Journal of Cleaner Production 181 (2018) 584–598

Contents lists available at ScienceDirect

Journal of Cleaner Production

journal homepage: www.elsevier.com/locate/jclepro

Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions

Jun-qing Li [a, b, c, *], Hong-yan Sang [b, **], Yu-yan Han [b], Cun-gang Wang [b], Kai-zhou Gao [b]

[a] School of Information Science and Engineering, Shandong Normal University, 250014, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] State Key Laboratory of Synthetic Automation for Process Industries, Northeastern University, Shenyang 110819, PR China

ARTICLE INFO

Article history:
Received 15 June 2017
Received in revised form
31 January 2018
Accepted 1 February 2018
Available online 16 February 2018

Keywords:
Hybrid flow shop scheduling problem
Multi-objective optimization
Setup energy consumption
Energy-aware

ABSTRACT

This paper proposes an energy-aware multi-objective optimization algorithm (EA-MOA) for solving the hybrid flow shop (HFS) scheduling problem with consideration of the setup energy consumptions. Two objectives, namely, the minimization of the makespan and the energy consumptions, are considered simultaneously. In the proposed algorithm, first, each solution is represented by two vectors: the machine assignment priority vector and the scheduling vector. Second, four types of decoding approaches are investigated to consider both objectives. Third, two efficient crossover operators, namely, Single-point Pareto-based crossover (SPBC) and Two-point Pareto-based crossover (TPBC) are developed to utilize the parent solutions from the Pareto archive set. Then, considering the problem structure, eight neighborhood structures and an adaptive neighborhood selection method are designed. In addition, a right-shifting procedure is utilized to decrease the processing duration for all machines, thereby improving the energy consumption objective of the given solution. Furthermore, several deep-exploitation and deep-exploration strategies are developed to balance the global and local search abilities. Finally, the proposed algorithm is tested on sets of well-known benchmark instances. Through the analysis of the experimental results, the highly effective proposed EA-MOA algorithm is compared with several efficient algorithms from the literature.

1. Introduction

The HFS scheduling problem is one generalization of the classical flow shop scheduling problem (FSSP), which has been verified to be a Non-deterministic Polynomial-time hard (NP-hard) problem (Ruiz and Vázquez Rodríguez, 2010; Ribas et al., 2010). In an HFS problem, two types of tasks should be considered simultaneously: assigning machines for each job and scheduling each job on each assigned machine. Therefore, the HFS problem is harder than the classical FSSP due to the additional consideration of parallel device selection for each job. Many published papers have discussed solving the HFS problem with many different algorithms. We can classify these algorithms by the number of stages in the considered problems. There are three types of problems: two-stage, three-stage, and *m*-stage. The two-stage problem is the HFS problem with two consecutive stages, while the *m*-stage problem has a series of *m* stages. Gupta (1988) studied the HFS problem with two stages where there is only one device in the second stage. Lin and Liao (2003) investigated the same problem with setup time and dedicated machines. Riane et al. (1998) developed an efficient heuristic for minimizing the makespan in a three-stage HFS problem. Carlier and Neron (2000) proposed an exact algorithm for solving the multi-processor flow shop. The benchmark problems that they generated were used in many studies as test problems.

The HFS with *m* stages is closer to the production reality. Therefore, it has been the focus of more research. Exact algorithms were first applied to solve the *m*-stage HFS problem, such as the Lagrange method (Chang and Liao, 1994) and the B&B algorithm

* Corresponding author. School of information science and engineering, Shandong Normal University, 250014, PR China.
** Corresponding author.
E-mail addresses: lijunqing@lcu-cs.com (J.-q. Li), sanghongyan@lcu-cs.com (H.-y. Sang).

https://doi.org/10.1016/j.jclepro.2018.02.004
0959-6526/

(Portmann et al., 1998). However, exact algorithms have limited ability to solve HFS problems with large scales. During recent years, heuristic and meta-heuristic algorithms have been developed to solve HFS problems, including genetic algorithm (GA) (Oguz and Ercan, 2005; Engin et al., 2011), artificial bee colony (ABC) (Li and Pan, 2015; Li et al., 2016a,b; Pan, 2016), iterated greedy (IG) (Ying et al., 2014), cuckoo search algorithm (CSA) (Marichelvam et al., 2014a,b), parallel tabu search algorithm (PTSA) (Bożejko et al., 2013), particle swarm optimization (PSO) (Liao et al., 2012; Chou, 2013), estimation of distribution algorithm (EDA) (Wang et al., 2015), bi-layer optimization approach (BLO) (Jiang et al., 2015), artificial immune system (AIS) (Chung and Liao, 2013), local search method (Lei and Guo, 2016), ant colony optimization (ACO) (Qin et al., 2015), bat algorithm (Marichelvam et al., 2013), and fruit-fly optimization algorithm (FOA) (Li et al., 2016a,b). Very recently, some hybrid meta-heuristics have also been designed to solve HFS problems, such as a hybrid of GA and TS (Sukkerd and Wuttipornpun, 2016), a hybrid of ABC algorithm and several heuristics (Pan et al., 2014), a hybrid of ABC and TS (Li and Pan, 2015), a combination of GA and imperialist competitive algorithms (ICA) (Moradinasab et al., 2013), and a hybrid of variable neighborhood search (VNS) algorithms (Li et al., 2014a,b). Some meta-heuristics have better global search abilities, while others have better local search abilities. Therefore, well-designed hybrid algorithms can always obtain better performances than single algorithms. However, most of the current literature about HFS problems has not considered machine differences in terms of power consumption capabilities.

In recent years, multi-objective optimization algorithms have been considered and studied in many fields (Deb et al., 2002; Deb and Jain, 2014; Zhang and Li, 2007; Marichelvam et al., 2014a,b; Wang and Liu, 2014; Huang et al., 2015; Tran and Ng, 2013; Shahvari and Logendran, 2016; Pan et al., 2011). Several multi-objective optimization algorithms have been proposed, such as NSGA-II (Deb et al., 2002), NSGA-III (Deb and Jain, 2014), and MOEA/D (Zhang and Li, 2007). Most of the published multi-objective algorithms have been investigated to solve continuous optimization problems. There is less literature on solving multi-objective HFS problems. Marichelvam et al. (2014a,b) proposed a discrete firefly algorithm to solve the HFS problem considering two objectives, i.e., makespan and the mean flow time. Wang and Liu (2014) investigated the HFS problem with minimization of the unavailability of the first stage machine and the makespan. Huang et al. (2015) developed a subgroup PSO approach for solving multi-objective two-stage HFS problems. Tran and Ng (2013) presented a hybrid water flow algorithm for this problem considering the minimization of the makespan and the total tardiness. Shahvari and Logendran (2016) presented a TS-based algorithm for the minimization of two objectives simultaneously, i.e., the weighted sum of the total weighted completion time and the total weighted tardiness. It can be concluded from the above analysis that there is less literature in which the multi-objective features in HFS problems are considered, especially with the consideration of the energy efficiency characteristics.

Nowadays, energy efficient algorithms are being investigated by increasing numbers of researchers (Gahm et al., 2016; Che et al., 2016). Zhang et al. (2014) utilized a time-indexed integer programming formulation to minimize the electricity cost and the carbon footprint under time-of-use tariffs in flow shop environments. For the permutation flow shop problems, Ding et al. (2016a,b) designed a multi-objective NEH algorithm (MONEH), where NEH is short for Nawaz et al. (1983), and a modified multi-objective iterated greedy (MMOIG) algorithm to minimize the total energy consumption and the makespan. For parallel machine scheduling problems, Ding et al. (2016a,b) proposed a time-interval-based mixed integer linear programming formulation to minimize the total electricity cost. Zhang and Chiong (2016) investigated an enhanced local search for minimizing the total weighted tardiness and the total energy consumption in job shop horizons. Luo et al. (2013) developed a hybrid algorithm based on the ant colony optimization method to solve the HFS problems considering the electric power cost (EPC) in the presence of time-of-use (TOU) electricity prices. Dai et al. (2013) presented a genetic simulated annealing algorithm for making a significant trade-off between the makespan and the total energy consumption in flexible flow shop horizons. For the same problem, Tang et al. (2016) utilized an improved particle swarm optimization method. Lu et al. (2017) considered two objectives namely the makespan and the energy consumption in permutation flow shop scheduling problem. There is less literature on minimization of both the makespan and the energy consumption in HFS problems, and there is no published literature in which the setup energy consumption is considered.

In realistic HFS environments, some stages contain multiple devices with different processing capabilities. In addition, each machine usually contains two states, i.e., the working state and the standby state. In each state, the machine will consume different volumes of energy. Furthermore, the setup energy consumption should be considered because it is significant in practice. The main reason for considering the setup energy consumption is that, setup energy consumption may occur when the setup operation is performed to clear the previous job from the certain container, for example, some types of iron in a torpedo. Different pairs of jobs may require different energy consumptions for the setup or clearing procedure. Therefore, in this study, we consider energy efficiency in HFS problems and minimize the energy consumptions and makespan. The rest of this paper is organized as follows: Section 2 gives the problem description. Then, the proposed algorithm is presented in Section 3. Section 4 reports the experimental results and compares them with those of other algorithms in the literature to evaluate the performance of the proposed algorithm. Finally, the last section presents the conclusions of our work.

2. Problem description

2.1. Notations and constraints

In an HFS problem, there are n tasks to be processed on m devices in a predefined order. All tasks and devices are available at time zero. Pre-emption is not allowed, that is, no task can be interrupted before the completion of its current operation. Setup times and setup energy consumptions are considered. Problem data are deterministic and known in advance. There are unlimited intermediate buffers between successive stages. The objective of an HFS problem is to schedule each task on each device such that the makespan and energy consumptions are minimized. The notations that are used in this paper are summarized below:

- Indices

i	index of jobs, $i = 1,2 \ldots,n$.
k	index of machines, $k = 1,2, \ldots,m$.
j	index of stages, $j = 1,2, \ldots, s$.

- Parameters

n	total number of jobs.
m	total number of machines.
s	total number of stages.
$O_{i,j}$	the jth operation of job i.
$p_{i,j}$	the processing time requirement of job i at stage j.
PM_j	set of parallel machines at stage j.
M_k	the kth machine.
pe_k	the energy consumption index of machine M_k for processing operations.
ie_k	the energy consumption index of machine M_k for standby state.
$ST_{i,h}$	the setup time for processing immediate consecutive jobs of i and h.
$SE_{i,h}$	the setup energy consumption for processing immediate consecutive jobs of i and h.
J	set of n jobs, $J=\{J_1, J_2, \ldots, J_n\}$.

- Decision variables

$s_{i,j}$	starting time of job i at stage j.
$c_{i,j}$	completion time of job i at stage j.
B^k_{time}	total busy time of M_k for processing operations.
I^k_{time}	total idle time of M_k.
c_{max}	the completion time of the last job on the last machine in the last stage.
TEC	total energy consumptions.

Based on the above notations and variables, two objectives are considered. The first objective is to minimize the maximum completion time $c_{\max} = \max_{1\le i\le n} c_{i,s}$. The second objective is to minimize the total energy consumptions (TEC), which includes two types of energy consumptions: (1) $TEC_1 = \sum_{k=1}^{m}(B^k_{time}\cdot pe_k + I^k_{time}\cdot ie_k)$: the energy consumptions during the processing and standby states for all machines, where I^k_{time} is calculated as follows: let t_1 be the last completion time on M_k, t_2 be the earliest starting time to process operations on M_k, then we get $I^k_{time} = t_1 - t_2 - B^k_{time}$; and (2) TEC_2: the energy consumptions during the setup process for each pair of consecutive jobs that are processed on the same machine.

The following assumptions and constraints are imposed in this study:

- All machines and operations are simultaneously available at time zero.
- There are no disruptions during processing times.
- Preemption is not considered, that is, all operations are processed continuously and not interrupted.
- There is infinite buffer or storage between any two consecutive stages.
- Each machine can process at most one operation at a time and each operation is processed on at most one machine at a time.
- The processing time for each operation is a positive integer.
- No two operations overlap on the same machine, i.e., the start time of the succeeding job must be greater than the completion time of its immediately preceding job plus the setup time between them.
- The starting time of any operation is greater than or equal to its release time from the previous stage.
- All operations are assigned strictly to one machine at each stage.
- Each operation has at most one immediately preceding or succeeding operation on the same machine.
- The set of parallel machines at different stage has no intersection with each other.

2.2. Example instance

The following example will help in illustrating this complex problem. Consider an instance with five jobs and three stages. There are two parallel machines in the first two stages and one machine in the last stage, that is, $n=5$, $m=5$, $PM_1=\{M_1, M_2\}$, $PM_2=\{M_3, M_4\}$, and $PM_3=\{M_5\}$. The processing times p_{ij}, the setup times ST_{ih}, the processing energy consumption index pe_k, the standby energy consumption index ie_k, and the setup energy consumption $SE_{i,h}$ are given as follows.

$$\left[p_{i,j}\right]_{5\times 3} = \begin{bmatrix} 20 & 30 & 30 \\ 30 & 30 & 30 \\ 30 & 30 & 30 \\ 40 & 30 & 30 \\ 30 & 30 & 30 \end{bmatrix}$$

$$[ST_{ih}] = \begin{bmatrix} 0 & 10 & 10 & 10 & 10 \\ 10 & 0 & 10 & 10 & 10 \\ 10 & 10 & 0 & 15 & 10 \\ 10 & 10 & 15 & 0 & 18 \\ 10 & 10 & 10 & 18 & 0 \end{bmatrix}$$

$$[pe_k]_{3\times 2} = \begin{bmatrix} 2.5 & 1.5 \\ 1 & 1.5 \\ 2 & - \end{bmatrix}$$

$$[ie_k]_{3\times 2} = \begin{bmatrix} 0.05 & 0.01 \\ 0.1 & 0.05 \\ 0.05 & - \end{bmatrix}$$

$$[SE_{ih}] = \begin{bmatrix} 0 & 10 & 10 & 10 & 10 \\ 10 & 0 & 10 & 10 & 10 \\ 10 & 10 & 0 & 15 & 10 \\ 10 & 10 & 15 & 0 & 18 \\ 10 & 10 & 10 & 18 & 0 \end{bmatrix}$$

The Gantt chart of a solution for the above problem instance is shown in Fig. 1, where each operation is represented by a rectangle that is labeled with the job number. For example, in the first stage, job J_1 is processed on the first machine followed by jobs J_3 and J_5. The makespan is 250. To compute the energy consumption objective value, we should compute the processing energy consumptions, the standby energy consumptions and the setup energy consumptions. For the example in Fig. 1, the energy consumption unit is mega joules (MJ) and the time unit is seconds. In the first stage, M_1 is busy processing J_1 from time 0–20 s. Therefore, M_1 consumes a processing energy of $2.5 \times 20 = 50$ MJ for processing J_1 in the first stage. In the first stage, M_1 is busy for $20 + 30 + 30 = 80$ s and consumes a processing energy of $2.5 \times 80 = 200$ MJ. In the second stage, M_3 is idle while waiting for J_4, and the standby energy consumption of M_3 is $0.1 \times 30 = 3$ MJ.

2.3. Pareto concepts

A multi-objective optimization (MOO) problem is generally formulated as follows (Deb et al., 2002; Deb and Jain, 2014; Zhang and Li, 2007):

$\min\{f_1(x) = z_1, \ldots, f_n(x) = z_n\}$

such that $x \in D$

where solution $x = [x_1, \ldots, x_n]$ is called the vector of decision variables, and D is the set of feasible solutions. We list the main Pareto concepts as follows:

- Pareto dominance: Let x and y be two solutions for the given problem. We write $x \prec y$ or x dominates y if $\forall j\, x_j \leq y_j$, and $x_j < y_j$ for at least one j.
- Pareto optimal solution: A solution x is Pareto optimal if there is no solution $y \in D$ that dominates x.
- Pareto front: The set of all Pareto optimal solutions in the objective space is generally called the non-dominated set or the Pareto front.

3. Proposed algorithm

3.1. Solution representation

For solving HFS problems, the classical solution representation is mainly classified into two categories: the permutation based representation, and the representation considering both the routing and the scheduling parts (Tang et al., 2016). In the permutation-based representation, each solution is represented by a string of integers. Each integer in the string corresponds to a task number. Thus, the length of the string is equal to n. In this study, we develop a novel encoding representation, which is given as follows:

- For the scheduling part, we introduce the permutation-based solution representation, because of its simplicity and ease of implementation. For the example problem in Fig. 1, the solution is {1,2,3,4,5}.
- For the routing part, we design a machine selection priority vector, which can be used to select machine as a priority vector. One example routing part vector is {{2,1},{1,2},{1}}, which indicates the following: (1) In the first stage, if two parallel machines that satisfy the selection rule that is defined in subsection 3.2 are waiting to process the same operation, M_2 will be selected first. (2) In the second stage, M_1 will be selected if the two machines that satisfy the selection condition are waiting for the same operation. It should be noted that the route priority is only used for the situation in which multiple machines that satisfy the selection condition are waiting for to process the same operation. For the above example, if only M_1 satisfies the selection rule in the first stage, then the operation will be processed on M_1 rather than on M_2.

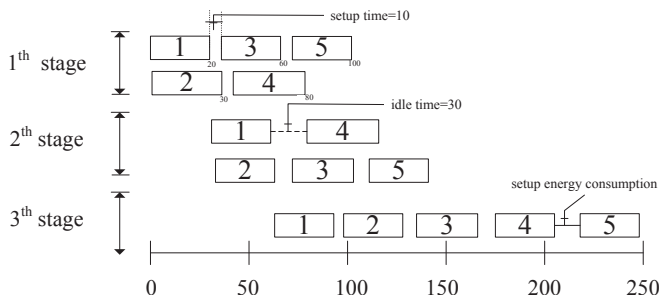


Fig. 1. Example Gantt chart.

3.2. Decoding

The solution encoding that is defined above contains no machine selection (routing) information in each stage. To decode a solution, we should consider two issues simultaneously, i.e., the job schedule and the machine assignment. The job scheduling rule is given as follows: (1) in the first stage, schedule each job one by one according to job sequence in the solution representation; and (2) in the other stages, each job will be scheduled as soon as it completes its previous operation.

For the machine assignment, we propose four types of decoding heuristics: (1) the first available machine strategy, which is named DE-I, where the first available machine is assigned to the waiting job. If more than one machine satisfies the available time simultaneously, then select the machine with the highest priority by referring to the machine priority vector; (2) the machine selection strategy with consideration of the minimum setup time, which is named DE-II. When one job is to be scheduled, it will select the machine on which the last processing operation has the minimum setup time with the current operation. If several machines satisfy the selection rule, then refer to the machine priority as well; (3) the machine selection strategy with consideration of the minimum setup energy consumption, which is named DE-III. When one job is to be scheduled, it will select the machine on which the last processing operation has the minimum setup energy consumptions for the current operation. The machine priority also will be referenced; and (4) the hybrid machine selection strategy, which is named DE-IV. When one job is to be scheduled, it will select the machine according to the following rules: (a) Select the machine with the first available time. (b) If there exist several machines with the same available time, select the one on which the last processing operation has the minimum setup energy consumption for the current operation. (c) If there are several available machines that satisfy the first two conditions, select the machine on which the last processing operation has the minimum setup time for the current operation.

3.3. Right shifting heuristic

For the given solution, if we maintain the completion time of the last processing operation on each machine at the last stage, and right-shift the other operations, then both the machine idle time and the energy consumption in the standby state for each machine will be decreased. Therefore, in this section, we present a right-shifting strategy, which is given in Fig. 2. As an example, Fig. 3 shows the Gantt chart before the right-shifting procedure, while Fig. 4 presents the Gantt chart after the right-shifting procedure. According to the two figures, the right-shifting procedure improves the solution. The time complexity of the right-shifting heuristic is $O(snm)$, where s is the number of stages, m is the number of machines, and n represents the number of jobs.

3.4. Initialization of the population

The initial population quality is crucial for the proposed algorithm. An initial population with high levels of quality and diversity may result in a faster convergence to accurate solutions. In this study, we employ the following simple initialization approach:

Step 1 Let counter $C_{nt} = 1$.
Step 2 If $C_{nt} = P_s$, stop the procedure; otherwise, randomly produce a solution.

Procedure *right-shifting*()

Input: a solution

Output: the scheduling Gantt chart

Step 1: For each stage, schedule each operation according to a selected decoding heuristic, as discussed in sub-section 3.2. The starting and completion times of each operation on each machine at each stage are recorded. Let the current maximum completion time be c_{max}^{1}.

Step2. Record the last processing operation on each machine at the last stage into the set *LP*. Let the completion time of each operation in *LP* be c_{max}^{1}.

Step 3: From the last stage to the first stage, right shift the other operations, except the operations in *LP*, as compactly as possible, that is, to decrease the idle time of each machine.

End

Fig. 2. Right-shifting heuristic.

Step 3 Evaluate the newly generated solution by using a randomly selected decoding heuristic, as discussed in section 3.2.
Step 4 If the newly generated solution is different from all solutions in the current population, add it into the population, use it to update the initial Pareto archive set, and set $C_{nt} = C_{nt} + 1$; otherwise, discard it.
Step 5 Return to Step 2.

3.5. Neighborhood structure

In this study, based on the problem structure, eight neighborhood structures are proposed:

- One swapping or insertion scheduling neighborhood, which is denoted by N_1. (1) Randomly select two positions in the scheduling vector; (2) Randomly perform one of the following two operations: (a) swap the two operations at the two positions; (b) insert one of the operations one position before or after the position of the other operation.
- One swapping or insertion routing neighborhood, which is denoted by N_2. (1) Randomly select one stage with more than two parallel machines; (2) randomly select two positions in the route vector; (3) Randomly perform one of the following two operations: (a) swap the two machines at the two selected positions; (b) insert one of the machines one position before or after the other position of the other machine.
- Multiple swapping or insertion scheduling neighborhood, which is denoted by N_3. (1) Randomly select a number h in [1,5]; (2) Perform one swapping or insertion scheduling neighborhood h times.
- Multiple swapping or insertion routing neighborhood, which is denoted by N_4. (1) Randomly select a number h in [1,5]; (2) Perform one swapping or insertion routing neighborhood h times.
- One scheduling and one routing neighborhood, which is denoted by N_5. Perform N_1 and N_2 simultaneously.
- One scheduling and multiple routing neighborhood, which is denoted by N_6. Perform N_1 and N_4 simultaneously.
- Multiple scheduling and one routing neighborhood, which is denoted by N_7. Perform N_2 and N_3 simultaneously.
- Multiple scheduling and multiple routing neighborhood, which is denoted by N_8. Perform N_2 and N_4 simultaneously.

3.6. Adaptive neighborhood structure

To balance the exploration and exploitation capabilities of the

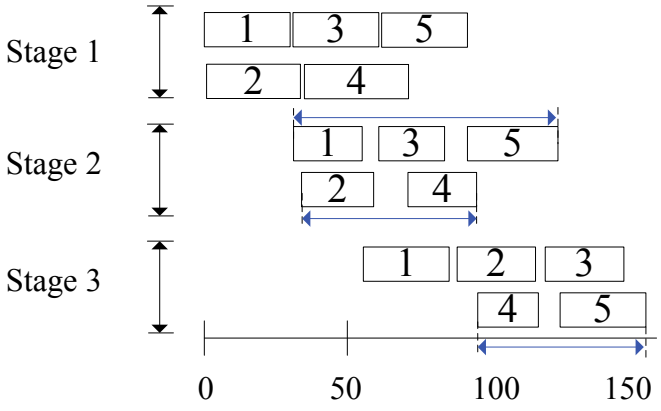


Fig. 3. Gantt chart before applying the right-shifting heuristic.

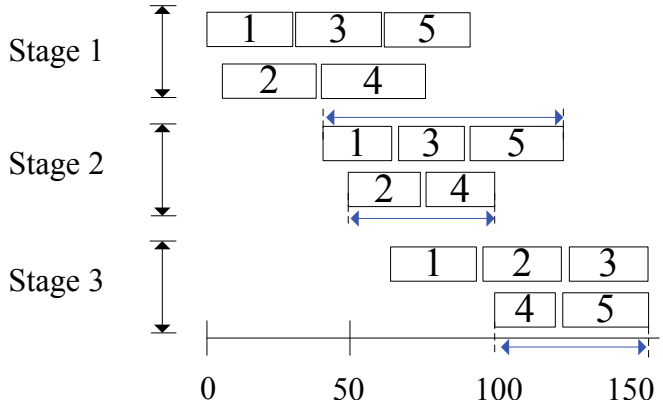


Fig. 4. Gantt chart after applying the right-shifting heuristic.

proposed algorithm, we introduce an extended version of the adaptive neighborhood structure in (Li et al., 2014a,b), to apply different neighborhood structures during the evolution phase. The detailed implementation of the adaptive neighborhood structure is as follows:

Step 1 In the initialization phase, the eight neighborhood structures are randomly inserted into the neighbor list (*NL*).
Step 2 During the evolution, each neighborhood structure in the first available position of the *NL* is selected to produce a neighboring solution. If the neighboring solution is not worse than the previous solution (i.e., it dominates or is non-dominated by the previous solution), the winner neighbor list (WNL) is updated with the selected neighborhood structure.
Step 3 The update processes of the two neighbor lists, namely, *NL* and *WNL*, are similar to those in (Li et al., 2014a,b), except that, in the proposed algorithm, during the deep-exploitation phase, *NL* is also used to select a neighborhood structure, and *WNL* is also updated if the newly generated neighboring solution is not worse than the previous one.

3.7. Local search phase

3.7.1. Exploitation phase

The local optimization method is generally considered the main component of an optimization algorithm (Pan et al., 2009). In the proposed algorithm, the detailed steps of the exploitation phase are given as follows:

Step 1 For each solution in the current population, randomly select one neighborhood structure and apply the selected neighborhood structure to generate one neighboring solution.
Step 2 Evaluate the newly generated solution and perform the following replacement processes:
Step 3 Update the current solution: If the newly generated solution dominates the current solution, then replace the latter; Step 4. Update the external Pareto archive set. (1) If the neighboring solution dominates any solution in the external Pareto archive set, then delete the solutions that are dominated by it and insert it into the external Pareto archive set. (2) If the newly generated solution is not dominated by any solution in the external Pareto archive set, then insert it into the set.

3.7.2. Deep-exploitation phase

To achieve a deep level of exploitation for the given solution, we propose a deep-exploitation function, which consists of the following steps:

Step 1 For each non-dominated solution in the Pareto archive set, perform the following steps:
Step 2 Generate a random number r. If r is less than the given deep-exploitation probability DE_p, then perform step 3; otherwise, stop the procedure.
Step 3 Perform the following steps DE_t times:
Step 4 Obtain a neighborhood structure from the neighbor list *NL*, and produce a neighboring solution S_n around the current solution S_c by using the selected neighborhood structure.
Step 5 Evaluate the newly generated neighboring solution S_n. Then, perform the following two procedures: update the current solution and update the external Pareto archive set, which are similar to steps 3 and 4 in sub-section 3.7.1

3.8. Global search phase

3.8.1. Crossover operators

In the proposed algorithm, the crossover operators are used to generate a new solution from other solutions. The most commonly used crossover operators, namely, Single-point crossover (SPC) and Two-point crossover (TPC), are included. Because the encoding mechanism is different from those in other studies, the SPC and TPC operators are also different. Furthermore, we proposed two additional efficient crossover operators: Single-point Pareto-based crossover (SPBC) and Two-point Pareto-based crossover (TPBC). The main difference between SPBC and SPC is that the former selects two parent solutions from the Pareto archive set in a random way.

Given two parent solutions p_1 and p_2 that have been randomly selected from the Pareto archive set, the newly generated child solutions are named c_1 and c_2. The detailed implementations of the crossover operators are described as follows:

- Single-point Pareto-based crossover: For the scheduling part, perform the following two steps: (1) for the two parent solutions p_1 and p_2, randomly select one position r_1 in the scheduling part; (2) let the elements before r_1 in the scheduling parts of c_1 and c_2 learn from p_1 and p_2, respectively, and let those in the following part learn from p_2 and p_1; (3) repair the elements in c_1 and c_2. For the routing part, perform the following four steps: (1) randomly select a stage r_s with more than one parallel machine; (2) in stage r_s, randomly select one machine position m_1; (3) let the elements before m_1 in the routing parts of c_1 and c_2 learn from p_1 and p_2, respectively, and let those in the following part learn from p_2 and p_1; (4) repair the elements in c_1 and c_2.
- Two-point Pareto-based crossover: For the scheduling part, perform the following two steps: (1) for the two parent solutions p_1 and p_2, randomly select two positions r_1 and r_2 in the scheduling part; (2) let the elements between r_1 and r_2 in the scheduling parts of c_1 and c_2 learn from p_2 and p_1, respectively; (3) fill in the blank positions in the scheduling parts of c_1 and c_2 with the remaining elements in p_1 and p_2, respectively, according to their occurrence orders. For the routing part, perform the following four steps: (1) randomly select a stage r_s with more than one parallel machine; (2) in stage r_s, randomly select two machine positions m_1 and m_2; (3) let the elements between m_1 and m_2 in the routing parts of c_1 and c_2 learn from p_2 and p_1, respectively; (4) fill in the blank positions in the routing parts of c_1 and c_2 with the remaining elements in p_1 and p_2, respectively, according to their occurrence orders. The numerical example of the two-point Pareto-based crossover is given in Fig. 5. In Fig. 5(b), the third stage is selected and the machine priority vectors that correspond to the selected stage are selected to perform the two-point crossover.

3.8.2. Deep exploration phase

To prevent the algorithm from becoming stuck in a local optimum, a solution is abandoned if it can't be improved after a specified number of generations (Pan et al., 2009; Fu et al., 2015). Then, a random solution replaces the abandoned solution to maintain a high level of diversity in the population. In this study, we apply the following steps:

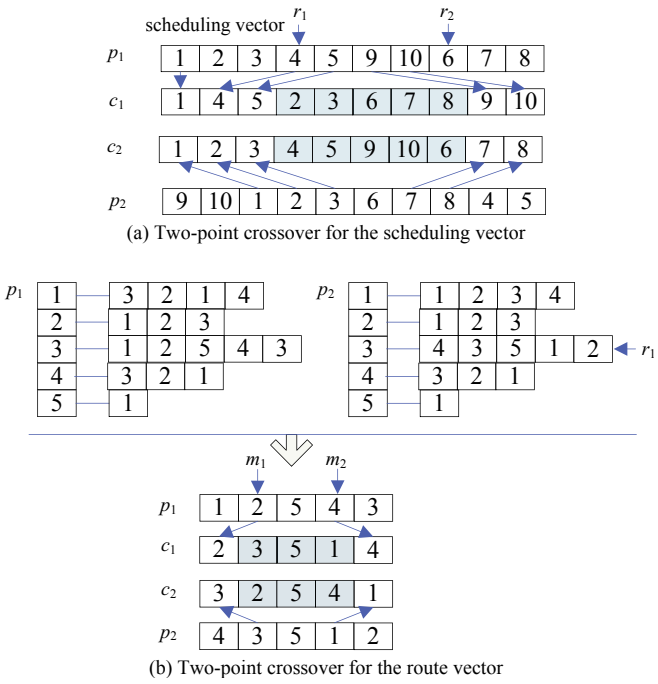


Fig. 5. Two-point Pareto-based crossover.

Step 1 By using the eight neighborhood structures, generate eight neighboring solutions around the abandoned solution and eight neighboring solutions around a randomly selected solution from the Pareto archive set.
Step 2 Apply the Pareto non-dominated sorting procedure to the sixteen neighboring solutions, and randomly select one solution in the first Pareto front as the current solution.

3.9. Framework of the proposed algorithm

The framework of the proposed algorithm is illustrated in Fig. 6. The detailed steps of the proposed EA-MOA algorithm are as follows:

Step 1 Initialization phase
  Step 1.1 Set the system parameters.
  Step 1.2 Initialize the population.
Step 2 Apply the non-dominated sorting function to the initial population, and initialize the Pareto archive set using the first level of the Pareto front of the initial population.
Step 3 If the stopping criterion is satisfied, output the best Pareto solutions; otherwise, perform steps 4 to 8.
Step 4 Exploitation search phase
  Step 4.1 For each solution in the current population, perform the following steps:
  Step 4.2 Select the first neighborhood structure in the winner neighbor list (*WNL*) to generate a neighboring solution.
  Step 4.3 Evaluate the newly generated solution, and update the winner neighbor list (*WNL*) by the following rules: (1) if the neighboring solution is not worse than the current solution, that is, if the neighboring solution dominates the current one or is non-dominated by the latter, then update the WNL by using the currently selected neighborhood structure; (2) if the neighboring solution is dominated by the current one, then update the NL.

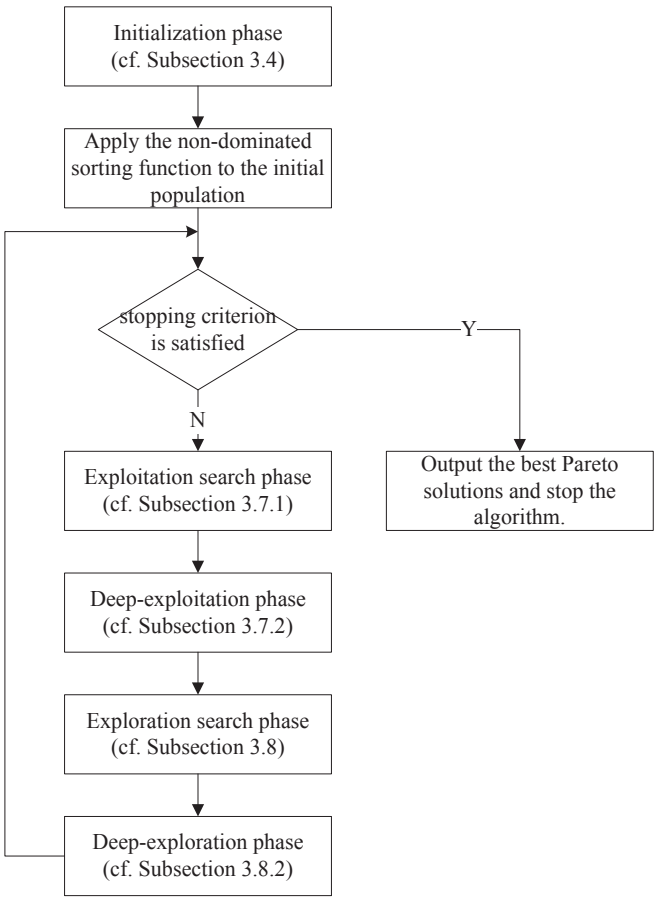


Fig. 6. Framework of the proposed algorithm.

Step 5 Deep-exploitation phase.
  Step 5.1 For each solution in the Pareto archive set, perform the following steps:
  Step 5.2 Perform the deep-exploitation procedure, as discussed in section 3.7.2.
  Step 5.3 Apply the non-dominated sorting function to the newly generated neighboring solutions and update the Pareto archive set by using neighboring solutions in the first Pareto front level.
  Step 5.4 Update the winner neighbor list (*WNL*) by using the neighborhood structures that have been used to generate a non-dominated neighboring solution.
Step 6 Exploration search phase.
  Step 6.1 Perform the following steps $P_{size}/2$ times:
  Step 6.2 Select two parent solutions by using the following two rules in a random way: (1) randomly select two solutions from the current population; (2) randomly select two solutions from the Pareto archive set.
  Step 6.3 Apply the single-point or two-point crossover function in a random way by using the two selected parent solutions.
  Step 6.4 Insert the two neighboring solutions into a temporary neighbor set *NS*.
  Step 6.5 Combine the current population with the neighbor set *NS* to generate a whole population, and apply the non-dominated sorting function to the whole population.

Step 6.6 Generate the next population from the first levels in the Pareto front. For the last Pareto level, use the crowding-distance approach.

Step 6.7 Update the winner neighbor list (*WNL*) by using the neighborhood structures that have been used to generate a non-dominated neighboring solution.

Step 7 Deep-exploration phase. If a solution in the population has not been improved during the limit trials, abandon it and utilize the deep-exploration function, as discussed in section 3.8.2.

4. Experiment results

This section discusses the computational experiments that were used to evaluate the performance of the proposed algorithm. Our algorithm was implemented in C++ on an Intel Core i5 3.3 GHz PC with 4GB memory. The compared algorithms were NSGA-II (Deb et al., 2002), MOEA/D (Zhang and Li, 2007), DBEA (Asafuddoula et al., 2015) and EADD (Li et al., 2015a,b). All four compared algorithms utilize the same coding mechanism, the same initialization function, and the same stopping criterion. Different from the proposed EA-MOA, the four compared algorithms include the following features: (1) for decoding, the compared algorithms use the first available machine first service rule; (2) for the mutation operator, the compared algorithms randomly select the swapping or insertion neighboring structure; and (3) for the crossover operator, the compared algorithms utilize the single-point crossover (SPC) operator. To verify the efficiency of the proposed heuristics, we also compared the algorithm without the proposed heuristics.

Two types of instances are tested for evaluating the performance of the proposed algorithm, which are given as follows.

(1) The extended versions of Carlier and Neron's benchmark problems (Carlier and Neron et al., 2000). In this study, we select and extend to instances with 10 jobs scale and instances with 15 jobs, where we add the setup time and setup energy consumption values for each instance to evaluate both the makespan and the energy consumption objectives. In this study, we rename the extended instances; for example, "cj10c5a2" denotes the extension of instance "j10c5a2";

(2) The large-scale problem instances, which include twenty instances, with problem scale ranging from 50 jobs to 200 jobs.

4.1. Setting parameters

Each instance can be characterized by the following parameters: the number of tasks (n), the number of machines (m), and the number of stages (s). After plenty of preliminary experiments, the parameters for the proposed EA-MOA algorithm are set as follows:

- Deep-exploitation probability DE_p: 0.5;
- Number of deep-exploitation iterations DE_t: 20;
- Population size P_{size}: 100;
- Pareto archive set size: $2 \times P_{size}$;
- Number of iterations after which a solution is abandoned: 20;
- Maximum number of iterations (stopping condition): 500.

4.2. Comparison metrics

In order to compare the performances of different algorithms in solving the multi-objective problems, we utilize the three performance metrics that were discussed in (Pan et al., 2009; Li et al., 2014a,b), i.e., the average Pareto distance V_{pd}, number of the non-dominated solutions V_{np}, and the ratio of the non-dominated solutions V_{rd}. Let S^P denote the reference solution set which is obtained by running all the compared algorithms for 3000 iterations. Let S^j ($j = 1,2,3,4$) represent the non-dominated solution set that is obtained by algorithm j, where $S^P = \cup S^j$. Then, the detailed computation processes of the three metrics are as follows.

(1) Average Pareto distance V_{pd}

Let $V_{pd} = \frac{1}{|S^P|}\sum_{y\in S^P} d_y(S^P)$ and $d_y(S^j) = \min_{x\in S^j}\left\{\sum_{i=1}^{2}\left(\frac{f_i(x)-f_i(y)}{f_i^{max}(.)-f_i^{min}(.)}\right)^2\right\}$, $y\in S^P$, where $f_i(.)$ represents the ith objective value, and $f_i^{max}(.)$ and $f_i^{min}(.)$ are the minimum and maximum, respectively, of the ith objective value in the Pareto referent point set S^P. $d_y(S^j)$ represents the shortest normalized distance from a reference solutions y in S^P to the solution set S^j. V_{pd} indicates the average of those shortest normalized distances from all the reference points to the solution set S^j.

The average Pareto distance is usually used to evaluate the spread and distribution of the obtained solution set. That is, a smaller V_{pd} indicates that the obtained solution set has a better distribution and better approximation to the reference set S^P. The most promising situation is that V_{pd} equals 0, which means that the set of obtained solutions is equal to the reference point set.

(2) Number of non-dominated solutions V_{np}

The number of non-dominated solutions is the number of obtained solutions that are not dominated by the reference solutions. A larger value of V_{np} indicates that there are more non-dominated solutions in the obtained solutions set S^j. The computational process uses the following formulation:$V_{np} = \{S^j - \{x\in S^j \,|\, \exists y\in S^P : y\prec x\}\}$, where $y\prec x$ means that solution y dominates solution x.

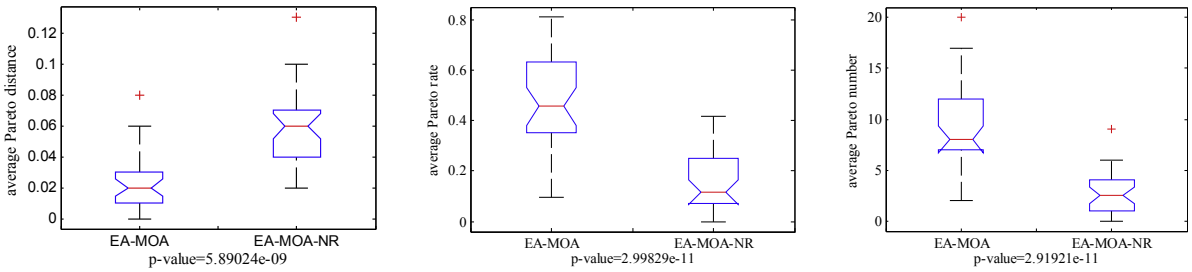
(3) Ratio of non-dominated solutions V_{rd}

The metric V_{rd} is used to compute the ratio of non-dominated solutions in the obtained solution set S^j. Obviously, a larger value of V_{rd} represents a solution set with a higher probability for the obtained solution to be a non-dominated solution. If V_{rd} is close to one, almost all of the solutions in the obtained solution set are equal to or near non-dominated solutions, whereas if V_{rd} is close to zero, almost all of the obtained solutions are dominated by solutions in the reference solution set. The computational process uses the following formulation:

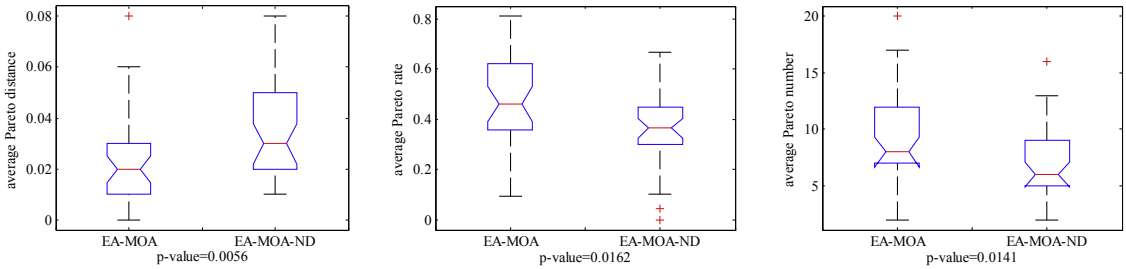
$$V_{rd} = \frac{V_{np}}{|S^j|}$$

4.3. Efficiency of the proposed components

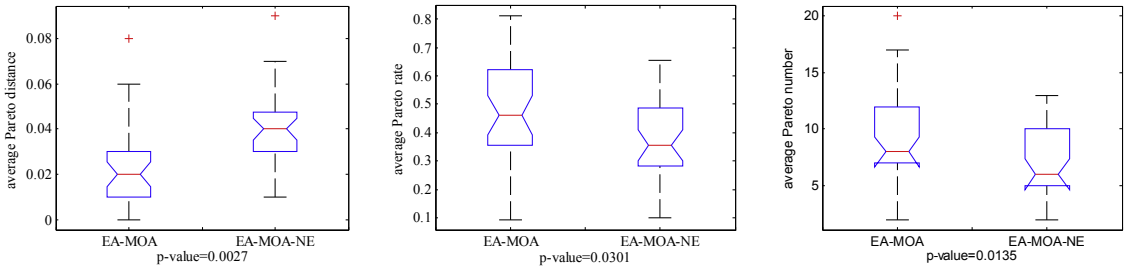
To verify the effectiveness of the proposed components in EA-



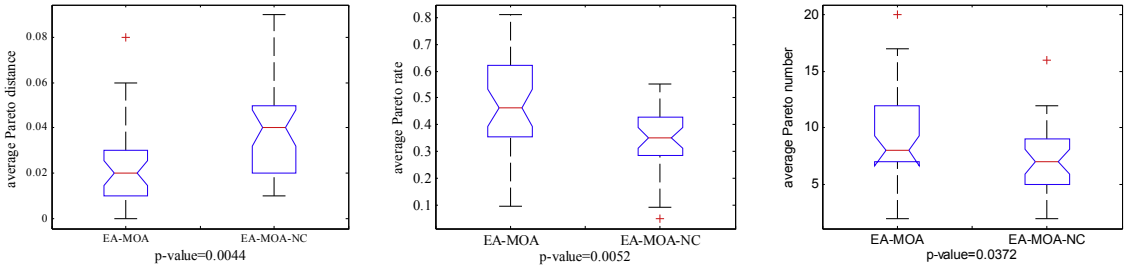
(a) Comparison results for the right-shifting heuristic



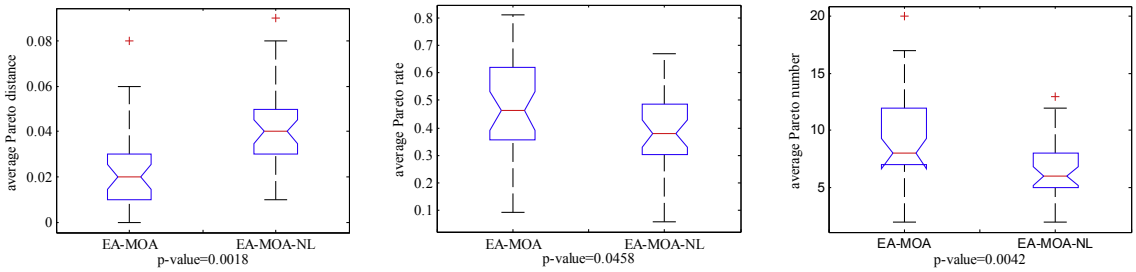
(b) Comparison results for the deep-exploitation heuristic



(c) Comparison results for the decoding heuristic



(d) Comparison results for the crossover heuristic



(e) Comparison results for the local search heuristic

Fig. 7. Means and 95% LSD interval for pairs of compared algorithms.

MOA, in this section, we conduct detailed comparisons of the algorithms with and without the proposed heuristics. The algorithm with all of the components is denoted as EA-MOA, the algorithm without the right-shifting heuristic is named EA-MOA-NR, the algorithm without the decoding heuristic is represented by EA-MOA-NE, the algorithm without the proposed crossover operator is denoted by EA-MOA-NC, the algorithm without the deep exploitation procedure is called EA-MOA-ND, and the algorithm without the local search methods is named EA-MOA-NL. It should be noted that: (1) EA-MOA-NE uses the first-available-machine-first-service method; and (2) EA-MOA-NC uses the single-point crossover operator. The parameters of the five pairs of algorithms are the same as those that were discussed in Section 4.1.

To check whether there exist significant differences between the compared pairs of algorithms, we performed a multifactor analysis of variance (ANOVA), where the five pairs of compared algorithms are considered as factors. Fig. 7 illustrate the means and the 95% LSD (Fisher's Least Significant Difference) interval for the best values for the five pairs of compared algorithms of the average Pareto distance, the average Pareto rate and the average Pareto number. There are significant differences between the five pairs of compared algorithms.

4.4. Comparisons with NSGA-II and MOEA/D

In this section, to verify the efficiency of the proposed EA-MOA algorithm, we performed detailed comparisons with the two canonical multi-objective algorithms, i.e., NSGA-II and MOEA/D. The test instances are the extended versions of Carlier and Neron's benchmark problems (Carlier and Neron et al., 2000).

4.4.1. Comparisons of the 10-jobs problems

The computational results for the 10-job problems are summarized in Table 1, which gives the comparison results of the average Pareto distance, the number of non-dominated solutions, and the ratio of the non-dominated solutions for the 10-job instance.

There are 13 columns in Table 1. The first column gives the tested problem name. Then, the next four columns tell the computational results on the average Pareto distance for the four compared algorithms, i.e., the proposed algorithm EA-MOA, NSGA-II, MOEA/D and EA-MOA-WR. The next four columns display the results on the number of non-dominated solutions, and the ratios of the non-dominated solutions are presented in the last four columns.

The following conclusions are drawn from the computational

Table 1
Comparisons of 10-job instances.

Problem	Pareto distance V_{pd}				Pareto number V_{np}				Pareto rate V_{rd}			
	EA-MOA	NSGA-II	MOEA/D	EA-MOA-WR	EA-MOA	NSGA-II	MOEA/D	EA-MOA-WR	EA-MOA	NSGA-II	MOEA/D	EA-MOA-WR
cj10c5a2	0.03	0.21	0.03	**0.02**	**10**	1	5	9	0.38	0.13	0.20	**0.50**
cj10c5a3	**0.00**	0.26	0.02	0.02	**17**	0	6	5	**0.77**	0.00	0.23	0.33
cj10c5a4	**0.01**	0.27	0.04	0.06	**14**	0	6	1	**0.58**	0.00	0.26	0.05
cj10c5a5	**0.04**	0.34	0.05	0.06	**4**	2	0	0	0.17	**0.25**	0.00	0.00
cj10c5a6	**0.03**	0.38	0.06	0.06	**9**	0	2	0	**0.45**	0.00	0.10	0.00
cj10c5b1	**0.03**	0.17	0.09	0.07	**5**	0	0	0	**0.33**	0.00	0.00	0.00
cj10c5b2	**0.04**	0.16	0.05	0.12	**8**	2	3	2	**0.44**	0.13	0.18	0.11
cj10c5b3	**0.03**	0.12	0.05	0.09	**4**	2	1	0	**0.31**	0.20	0.06	0.00
cj10c5b4	**0.07**	0.21	0.08	0.09	**2**	0	1	1	**0.10**	0.00	0.06	0.08
cj10c5b5	**0.02**	0.19	0.06	0.07	**11**	0	0	0	**0.50**	0.00	0.00	0.00
cj10c5b6	**0.04**	0.12	0.06	0.08	**2**	0	1	1	**0.10**	0.00	0.07	0.06
cj10c5c1	**0.03**	0.20	0.07	0.07	**8**	1	0	1	**0.44**	0.09	0.00	0.05
cj10c5c2	**0.02**	0.16	0.03	0.05	**12**	2	9	4	**0.43**	0.29	0.38	0.20
cj10c5c3	**0.02**	0.14	0.05	0.05	**13**	2	4	1	**0.57**	0.18	0.22	0.06
cj10c5c4	**0.02**	0.17	0.03	0.03	**7**	2	5	7	0.26	0.25	0.18	**0.30**
cj10c5c5	**0.02**	0.11	0.04	0.05	**11**	2	3	1	**0.48**	0.20	0.12	0.05
cj10c5c6	**0.02**	0.25	0.06	0.07	**6**	1	0	1	**0.27**	0.07	0.00	0.06
cj10c5d1	**0.03**	0.13	0.05	0.04	**9**	1	2	4	**0.32**	0.13	0.13	0.29
cj10c5d2	**0.03**	0.10	0.08	0.11	**9**	1	1	0	**0.39**	0.11	0.04	0.00
cj10c5d3	**0.03**	0.07	0.04	0.06	**11**	3	4	2	**0.38**	0.33	0.18	0.12
cj10c5d4	**0.03**	0.13	0.05	0.07	**7**	1	2	0	**0.29**	0.08	0.14	0.00
cj10c5d5	**0.02**	0.03	0.05	0.06	**9**	6	3	3	0.35	**0.67**	0.14	0.13
cj10c5d6	**0.04**	0.16	0.05	0.16	**6**	2	3	1	**0.33**	0.20	0.30	0.05
cj10c10a1	**0.03**	0.41	0.06	0.07	**5**	0	4	1	0.24	0.00	**0.31**	0.05
cj10c10a2	0.03	0.36	**0.02**	**0.02**	**9**	0	8	5	**0.47**	0.00	0.42	0.45
cj10c10a3	**0.02**	0.32	0.04	0.06	**7**	0	3	1	**0.44**	0.00	0.19	0.06
cj10c10a4	**0.03**	0.72	0.05	0.07	**9**	0	1	4	**0.64**	0.00	0.06	0.21
cj10c10a5	**0.04**	0.28	0.07	0.07	**9**	0	0	1	**0.35**	0.00	0.00	0.08
cj10c10a6	**0.02**	0.37	0.06	0.05	**5**	0	2	2	**0.31**	0.00	0.12	0.14
cj10c10b1	**0.01**	0.31	0.07	0.13	**11**	0	0	1	**0.73**	0.00	0.00	0.06
cj10c10b2	**0.02**	0.29	0.07	0.07	**6**	0	0	1	**0.30**	0.00	0.00	0.05
cj10c10b3	**0.01**	0.25	0.05	0.05	**14**	1	3	3	**0.67**	0.17	0.16	0.17
cj10c10b4	0.04	0.27	**0.03**	0.07	**11**	0	6	3	**0.58**	0.00	0.33	0.14
cj10c10b5	**0.02**	0.16	0.08	0.05	**3**	2	1	2	0.21	0.18	0.04	**0.22**
cj10c10b6	**0.04**	0.12	0.08	0.07	**6**	2	0	0	**0.30**	0.20	0.00	0.00
cj10c10c1	**0.03**	0.22	0.04	0.08	**8**	1	7	0	**0.42**	0.13	0.35	0.00
cj10c10c2	**0.02**	0.14	0.06	0.06	**11**	3	1	1	**0.39**	0.38	0.04	0.08
cj10c10c3	**0.03**	0.15	0.05	0.06	**13**	1	2	3	**0.52**	0.09	0.09	0.15
cj10c10c4	**0.03**	0.09	0.07	0.07	**10**	3	2	2	**0.45**	0.30	0.09	0.12
cj10c10c5	**0.03**	0.17	0.06	0.03	**7**	1	3	4	**0.33**	0.11	0.18	0.33
cj10c10c6	0.04	0.07	**0.03**	0.05	**8**	2	7	2	**0.35**	0.25	0.33	0.15
mean	**0.03**	0.21	0.05	0.07	8.44	1.15	2.71	1.95	**0.40**	0.12	0.14	0.12

- The best values are in bold.

results for the average Pareto distance: (1) In solving the 41 10-job problems, namely, “cj10c5a2” to “cj10c10c6”, the proposed EA-MOA algorithm obtained 37 best values for the average Pareto distance, which is better than the second best algorithm MOEA/D, which obtained just three best values. In addition, the algorithm EA-MOA-WR outperforms the NSGA-II algorithm. (2) According to the last line in the table, on average, the proposed algorithm outperforms the other compared algorithms. (3) Without the right-shifting procedure, the algorithm EA-MOA-WR shows slightly worse performance than EA-MOA, which verifies the efficiency and effectiveness of the right-shifting function.

It can be concluded from the computational results for the number of the non-dominated solutions that: (1) for solving the 41 10-job scale problems, the proposed EA-MOA algorithm obtained 41 best values for the number of Pareto solutions, which is obviously better than the other compared algorithms; (2) from the last line in the table, we can see that on average, the proposed algorithm performs the best, which is obviously better than the second best algorithm MOEA/D; and (3) EA-MOA shows better performance than EA-MOA-WR, which further verify the efficiency and effectiveness of the right-shifting function.

The comparison results for the ratio of Pareto solutions in Table 1 show that: (1) for solving the 41 10-job problems, namely, “cj10c5a2” to “cj10c10c6”, the proposed EA-MOA algorithm obtained 36 best values for the ratio of Pareto solutions, which is better than the second best algorithm EA-MOA-WR, which obtained just three best values; and (2) according to the last line in the table, on average, the proposed algorithm outperforms the other compared algorithms.

4.4.2. Comparisons of the 15-jobs problems

The computational results for solving the 15-job problems are summarized in Table 2. The following conclusions are obtained from Table 2: (1) In comparisons of the average Pareto distance, for solving the 41 10-job problems, namely, “cj10c5a2” to “cj10c10c6”, the proposed EA-MOA algorithm obtained 35 best values for the average Pareto distance, which is better than the second-best algorithm MOEA/D, which obtained just three best values. In addition, the EA-MOA-WR algorithm outperforms the NSGA-II algorithm. According to the last line in the table, on average, the proposed algorithm performs better than the other compared algorithms. (2) In comparisons of the number of Pareto solutions, for solving the 41 10-job problems, the proposed EA-MOA algorithm obtained all the best values for the number of Pareto solutions, which is better than the other compared algorithms. According to the last line in the table, on average, the proposed algorithm performs the best. (3) In comparisons of the ratio of Pareto solutions, for solving the 41 10-job problems, namely, “cj10c5a2” to “cj10c10c6”, the proposed EA-MOA algorithm obtained 31 best values for the ratio of Pareto solutions, which is better than the second-best algorithms MOEA/D and NSGA-II, which each obtained just four best values. According to the last line in the table, on average, the proposed algorithm outperforms the other compared algorithms.

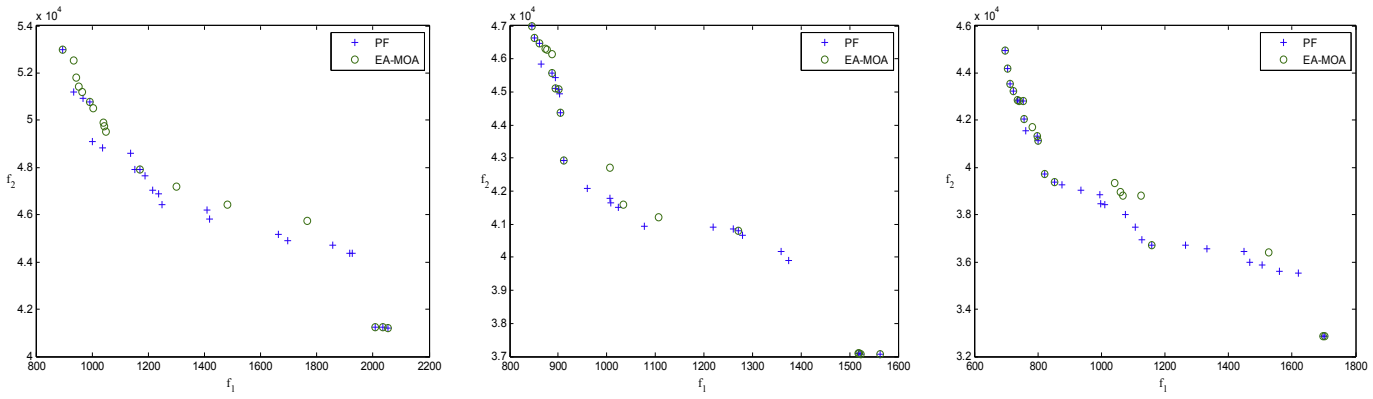
4.4.3. Comparisons analysis

According the comparison results for solving different scales of instances, we can see that the proposed EA-MOA algorithm shows better performance than the other compared algorithms. The main

Table 2
Comparisons for the instances with 15 jobs.

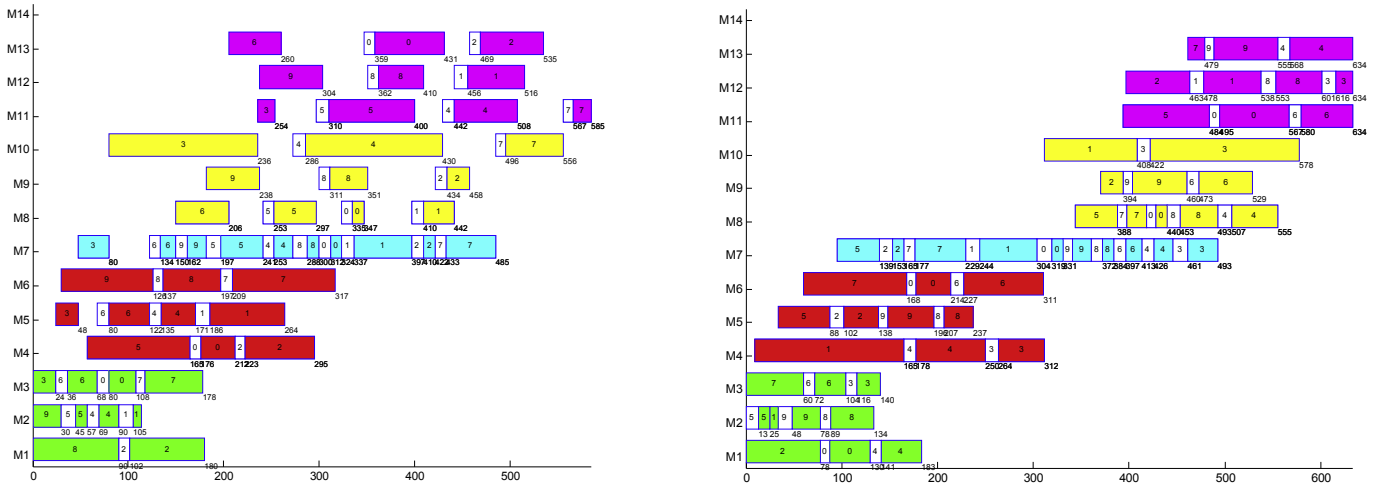
Problem	Pareto distance V_{pd}				Pareto number V_{np}				Pareto rate V_{rd}			
	EA-MOA	NSGA-II	MOEA/D	EA-MOA-WR	EA-MOA	NSGA-II	MOEA/D	EA-MOA-WR	EA-MOA	NSGA-II	MOEA/D	EA-MOA-WR
cj15c5a1	0.03	0.74	0.04	**0.02**	**6**	0	3	4	0.353	0.000	0.136	**0.364**
cj15c5a2	**0.01**	0.22	0.06	0.05	**12**	0	5	0	**0.632**	0.000	0.357	0.000
cj15c5a3	**0.02**	0.43	0.02	0.06	**15**	0	4	3	**0.714**	0.000	0.211	0.167
cj15c5a4	**0.02**	0.41	0.04	0.05	**11**	0	2	4	**0.786**	0.000	0.111	0.250
cj15c5a5	**0.01**	0.35	0.03	0.02	**12**	0	2	6	**0.667**	0.000	0.087	0.353
cj15c5a6	**0.03**	0.43	0.05	0.06	**7**	1	4	1	**0.467**	0.250	0.222	0.071
cj15c5b1	**0.03**	0.23	0.04	0.09	**6**	0	3	1	**0.333**	0.000	0.231	0.083
cj15c5b2	0.06	0.24	0.06	**0.04**	**2**	0	2	5	0.100	0.000	0.182	**0.417**
cj15c5b3	**0.06**	0.24	0.07	0.13	**4**	0	2	0	**0.211**	0.000	0.143	0.000
cj15c5b4	**0.02**	0.17	0.07	0.07	**7**	0	4	5	**0.583**	0.000	0.200	0.250
cj15c5b5	**0.02**	0.27	0.05	0.05	**8**	1	5	3	**0.421**	0.083	0.250	0.188
cj15c5b6	**0.08**	0.15	0.09	0.08	**2**	2	2	1	0.095	0.143	**0.222**	0.100
cj15c5c1	0.04	0.08	**0.03**	0.04	**8**	4	7	4	0.364	**0.571**	0.269	0.235
cj15c5c2	**0.01**	0.06	0.02	0.03	**10**	6	10	9	0.476	**0.667**	0.385	0.333
cj15c5c3	**0.03**	0.17	0.03	0.07	**5**	3	4	2	0.217	0.231	**0.250**	0.091
cj15c5c4	**0.01**	0.12	0.03	0.04	**20**	1	6	3	**0.741**	0.091	0.231	0.115
cj15c5c5	0.02	0.19	**0.01**	0.03	**13**	4	12	2	**0.542**	0.308	0.480	0.080
cj15c5c6	**0.02**	0.19	0.03	0.06	**11**	4	7	2	**0.500**	0.400	0.304	0.087
cj15c5d1	0.03	0.81	**0.01**	0.04	**8**	0	7	4	**0.800**	0.000	0.583	0.222
cj15c5d2	**0.03**	0.13	0.06	0.10	**10**	2	6	1	**0.385**	0.250	0.333	0.043
cj15c5d3	**0.02**	0.12	0.06	0.07	**5**	4	1	0	0.217	**0.364**	0.037	0.000
cj15c5d4	**0.01**	0.08	0.07	0.10	**5**	3	3	1	0.208	**0.273**	0.150	0.050
cj15c5d5	**0.01**	0.07	0.02	0.04	**17**	2	9	4	**0.586**	0.222	0.391	0.190
cj15c5d6	**0.03**	0.17	0.04	0.08	**7**	2	3	2	**0.368**	0.182	0.103	0.095
cj15c10a1	0.04	0.38	**0.03**	0.04	**7**	0	7	3	**0.438**	0.000	0.304	0.167
cj15c10a2	**0.02**	0.22	0.05	0.06	**8**	0	5	0	**0.364**	0.000	0.192	0.000
cj15c10a3	**0.03**	0.22	0.05	0.07	**7**	2	5	4	**0.438**	0.167	0.250	0.250
cj15c10a4	**0.00**	0.32	0.06	0.07	**17**	0	2	1	**0.810**	0.000	0.118	0.091
cj15c10a5	**0.01**	0.18	0.02	0.06	**12**	0	12	2	0.462	0.000	**0.480**	0.118
cj15c10a6	**0.03**	0.37	0.06	0.07	**9**	1	4	1	**0.450**	0.077	0.211	0.056
cj15c10b1	**0.01**	0.51	0.02	0.04	**9**	0	8	4	**0.474**	0.000	0.296	0.154
cj15c10b2	**0.03**	0.47	0.07	0.08	**12**	0	0	1	**0.667**	0.000	0.000	0.063
cj15c10b3	**0.03**	0.72	0.06	0.08	**7**	0	6	5	0.333	0.000	**0.353**	0.263
cj15c10b4	**0.01**	0.48	0.02	0.03	**14**	0	9	6	**0.778**	0.000	0.500	0.300
mean	**0.03**	0.29	0.04	0.06	**9.21**	1.24	5.03	2.76	**0.470**	0.126	0.252	0.154

- The best values are in bold.



(a) Pareto front for "cj15c10b2" (b) Pareto front for "cj15c10b3" (c) Pareto front for "cj15c10b4"

Fig. 8. Pareto front results.



(a) First Gantt charts for "cj10c5a2" (b) Second Gantt charts for "cj10c5a2"

Fig. 9. Gantt charts for "cj10c5a2".

reasons are as follows: (1) the proposed hybrid decoding methods that are embedded in the algorithm balance the diversity and the performance of the solutions; (2) the proposed right-shifting procedure further enhances the solution quality; and (3) the proposed exploration and exploitation procedures strengthen the balance of the global and local search abilities.

Fig. 8 (a)–(c) show the Pareto front charts for solving the last three 15-job instances, i.e., "cj15c10b2" to "cj15c10b4". According to the three figures, the proposed algorithm obtains solutions that are close to the Pareto front and well-distributed.

Fig. 9 (a) and (b) report the Gantt charts for two of the obtained non-dominated solutions, which show the effectiveness of the proposed EA-MOA algorithm.

4.5. Comparisons on large-scale instances

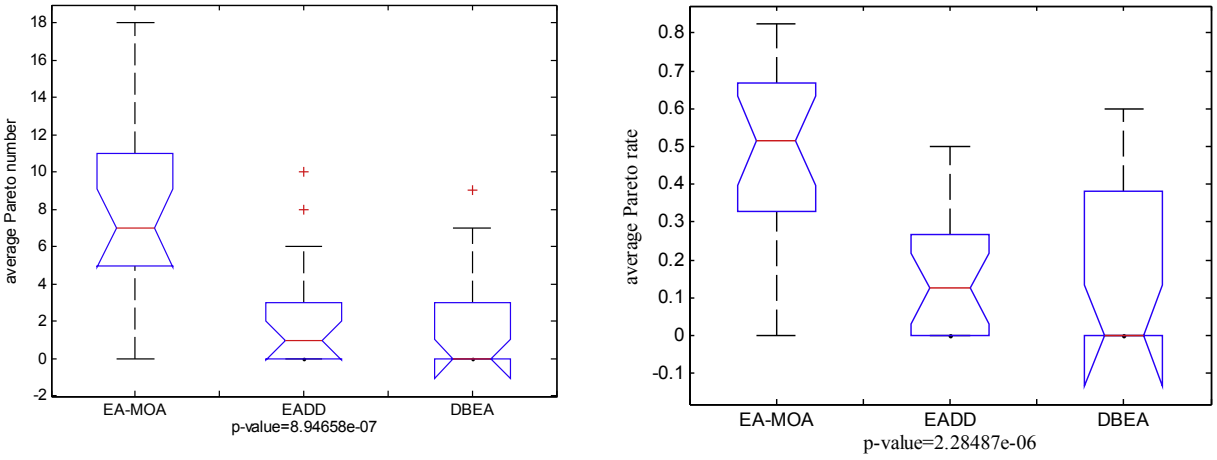
To further verify the performance of the proposed algorithm in solving large-scale problems, we randomly generate four types of large-scale instances, namely, 50-job, 100-job, 150-job, and 200-job instances. We also coded two recently published efficient algorithms, i.e., DBEA (Asafuddoula et al., 2015) and EADD (Li et al., 2015a,b), to make detailed comparisons with the proposed algorithms. The three compared algorithms are tested on the same problem instances and in the same computing environment with the same stopping criterion. The detailed comparison results after 30 independent runs are given in Table 3.

Table 3 gives the results for the 20 large-scale problems obtained by each compared algorithm. The first column gives the problem name, and the second column lists the problem scale in terms of the total number of jobs. Then, the results that were obtained by the proposed EA-MOA, EADD, and DBEA are reported in the following columns. From the comparison results, we observe the following: (1) In comparisons of the average Pareto distance, the proposed EA-MOA algorithm obtained 15 optimal values out of the given twenty large-scale instances, which is significantly better than the other two compared algorithms. The second best algorithm, namely DBEA, only obtained three optimal values. According to the last row, on average, the proposed EA-MOA algorithm obtained an average value of 0.035, which is better than those of the other compared algorithms. (2) In comparisons of the number of Pareto solutions, the proposed EA-MOA algorithm obtained 15 better values out of the given twenty large-scale instances, which is significantly better than the other two compared algorithms. The second-best algorithm, namely EADD, only obtained four optimal values. According to the last row in the table, on average, the proposed EA-MOA algorithm obtained an average value of 7.65, which

Table 3
Comparisons for the 20 large-scale problems.

Problem	Pareto distance V_{pd}			Pareto number V_{np}			Pareto rate V_{rd}		
	EA-MOA	NSGA-II	MOEA/D	EA-MOA	NSGA-II	MOEA/D	EA-MOA	NSGA-II	MOEA/D
cj50-1	**0.012**	0.182	0.226	**8**	0	0	**0.667**	0.000	0.000
cj50-2	**0.000**	0.264	0.311	**6**	0	0	**0.667**	0.000	0.000
cj50-3	**0.000**	0.119	0.130	**11**	0	0	**0.688**	0.000	0.000
cj50-4	**0.000**	0.066	0.033	**8**	1	1	**0.667**	0.125	0.200
cj50-5	**0.000**	0.042	0.049	**9**	0	0	**0.643**	0.000	0.000
cj100-1	**0.000**	0.191	0.179	**12**	0	0	**0.600**	0.000	0.000
cj100-2	**0.013**	2.052	2.275	**11**	0	0	**0.579**	0.000	0.000
cj100-3	**0.490**	0.493	0.516	0	**1**	0	**0.000**	0.250	**0.000**
cj100-4	0.076	**0.001**	0.072	**3**	2	**3**	0.250	**0.333**	**0.333**
cj100-5	0.092	0.249	**0.065**	**1**	1	0	0.111	**0.125**	0.000
cj150-1	**0.000**	0.004	0.020	**13**	3	4	0.406	0.214	**0.444**
cj150-2	**0.001**	0.009	0.298	**10**	4	0	**0.500**	0.190	0.000
cj150-3	**0.000**	0.027	0.037	5	**6**	0	**0.500**	0.500	0.000
cj150-4	**0.000**	0.006	0.018	**14**	10	1	**0.824**	0.500	0.143
cj150-5	0.014	**0.006**	0.027	2	**3**	0	**0.125**	0.300	0.000
cj200-1	**0.000**	0.001	0.000	**18**	0	3	**0.529**	0.000	0.429
cj200-2	0.002	0.002	**0.000**	5	**8**	7	0.238	0.286	**0.500**
cj200-3	**0.006**	0.028	0.008	**5**	3	**5**	0.417	0.200	**0.600**
cj200-4	**0.000**	0.163	0.308	**6**	0	0	**0.750**	0.000	0.000
cj200-5	0.001	0.071	**0.000**	6	0	**9**	0.500	0.000	**0.529**
mean	**0.035**	0.199	0.229	**7.65**	2.1	1.7	**0.483**	0.151	0.159

- The best values are in bold.



(a) Means and 95% LSD interval for the Pareto number.

(b) Means and 95% LSD interval for the Pareto rate

Fig. 10. Means and 95% LSD interval for EA-MOA, EADD, and DBEA.

is better than the values that were obtained by the other compared algorithms. (3) In comparisons of the ratio of Pareto solutions, the proposed EA-MOA algorithm obtained 14 better values out of the given twenty large-scale instances, which is significantly better than the other two compared algorithms. The second-best algorithm, namely DBEA, only obtained six optimal values. The proposed EA-MOA algorithm obtained an average value of 0.483, which is better than the values that were obtained by the other compared algorithms.

To check whether the observed differences from the above table are significant, we also performed a multifactor analysis of variance (ANOVA), where the three compared algorithms are considered factors. Fig. 10 (a) and (b) illustrate the means and the 95% LSD (Fisher's Least Significant Difference) intervals for the best values of the three compared algorithms for the average Pareto number and the average Pareto rate, respectively. There is a statistically significant difference between the three compared algorithms.

EA-MOA performs better in solving the instances with 50, 100, and 150 jobs. However, for solving the problem with 200 jobs, the proposed algorithm only slightly outperforms the other two algorithms. The main reason may be that, in solving the 200-job instance, the proposed algorithm takes more time because it considers the right-shift heuristic.

5. Conclusions

In this study, a hybrid energy-aware multi-objective optimization algorithm is proposed for solving the HFS problem with the objective of minimizing the energy consumption and the makespan simultaneously. The main contributions of the proposed algorithm are as follows: (1) a well-designed encoding and decoding mechanism that is tailored to the features of the problem is proposed; (2) eight types of neighborhood structures and an adaptive neighborhood structure selection mechanism are designed to enhance both

the exploitation and exploration capabilities; (3) a right-shifting approach is designed to improve the energy consumption objective of the given solution; and (4) to balance the global and local search abilities, a hybrid deep-exploitation and deep-exploration method is designed.

The proposed algorithm is tested on problems with different scales. Several efficient algorithms are compared with the proposed algorithm. Experimental results show the robustness and efficiency of the proposed EA-MOA algorithm. Future work will include the introduction of a local optimization method or other efficient heuristics to improve the search capabilities of the proposed algorithm, and the application of the proposed algorithm to other problems, such as multi-objective HFS problems under dynamic environments and the multi-objective flexible flow shop scheduling problems.

Acknowledgements

This research is partially supported by National Science Foundation of China under Grant 61773192, 61773246, 61603169 and 61503170, Shandong Province Higher Educational Science and Technology Program (J17KZ005), Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education (K93-9-2017-02), and State Key Laboratory of Synthetical Automation for Process Industries (PAL-N201602).

References

Asafuddoula, M., Ray, T., Sarker, R., 2015. A decomposition-based evolutionary algorithm for many objective optimization. IEEE Trans. Evol. Comput. 19 (3), 445–460.
Bożejko, W., Pempera, J.A., Smutnicki, C.A., 2013. Parallel tabu search algorithm for the hybrid flow shop problem. Comput. Ind. Eng. 65 (3), 466–474.
Carlier, J., Néron, E., 2000. An exact method for solving the multi-processor flow-shop. Rairo-Oper. Res. 34 (1), 1–25.
Chang, S.C., Liao, D.Y., 1994. Scheduling flexible flow shops with no setup effects. IEEE Trans. Robot. Autom. 10 (2), 112–122.
Che, A., Zeng, Y., Lyu, K., 2016. An efficient greedy insertion heuristic for energy-conscious single machine scheduling problem under time-of-use electricity tariffs. J. Clean. Prod. 129, 565–577.
Chou, F.D., 2013. Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks. Int. J. Prod. Econ. 141 (1), 137–145.
Chung, T.P., Liao, C.J., 2013. An immunoglobulin-based artificial immune system for solving the hybrid flow shop problem. Appl. Soft Comput. 13 (8), 3729–3736.
Dai, M., Tang, D., Giret, A., Salido, M.A., Li, W.D., 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robot. Cim-Int. Manuf 29, 418–429.
Deb, K., Jain, H., 2014. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. IEEE Trans. Evol. Comput. 18 (4), 577–601.
Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.A.M., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6 (2), 182–197.
Ding, J.Y., Song, S., Wu, C., 2016a. Carbon-efficient scheduling of flow shops by multi-objective optimization. Eur. J. Oper. Res. 248, 758–771.
Ding, J.Y., Song, S., Zhang, R., Wu, C., 2016b. Parallel machine scheduling under time-of-use electricity prices: new models and optimization approaches. IEEE Trans. Autom. Sci. Eng. 13, 1138–1154.
Engin, O., Ceran, G., Yilmaz, M.K., 2011. An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Appl. Soft Comput. 11 (3), 3056–3065.
Fu, J., Faust, J., Chachuat, B., Mitsos, A., 2015. Local optimization of dynamic programs with guaranteed satisfaction of path constraints. Automatica 62, 184–192.
Gahm, C., Denz, F., Dirr, M., Tuma, A., 2016. Energy-efficient scheduling in manufacturing companies: a review and research framework. Eur. J. Oper. Res. 248 (3), 744–757.
Gupta, J.N.D., 1988. Two-stage, hybrid flow shop scheduling problem. J. Oper. Res. Soc. 39, 359–364.
Huang, R.H., Yang, C.L., Hsu, C.T., 2015. Multi-objective two-stage multiprocessor flow shop scheduling—a subgroup particle swarm optimisation approach. Int. J. Syst. Sci. 46 (16), 3010–3018.
Jiang, S., Liu, M., Hao, J., Qian, W., 2015. A bi-layer optimization approach for a hybrid flow shop scheduling problem involving controllable processing times in the steelmaking industry. Comput. Ind. Eng. 87, 518–531.
Lei, D., Guo, X., 2016. Hybrid flow shop scheduling with not-all-machines options via local search with controlled deterioration. Comput. Oper. Res. 65, 76–82.
Li, J., Pan, Q., 2015. Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. Inf. Sci. 316, 487–502.
Li, J., Pan, Q., Wang, F., 2014a. A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem. Appl. Soft Comput. 24, 63–77.
Li, J.Q., Pan, Q.K., Tasgetiren, M.F., 2014b. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. Appl. Math. Model. 38 (3), 1111–1132.
Li, D., Meng, X., Liang, Q., Zhao, J., 2015a. A heuristic-search genetic algorithm for multi-stage hybrid flow shop scheduling with single processing machines and batch processing machines. J. Intell. Manuf. 26 (5), 873–890.
Li, K., Deb, K., Zhang, Q., Kwong, S., 2015b. An evolutionary many-objective optimization algorithm based on dominance and decomposition. IEEE Trans. Evol. Comput. 19 (5), 694–716.
Li, J., Pan, Q., Duan, P., 2016a. An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping. IEEE. T. Cyber. 46 (6), 1311–1324.
Li, J.Q., Pan, Q.K., Mao, K.A., 2016b. Hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems. IEEE Trans. Autom. Sci. Eng. 13 (2), 932–949.
Liao, C.J., Tjandradjaja, E., Chung, T.P., 2012. An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. Appl. Soft Comput. 12 (6), 1755–1764.
Lin, H.T., Liao, C.J., 2003. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. Int. J. Prod. Econ. 86 (2), 133–143.
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.
Luo, H., Du, B., Huang, G.Q., Chen, H., Li, X., 2013. Hybrid flow shop scheduling considering machine electricity consumption cost. Int. J. Prod. Econ. 146, 423–439.
Marichelvam, M.K., Prabaharan, T., Yang, X.S., Geetha, M., 2013. Solving hybrid flow shop scheduling problems using bat algorithm. Int. J. Logist. Econ. Glob. 5 (1), 15–29.
Marichelvam, M.K., Prabaharan, T., Yang, X.S., 2014a. A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. IEEE Trans. Evol. Comput. 18 (2), 301–305.
Marichelvam, M.K., Prabaharan, T., Yang, X.S., 2014b. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. Appl. Soft Comput. 19, 93–101.
Moradinasab, N., Shafaei, R., Rabiee, M., Ramezani, P., 2013. No-wait two stage hybrid flow shop scheduling with genetic and adaptive imperialist competitive algorithms. J. Exp. Theor. Artif. Intell. 25 (2), 207–225.
Nawaz, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11 (1), 91–95.
Oğuz, C., Ercan, M.F., 2005. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. J. Sched. 8 (4), 323–351.
Pan, Q.K., 2016. An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling. Eur. J. Oper. Res. 250 (3), 702–714.
Pan, Q.K., Wang, L., Qian, B., 2009. A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. Comput. Oper. Res. 36, 2498–2511.
Pan, Q.K., Tasgetiren, M.F., Suganthan, P.N., Chua, T.J., 2011. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf. Sci. 181, 2455–2468.
Pan, Q.K., Wang, L., Li, J.Q., Duan, J.H., 2014. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. Omega-Int. J. Manage. S. 45, 42–56.
Portmann, M.C., 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.
Qin, W., Zhang, J., Song, D., 2015. An improved ant colony algorithm for dynamic hybrid flow shop scheduling with uncertain processing time. J. Intell. Manuf. https://doi.org/10.1007/s10845-015-1144-3.
Riane, F., Artiba, A., Elmaghraby, S.E., 1998. A hybrid three-stage flowshop problem: efficient heuristics to minimize makespan. Eur. J. Oper. Res. 109 (2), 321–329.
Ribas, I., Leisten, R., Framiñan, J.M., 2010. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Comput. Oper. Res. 37 (8), 1439–1454.
Ruiz, R., Vázquez Rodríguez, J.A., 2010. The hybrid flow shop scheduling problem. Eur. J. Oper. Res. 205, 1–18.
Shahvari, O., Logendran, R., 2016. Hybrid flow shop batching and scheduling with a bi-criteria objective. Int. J. Prod. Econ. 179, 239–258.
Sukkerd, W., Wuttipornpun, T., 2016. Hybrid genetic algorithm and tabu search for finite capacity material requirement planning system in flexible flow shop with assembly operations. Comput. Ind. Eng. 97, 157–169.
Tang, D., Dai, M., Salido, M.A., Giret, A., 2016. Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. Comput. Ind. 81, 82–95.
Tran, T.H., Ng, K.M., 2013. A hybrid water flow algorithm for multi-objective flexible flow shop scheduling problems. Eng. Optim. 45 (4), 483–502.
Wang, S., Liu, M., 2014. Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method. Int. J. Prod. Res. 52 (5), 1495–1508.
Wang, S.Y., Wang, L., Liu, M., Xu, Y., 2015. An order-based estimation of distribution

algorithm for stochastic hybrid flow-shop scheduling problem. Int. J. Comput. Integrated Manuf. 28 (3), 307–320.
Ying, K.C., Lin, S.W., Wan, S.Y., 2014. Bi-objective reentrant hybrid flowshop scheduling: an iterated Pareto greedy algorithm. Int. J. Prod. Res. 52 (19), 5735–5747.
Zhang, R., Chiong, R., 2016. Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. J. Clean. Prod. 112, 3361–3375.
Zhang, Q., Li, H., 2007. MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. 11 (6), 712–731.
Zhang, H., Zhao, F., Fang, K., Sutherland, J.W., 2014. Energy-conscious flow shop scheduling under time-of-use electricity tariffs. CIRP Ann. - Manuf. Technol. 63, 37–40.