 4 × 3	**493**	565	608	687	612	240 × 4 × 3	**1548**	1748	1669	2219	1981
80 × 4 × 4	**354**	442	471	535	458	240 × 4 × 4	**1141**	1344	1228	1681	1464
80 × 4 × 5	**262**	344	373	415	383	240 × 4 × 5	**977**	990	979	1311	972
100 × 3 × 3	**561**	650	701	863	715	260 × 3 × 3	**1655**	1871	1767	2421	2147
100 × 3 × 4	**444**	524	469	679	561	260 × 3 × 4	**1229**	1413	1309	1772	1554
100 × 3 × 5	**329**	414	357	515	424	260 × 3 × 5	**983**	1175	1057	1449	1263
100 × 4 × 3	**576**	667	700	863	713	260 × 4 × 3	**1692**	1897	1785	2432	2172
100 × 4 × 4	**451**	538	574	679	561	260 × 4 × 4	**1260**	1459	1358	1812	1601
100 × 4 × 5	**334**	418	355	516	423	260 × 4 × 5	**992**	1195	1069	1474	1254
120 × 3 × 3	**735**	856	788	1060	924	280 × 3 × 3	**1868**	2072	2003	2681	2177
120 × 3 × 4	**548**	663	611	820	697	280 × 3 × 4	**1369**	1574	1471	1993	1615
120 × 3 × 5	**429**	535	463	648	526	280 × 3 × 5	**1043**	1234	1125	1543	1212
120 × 4 × 3	**752**	885	817	1072	916	280 × 4 × 3	**1919**	2114	2037	2720	2227
120 × 4 × 4	**567**	679	604	829	687	280 × 4 × 4	**1399**	1587	1488	2002	1728
120 × 4 × 5	**429**	544	472	649	535	280 × 4 × 5	**1065**	1251	1152	1579	1227

(Continued)

Table 4 (continued)

Instance	CICA	ICA	MEASL	IG	HLNS	Instance	CICA	ICA	MEASL	IG	HLNS
140 × 3 × 3	**842**	952	895	1268	969	300 × 3 × 3	**2034**	2259	2261	2868	2356
140 × 3 × 4	**619**	737	657	951	790	300 × 3 × 4	**1446**	1662	1452	2101	1660
140 × 3 × 5	**483**	600	518	763	623	300 × 3 × 5	**1164**	1350	1248	1679	1314
140 × 4 × 3	**842**	956	914	1262	1087	300 × 4 × 3	**2076**	2359	2374	2890	2583
140 × 4 × 4	**619**	740	667	951	793	300 × 4 × 4	**1492**	1685	1557	2108	1657
140 × 4 × 5	**492**	607	522	755	622	300 × 4 × 5	**1177**	1374	1253	1688	1342

Table 5: Computational results of five algorithms on MAX

Instance	CICA	ICA	MEASL	IG	HLNS	Instance	CICA	ICA	MEASL	IG	HLNS
10 × 3 × 3	**57**	57	63	63	60	160 × 3 × 3	**1077**	1225	1170	1539	1367
10 × 3 × 4	**58**	67	58	63	66	160 × 3 × 4	**783**	932	862	1149	999
10 × 3 × 5	**56**	63	56	68	67	160 × 3 × 5	**616**	754	651	898	774
10 × 4 × 3	**56**	73	61	73	76	160 × 4 × 3	**1085**	1239	1147	1541	1380
10 × 4 × 4	55	64	**53**	67	63	160 × 4 × 4	**794**	943	867	1153	1003
10 × 4 × 5	**55**	70	60	74	77	160 × 4 × 5	**597**	763	665	900	768
20 × 3 × 3	**144**	163	147	174	174	180 × 3 × 3	**1216**	1355	1327	1717	1534
20 × 3 × 4	**115**	132	115	137	150	180 × 3 × 4	**865**	1022	917	1254	1067
20 × 3 × 5	102	118	97	**95**	116	180 × 3 × 5	**697**	856	757	1028	891
20 × 4 × 3	**142**	164	150	175	170	180 × 4 × 3	**1239**	1379	1328	1702	1566
20 × 4 × 4	115	133	**113**	137	140	180 × 4 × 4	**884**	1039	936	1275	1096
20 × 4 × 5	96	122	**95**	121	114	180 × 4 × 5	**718**	896	768	1041	1086
40 × 3 × 3	**259**	290	266	336	304	200 × 3 × 3	**1322**	1478	1408	1864	1694
40 × 3 × 4	184	226	**181**	258	223	200 × 3 × 4	**994**	1143	1044	1398	1231
40 × 3 × 5	153	196	**149**	223	178	200 × 3 × 5	**779**	941	841	1144	961
40 × 4 × 3	274	286	**262**	338	318	200 × 4 × 3	**1340**	1520	1446	1889	1715
40 × 4 × 4	**173**	212	192	266	234	200 × 4 × 4	**1014**	1149	1047	1448	1243
40 × 4 × 5	152	188	148	218	**147**	200 × 4 × 5	**801**	975	857	1156	976
60 × 3 × 3	**339**	396	422	526	464	220 × 3 × 3	**1540**	1676	1581	2082	1898
60 × 3 × 4	**275**	305	335	404	346	220 × 3 × 4	**1050**	1243	1153	1534	1371
60 × 3 × 5	**197**	246	263	300	256	220 × 3 × 5	**861**	1020	914	1250	986
60 × 4 × 3	**357**	407	442	542	462	220 × 4 × 3	**1487**	1681	1625	2123	1896
60 × 4 × 4	**266**	311	332	400	349	220 × 4 × 4	**1077**	1279	1125	1555	1344
60 × 4 × 5	**200**	259	268	317	252	220 × 4 × 5	**851**	1046	919	1262	1084
80 × 3 × 3	**517**	557	598	733	637	240 × 3 × 3	**1557**	1737	1666	2249	2012
80 × 3 × 4	**371**	449	490	547	478	240 × 3 × 4	**1147**	1321	1230	1695	1487
80 × 3 × 5	**277**	353	385	434	342	240 × 3 × 5	997	987	**966**	1352	989
80 × 4 × 3	**516**	576	623	701	682	240 × 4 × 3	**1581**	1775	1709	2282	2043
80 × 4 × 4	**382**	452	486	551	544	240 × 4 × 4	**1173**	1359	1274	1722	1519
80 × 4 × 5	**293**	349	392	433	429	240 × 4 × 5	1037	1154	1052	1341	**1009**
100 × 3 × 3	**599**	663	715	884	766	260 × 3 × 3	**1694**	1902	1796	2470	2191

(Continued)

Table 5 (continued)

Instance	CICA	ICA	MEASL	IG	HLNS	Instance	CICA	ICA	MEASL	IG	HLNS
100 × 3 × 4	**465**	535	499	700	588	260 × 3 × 4	**1249**	1432	1353	1827	1596
100 × 3 × 5	**350**	429	373	524	444	260 × 3 × 5	**1027**	1196	1124	1488	1302
100 × 4 × 3	**598**	680	719	888	744	260 × 4 × 3	**1745**	1920	1854	2491	2220
100 × 4 × 4	**475**	550	588	710	587	260 × 4 × 4	**1290**	1477	1458	1846	1661
100 × 4 × 5	**350**	431	372	532	439	260 × 4 × 5	**1038**	1212	1109	1518	1278
120 × 3 × 3	**749**	865	820	1089	960	280 × 3 × 3	**1918**	2105	2046	2726	2205
120 × 3 × 4	**562**	670	637	847	714	280 × 3 × 4	**1409**	1596	1508	2024	1785
120 × 3 × 5	**444**	550	484	664	549	280 × 3 × 5	**1077**	1253	1158	1578	1261
120 × 4 × 3	**776**	899	844	1093	965	280 × 4 × 3	**1958**	2124	2100	2768	2365
120 × 4 × 4	**590**	691	617	850	721	280 × 4 × 4	**1437**	1606	1596	2061	1794
120 × 4 × 5	**447**	560	495	688	556	280 × 4 × 5	**1098**	1284	1193	1613	1266
140 × 3 × 3	**884**	964	936	1295	999	300 × 3 × 3	**2112**	2298	2293	2937	2416
140 × 3 × 4	**648**	752	704	981	826	300 × 3 × 4	**1477**	1686	1591	2159	1690
140 × 3 × 5	**495**	619	535	793	640	300 × 3 × 5	**1222**	1372	1314	1737	1361
140 × 4 × 3	**876**	975	940	1301	1114	300 × 4 × 3	**2131**	2305	2435	2938	2686
140 × 4 × 4	**646**	754	701	958	816	300 × 4 × 4	**1530**	1701	1626	2176	1712
140 × 4 × 5	**503**	622	556	793	640	300 × 4 × 5	**1204**	1391	1285	1718	1389

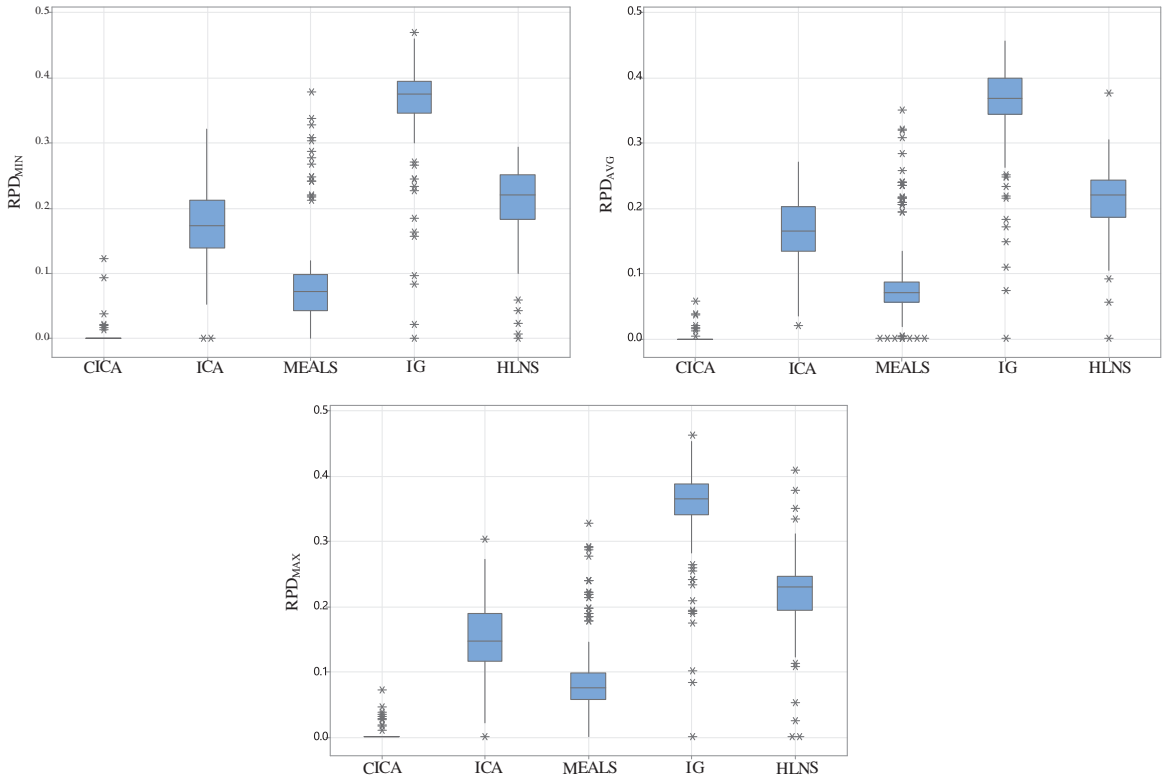

Figure 3: Box plots of five algorithms

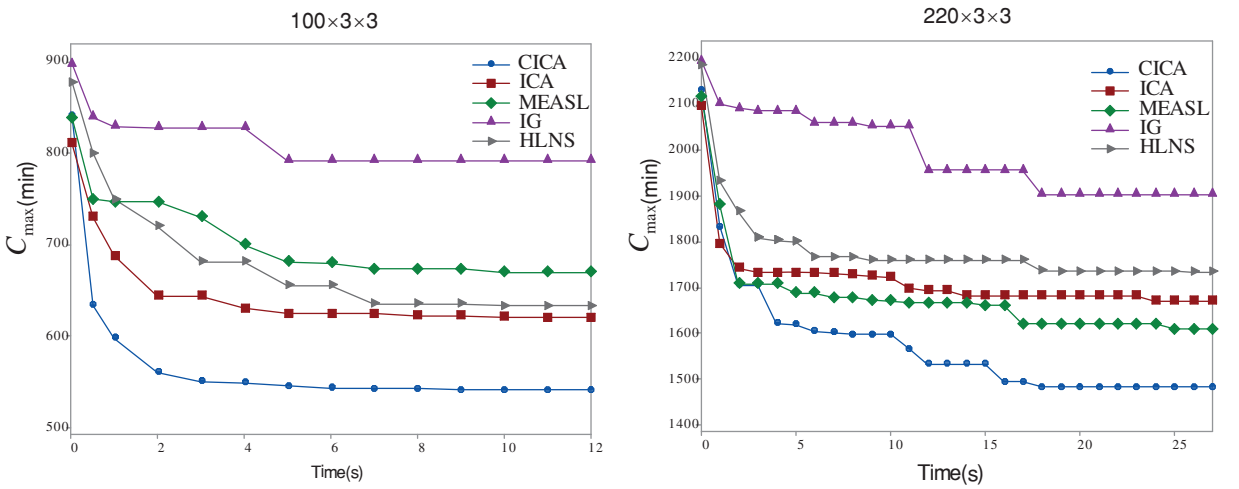

Figure 4: convergence curves of instances 100 × 3 × 3 and 220 × 3 × 3

As shown in Table 3, CICA obtains smaller MIN than ICA on all instances and MIN of CICA is lower than that of ICA by at least 20 on 87 instances. CICA converges better than ICA. This conclusion also can be obtained from Figs. 3 and 4. It can be also found from Tables 4, 5 and Figs. 3, 4 that CICA performs significantly than ICA on AVG and MAX. CICA obtain smaller AVG and MAX than ICA on all instances. It can be concluded that cooperation-based assimilations, combination-based assimilations and new imperial competition process have a positive impact on the performance of CICA.

It can be found from Table 3 that CICA performs better than its comparative algorithms on MIN. CICA produces smaller MIN than three comparative algorithms on 90 of 96 instances; moreover, MIN of CICA is less than that of MEASL by at least 20 on 73 instances, that of IG by at least 20 on 87 instances and that of HLNS by at least 20 on 85 instances. When l and m are the same, with increasing of n, the gap between MIN of CICA and three comparative algorithms is also increasing. CICA has better convergence than MEASL, IG and HLNS. This conclusion can also be drawn from Figs. 3 and 4.

Table 4 describes that CICA obtains smaller AVG than MEASL, IG and HLNS on 88 instances; moreover, AVG of CICA is better than that of its all comparative algorithms by at least 20 on 75 instances. CICA possesses better average performance than its three comparative algorithms. Fig. 3 also illustrates notable average performance differences between CICA and each comparative algorithm.

It also can be seen from Table 5 that MAX of CICA only exceeds that of three comparative algorithms only 10 instances. CICA has smaller MAX than comparative algorithms by at least 20 on 76 instances. Fig. 3 also demonstrates that CICA possesses better stability than its comparative algorithms.

As analyzed above, CICA performs better convergence, average performance and stability than its co-mparative algorithms. The good performances of CICA mainly result from its new strategies. Cooperation-based assimilations of empires 1,4, and combination-based assimilations of empires can effectively improve quality of empires and avoid the waste of computing resources. High diversity can be kept by new imperial competition process. These strategies can make good balance between

exploration and exploitation, thus, CICA is a very promising method for solving parallel BPM scheduling problem with release times.

5 Conclusions and Future Topics

This study examines a scheduling problem involving unrelated parallel batch processing machines (BPM) with release times, a scenario that is derived from the moulding process in a foundry. The Cooperated Imperialist Competitive Algorithm (CICA), which does not use the number of empires as a parameter and maintains four empires throughout the search process, is introduced. The algorithm provides two new methods for assimilation through cooperation and combination between empires, and presents a new form of imperialist competition. Extensive experiments are conducted to compare CICA with existing methods and test its performance. The computational results demonstrate that CICA is highly competitive in solving the considered parallel BPM scheduling problems. BPM is prevalent in many real-life manufacturing processes, such as foundries, and scheduling problems involving BPM are more complex than those without BPM. Future research will focus on scheduling problems with BPM, such as the hybrid flow shop scheduling problem with the BPM stage. We aim to use the knowledge of the problem and new optimization mechanisms in CICA to solve these problems. To obtain high-quality solutions, new optimization mechanisms, such as machine learning, are incorporated into meta-heuristics like the imperialist competitive algorithms. We also plan to apply new meta-heuristics, such as teaching-learning-based optimization, to solve the problem. Energy-efficient scheduling with BPM is another future topic. Furthermore, the application of CICA to other problems is also worth further investigation.

Acknowledgement: The authors would like to thank the editors and reviewers for their valuable work, as well as the supervisor and family for their valuable support during the research process.

Funding Statement: This research was funded by the National Natural Science Foundation of China (Grant Number 61573264).

Author Contributions: Conceptualization, Deming Lei; methodology, Heen Li; software, Heen Li; validation, Deming Lei; formal analysis, Deming Lei; investigation, Deming Lei; resources, Heen Li; data curation, Heen Li; writing—original draft preparation, Deming Lei, Heen Li; writing—review and editing, Deming Lei; visualization, Heen Li; supervision, Deming Lei; project administration, Deming Lei. All authors have read and agreed to the published version of the manuscript.

Availability of Data and Materials: All the study data are included in the article.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] R. Uzsoy, "Scheduling a single batch processing machine with non-identical job sizes," *Int. J. Prod. Res.*, vol. 32, no. 7, pp. 1615–1635, Jul. 1993. doi: 10.1080/00207549408957026.

[2] C. Y. Lee, "Minimizing makespan on a single batch processing machine with dynamic job arrivals Int," *J. Prod. Res.*, vol. 37, no. 1, pp. 219–236, 1999. doi: 10.1080/002075499192020.

[3] 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. Res.*, vol. 87, no. 2, pp. 141–147, Jan. 2004.

[4] S. C. Zhou, L. N. Xing, X. Zheng, N. Du, L. Wang and Q. F. Zhang, "A self-adaptive differential evolution algorithm for scheduling a single batch-processing machine with arbitrary job sizes and release times," *IEEE Trans. Cybern.*, vol. 51, no. 3, pp. 1430–1442, Mar. 2019. doi: 10.1109/TCYB.2019.2939219.

[5] Y. G. Yu, L. D. Liu, and Z. Y. Wu, "A branch-and-price algorithm to perform single-machine scheduling for additive manufacturing," *J Manag. Sci. Eng.*, vol. 8, no. 2, pp. 273–286, Jun. 2023. doi: 10.1016/j.jmse.2022.10.001.

[6] H. B. Zhang, Y. Yang, and F. Wu, "Just-in-time single-batch-processing machine scheduling," *Comput. Oper. Res.*, vol. 140, no. C, pp. 105675, 2022. doi: 10.1016/j.cor.2021.105675.

[7] R. Xu, H. P. Chen, and X. P. Li, "A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system," *Int. J. Prod. Econ.*, vol. 145, no. 1, pp. 371–386, Sep. 2013. doi: 10.1016/j.ijpe.2013.04.053.

[8] L. Jiang, J. Pei, X. B. Liu, P. M. Pardalos, Y. J. Yang and X. F. Qian, "Uniform parallel batch machines scheduling considering transportation using a hybrid DPSO-GA algorithm," *Int. J. Adv. Manuf. Technol.*, vol. 89, no. 5–8, pp. 1887–1900, Aug. 2016. doi: 10.1007/s00170-016-9156-5.

[9] R. Zhang, P. Chang, S. J. Song, and C. Wu, "A multi-objective artificial bee colony algorithm for parallel batch-processing machine scheduling in fabric dyeing processes," *Knowl. Based Syst.*, vol. 116, no. C, pp. 114–129, Jan. 2017. doi: 10.1016/j.knosys.2016.10.026.

[10] Z. H. Jia, J. H. Yan, J. Y. T. Leung, K. Li, and H. P. Chen, "Ant colony optimization algorithm for scheduling jobs with fuzzy processing time on parallel batch machines with different capacities," *Appl. Soft Comput*, vol. 75, no. 1, pp. 548–561, Feb. 2019. doi: 10.1016/j.asoc.2018.11.027.

[11] M. Liu, F. Chu, J. K. He, D. P. Yang, and C. B. Chu, "Coke production scheduling problem: A parallel machine scheduling with batch preprocessings and location-dependent processing times," *Comput. Oper. Res.*, vol. 104, no. 7, pp. 37–48, Apr. 2019. doi: 10.1016/j.cor.2018.12.002.

[12] Z. H. Jia, S. Y. Huo, K. Li, and H. P. Chen, "Integrated scheduling on parallel batch processing machines with non-identical capacities," *Eng. Optim.*, vol. 52, no. 4, pp. 715–730, May 2019. doi: 10.1080/0305215X.2019.1613388.

[13] K. Li, H. Zhang, C. B. Chu, Z. H. Jia, and J. F. Chen, "A bi-objective evolutionary algorithm scheduled on uniform parallel batch processing machines," *Expert. Syst. Appl.*, vol. 204, no. 6, pp. 117487, Oct. 2022. doi: 10.1016/j.eswa.2022.117487.

[14] X. Xin, M. I. Khan, and S. G. Li, "Scheduling equal-length jobs with arbitrary sizes on uniform parallel batch machines," *Open Math.*, vol. 21, no. 1, pp. 228–249, Jan. 2023. doi: 10.1515/math-2022-0562.

[15] J. E. C. Arroyo and J. Y. T. Leung, "An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times," *Comput. Ind. Eng.*, vol. 105, no. C, pp. 84–100, Mar. 2017. doi: 10.1016/j.cie.2016.12.038.

[16] S. J. Lu, X. B. Liu, J. Pei, M. T. Thai, and P. M. Pardalos, "A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity," *Appl. Soft Comput*, vol. 66, no. 1, pp. 168–182, May 2018. doi: 10.1016/j.asoc.2018.02.018.

[17] S. C. Zhou, J. H. Xie, N. Du, and Y. Pang, "A random-keys genetic algorithm for scheduling unrelated parallel batch processing machines with different capacities and arbitrary job sizes," *Appl. Math. Comput.*, vol. 334, pp. 254–268, Oct. 2018.

[18] J. M. Zhang, X. F. Yao, and Y. Li, "Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing," *Int. J. Prod. Res.*, vol. 58, no. 8, pp. 2263–2282, May 2019. doi: 10.1080/00207543.2019.1617447.

[19] M. Kong, X. B. Liu, J. Pei, P. M. Pardalos, and N. Mladenovic, "Parallel-batching scheduling with nonlinear processing times on a single and unrelated parallel machines," *J. Glob. Optim.*, vol. 78, no. 4, pp. 693–715, Sep. 2018. doi: 10.1007/s10898-018-0705-3.

[20] C. Song, "A self-adaptive multiobjective differential evolution algorithm for the inrelated parallel batch processing machine scheduling problem," *Math. Probl. Eng.*, vol. 2022, pp. 5056356, Sep. 2022.

[21] H. Zhang, K. Li, C. B. Chu, and Z. H. Jia, "Parallel batch processing machines scheduling in cloud manufacturing for minimizing total service completion time," *Comput. Oper. Res.*, vol. 146, pp. 1–20, Oct. 2022.
[22] A. Fallahi, B. Shahidi-Zadeh, and S. T. A. Niaki, "Unrelated parallel batch processing machine scheduling for production systems under carbon reduction policies: NSGA-II and MOGWO metaheuristics," *Soft Comput.*, vol. 27, no. 22, pp. 17063–17091, Jul. 2023. doi: 10.1007/s00500-023-08754-0.
[23] X. Xiao, B. Ji, S. S. Yu, and G. H. Wu, "A tabu-based adaptive large neighborhood search for scheduling unrelated parallel batch processing machines with non-identical job sizes and dynamic job arrivals," *Flex. Serv. Manuf. J.*, vol. 62, no. 12, pp. 2083, Mar. 2023. doi: 10.1007/s10696-023-09488-9.
[24] H. Zhang, K. Li, Z. H. Jia, and C. B. Chu, "Minimizing total completion time on non-identical parallel batch machines with arbitrary release times using ant colony optimization," *Eur. J. Oper. Res.*, vol. 309, no. 3, pp. 1024–1046, Sep. 2023. doi: 10.1016/j.ejor.2023.02.015.
[25] W. Jiang, Y. L. Shen, L. X. Liu, X. C. Zhao, and L. Y. Shi, "A new method for a class of parallel batch machine scheduling problem," *Flex Serv. Manuf. J.*, vol. 34, no. 2, pp. 518–550, Apr. 2022. doi: 10.1007/s10696-021-09415-w.
[26] B. Ji, X. Xiao, S. S. Yu, and G. H. Wu, "A hybrid large neighborhood search method for minimizing makespan on unrelated parallel batch processing machines with incompatible job families," *Sustain.*, vol. 15, no. 5, pp. 3934, Feb. 2023. doi: 10.3390/su15053934.
[27] J. W. Ou, L. F. Lu, and X. L. Zhong, "Parallel-batch scheduling with rejection: Structural properties and approximation algorithms," *Eur. J. Oper. Res.*, vol. 310, no. 3, pp. 1017–1032, Nov. 2023. doi: 10.1016/j.ejor.2023.04.019.
[28] E. Santos-Meza, M. O. Santos, and M. N. Arenales, "A lot-sizing problem in an automated foundry," *Eur. J. Oper. Res.*, vol. 139, no. 3, pp. 490–500, Jun. 2002. doi: 10.1016/S0377-2217(01)00196-5.
[29] S. K. Gauri, "Modeling product-mix planning for batches of melt under multiple objectives in a small scale iron foundry," *Producti. Manag.*, vol. 3, pp. 189–196, Feb. 2009. doi: 10.1007/s11740-009-0152-6.
[30] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition," presented at the 2007 IEEE Cong. on Evoluti. Computati., Singapore, Sep. 25–28, 2007.
[31] M. Zare *et al.*, "A global best-guided firefy algorithm for engineering problems," *J. Bionic Eng.*, vol. 20, no. 5, pp. 2359–2388, May 2023. doi: 10.1007/s42235-023-00386-2.
[32] G. Hu, Y. X. Zheng, L. Abualigah, and A. G. Hussien, "DETDO: An adaptive hybrid dandelion optimizer for engineering optimization," *Adv Eng. Inform.*, vol. 57, pp. 102004, Aug. 2023. doi: 10.1016/j.aei.2023.102004.
[33] G. Hu, Y. Guo, G. Wei, and L. Abualigah, "Genghis Khan shark optimizer: A novel nature-inspired algorithm for engineering optimization," *Adv. Eng. Inform.*, vol. 58, pp. 102210, Oct. 2023. doi: 10.1016/j.aei.2023.102210.
[34] M. Ghasemi, M. Zaew, A. Zahedi, P. Trojovsky, L. Abualigah and E. Trojovska, "Optimization based on performance of lungs in body: Lungs performance-based optimization (LPO)," *Comput. Methods Appl. Mech. Eng.*, vol. 419, pp. 116582, Feb. 2024. doi: 10.1016/j.cma.2023.116582.
[35] S. Hosseini and A. A. Khaled, "A survey on the imperialist competitive algorithm metaheuristic: Implementation in engineering domain and directions for future research," *Appl. Soft Comput.*, vol. 24, no. 1, pp. 1078–1094, Nov. 2014. doi: 10.1016/j.asoc.2014.08.024.
[36] C. Z. Guo, M. Li, and D. M. Lei, "Multi-objective flexible job shop scheduling problem with key objectives," presented at the 2019 34rd YAC, Jinzhou, China, Jun. 06–08, 2019.
[37] D. M. Lei, Z. X. Pan, and Q. Y. Zhang, "Researches on multi-objective low carbon parallel machines scheduling," *J. Huazhong Univ. Sci. Technol. Med. Sci.*, vol. 46, no. 8, pp. 104–109, Aug. 2018.
[38] M. Li, B. Su, and D. M. Lei, "A novel imperialist competitive algorithm for fuzzy distributed assembly flow shop scheduling," *J. Intell.*, vol. 40, no. 3, pp. 4545–4561, Mar. 2021. doi: 10.3233/JIFS-201391.

[39] J. F. Luo, J. S. Zhou, and X. Jiang, "A modification of the imperialist competitive algorithm with hybrid methods for constrained optimization problems," *IEEE Access*, vol. 9, pp. 1, Dec. 2021. doi: 10.1109/ACCESS.2021.3133579.

[40] D. M. Lei and J. L. Li, "Distributed energy-efficient assembly scheduling problem with transportation capacity," *Symmetry*, vol. 14, no. 11, pp. 2225, Oct. 2022. doi: 10.3390/sym14112225.

[41] B. Yan, Y. P. Liu, and Y. H. Huang, "Improved discrete imperialist competition algorithm for order scheduling of automated warehouses," *Comput. Ind. Eng.*, vol. 168, pp. 108075, Jun. 2019. doi: 10.1016/j.cie.2022.108075.

[42] T. You, Y. L. Hu, P. J. Li, and Y. G. Tang, "An improved imperialist competitive algorithm for global optimization," *Turk. J. Elec. Eng. Comp. Sci.*, vol. 27, no. 5, pp. 3567–3581, Sep. 2019. doi: 10.3906/elk-1811-59.

[43] H. Yu, J. Q. Li, X. L. Chen, and W. M. Zhang, "A hybrid imperialist competitive algorithm for the outpatient scheduling problem with switching and preparation times," *J. Intell.*, vol. 42, no. 6, pp. 5523–5536, Apr. 2022.

[44] Y. B. Li, Z. P. Yang, L. Wang, H. T. Tang, L. B. Sun and S. S. Guo, "A hybrid imperialist competitive algorithm for energy-efficient flexible job shop scheduling problem with variable-size sublots," *Comput. Ind. Eng.*, vol. 172, no. B, pp. 108641, Oct. 2022.

[45] D. M. Lei, H. Y. Du, and H. T. Tang, "An imperialist competitive algorithm for distributed assembly flowshop scheduling with $Pm \rightarrow 1$ layout and transportation," *J. Intell.*, vol. 45, no. 1, pp. 269–284, Jan. 2023.

[46] M. Yazdani, S. M. Khalili, and F. Jolai, "A parallel machine scheduling problem with two-agent and tool change activities: An efficient hybrid metaheuristic algorithm," *Int. J. Comput. Integr. Manuf.*, vol. 29, no. 10, pp. 1075–1088, Jul. 2016.

[47] D. M. Lei and S. S. He, "An adaptive artificial bee colony for unrelated parallel machine scheduling with additional resource and maintenance," *Expert. Syst. Appl.*, vol. 205, pp. 117577, Nov. 2022.

[48] J. Deng, L. Wang, S. Y. Wang, and X. L. Zheng, "A competitive memetic algorithm for the distributed two-stage assembly flow-shop scheduling problem," *Int. J. Prod. Res.*, vol. 54, no. 12, pp. 3561–3577, Aug. 2015. doi: 10.1080/00207543.2015.1084063.

[49] J. Deng and L. Wang, "A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem," *Swarm Evol. Comput.*, vol. 33, pp. 121–131, Feb. 2017.