Complex & Intelligent Systems (2021) 7:311–326
https://doi.org/10.1007/s40747-020-00193-w

ORIGINAL ARTICLE

Discrete imperialist competitive algorithm for the resource-constrained hybrid flowshop problem with energy consumption

Xin-rui Tao[1] · Jun-qing Li[1,2] · Ti-hao Huang[1] · Peng Duan[1]

Received: 7 May 2020 / Accepted: 27 August 2020 / Published online: 29 September 2020

Abstract
The resource-constrained hybrid flowshop problem (RCHFS) has been investigated thoroughly in recent years. However, the practical case that considers both resource-constrained and energy consumption still has rare research. To address this issue, a discrete imperialist competitive algorithm (DICA) was proposed to minimize the makespan and energy consumption. In the proposed algorithm, first, each solution was represented by a two-dimensional vector, where one vector represented the scheduling sequence and another one showed the machine assignment. Then, a decoding method considering the resource allocation was designed. Finally, we combined DICA and the simulated annealing algorithm (SA) to improve the performance of the proposed approach. Furthermore, we tested the proposed algorithm based on a randomly generated set of real shop scheduling system instances and compared with the existing heuristic algorithms. The results confirmed that the proposed algorithm can solve the RCHFS with high efficiency.

Keywords Hybrid flowshop · Imperialist competitive algorithm · Resource-constrained

Introduction

Hybrid flowshop scheduling problem (HFSP) is a generalization of the conventional flowshop scheduling problem. Compared with other types of scheduling problems, the multi-process and multi-stage characteristics of HFSP are deemed more realistic. In a typical manufacturing industry, various dynamic events may occur in the actual production process, such as limited resources and machine breakdown. Therefore, it is necessary to investigate the RCHFS taking this into consideration. As the major source of global warming, manufacturing activities are required to satisfy the regulations on environment protection and energy consumption. In this paper, the resource-constrained hybrid flowshop problem with energy consumption was studied, and this problem has significant practice relevance.

✉ Jun-qing Li
lijunqing@lcu-cs.com

1 School of Computer, Liaocheng University, Liaocheng 252059, China

2 School of Information Science and Engineering, Shandong Normal University, Jinan 250014, China

With regard to the conventional HFSP, many variants and solutions have been discussed by Rubén Ruiz et al. [1]. To solve HFSP, several researchers have applied the exact algorithms, such as the Lagrangian relaxation algorithm [2–4] and the branch and bound algorithm [5–7]. However, with an increase in the scale of the problem, the resolution of accurate algorithms has become limited, and consequently, metaheuristic or heuristic algorithms have been used more widely to solve this problem. For example, the genetic algorithm (GA) has been applied by Behnamian et al. [8] and Dugardin et al. [9]. Then, the simulated annealing algorithm (SA) has been adopted by Elmi and Topaloglu to solve the problem of multi-robot scheduling [10, 11]. The tabu search algorithm has been considered for the two-stage hybrid flowshop problem by Figielska [12]. The discrete firefly algorithm has been proposed by Marichelvam to solve the bi-objective HFSP [13]. The imperialist competitive algorithm has been considered by Naderi and Yazdani to cope with the HFSP with sublots and setup times [14]. In addition, it should be mentioned that the hybrid artificial bee colony algorithm [15], the hybrid fruit fly optimization algorithm [16], the hybrid squirrel search [17], and others have also achieved good results in the field of HFSP.

RCHFS defined as a variant of HFSP has also been investigated by many researchers in recent years. The Lagrangian decomposition and coordination approach has been developed by Nishi [18]. The branch and bound algorithm was proposed by Sural et al. [19]. The approach combining the Bat algorithm and variable neighborhood search (BA-VNS) has been proposed by Pei et al. [20] for the dual resource-constrained scheduling problem. Li et al., have investigated the improved artificial bee colony algorithm (ABC) and the two-phase-based encoding mechanism [21–23]. The polynomial algorithm has been proposed by Cheng et al. [24] for different cases. The search algorithm based on GA has been proposed by Leu and Hwang, and sensitivity analysis has been performed accordingly [25].

In recent years, green scheduling in the manufacturing industry has become one of the main directions of research and development. Gao et al. [26] aims to provide a comprehensive literature review of production scheduling for intelligent manufacturing systems with the energy-related constraints and objectives. A unified framework for automated design of production scheduling heuristics with genetic programming is developed [27]. A genetic-SA has been adopted by Dai et al. [28] to achieve a significant advance in this field. The improved multi-objective evolutionary algorithm based on decomposition has also been considered [29]. A Pareto-based chemical-reaction optimization algorithm has been proposed by Li et al. [30]. A new concept of teaching–learning-based optimization (TLBO) has been considered to minimize total energy consumption and total tardiness by Lei et al. [31]. The multi-objective iterative greedy algorithm has been proposed by Ding et al. [32] to solve the bi-objective problem. In addition, a hybrid iterated greedy algorithm [33], an efficient multi-objective algorithm [34], and an improved Jaya algorithm [35] has been studied in the multi-objective field. Many experts have also applied energy-aware models [36–38], turn off/on scheme [39], different types of machines [40], and so on.

The main contributions of the present paper are as follows: (1) we considered resource-constrained and energy consumption based on HFSP and proposed corresponding models. To the best of our knowledge, it is the first work to consider the HFSP with resource constrained and energy consumption objective; (2) a novel DICA method was proposed to solve the RCHFS with energy consumption. In addition, we considered the encoding and decoding strategies that were adapt to this problem. (3) DICA and SA were combined to improve the performance of the proposed algorithm.

The rest of this article is organized as follows. Section "Problem description" briefly describes the research problem. In Sect. "Proposed algorithm", we describe the canonical ICA and improved algorithms, encoding, decoding, and the local search process. Then, in Sect. "Numerical experiments", we discuss the experiments conducted on a randomly generated set of instances of the actual shop scheduling system and compare the obtained results with those of several other algorithms. Finally, Sect. "Conclusions" summarizes the overall conclusions.

Problem description

Problem definition

The problem considered in the present study can be formally described as follows. There are n jobs that require s stages of processing. In the entire process, there is at least one stage using two or more parallel processing machines. After completion of processing in the first stage, the work can be continued in the second stage. When a machine is available in the second stage, the work can be processed in the second stage, and if the first stage of processing is completed, the work can remain in an infinite capacity buffer space until the machine becomes idle in the second stage. In phase 2, the jobs can be processed on any processor, and interruption is not allowed after processing begins.

We assume that there are h resources in the shop comprising $R_1, R_2, \ldots R_h$. The number of resources corresponding to each type is defined in advance. The process related to each machine is based on cooperation among different types of resources. Therefore, to start the operation, it is necessary to ensure that the required machines and resources are available simultaneously.

In addition, in the realistic HFSP, there is at least one stage with multiple machines having different processing capabilities in which each machine has two states, namely the processing state and the standby state. The amount of energy consumed by different states is different. The objective is to minimize the makespan and energy consumptions.

The notations used in this study are summarized below.

Indices	
i	Job index, $i = 1, 2, \ldots n$
$k\mathrm{d_i}$	Machine index, $k = 1, 2, \ldots m$
j	Phase index, $j = 1, 2, \ldots g$
q	Resource type index, $q = 1, 2, \ldots h$
$O_{i,j}$	jth operation of job i
Parameters	
n	Total number of jobs
m	Total number of machines
g	Total number of stages
h	Total number of resource types
L	A large number

$p_{i,j}$	Processing time for job i at stage j
M_j	Machine number at stage j
pe_k	Energy consumption index of machine k for processing operations
se_k	Energy consumption index of machine k for standby operations
Decision variables	
$B_{i,j,k}$	Beginning time of the job i at stage j of machine k
$E_{i,j,k}$	Ending time of the job i at stage j of machine k
$y_{i,j,k}$	Binary value set to 1 if job i is assigned at machine k at stage j; otherwise, $y_{i,j,k}$ is set to 0
$x_{i,l}$	Binary value set to 1 if job i is assigned to position l; otherwise, $x_{i,l}$ is set to 0
$R_{i,k,r}$	Binary value set to 1 if job i is conducted on machine k with r resource consumption; otherwise, $R_{i,k,r}$ is set to 0
$C_{\max}$	The completion time
TEC	Total energy consumption

The objective is expressed as:

$$\text{Minimize} \quad w * C_{\max} + (1-w)\text{TEC} \tag{1}$$

$$\text{Subject to} \quad \sum_{l=1}^{n} x_{i,l} = 1, i = 1, 2, \ldots n; \tag{2}$$

$$\sum_{i=1}^{n} x_{i,l} = 1, l = 1, 2, \ldots n; \tag{3}$$

$$\sum_{k=1}^{m_j} y_{i,j,k} = 1, i = 1, 2, \ldots n; j = 1, 2, \ldots g; \tag{4}$$

$$E_{i,j,k} = B_{i,j,k} + p_{i,j}, i = 1, 2, \ldots n; \quad j = 1, 2, \ldots g, \quad k = 1, 2, \ldots M_j; \tag{5}$$

$$E_{i,j,k} \le B_{i,(j+1),k'}, i = 1, 2, \ldots n; \quad j = 1, 2, \ldots g; \quad k = 1, 2, \ldots M_j; k' = 1, 2, \ldots M_{j+1}; \tag{6}$$

$$\sum_{i=1}^{n} x_{i,l} B_{i,j,k} \le \sum_{i=1}^{n} x_{i,(l+1)} B_{i,j,k}, l = 1, 2, \ldots n; \quad j = 1, 2, \ldots g, \quad k = 1, 2, \ldots M_j; \tag{7}$$

$$\sum_{i=1}^{n} x_{i,l1} y_{i,j,k} E_{i,j,k} \le \sum_{i=1}^{n} x_{i,l2} y_{i,j,k} B_{i,j,k} + \left(1 - \sum_{i=1}^{n} x_{i,l2} y_{i,j,k} B_{i,j,k}\right) * L, \quad j = 1, 2, \ldots g; \quad l_1, l_2 = 1, 2, \ldots n \quad l_1 \le l_2; \quad k = 1, 2, \ldots m_j; \tag{8}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{g} \sum_{k=1}^{M_j} R_{i,k,r} \times y_{i,j,k} \le R_h, \quad i = 1, 2, \ldots n; \quad k = 1, 2, \ldots m_j; \quad h = 1, 2, \ldots q; \tag{9}$$

$$\text{TEC} = E_1 + E_2; \tag{10}$$

$$E_1 = \sum_{i=1}^{n} \sum_{j=1}^{g} \sum_{k=1}^{M_j} y_{i,j,k} \times p_{i,j} \times \mathrm{pe}_k; \tag{11}$$

$$E_2 = \sum_{i=1}^{n} \sum_{j=1}^{g} \sum_{k=1}^{M_j} \left(B_{i,j+1,k'} - E_{i,j,k} - \sum_{i=1}^{n} \sum_{j=1}^{g} y_{i,j,k} \times y_{i,j+1,k}\right) \times \mathrm{se}_k; \tag{12}$$

Equation (1) indicates schedule objective. Equation (2) ensures that each priority position can only correspond to one job. Equation (3) ensures that each job has only one priority position. Equation (4) means that only one machine can process each job at any stage. Equation (5) represents the relationship between the completion time and the start time of the process at the same stage. Equation (6) means that the same job must complete the current process before proceeding to the next stage. Equation (7) indicates that the earlier job in the scheduling arrangement of the same stage, the earlier the processing time begins. Equation (8) indicates that the backward job assigned to the same machine at the same stage must wait for the predecessor job to be processed before proceeding. Equation (9) promise the consumed resource is within the limitation. Equations (10)–(12) are the energy consumption, where represents the energy consumption when the machines stay at the processing state; represents the energy consumption when the machines stay at the idle state.

Illustrative example

In this subsection, we present the following example to illustrate the research problem and the solution method. There are five jobs and two stages. There is one machine in the first stage and two parallel machines in the second stage. The total number of resource types is set to two. Table 1 represents the processing time for each job in each stage. Table 2 outlines the resources and energy required by the machines in each stage. Here, Y indicates that the corresponding resources are required, and N denotes that the corresponding machines do not need the resources.

Figure 1a represents the machine Gantt chart for the sample solution in which the sequence of jobs in the first stage is as follows: J_1, J_2, J_3, J_4, J_5. It is evident that J_2 can enter the second stage immediately after the first one is finished

Table 1 Processing time of each job

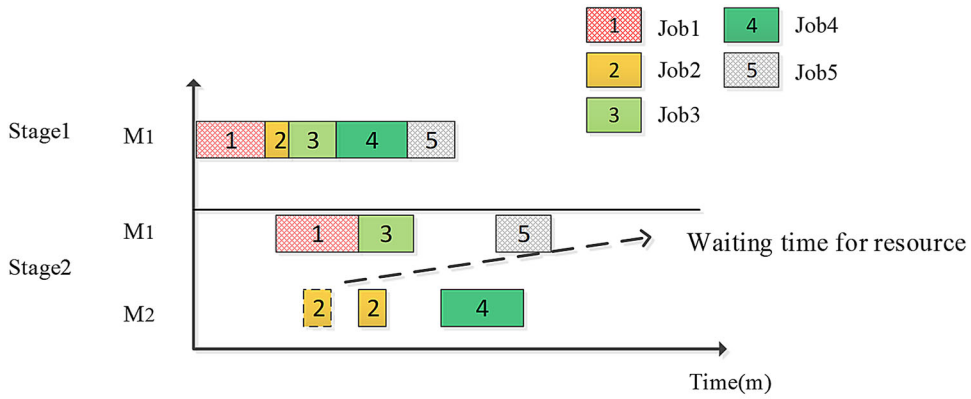
Stage	Machine	Processing time				
		J_1	J_2	J_3	J_4	J_5
1	M_1	3	1	2	3	2
2	M_2	4	1	2	3	3
	M_3	4	1	2	3	3

Table 2 Resources and energy required by the machine

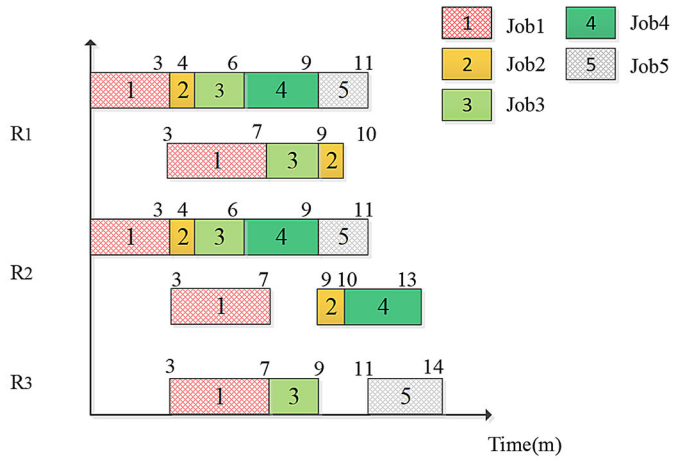
Stage	Machine	Resource type			Energy consumption	
		I	II	III	Processing	Standby
1	M_1	Y	Y	N	2.5	0.1
2	M_2	Y	N	Y	2	0.05
	M_3	Y	Y	N	3	0.15

as the second stage machine is idle; however, J_2 has to be delayed as resources R_1 are insufficient to start processing until time 9. In Fig. 1b, it can be seen that the occupancy of each resource type in each time period does not exceed the total number of resources currently used in any time period; this proves that the solution is feasible. Table 3 illustrates the total consumption of resources during each time period.

For the example in Fig. 1a, the energy consumption unit is mega joules (MJ) and the time unit is seconds. M_1 needs to process the five jobs in the first stage; the processing time is 11; the processing energy consumption is $11 \times 2.5 = 27.5$; and there is no standby energy consumption. In the second stage, the processing time of M_2 is 9; the standby time is 2; the energy consumption of M_2 is $9 \times 2 + 2 \times 0.05 = 18.1$; the processing time of M_3 is 4; and there is no standby time, so that the energy consumption is $4 \times 3 = 12$. Finally, the total energy consumption corresponding to the example is 27.5 + 18.1 + 12 = 57.6.



(a) Machine Gantt chart.



(b) Resource Gantt chart.

Fig. 1 Gantt charts for the solution of the example case

Proposed algorithm

Canonical ICA

Inspired by the imperialist colonial competition mechanism, a new swarm intelligence optimization algorithm called ICA was proposed by Gargari and Lucas in 2007 [41]. After that, ICA was applied widely to various practical optimization problems. For example, it was used by Zahra et al. [42] to find the optimal solution in a shorter time compared with the other existing algorithms. Lucas et al. [43] designed a low-speed, single-sided linear induction motor and demonstrated that ICA performed better than GA in this case. Niknam et al. [44] combined ICA with the K-means algorithm and successfully applied this hybrid method to the task of processing data clusters.

Recent research has demonstrated that ICA is an efficient algorithm for the job shop scheduling problems, such as the flowshop and single machine scheduling ones [45–49]. ICA has advantages in terms of its rapid convergence speed, and many improved algorithms have been proposed on its basis. For example, the chaotic mapping has been applied by Bahrami et al. [50]to determine the changes in directions of colonies in an assimilation operator. Lin et al. [51] have proposed applying perturbation to the ICA algorithm and replaced the relatively weak imperialist countries with artificial ones to enhance the information exchange between empires.

Similar to other evolutionary algorithms, ICA starts with a set of initial solutions called populations. Each individual in a population is denoted as a "chromosome" in GA and a "country" in ICA. Each country is divided into the two parts: imperialist countries and colonies. Countries with better fitness are considered to be empires, and those with worse fitness are denoted as colonies of empires. Competition occurs between the empires, and a powerful empire is likely to seize a colony from a weak one. The entire competition process lasts until only one empire remains; the maximum number of iterations of the algorithm is reached; or the optimal solution is found. The canonical ICA process involves producing the initial empire, assimilating the colonies, proceeding with competition among the empires, and executing the disappearance of empires [52].

Algorithm 1. Canonical ICA

Input: infeasible solution

Output: feasible solution

```
Initialization: Generate an initial population P
Construct an initial empire
While the stopping criterion is not met, do
    For i=1 to Nim, do
        Mutate the imperialist
        Assimilate colonies by the mutated imperialist
    End for
    Update the imperialists
    Do an imperialist competitive strategy
    For i=1 to Nim, do
        Do an imperialist development strategy
    End for
    Apply local search on the imperialist
End while
```

Empire initialization

In ICA, the individuals that constitute the population together are denoted as a country.

For a D-dimensional optimization problem, a country can be represented as follows (13):

$$\text{country} = [P_1, P_2, \ldots P_N]. \tag{13}$$

The initial country generated by the schedule is denoted as N_{pop} from which N_{imp} with the best fitness value are selected. One country is set as an empire, and the remainder comprise N_{col}. The countries are distributed as colonies and are randomly assigned to various empires according to their power. The powers of the empires are standardized according to Eq. (14):

$$C_n = c_n - \max\{c_i\}, \tag{14}$$

where c_n is the fitness value of the nth empire, and C_n is its standardized fitness value. Based on the standardized fitness values, Eq. (15) is defined to calculate the power of each empire:

$$P_n = \frac{C_n}{\sum_{i=1}^{N_{\text{imp}}} c_i}, \tag{15}$$

where C_n is the total normalized cost of each empire; $\sum_{i=1}^{N_{\text{imp}}} c_i$ is the total empire cost; and P_n is the normalized power of each imperialist. The empire with the greater power will have more colonies. Based on P_n, each empire is divided into the number of colonies it should have according to Eq. (16):

$$\text{N.C.}_n = \text{round}\{\ P_n * N_{\text{col}}\}, \tag{16}$$

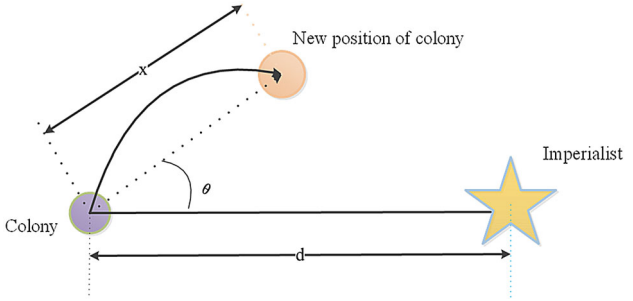


Fig. 2 Moving colonies toward their relevant imperialist in a randomly deviated direction

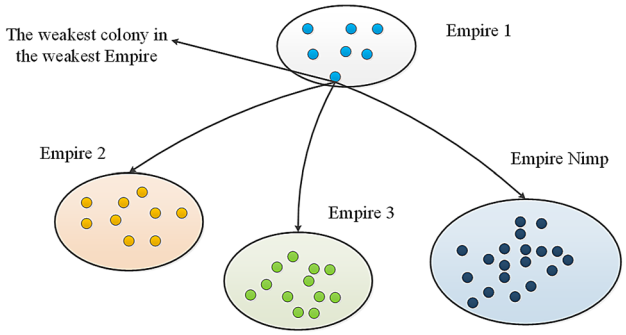


Fig. 3 Imperialist competition

where N.C.$_n$ is the number of colonies initially owned by each empire. According to this number, the colonies are randomly assigned to each empire. The initialization of all empires and colonies is then complete.

Colonial assimilation

In the middle of eighteenth century, world imperialism continued to divide and plunder the colonies. To control the colonies in a better manner, powerful empires promoted customs, culture, and political ideas in the colonies. Accordingly, the colonial assimilation stage of ICA is aimed at imitating this process. The concept of this assimilation process implies forcing all colonies within an empire tend to the local optimal solution within the empire. The process of assimilation can be achieved by moving the colonies into the empire. The corresponding movement process is represented in Fig. 2.

As illustrated in Fig. 2, the colonies move to the empire by a distance x to the new location, and x is defined as

$$X \sim U(0, \beta * d), \tag{17}$$

where β is a real number greater than 1, and d is the distance between the colony and the empire. The condition $\beta > 1$ is required to ensure that the colonies can approach the colonial countries from different directions.

A random offset variable θ is added after the move to perform a wider search around the empire. θ is subject to a uniformly distributed random number:

$$\theta \sim U(-\gamma, \gamma), \tag{18}$$

where γ is a parameter used to adjust deviation from the original direction. The values of β and γ are typically set to 2 and $\pi/4$, respectively, as shown in Fig. 2.

Empire competition

In the imperialism competition stage, we need to select new colonies among weak empires to move to a powerful empire. Therefore, the problem also involves the task of identifying weak empires. In the present study, we propose the two strategies that can be randomly selected to find a weak empire. First, the empires with less colonies can be treated as weak empires. Then, we sort empires by the number of colonies to identify the weak empires and disrupt their colonial order before selecting *selectNum* colonies as the ones belonging to the powerful empires. Second, we can also sort the fitness of the colonies within the empire from small to large, before selecting the least fit as the weak empire and then disrupting the colonial order of the weak empires. We select some of these colonies to move to the powerful empires. The imperialist competition process is represented in Fig. 3.

Empire demise

After a period of time, all except the most powerful empire collapse, and all colonies are included under the control of this unique empire. An empire will perish if it loses all of its colonies, and the algorithm ends when the last empire that is left, or the maximum number of iterations is reached.

Problem encoding

We use a two-phase coding mechanism composed of a two-vector-based (TVB) representation and a machine Gantt-based solution (MGB) representation. We use these two representations, as in the earlier evolutionary stage, such two-vector-based representation can be used to identify a promising search space with a wide search capacity. In the later evolution stage, we employ the solution representation based on a machine Gantt diagram to encode the detailed scheduling for each machine and to search through a suffi-

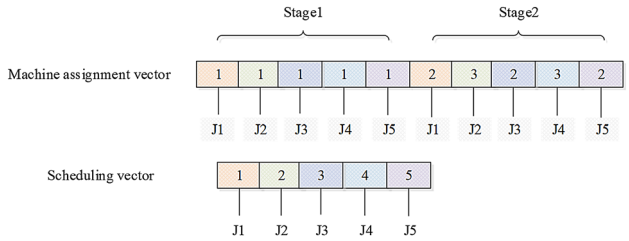


Fig. 4 TVB solution representation example

ciently large search space. Details of the two coding strategies are described as follows.

TVB representation

Machine allocation and operation ordering are considered in the encoding process for the two-vector-based solution representation, as they are commonly used for the standard hybrid flowshop scheduling problems. The resource selection task is dynamically completed during the decoding process. The first vector is called the machine assignment vector, in which each element is used to identify the machine number assigned for the corresponding operation.

The scheduling vector is set as $\{1, 2, 3, 4, 5\}$, which defines the process order for the first stage. Each job is transferred to the next stage immediately after the current phase is completed, and the process starts when the assigned machines and resources are available. Therefore, the process order may change in the later stages because of unavailability of the machines and resources. Figure 4 presents the TVB solution representation example based on Fig. 1a.

MGB representation

The second type of solution representation called the MGB representation is designed on the basis of the current Gantt diagram for the machine, as illustrated in Fig. 1a. The solution requires the vectors for each machine in each phase for which the scheduling sequence for the assigned operations is provided. In the case of the example presented in Fig. 1a, the MGB solution representation can be {1,2,3,4,5}, {{1,3,5}, or {2,4}}. The first vector corresponds to the coding in the first stage, and the second vector is the machine coding in the second stage.

Problem decoding

In the decoding process, we use the TVB and MGB solution representations to determine the start time for each operation and the suitable resources for each machine. The three following conditions need to be satisfied simultaneously to start the operation: The job arrives at the specified machine; the specified machine is available; and the resources required by the specified machine are available (Table 3).

Table 3 Resource occupancy

During	Resource (occupation number)
[0, 3]	{1, 1, 0}
[3, 4]	{2, 2, 1}
[4, 6]	{2, 2, 1}
[6, 7]	{2, 2, 1}
[7, 9]	{2, 1, 1}
[9, 10]	{2, 2, 1}
[10, 11]	{1, 2, 0}
[11, 13]	{0, 1, 1}

In the subsequent stages, each job can be passed to the following stages immediately after its completion time in the current stage. If the assigned machine and the required resources are available, the operation can be scheduled. If the required resources are not available, the operation should be postponed until the availability of the required resources is confirmed.

In operation $O_{i,j}$, when we calculate its starting time $B_{i,j,k}$, we consider the following conditions: (1) the completion time of J_i in the previous stage denoted as $E_{i,j-1,k'}$; (2) the available machine time I_k; and (3) the maximum time available for the resources required by machine k. Therefore, $B_{i,j,k}$ can be computed as follows:

$$B_{i,j,k} = \max\left(B_{i,j-1,k'} + P_{i,j},\, I_k,\, AR_{i,k,r}\right), \tag{19}$$

$$AR_{i,k,r} = \max_{r \in R_h}(A_r), \tag{20}$$

where A_r is the time available for resource $r \in R_h$, and $AR_{i,k,r}$ is the maximum time available for all resources required by machine k.

If $O_{i,j}$ is the first operation of J_i, then the starting time $B_{i,j,k}$ can be calculated as follows:

$$B_{i,j,k} = \max(I_k,\, AR_{i,k,r}). \tag{21}$$

After completion of $O_{i,j}$, the resources $R_{i,k,r}$ are consumed and then, immediately released to be used in other operations. Therefore, the available time for $R_{i,k,r}$ and I_k can be updated as follows:

$$I_k = AR_{i,k,r} = B_{i,j,k} + P_{i,j}. \tag{22}$$

After scheduling all operations, the makespan of the system can be calculated as follows:

$$C\max = \max_{i=1,\ldots,n}\left(B_{i,j,k} + P_{i,j}\right). \tag{23}$$

Local search strategy

Local search for the TVB coding method

In the present study, we consider the representation of solutions based on the two vectors and use a local search strategy operator, which is described as follows.

Step 1 For the machine assignment vectors, we randomly select an operation and assign it to different available machines

Step 2 For the scheduling vectors, we randomly use the swap and insert operators. With regard to the swap operator, we randomly select two job numbers in the scheduling vector and then swap the two jobs, aiming to generate different solutions. In the case of the insertion operator, we randomly select two jobs in the scheduling vector before deleting one job and inserting it instead of another selected job

Local search for the MGB coding method

Considering the representation of the MGB solution, we use a new local search operator, which is described as follows.

Step 1 Calculate the completion time for each machine in the last stage

Step 2 Identify the machines with the longest completion time and store them in a vector denoted as B_M

Step 3 Randomly select a machine B_m from B_M and store all jobs that are being processed on B_M in a vector denoted as B_J

Step 4 Randomly select a job B_j in B_J; then, randomly select a stage j; and find the designated machine M_J that handles B_j in stage j

Step 5 Delete job B_j from M_j; randomly allocate B_j to another machine in stage j; and find the best location for B_j in the newly allocated machine

The improved ICA

In the canonical ICA, the empires are fixed until they are swapped for colonies and transformed into them; this is considered as a disadvantage of ICA. Therefore, we attempt to solve this problem by hybridizing the SA and ICA. By running SA from imperialist positions to improve their status, a new operator called imperial improvement is added to the canonical ICA.

SA is a stochastic optimization algorithm inspired by the annealing process in metallurgy. Starting from a higher initial temperature, SA stochastically finds the global optimal solution of the objective function in the solution space with the decreasing temperature parameters. In other words, the local optimal solution can be found probabilistically, and the global optimal solution can be eventually reached. Similarly to natural annealing, SA solutions for optimization problems are heated (randomly generated). Then, the solution can select one of the nearby locations in the neighborhood with a specific acceptance probability value. The acceptance probability depends on a global parameter T, namely temperature, which decreases as the iterations of the algorithm are being executed.

Acceptance probability is formulated as a Boltzmann–like equation proposed by Kirkpatrick et al. [53]. Generally, if the current solution is represented by x, and the newly created solution is represented by x^{new}, the acceptance probability of x^{new} is defined as $P(x, x^{\text{new}}, T)$. If x^{new} is better than x, it is accepted, meaning that $P(x, x^{\text{new}}, T) = 1$; otherwise, the value of $P(x, x^{\text{new}}, T)$ is randomly obtained in the interval [0, 1]. The acceptance probability formula is defined as $(x, x^{\text{new}}, T) = e^{\frac{-\Delta}{T}}$, where Δ is the difference between the fitness of x^{new} and x; namely $\Delta = f(x^{\text{new}}) - f(x)$. There are several cooling schemes, which can be applied in the case when the exponential cooling is considered. The temperature at iteration k of the algorithm is defined according to Eq. (24):

$$T_k = \alpha T_{k-1} = \alpha^k T_0, \tag{24}$$

where $0 < \alpha < 1$ is the reduction rate of the temperature. Evidently, the smaller α is, the slower the temperature drops. The pseudocode of SA is described as follows:

Algorithm 2. SA

Input: infeasible solution

Output: feasible solution

1. Initialize the temperature
2. Create a random solution x and evaluate $f(x)$
3. Store x in x_{best}
4. **While** stopping criteria are not satisfied
5. Create a neighbor solution x^{new} and evaluate $f(x^{new})$
6. Accept x^{new} with probability of $P(x, x^{new}, T)$
7. **If** x is better than x_{best}, store x in x_{best}
8. Reduce temperature
9. **End while**

In general, SA implements neighborhood structures to create new points in the search space. Different types of problems can have different solutions. We define a random process for RCHFS to create neighborhoods. The following steps are required to execute this process:

Algorithm 3. The neighborhood creation scheme

Input: imperialists

Output: improved imperialists

1. With a probability of 1/2, go to step 2; otherwise go to step 3.
2. Select a random element in vector v_1 (the machine assignment vector) and replace it by a random number in the interval [0, 1]. Then, go to 6.
3. With a probability of 1/2, go to step 4; otherwise go to step 5.
4. Select two distinct elements in v_2 (the scheduling vector) and swap their locations. Then, go to step 6.
5. Select two distinct elements in v_2 and reverse the sequence between them (including themselves).
6. **End**

Numerical experiments

Experimental instances

To solve the RCHFS problem and verify the validity of the DICA algorithm, we randomly generated 20 large-scale test cases for RCHFS problems based on the actual production data. The number of identical factors for each test problem was randomly generated. The test set of instances was classified into the four categories according to the number of jobs (50, 100, 150, and 200). To test the efficiency of the proposed DICA algorithm in the environments with different complexity, each category was divided into the five sub-problems with different numbers of stages (2–10). The instances can be found in the website: https://www.researchgate.net/publication/334735417_RCHFS-suanli. For example, the code instance 1 indicates that the problem is characterized with 50 jobs, two stages, and nine parallel machines in the first stage. Through the experiments, we set the empire competition selection rate to 0.2. In addition, Zandieh et al. [54] considered that the robustness of the algorithm was better when the parameters were set as in Table 4.

Experimental parameters

We conducted a preliminary experiment to appropriately set the system parameters. The key parameters considered in the experiment were as follows: (1) the population size (P_m) comprising the total number of individuals in the experiment; (2) the probability of crossover (P_c) that determines the likelihood of each individual's crossover operation; and (3) the probability of mutation (P_m), which defines the probability of mutation operation.

Table 4 SA parameter values

Parameter	Value
The beta factor of SA	1.9
T_0 factor of SA	500
Maximum iteration of SA	20

Table 5 Key parameter levels

Parameter	Values			
	1	2	3	4
P_s	50	100	150	200
P_c	0.10	0.30	0.50	0.60
P_m	0.05	0.10	0.20	0.30

Table 6 Response values

P_s	P_c	P_m	Average fitness
1	1	1	538.87
1	2	2	539.19
1	3	3	539.21
1	4	4	539.03
2	1	2	539.45
2	2	1	539.01
2	3	4	539.17
2	4	3	538.97
3	1	3	539.43
3	2	4	539.48
3	3	1	539.09
3	4	2	539.28
4	1	4	539.51
4	2	3	539.84
4	3	2	539.22
4	4	1	539.31

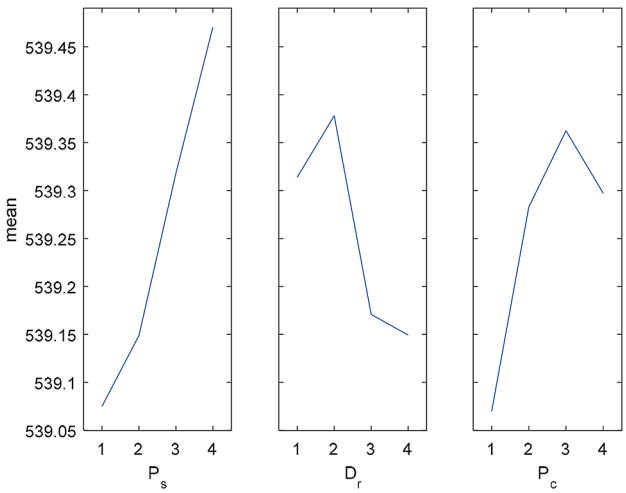


Fig. 5 Factor level trends for the three key parameters

The Taguchi method of design of experiment (DOE) [55] is utilized to test effects of these parameters on the

Table 7 Comparison with DICA and CPLEX solver

Ins	Scale	Best	Fitness		RPI	
			DICA	CPLEX	DICA	CPLEX
Inst 1	3-3-2	141.36	**141.36**	143.69	**0.00**	1.65
Inst 2	4-4-2	252.26	**252.26**	**252.26**	**0.00**	**0.00**
Inst 3	5-4-2	173.96	**173.96**	175.58	**0.00**	0.93
Inst 4	5-5-2	152.20	**152.20**	154.54	**0.00**	1.54
Inst 5	5-5-3	232.81	**232.81**	241.15	**0.00**	3.58
Inst 6	6-4-2	127.24	**127.24**	132.11	**0.00**	3.83
Inst 7	6-5-2	127.19	**127.19**	135.84	**0.00**	6.80
Inst 8	6-5-3	198.36	**198.36**	203.58	**0.00**	2.63
Inst 9	7-4-2	146.41	**146.41**	150.75	**0.00**	2.96
Inst 10	7-5-2	139.62	**139.62**	146.64	**0.00**	5.03
Inst 11	7-5-3	186.80	**186.80**	–	**0.00**	–
Inst 12	8-4-2	159.06	**159.06**	**159.06**	**0.00**	**0.00**
Inst 13	8-5-2	160.15	**160.15**	165.16	**0.00**	3.13
Inst 14	8-5-3	190.08	**190.08**	–	**0.00**	–
Mean		170.53	**170.53**	171.67	**0.00**	2.29

Table 8 Comparison of RPI values for the decoding strategy

Ins	Best	Decoding		Dev	
		1	2	1	2
Inst 1	1068.00	1107.00	**1068.00**	3.65	**0.00**
Inst 2	3010.00	3479.00	**3010.00**	15.58	**0.00**
Inst 3	3038.00	3460.00	**3038.00**	13.89	**0.00**
Inst 4	8504.00	9046.00	**8504.00**	6.37	**0.00**
Inst 5	5497.00	6466.00	**5497.00**	17.62	**0.00**
Inst 6	1965.00	1965.00	**1967.00**	**0.00**	0.10
Inst 7	11799.00	12046.00	**11799.00**	2.09	**0.00**
Inst 8	8582.00	8582.00	**6374.00**	34.64	**0.00**
Inst 9	5759.00	6766.00	**5759.00**	17.48	**0.00**
Inst 10	6965.00	8650.00	**6965.00**	24.19	**0.00**
Inst 11	4191.00	4826.00	**4191.00**	15.15	**0.00**
Inst 12	13468.00	13468.00	**12433.00**	8.32	**0.00**
Inst 13	18668.00	20957.00	**18668.00**	12.26	**0.00**
Inst 14	18555.00	20831.00	**18555.00**	12.26	**0.00**
Inst 15	19290.00	23309.00	**19290.00**	20.83	**0.00**
Inst 16	12370.00	12478.00	**12370.00**	0.87	**0.00**
Inst 17	9110.00	9868.00	**9110.00**	8.32	**0.00**
Inst 18	6780.00	8957.00	**6780.00**	32.10	**0.00**
Inst 19	17748.00	19866.00	**17748.00**	11.93	**0.00**
Inst 20	21315.00	25344.00	**21315.00**	18.90	**0.00**
Mean		11073.50	**9722.05**	13.82	**0.00**

performance of the proposed algorithm, the levels of the three critical parameters determined according to the preliminary experiment are shown in Table 5. An orthogonal array L16(4^3) was used to tune the parameters. The proposed

Table 9 Comparison of RPI values for the SA strategy

Ins	Best	SA		Dev	
		1	2	1	2
Inst 1	1145.23	1347.04	**1145.23**	17.62	**0.00**
Inst 2	3386.53	3595.95	**3386.53**	6.18	**0.00**
Inst 3	3487.00	3775.60	**3487.00**	8.28	**0.00**
Inst 4	9669.04	9996.61	**9669.04**	3.39	**0.00**
Inst 5	6593.97	6978.80	**6593.97**	5.84	**0.00**
Inst 6	2031.07	2183.17	**2031.07**	7.49	**0.00**
Inst 7	10961.31	11656.45	**10961.31**	6.34	**0.00**
Inst 8	7919.68	7955.12	**7919.68**	0.45	**0.00**
Inst 9	6838.27	6998.24	**6838.27**	2.34	**0.00**
Inst 10	8900.45	9602.83	**8900.45**	7.89	**0.00**
Inst 11	4468.51	**4468.51**	**4508.27**	**0.00**	0.89
Inst 12	12073.68	12484.75	**12073.68**	3.40	**0.00**
Inst 13	21734.24	22782.69	**21734.24**	4.82	**0.00**
Inst 14	18940.59	19976.35	**18940.59**	5.47	**0.00**
Inst 15	22674.61	24681.01	**22674.61**	8.85	**0.00**
Inst 16	10771.71	11190.27	**10771.71**	3.89	**0.00**
Inst 17	9456.48	9944.88	**9456.48**	5.16	**0.00**
Inst 18	9108.13	**9108.13**	9115.39	**0.00**	0.08
Inst 19	18464.75	19631.39	**18464.75**	6.32	**0.00**
Inst 20	24320.45	25275.92	**24320.45**	3.93	**0.00**
Mean		11181.69	**10649.64**	5.38	**0.05**

algorithm was run 30 times for each independent combination of parameters. We then calculated the average RPI value obtained by the algorithm used for comparison as the response variable and the results are presented in Table 6. As shown in Fig. 5, the P_s is set to 50, the P_c is set to 0.6 and the P_m is set to 0.05, so that the algorithm has the best performance. In addition, we set the empire competition selection rate to 0.1 [41, 42].

Performance compared with the exact solver CPLEX

To investigate the validation of the proposed optimization model, we employ an exact solver IBM ILOG CPLEX 12.7 to calculate the proposed model for the fourteen small-scale instances. The small-scale instances are generated at random. In the exact solver, the maximum number of threads is 3, and the CPU time limits is set to 3 h for each run.

Table 7 shows the comparison results between the DICA algorithm and CPLEX solver. The first column gives the instance number, and the second column shows the problem scale (where 4–4–2 means the instance includes 4 jobs, 4 machines and 2 stages). We run the DICA algorithm 10 times for each instance and take the minimum value. The last two columns show the RPI values for the two methods. We can see that: (1) the proposed DICA algorithm obtains a higher

Table 10 Comparison of the best solutions obtained by the considered algorithms

Ins	Best	Fitness				DEV			
		HICA	DABC	LABC	DICA	HICA	DABC	LABC	DICA
Inst 1	0.496	0.500	0.501	**0.496**	0.504	0.80	1.21	**0.00**	1.81
Inst 2	0.586	0.625	0.607	0.601	**0.586**	6.65	3.63	2.58	**0.00**
Inst 3	0.538	0.564	0.546	0.547	**0.538**	4.68	1.45	1.54	**0.00**
Inst 4	0.617	0.644	0.619	0.632	**0.617**	4.48	0.33	2.42	**0.00**
Inst 5	0.605	0.638	0.631	**0.605**	0.614	5.46	4.30	**0.00**	1.49
Inst 6	0.496	0.545	0.517	0.520	**0.496**	9.68	4.13	4.67	**0.00**
Inst 7	0.640	0.660	0.653	**0.640**	0.642	2.96	2.03	**0.00**	0.27
Inst 8	0.634	0.650	**0.634**	0.641	0.637	2.52	**0.00**	1.10	0.41
Inst 9	0.609	0.632	0.620	0.611	**0.609**	3.72	1.78	0.34	**0.00**
Inst 10	0.612	0.647	0.620	0.628	**0.612**	5.74	1.26	2.52	**0.00**
Inst 11	0.596	0.598	0.604	**0.596**	0.606	0.16	1.34	**0.00**	1.68
Inst 12	0.653	0.663	0.657	**0.653**	0.654	1.37	0.45	**0.00**	0.17
Inst 13	0.624	0.638	0.629	0.630	**0.624**	2.35	0.86	1.01	**0.00**
Inst 14	0.661	0.667	0.663	0.664	**0.661**	0.91	0.27	0.56	**0.00**
Inst 15	0.632	0.650	**0.632**	0.643	0.633	2.85	**0.00**	1.90	0.17
Inst 16	0.663	0.673	0.665	**0.663**	0.666	1.35	0.15	**0.00**	0.42
Inst 17	0.626	0.646	0.632	0.637	**0.626**	3.28	0.93	1.69	**0.00**
Inst 18	0.613	**0.613**	0.619	0.618	0.613	**0.00**	1.35	0.98	0.10
Inst 19	0.649	0.656	0.652	0.652	**0.649**	1.08	0.43	0.41	**0.00**
Inst 20	0.651	0.672	0.654	0.656	**0.651**	3.29	0.45	0.80	**0.00**
Mean		0.629	0.618	0.617	0.612	3.17	1.32	1.13	0.33

Table 11 Comparison of the average solutions obtained by the considered algorithms

Ins	Best	Fitness				Dev			
		HICA	DABC	LABC	DICA	HICA	DABC	LABC	DICA
Inst 1	0.533	0.628	0.543	0.543	**0.533**	17.93	1.88	1.92	**0.00**
Inst 2	0.616	0.658	0.624	**0.616**	0.617	6.81	1.33	**0.00**	0.16
Inst 3	0.566	0.609	0.579	0.567	**0.566**	7.59	2.21	0.16	**0.00**
Inst 4	0.643	0.675	0.650	0.649	**0.643**	4.89	1.15	0.87	**0.00**
Inst 5	0.636	0.676	0.649	**0.636**	0.640	6.22	1.94	**0.00**	0.50
Inst 6	0.542	0.606	0.569	0.547	**0.542**	11.81	4.89	0.84	**0.00**
Inst 7	0.653	0.676	0.656	**0.653**	0.654	3.40	0.34	**0.00**	0.07
Inst 8	0.648	0.674	0.658	**0.648**	0.649	3.93	1.40	**0.00**	0.16
Inst 9	0.631	0.678	**0.631**	0.632	0.636	7.48	**0.00**	0.27	0.79
Inst 10	0.634	0.682	**0.634**	0.643	0.642	7.49	**0.00**	1.38	1.13
Inst 11	0.622	0.666	0.640	0.635	**0.622**	7.07	2.88	2.07	**0.00**
Inst 12	0.664	0.682	0.671	0.671	**0.664**	2.78	1.05	1.04	**0.00**
Inst 13	0.646	0.668	0.660	0.668	**0.646**	3.46	2.11	3.45	**0.00**
Inst 14	0.671	0.694	0.679	0.687	**0.671**	3.37	1.12	2.40	**0.00**
Inst 15	0.642	0.669	**0.642**	0.653	0.644	4.19	0.00	1.63	0.33
Inst 16	0.673	0.688	0.681	0.679	**0.673**	2.29	1.14	0.97	**0.00**
Inst 17	0.644	0.681	0.655	0.648	**0.644**	5.70	1.73	0.58	**0.00**
Inst 18	0.630	0.663	0.641	0.636	**0.630**	5.21	1.66	0.99	**0.00**
Inst 19	0.663	0.690	**0.663**	0.666	0.665	4.17	**0.00**	0.54	0.35
Inst 20	0.664	0.688	**0.664**	0.671	0.665	3.57	**0.00**	1.01	0.11
Mean		0.668	0.639	0.637	0.632	5.97	1.34	1.01	0.18

Table 12 Comparison of the worst solutions obtained by the considered algorithms

Ins	Best	Fitness				Dev			
		HICA	DABC	LABC	DICA	HICA	DABC	LABC	DICA
Inst 1	0.575	0.736	0.612	0.594	**0.575**	28.06	6.39	3.26	**0.00**
Inst 2	0.638	0.712	0.656	0.647	**0.638**	11.59	2.79	1.42	**0.00**
Inst 3	0.589	0.655	0.611	0.603	**0.589**	11.09	3.70	2.34	**0.00**
Inst 4	0.656	0.703	0.671	**0.656**	0.657	7.16	2.28	**0.00**	0.11
Inst 5	0.657	0.728	0.679	0.669	**0.657**	10.79	3.38	1.86	**0.00**
Inst 6	0.568	0.735	0.575	0.582	**0.568**	29.51	1.32	2.60	**0.00**
Inst 7	0.666	0.697	0.710	0.677	**0.666**	4.66	6.61	1.56	**0.00**
Inst 8	0.659	0.698	0.713	0.671	**0.659**	6.01	8.25	1.91	**0.00**
Inst 9	0.646	0.728	0.651	0.651	**0.646**	12.67	0.70	0.64	**0.00**
Inst10	0.655	0.736	0.660	**0.655**	0.666	13.05	1.52	**0.00**	1.70
Inst 11	0.639	0.700	**0.639**	0.643	0.640	9.54	**0.00**	0.62	0.22
Inst 12	0.671	0.698	0.691	0.686	**0.671**	4.13	2.99	2.25	**0.00**
Inst 13	0.659	0.692	0.673	**0.659**	0.660	4.93	2.09	**0.00**	0.09
Inst 14	0.682	0.711	0.687	0.686	**0.682**	4.27	0.81	0.55	**0.00**
Inst 15	0.652	0.694	0.661	0.658	**0.652**	6.39	1.37	0.95	**0.00**
Inst 16	0.675	0.703	0.678	**0.675**	0.679	3.66	0.04	**0.00**	0.47
Inst 17	0.659	0.727	0.674	0.669	**0.659**	10.33	2.22	1.48	**0.00**
Inst 18	0.645	0.704	0.660	0.651	**0.645**	9.20	2.42	1.00	**0.00**
Inst 19	0.675	0.725	0.684	**0.675**	0.677	7.40	1.33	**0.00**	0.35
Inst 20	0.674	0.708	**0.674**	0.678	0.677	5.04	**0.00**	0.59	0.57
Mean		0.710	0.663	0.654	0.648	9.97	2.51	1.15	0.18

solution quality; (2) for larger-scale calculation examples, the CPLEX solver shows worse performance. ('-' indicates that CPLEX solver cannot find feasible solution in 3 h. Bold font indicates the optimal value in the comparison algorithms.) The performance measure is relative percentage increase (RPI), which is calculated as follows:

$$\mathrm{RPI}(C) = \frac{C_c - C_b}{C_b} \times 100, \tag{25}$$

where C_b is the best solution found by all the compared algorithms, while C_c is the best solution generated by a given algorithm.

Efficiency of the solution representation

To investigate the efficiency of the solution representation defined above, we considered a permutation-based representation to compare with the proposed algorithm. The parameter sets for the two considered algorithms were the same as those defined above. The main difference between the two algorithms considered in the comparison was the embedding of a two-level coding mechanism in DICA. Table 8 presents the comparison of the detailed results obtained by the two algorithms after 30 independent runs. The instance name is presented in the first column of Table 8; the scale, in the second column. The RPI values obtained for DICA and DICAnd are presented in the following two columns. The results presented in Table 8 can be summarized as follows: (1) All optimal values were obtained by DICA, and DICAnd provided none of them; (2) the last row in the table also demonstrates that DICA performed better than DICAnd; and (3) we achieved the better performance using the proposed two-phase encoding method. Table 9 shows the comparison of the average running results of our algorithm and the canonical ICA, we can see that DICA has a significant advantage.

The advantages of the proposed two-phase coding method are as follows: (1) The two vector-based coding methods are adopted in the early stage of evolution; therefore, they can allow rapidly locating the optimal search space and improving the search efficiency; (2) in the second part of the evolution process, the proposed algorithm performs a fine-grained search at the optimal search position using the Gantt chart-based representation and, thus, allows improving the quality of the solution; and (3) the proposed algorithm uses a two-phase representation to balance the exploration and development capacities. To determine whether the results were significantly different, we also performed a multivariate analysis of variance often used to compare the performance of different heuristics.

Comparisons with other efficient algorithms

Finally, to assess the efficiency of the proposed DICA algorithm, we compared it with the discrete DABC algorithm [56], LABC [57] and hybrid ICA (HICA) [58]. We integrate energy consumption and makespan into one objective function by weighted sum approach, which is used by many researchers [59, 60]. The total energy consumption and makespan are normalized and can be calculated using Eqs. (26) and (27), respectively. The final objective function of weighted sum is presented as Eq. (28).

$$F_{\text{value}} = F/F_{\max}, \tag{26}$$

$$E_{\text{value}} = E/E_{\max}, \tag{27}$$

$$\min(f) = w * F_{\text{value}} + (1-w) * E_{\text{value}}; \tag{28}$$

where $F_{\max}$ and $E_{\max}$ are the maximum makespan and maximum energy consumption, respectively. w is the weight to reflect the significance of F which is defined in the range $0 \leq w \leq 1$, and we set w to 0.8.

Each algorithm was run 30 times on the same computer considering 20 test cases. The comparative experimental results are presented in Tables 10, 11, and 12. The instance name is presented in the first column. The second column presents the best value for each instance. The following three columns provide the best values obtained by each algorithm considered in the comparison. To intuitively compare the quality of the solutions obtained by the three algorithms, we calculated the percentage deviation for the solutions. The corresponding results are presented in the last three columns.

The results outlined in Table 10, 11, and 12 can be summarized as follows: (1) The DICA algorithm obtained 11, 11, and 13 optimal solutions for a given example, which were far greater compared with the results of the other considered algorithms; and (2) as presented in the last row, the average validity period and average percentage deviation were lower in the case of DICA compared with the other algorithms. In conclusion, the experimental results indicated that the proposed DICA algorithm is a more efficient method for solving the RCHFS problem compared with other recently proposed algorithms. In addition, we randomly selected four instances "Instance1, Instance5, Instance11, Instance17". Figures 6a–d record the convergence curves of these examples, showing the effectiveness of the proposed DICA algorithm. We can clearly see that among the convergence curves of each example, the DICA algorithm is the most efficient. Figure 7 shows a resource Gantt chart for Example 6, representing 50 jobs and 10 machines in two stages, requiring three resources corresponding to R_1, R_2, R_3.

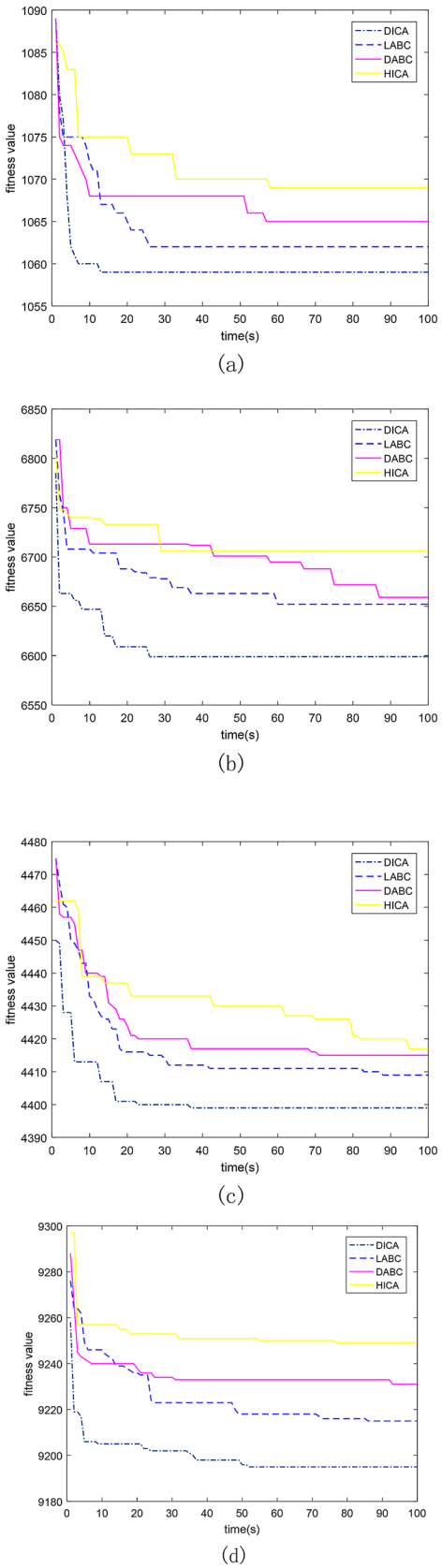


Fig. 6 The convergence curve of instances

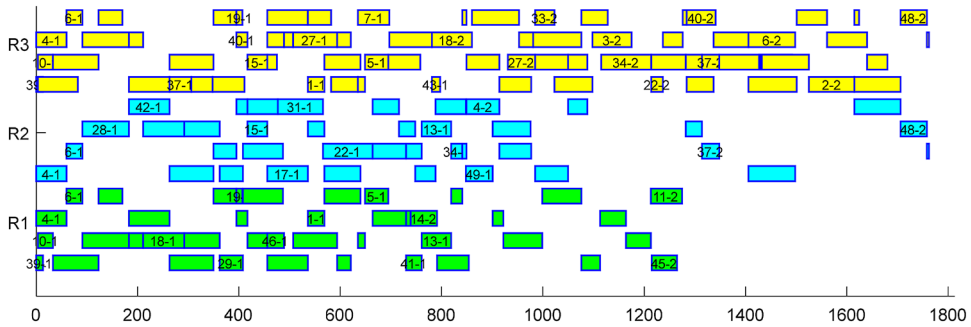


Fig. 7 The resource Gantt chart

Comparative analysis

The conducted experimental comparison has led to the following conclusions with regard to the proposed algorithm: (1) The two-phase coding mechanism can be used to improve the search capacity of the algorithm in different evolution stages, thereby improving the performance of the algorithm; (2) the decoding process considers resource constraints to provide the feasible and efficient solution; and (3) the local search process can efficiently utilize the two-phase encoding mechanism to improve the search capacity of the algorithm.

Conclusions

In this paper, we proposed the DICA method for a variety of RCHFS problems associated with energy. The main contributions of the present study are as follows: (1) we considered resource-constrained and energy consumption based on HFSP and proposed corresponding models. To the best of our knowledge, it is the first work to consider the HFSP with resource constrained and energy consumption objective; (2) a novel DICA method was proposed to solve the RCHFS with energy consumption. In addition, we considered the encoding and decoding strategies that were adapt to this problem. (3) DICA and SA were combined to improve the performance of the proposed algorithm. We verify the effectiveness of the proposed algorithm by comparing with CPLEX and other advanced algorithms. In future research, we aim to apply the proposed DICA method to solve other types of scheduling problems, such as distributed green hybrid flowshop scheduling problems. The proposed algorithm was tested on the problems of different scales having various structures. Several efficient existing algorithms were compared to the proposed DICA. The experimental results confirmed the efficiency of the proposed algorithm. Future research will be mainly focused on the following issues: (1) applying the proposed algorithm to solve other types of scheduling problems, such as robot-related problems in a hybrid flowshop and distributed optimization problems with more realistic constraints; (2) considering two or more goals and developing a multi-objective optimization algorithm to solve more realistic green intelligent optimization problems; and (3) solving large-scale scheduling problems, such as steelmaking scheduling, chemical production or parking lot scheduling.

Acknowledgements This research is partially supported by National Science Foundation of China under Grant nos. 61773192, 61803192, 61773246, Shandong Province Higher Educational Science and Technology Program (J17KZ005).

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flow shop scheduling problem. Eur J Oper Res 205(1):1–18
2. Tang LX, Xuan H (2006) Lagrangian relaxation algorithms for real-time hybrid flowshop scheduling with finite intermediate buffers. J Oper Res Soc 57(3):316–324
3. Tang L, Xuan H, Liu J (2006) A new Lagrangian relaxation algorithm for hybrid flowshop scheduling to minimize total weighted completion time. Comput Oper Res 33(11):3344–3359
4. Portmann MC, Vignier A, Dardilhac D, Dezalay D (1998) Branch and bound crossed with GA to solve hybrid flowshops. Eur J Oper Res 107(2):389–400
5. Fattahi P, Hosseini MH, Jolai F, Tavakkoli-Moghaddam R (2014) A branch and bound algorithm for hybrid flow shop scheduling prob-

lem with setup time and assembly operations. Appl Math Model 38(1):119–134
6. Wang S, Liu M, Chu C (2015) A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling. Int J Prod Res 53(4):1143–1167
7. Zhang W, Yin C, Liu J, Linn RJ (2005) Multi-job lot streaming to minimize the mean completion time in m-1 hybrid flowshops. Int J Prod Econ 96(2):189–200
8. Behnamian J, Ghomi SF, Zandieh M (2009) A multi-phase covering Pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic. Expert Syst Appl 36(8):11057–11069
9. Dugardin F, Yalaoui F, Amodeo L (2010) New multi-objective method to solve reentrant hybrid flow shop scheduling problem. Eur J Oper Res 203(1):22–31
10. Elmi A, Topaloglu S (2014) Scheduling multiple parts in hybrid flow shop robotic cells served by a single robot. Int J Comput Integr Manuf 27(12):1144–1159
11. Elmi A, Topaloglu S (2013) A scheduling problem in blocking hybrid flow shop robotic cells with multiple robots. Comput Oper Res 40(10):2543–2555
12. Figielska E (2014) A heuristic for scheduling in a two-stage hybrid flowshop with renewable resources shared among the stages. Eur J Oper Res 236(2):433–444
13. Marichelvam MK, Prabaharan T, Yang XS (2013) A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. IEEE Trans Evol Comput 18(2):301–305
14. Naderi B, Yazdani M (2014) A model and imperialist competitive algorithm for hybrid flow shops with sublots and setup times. J Manuf Syst 33(4):647–653
15. Song MX, Li JQ, Han YQ, Han YY, Liu LL, Sun Q (2020) Metaheuristics for solving the vehicle routing problem with the time windows and energy consumption in cold chain logistics. Appl Soft Comput. https://doi.org/10.1016/j.asoc.2020.106561
16. Li JQ, Pan QK, Mao K (2015) A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems. IEEE Trans Autom Sci Eng 13(2):932–949
17. Khare A, Agrawal S (2019) Scheduling hybrid flowshop with sequence-dependent setup times and due windows to minimize total weighted earliness and tardiness. Comput Ind Eng 135:780–792
18. Nishi T, Konishi M, Hasebe S (2004) A decentralized scheduling method for flowshop problems with resource constraints. Electr Eng Jpn 149(1):44–51
19. Süral H, Kondakci S, Erkip N (1992) Scheduling unit-time tasks in renewable resource constrained flowshops. Zeitschrift für Oper Res 36(6):497–516
20. Pei J, Liu X, Fan W, Pardalos PM, Lu S (2019) A hybrid BA-VNS algorithm for coordinated serial-batching scheduling with deteriorating jobs, financial budget, and resource constraint in multiple manufacturers. Omega 82:55–69
21. Li J, Pan Q, Duan P (2016) An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping. IEEE Trans Cybernet 46(6):1311–1324
22. Li J, Han Y, Duan PY, Han Y, Niu B, Li C, Zheng Z, Liu Y (2020) Meta-heuristic algorithm for solving vehicle routing problems with time windows and synchronized visit constraints in prefabricated systems. J Cleaner Prod. https://doi.org/10.1016/j.jclepro.2019.119464
23. Li J, Liu Z, Li C, Zheng Z (2020) Improved artificial immune system algorithm for Type-2 fuzzy flexible job shop scheduling problem. IEEE Trans Fuzzy Syst. https://doi.org/10.1109/TFUZZ.2020.3016225
24. Cheng T, Lin B, Huang H (2012) Resource-constrained flowshop scheduling with separate resource recycling operations. Comput Oper Res 39(6):1206–1212
25. Leu S, Hwang S (2002) GA-based resource-constrained flow-shop scheduling model for mixed precast production. Autom Constr 11(4):439–452
26. Gao K, Huang Y, Sadollah A et al (2020) A review of energy-efficient scheduling in intelligent production systems. Complex Intell Syst 6:237–249
27. Nguyen S, Mei Y, Zhang M (2017) Genetic programming for production scheduling: a survey with a unified framework. Complex Intell Syst 3:41–66
28. Dai M, Tang D, Giret A, Salido MA, Li WD (2013) Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robot Comput Integr Manuf 29(5):418–429
29. Jiang ED, Wang L (2019) An improved multi-objective evolutionary algorithm based on decomposition for energy-efficient permutation flow shop scheduling problem with sequence-dependent setup time. Int J Prod Res 57(6):1756–1771
30. Li JQ, Pan QK, Duan PY, Sang HY (2019) Solving multi-area environmental/economic dispatch by a Pareto-based chemical-reaction optimization algorithm. IEEE/CAA J Autom Sin 6(5):1240–1250
31. Lei D, Gao L, Zheng Y (2017) A novel teaching-learning-based optimization algorithm for energy-efficient scheduling in hybrid flow shop. IEEE Trans Eng Manage 65(2):330–340
32. Ding JY, Song S, Wu C (2016) Carbon-efficient scheduling of flow shops by multi-objective optimization. Eur J Oper Res 248(3):758–771
33. Li JQ, Du Y, Gao KZ, Duan PY, Gong DW, Pan QK, Suganthan PN (2020) A hybrid iterated greedy algorithm for a crane transportation flexible job shop problem. IEEE Trans Autom Sci Eng (**In press**)
34. Li JQ, Tao XR, Jia BX, Han YY, Liu C, Duan P, Zheng ZX, Sang HY (2019) Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots. Swarm Evolution Comput. https://doi.org/10.1016/j.swevo.2019.100600
35. Li JQ, Deng JW, Li CY, Han YY, Tian J, Zhang B, Wang CG (2020) An improved jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. Knowl-Based Syst. https://doi.org/10.1016/j.knosys.2020.106032
36. Bruzzone AA, Anghinolfi D, Paolucci M, Tonelli F (2012) Energy-aware scheduling for improving manufacturing process sustainability: a mathematical model for flexible flow shops. CIRP Ann 61(1):459–462
37. Zhang H, Zhao F, Fang K, Sutherland JW (2014) Energy-conscious flow shop scheduling under time-of-use electricity tariffs. CIRP Ann 63(1):37–40
38. Fazli Khalaf A, Wang Y (2018) Energy-cost-aware flow shop scheduling considering intermittent renewables, energy storage, and real-time electricity pricing. Int J Energy Res 42(12):3928–3942
39. Lu C, Gao L, Li X, Pan Q, Wang Q (2017) Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. J Clean Prod 144:228–238
40. Huang RH, Yu SC, Chen PH (2017) Energy-Saving Scheduling in a Flexible Flow Shop Using a Hybrid Genetic Algorithm. Journal of Environmental Protection 8(10):1037
41. Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. 2007 IEEE congress on evolutionary computation. IEEE 2007:4661–4667
42. Pooranian Z, Shojafar M, Javadi B et al (2014) Using imperialist competition algorithm for independent task scheduling in grid computing. J Intell Fuzzy Syst 27(1):187–199
43. Lucas C, Nasiri-Gheidari Z, Tootoonchian F (2010) Application of an imperialist competitive algorithm to the design of a linear induction motor. Energy Convers Manage 51(7):1407–1411
44. Niknam T, Taherian Fard E, Pourjafarian N, Rousta A (2011) An efficient hybrid algorithm based on modified imperialist compet-

itive algorithm and K-means for data clustering. Eng Appl Artif Intell 24(2):306–317
45. Lei D, Yuan Y, Cai J, Bai D (2020) An imperialist competitive algorithm with memory for distributed unrelated parallel machines scheduling. Int J Prod Res 58(2):597–614
46. Karimi S, Ardalan Z, Naderi B, Mohammadi M (2017) Scheduling flexible job-shops with transportation times: mathematical models and a hybrid imperialist competitive algorithm. Appl Math Model 41:667–682
47. Lei D, Li M, Wang L (2019) A two-phase meta-heuristic for multiobjective flexible job shop scheduling problem with total energy consumption threshold. IEEE Trans Cybern 49(3):1097–1109
48. Mohammadi S, Kakaie R, Ataei M, Pourzamani E (2017) Determination of the optimum cut-off grades and production scheduling in multi-product open pit mines using imperialist competitive algorithm (ICA). Resourc Policy 51:39–48
49. Karimi N, Zandieh M, Najafi AA (2011) Group scheduling in flexible flow shops: a hybridised approach of imperialist competitive algorithm and electromagnetic-like mechanism. Int J Prod Res 49(16):4965–4977
50. Bahrami H, Faez K, Abdechiri M (2010) Imperialist competitive algorithm using chaos theory for optimization. In: 2010 12th International Conference on Computer Modelling and Simulation. IEEE, pp 98–103
51. Lin JL, Cho CW, Chuan HC (2013) Imperialist competitive algorithms with perturbed moves for global optimization. Appl Mech Mater Trans Tech Publ 284:3135–3139
52. Lin JL, Tsai YH, Yu CY et al (2012) Interaction enhanced imperialist competitive algorithms. Algorithms 5(4):433–448
53. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
54. Zandieh M, Khatami AR, Rahmati SHA (2017) Flexible job shop scheduling under condition-based maintenance: improved version of imperialist competitive algorithm. Appl Soft Comput 58:449–464
55. Montgomery DC (2005) Design and analysis of experiments. Wiley, Arizona
56. Li J, Duan P, Sang H, Wang S, Liu Z, Duan P (2018) An efficient optimization algorithm for resource-constrained steelmaking scheduling problems. IEEE Access 6:33883–33894
57. Li J, Han Y (2020) A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system. Cluster Comput. https://doi.org/10.1007/s10586-019-03022-z
58. Goldansaz SM, Jolai F, Zahedi Anaraki AH (2013) A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop. Appl Math Model 37(23):9603–9616
59. Yan J, Li L, Zhao F et al (2016) A multi-level optimization approach for energy-efficient flexible flow shop scheduling. J Clean Prod 2016:1543–1552
60. Luo H, Du B, Huang GQ et al (2013) Hybrid flow shop scheduling considering machine electricity consumption cost. Int J Prod Econ 146(2):423–439