Applied Soft Computing 55 (2017) 226–237

Contents lists available at ScienceDirect

Applied Soft Computing

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

Bi-criteria ant colony optimization algorithm for minimizing makespan and energy consumption on parallel batch machines

Zhao-hong Jia [a,b], Yu-lan Zhang [b], Joseph Y.-T. Leung [c,d,*], Kai Li [c]

[a] Key Lab of Intelligent Computing and Signal Processing of Ministry of Education, PR China
[b] School of Computer Science and Technology, Anhui University, Hefei, Anhui 230039, PR China
[c] School of Management, Hefei University of Technology, Hefei, Anhui 230009, PR China
[d] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA

ARTICLE INFO

Article history:
Received 17 November 2016
Received in revised form
23 December 2016
Accepted 26 January 2017
Available online 2 February 2017

Keywords:
Parallel batch machines
Ant colony optimization algorithm
Makespan
Energy consumption
Green manufacturing

ABSTRACT

We investigate the problem of minimizing the makespan and the total electric power cost simultaneously on a set of parallel identical batch-processing machines, where the jobs with non-identical sizes dynamically arrive. To address the bi-criteria problem, a Pareto-based ant colony optimization (PACO) algorithm is proposed. Depending on whether the current batch being delayed after the job is added into, two candidate lists are constructed to narrow the search space. Moreover, heuristic information is designed for each candidate list to guide the search. In addition, the objective-oriented local optimization methods are applied to improve the solution quality. Finally, the proposed algorithm is compared with existing multi-objective algorithms through extensive simulation experiments. The experimental results indicate that the proposed algorithm outperforms all of the compared algorithms, especially for large-scale problems.

1. Introduction

Scheduling on batch processing machines (BPMs) [1–3], extended from classical scheduling, has a strong application background, such as casting industry, furniture manufacturing industry, metal industry, aeronautical industry, pharmaceutical industry, and logistics freight [4]. Parallel batch (p-batch) scheduling problem originates from the chip production of semiconductor manufacturing process, specifically, the final stage of chip testing. P-batch scheduling differs from classical scheduling in that more than one job can be processed on a machine simultaneously. The processing time of a batch, consisting of several jobs, equals to the longest processing time of all the jobs in the batch. Generally, the objectives of p-batch scheduling problems are related to the processing times.

Recently, the issues of environmental protection and energy conservation in manufacturing industry have been paid more attention. Improving the utilization efficiency of resources and energy has become critical for sustainable development of modern industrial companies. Therefore, the United Nations Environment Programme proposed the theory of Green Economy (GE). Green manufacturing, one of the important components of GE, refers to reducing pollution and saving energy consumption during the production process. A typical example related to green scheduling is aluminum manufacturing [5,6]. In general, there are several furnaces being used to heat the aluminum ingots, and more than one ingot can be placed in one furnace, which corresponds to a batch processing machine. When idle, the furnaces are not completely closed, resulting in additional energy consumption. Moreover, the electricity price varies over several periods in most industrial areas. Hence, it is greatly significant to ensure the timely completion of the production with the lowest electricity cost. However, most studies on batch scheduling problems take the completion time or tardiness into consideration, instead of objectives related to energy efficiency.

In this paper, to address the problem of minimizing both the makespan and the total electric power cost (EPC) simultaneously, a Pareto-based ant colony optimization (PACO) algorithm is proposed and compared with three other multi-objective optimization algorithms.

The rest of this paper is organized as follows: Section 2 introduces the related work of batch scheduling problems. Section 3 illustrates the problem under study and the mixed integer mathematical model. Section 4 presents the proposed multi-objective

* Corresponding author at: Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA.
E-mail addresses: zhjia@mail.ustc.edu.cn (Z.-h. Jia), 15256090482@163.com (Y.-l. Zhang), leung@njit.edu (J.Y.-T. Leung), hfutlk@139.com (K. Li).

http://dx.doi.org/10.1016/j.asoc.2017.01.044
1568-4946/

ACO algorithm. Section 5 provides the comparative experiments and the results under seven experimental metrics. The paper is concluded in Section 6.

2. Literature review

Due to its practical significance, more and more researchers are concerned with the batch scheduling problems (BSP) with high computational complexity. Ikura and Gimple [1] first studied the simplest BSP model of a single BPM. They proposed an effective algorithm to find the feasible solution for minimizing the makespan with suitable release times and delivery times. Lee et al. [2] presented several dynamic programming algorithms to minimize the maximum lateness and the maximum tardiness and a polynomial time dynamic programming algorithm to minimize the number of tardy jobs of a single BPM according to the characteristics of semiconductor chip manufacturing process. Chandru et al. [7] presented an optimal branch and bound algorithm and some heuristic algorithms to minimize the total completion time on a single BPM with identical job size. Ghazvini and Dupont [8] proposed two heuristics for the problem of minimizing total completion time with non-identical job sizes on a single machine, where one of the two shows better performance. Uzsoy [9] also considered the batch scheduling with non-identical job sizes. He proved that minimizing the total completion time and the makespan on a single BPM are NP-hard and presented several heuristics. Uzsoy and Yang [10] gave some effective heuristics to minimize the total weighted completion time for a single BPM. Besides the heuristic and the dynamic programming algorithms, some researchers applied meta-heuristic algorithms for a single batch-processing machine with non-identical job sizes. Melouk et al. [11] presented a simulated annealing (SA) algorithm to minimize the makespan, which outperforms the CPLEX solver. To minimize the makespan on a single BPM, Chou et al. [12] proposed two different hybrid genetic algorithm (GA), whose performance are satisfactory.

The BSP on the parallel machines (BSPP) is an extension of the BSP on a single machine. Chandru et al. [7] provided several heuristics to deal with the BSPP for the total completion time minimization, denoted by $\sum C_j$. Koh et al. [13] proposed a number of heuristic algorithms and a GA to minimize $C_{\max}$, $\sum C_j$ and $\sum W_j * C_j$ of the BSPP with incompatible job families. Chang et al. [14] considered the BSPP with non-identical job sizes and presented a SA algorithm and compared it with the CPLEX solver. The SA shows better performance in terms of both the solution quality and the computation time. After that, to minimize the makespan of the BSPP, Damodaran et al. [15] provided several heuristic algorithms that outperformed the mixed-integer solver in the comparative experiments.

Recently, researchers pay more attention on the BSP with dynamic job arrivals. Lee and Uzsoy [16] investigated the problem with dynamic job arrivals and provided polynomial and pseudo-polynomial algorithms for several special cases. In addition, they gave an efficient heuristic for the general case. The simulated results verified the excellent average performance of the proposed heuristic. Damodaran et al. [17,18] provided a greedy randomized adaptive search procedure and several constructive heuristics to address the BSPP problem with arbitrary job arrivals. Li et al. [19] presented a polynomial-time approximation scheme (PTAS) to solve the BSPP with arbitrary job arrivals. Chung et al. [20] proposed a mixed integer linear programming model and presented three heuristics to minimize the makespan for the BSPP problem with non-identical ready time. Wang and Chou [21] constructed a mixed integer mathematical model and proposed two meta-heuristic algorithms by combining SA and GA with dynamic programming for the BSPP problem with different release times. Zhang et al. [22] proposed a hybrid local search algorithm with path relinking to minimize the makespan on a single BPM with different job sizes, processing times and job arrivals. Arroyo and Leung [23] proposed several heuristics to minimize the makespan on unrelated BSPP with non-identical job sizes and unequal ready times.

To be close to the real situation, two objectives, rather than one, are considered. Kashan et al. [24] optimized the two objectives of makespan and the maximum tardiness on a single BPM with non-identical sizes simultaneously. The proposed multi-objective GA shows good performance on both the quality and the diversity of the solutions. Xu et al. [25] provided an ant colony system (ASC) to minimize the makespan and the maximum tardiness on parallel BPMs. With an effective solution construction mechanism, the proposed ASC exhibits better performance than the compared algorithms. Wang et al. [26] proposed an exact ϵ-constraint and two heuristic algorithms to minimize the makespan and the total energy costs on a single BPM with non-identical job sizes, the time-of-use electricity prices, and different energy consumption rates of the machines. Nevertheless, the investigation of multi-objective BPM problems is limited according to the literatures that we have obtained.

For a detailed view on BPM problems, the readers are referred to [27,28].

3. Problem description

The studied problem can be denoted as $P_m \mid p\text{-}batch,\ r_j,\ p_j,\ s_j,\ C \mid (C_{\max}, EPC)$. Assume a job set J with n jobs is to be processed on the machine set M with m parallel BPMs. Each job J_j ($j = 1, 2, \ldots, n$) in J is associated with a release time r_j, a processing time p_j and a job size s_j. Each machine M_i ($i = 1, 2, \ldots, m$) has a capacity denoted by C. Since the machines are generally not closed during processing, each machine has two states in the process, namely, the processing state and the idle state, where the processing power and idle power of machine M_i are denoted by PW_i and SW_i, respectively. All machines are available at time zero and any job must be completed without interruption. The jobs in the same batch are processed simultaneously. Let B denote the batch set and the kth batch scheduled on M_i is denoted by B_{ki}. The release time of B_{ki}, denoted by R_{ki}, is the largest arrival time of the jobs in B_{ki}, i.e., $R_{ki} = \max\{r_j \mid J_j \in B_{ki}\}$. The processing time of B_{ki}, denoted by PT_{ki}, is the longest processing time among the jobs in B_{ki}, i.e., $PT_{ki} = \max\{p_j \mid J_j \in B_{ki}\}$. The size of B_{ki}, denoted by S_{ki}, is the total size of all the jobs in B_{ki}, i.e., $S_{ki} = \sum_{J_j \in B_{ki}} s_j$. The size of each batch cannot exceed C, i.e., $S_{ki} \le C$.

After a feasible solution σ is constructed, the starting time and completion time of B_{ki} on machine M_i are determined, denoted by ST_{ki} and CT_{ki} respectively, and defined as $ST_{ki} = \max\{R_{ki}, CT_{(k-1)i}\}$ and $CT_{ki} = ST_{ki} + PT_{ki}$, where $B_{(k-1)i}$ is the batch executed immediately before B_{ki} and $CT_{0i} = 0$. The first optimization objective $C_{\max}$ is the maximum completion time among all batches in σ, i.e., $C_{\max} = \max_{B_{ki} \in \sigma}\{CT_{ki}\}$. The second objective, i.e., the total electricity cost, is denoted by EPC. The energy consumption at time t, denoted by $f(t)$, is the sum of the processing power and idle power of all the machines from time 0 to t.

The decision variables Y_{ki}, X_{jki} and z_i^t are defined as:

$$Y_{ki} = \begin{cases} 1, & \text{if batch } B_{ki} \text{ is created for machine } M_i \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$X_{jki} = \begin{cases} 1, & \text{if job } J_j \text{ is grouped into } B_{ki} \text{ on } M_i \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

$$z_i^t = \begin{cases} 1, & \text{if there is a batch processing on } M_i \text{ at time } t \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Based on the above descriptions, the mathematical model of the problem can be formulated as follows:

$$\text{Minimize } C_{\max} \quad (4)$$

$$\text{Minimize } EPC = \int_0^{C_{\max}} f(t) \sum_{i=1}^{m} (PW_i * z_i{}^t + SW_i * (1 - z_i{}^t))dt \quad (5)$$

$$\text{s.t. } X_{jki} \le Y_{ki} \quad i = 1, \ldots, m; \quad j = 1, \ldots, n; \quad k = 1, \ldots, n \quad (6)$$

$$\sum_{i=1}^{m} \sum_{k=1}^{n} X_{jki} = 1 \quad j = 1, \ldots, n \quad (7)$$

$$\sum_{j=1}^{n} s_j X_{jki} \le C \quad i = 1, \ldots, m; \quad k = 1, \ldots, n \quad (8)$$

$$PT_{ki} \ge p_j X_{jki} \quad i = 1, \ldots, m; \quad j = 1, \ldots, n; \quad k = 1, \ldots, n \quad (9)$$

$$ST_{ki} \ge r_j X_{jki} \quad i = 1, \ldots, m; \quad j = 1, \ldots, n; \quad k = 1, \ldots, n \quad (10)$$

$$ST_{ki} \ge ST_{(k-1)i} Y_{(k-1)i} + PT_{(k-1)i} Y_{(k-1)i}$$
$$i = 1, \ldots, m; \quad k = 2, \ldots, n \quad (11)$$

$$CT_{ki} = ST_{ki} Y_{ki} + PT_{ki} Y_{ki} \quad i = 1, \ldots, m; \quad k = 1, \ldots, n \quad (12)$$

$$C_{\max} \ge CT_{ki} \quad i = 1, \ldots, m; \quad k = 1, \ldots, n \quad (13)$$

$$Y_{ki}, X_{jki}, z_i^t \in \{0, 1\} \quad i = 1, \ldots, m; \quad j = 1, \ldots, n; \quad k = 1, \ldots, n \quad (14)$$

$$z_i^t \in \{0, 1\} \quad 0 \le t \le C_{\max}, \quad i = 1, \ldots, m \quad (15)$$

Objective (4) is to minimize the makespan. Objective (5) is to minimize the *EPC* and $f(t)$, denotes the electricity price function. Constraint (6) ensures that the jobs can be assigned into a batch only when the batch is created. Constraint (7) ensures that each job can be assigned to only one batch on one machine. Constraint (8) ensures that the total size of all jobs in one batch does not exceed the machine capacity. Constraint (9) denotes that the processing time of batch B_{ki} is the longest time among all the jobs in B_{ki}. Constraint (10) denotes that the starting time of batch B_{ki} is at least the largest arriving time among all the jobs in B_{bi}. Constraint (11) ensures that the processing of the batches cannot be interrupted, and a batch cannot start until the completion of the previous batch on the same machine. Constraint (12) defines the completion time of B_{ki}. Constraint (13) shows the completion time of each batch B_{ki} is not more than the makespan. Constraints (14) and (15) define the binary variables.

4. Pareto-based ant colony optimization algorithm

The proposed PACO is a bi-objective ant colony optimization algorithm. In PACO, the preferences for the two objectives are utilized to control the search directions of the ants in the solution space. The problem is generally able to be solved in two stages, forming the batches and scheduling the batches. To improve the optimization efficiency, PACO combines the solution of the two subproblems together. That is, a feasible solution is directly constructed by assigning the appropriate jobs, one by one, to the current batch on one machine. The details of the PACO are described in Section 4.2.

4.1. Ant colony optimization

ACO is a meta-heuristic algorithm for complex discrete optimization problems. First proposed in 1991, the ACO algorithm was successfully applied to the TSP problem [29]. It has drawn extensive attention from the researchers so that a number of ACO-based algorithms have been proposed by adopting different search strategies. Bullnheimer et al. [30] proposed a new rank based version of the ant system with better worst case behavior. Stützle and Hoos [31] presented the max-min ant system (MMAS). By limiting the value of the pheromone trails, the MMAS shows a promising performance in preventing premature convergence. Pilat and White [32] provided an improved ACO algorithm combined with GA.

In recent years, ACO algorithms have been increasingly used to solve scheduling problems. Kashan and Karimi [33] gave an ant colony framework to minimize the total weighted completion time on a single BPM (SBPM) with incompatible job families. The proposed algorithm outperforms the existing heuristics and meta-heuristics with increasing problem sizes. Cheng et al. [34] applied ACO to the makespan minimization of the SBPM problem with unequal-size jobs under fuzzy environment. The proposed ACO algorithm outperforms the other two meta-heuristics based on GA [35] and SA [11], respectively, according to the simulated experimental results. Xu et al. [36] presented an ACO algorithm to minimize the makespan of the SBPM problem with different job release times. The proposed algorithm is superior to CPLEX and GA according to the simulation results. Cheng et al. [37] proposed an ACO to minimize the total cost of production and distribution for the manufacturers. Jia et al. [38] proposed a meta-heuristic based on the MMAS to minimize the makespan on BSPP with non-identical job sizes and incompatible job families.

With the development of the real-world applications, ACO has been extended to deal with multi-objective optimization problems. Yagmahan and Yenisey [39] utilized ACO algorithm to solve the flow shop scheduling problem. The proposed algorithm is more effective and better than existing heuristics. Li et al. [40] proposed an ACO algorithm to minimize the makespan and total weighted tardiness simultaneously on the parallel BPMs with incompatible job families, dynamic job arrivals, and sequence-dependent setup times. Du et al. [41] proposed an improved ACO algorithm to minimize the makespan and the energy consumption in a hybrid flow shop. The proposed ACO incorporates users' preference into the process of the solution construction to narrow the search and magnify certain Pareto frontier where the users are interested in.

However, few studies have applied the ACO algorithms to minimizing the makespan and the cost simultaneously on BPMs with different job release times. When solving the multi-objective problems, from a given initial point, each ant starts choosing the next unscheduled jobs based on the decision rule, and then update the corresponding pheromone trails. The search of the ants is guided by both the pheromone trails and heuristic information associated with the studied problem. Once the construction of a solution is completed, a local optimization algorithm is applied to improve it. In each iteration, after all the ants have generated their solutions, we update the Pareto-optimal set. The pheromone trails are then updated with the solutions in the new Pareto-optimal set. However, different ACO algorithms employ different strategies to choose the nodes and update the pheromone trails.

4.2. Solution construction

4.2.1. Encoding

Encoding is the first step of ACO algorithms. To reduce the solution search space, a common way is to employ a meta-heuristic

algorithm to form the batches first, then use a heuristic algorithm to schedule the batches on the BPMs. This strategy is called "batch first, schedule second" [18]. Different from this, we adopt a new construction method, which obtains the batches and the machine information at the same time after one job is assigned to a batch. Let B_b denote an arbitrary batch in B. Table 1 gives an example of the encoding of a solution with nine jobs. The solution is encoded by a 2×9 vector. Each column j corresponds to the assignment of job J_j. The numbers in the first row and second row are the batch indexes and machine indexes of the corresponding job. In this mechanism, once an ant adds job J_j to batch B_b on machine M_i, the assignment of job J_j is confirmed. A complete solution is constructed when the locations of n jobs are determined.

4.2.2. Pheromone trails

As mentioned above, there are two objectives to be optimized simultaneously, i.e., the makespan and the EPC. Instead of a single pheromone matrix, an individual pheromone trail matrix is built for each objective in PACO, so that the impact between different objectives can be distinguished. We define the pheromone trails τ_{hj} as the desirability of grouping job J_h in a batch with job J_j [36]. In the process of constructing the solutions, the pheromone between batch B_b and job J_j in the candidate list for the oth objective τ^o_{bj}, is defined to be the average pheromone value of the total pheromone of the candidate job and the jobs grouped in the current batch B_b.

$$\tau^o_{bj} = \frac{\sum_{J_h \in B_b} \tau^o_{hj}}{\mid B_b \mid} (J_j \in CL_o), \tag{16}$$

where $o \in \{1, 2\}$ denotes the objective index. That is, τ^1_{hj} is the pheromone for $C_{\max}$, and τ^2_{hj} is the pheromone for EPC. $|B_b|$ is the number of jobs in the current batch B_b, h is the job index in B_b, and J_j is the job in the candidate list CL_o.

4.2.3. Candidate lists

Considering the NP-hardness of the problem, we use the hierarchical candidate lists to reduce the search space and improve the search ability effectively. Two different candidate lists CL_1 and CL_2 are proposed according to whether the current batch will be delayed when the candidate job is inserted. ST_b denotes the starting time of current batch B_b. During the construction of solutions, the jobs in candidate list CL_1 are considered first. If CL_1 is empty, then CL_2 are generated. CL_1 is constructed as follows:

$$CL_1 = \{J_j \mid s_j \le C - S_b \wedge r_j \le ST_b\}. \tag{17}$$

The size of job J_j in CL_1 has to be no more than the residual space of the current batch B_b. Additionally, the release time of job J_j does not exceed the starting time of B_b, which avoids batch B_b being delayed. CL_2 is constructed as in Eq. (18).

$$CL_2 = \{J_j \mid s_j \le C - S_b \wedge r_j > ST_b\}. \tag{18}$$

Each job in CL_2 must satisfy the machine capacity constraint and the release time of each one being more than the starting time of batch B_b.

4.2.4. Heuristic information

Heuristic information is designed according to the specific problem. It is difficult to present the heuristic information for the EPC since the factors affecting the EPC are complex. Due to the EPC being directly associated with $C_{\max}$, we only define the heuristic information for $C_{\max}$ based on different candidate lists. The release time of each job in candidate list CL_1 is less than the starting time of batch B_b, thus only the processing times of jobs in CL_1 are left for consideration. The heuristic information based on CL_1, η^1_{bj}, is defined as follows:

$$\eta^1_{bj} = \frac{1}{\mid PT_b - p_j \mid + 1}. \tag{19}$$

where PT_b is the processing time of current batch B_b under construction. The candidate list CL_2 is constructed when CL_1 is empty. When selecting the jobs in CL_2, both the processing times and the release times related to the jobs should be considered. Thus, for the jobs in CL_2, the heuristic information η^2_{bj} is formulated as follows:

$$\eta^2_{bj} = \frac{1}{\mid PT_b - p_j \mid + 1} \cdot \frac{1}{r_j - ST_b}. \tag{20}$$

Therefore, the jobs with smaller release times have higher priority to be selected than those arrive later. In this way, the delay will be kept as small as possible.

4.2.5. Decision rule

PACO simplifies the solution by merging the batch formation and the scheduling of batches into one step. During the solution construction, each ant chooses a machine to generate a new batch as follows:

$$\arg\min_{M_i \in M} \left\{ \sum_{o=1}^{2} V^o * f^o_i \right\}, \tag{21}$$

where f^1_i is the value of the completion time of machine M_i and f^2_i is the value of the EPC of M_i. V^1 and V^2 are the weights of f^1_i and f^2_i, and $V^1 + V^2 = 1$. V^1 is a real number generated by the uniform distribution within [0.8, 1], which is determined by the results of the preliminary experiments.

When the current batch B_b is empty, each ant chooses an unscheduled job randomly to increase the diversity of feasible solutions. If B_b is not empty, each ant selects a job from CL_1 or CL_2 and adds it into B_b according to the probability formulated as follows:

$$P_{bj} = \begin{cases} \dfrac{(\sum_{o=1}^{2} \tau^o_{bj} \nu_o)^{\alpha} (\eta^1_{bj})^{\beta}}{\sum_{J_x \in CL_1} (\sum_{o=1}^{2} \tau^o_{bx} \nu_o)^{\alpha} (\eta^1_{bx})^{\beta}}, & \text{if } J_j \in CL_1 \\ \dfrac{(\sum_{o=1}^{2} \tau^o_{bj} \nu_o)^{\alpha} (\eta^2_{bj})^{\beta}}{\sum_{J_x \in CL_2} (\sum_{o=1}^{2} \tau^o_{bx} \nu_o)^{\alpha} (\eta^2_{bx})^{\beta}}, & \text{if } J_j \in CL_2 \\ 0, & \text{otherwise.} \end{cases} \tag{22}$$

where ν_o, a real number in (0,1), refers to the user's preference to the oth objective, and $\nu_1 + \nu_2 = 1$. The probability distribution is influenced by α and β, determining the relative importance of pheromone trails and heuristic information.

4.3. Update of pheromone trails

There are two ways to update the pheromone trails: the local update and the global update. The local pheromone update is performed on the pheromone matrix of each objective once an ant adds a job into a non-empty batch, which reduces the probability of selecting the path ever been accessed so as to strengthen the search in unknown solution space. The local update is formulated as:

$$\tau^o_{hj} = (1 - \rho_l) \cdot \tau^o_{hj} + \rho_l \cdot \tau_0, \tag{23}$$

where τ_0 is the initial value of pheromone trails and $\rho_l \in (0, 1)$ is a parameter to control the speed of pheromone evaporation.

Table 1
An example of solution encoding in PACO.

J_j	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9
b	2	1	3	5	4	2	3	5	1
i	3	4	2	4	1	3	2	4	4

Table 2
Parameters of the batches in the initial solution σ_0.

B_b	B_1	B_2	B_3	B_4	B_5
R_b	5	2	3	3	12
PT_b	4	5	4	7	5

The global pheromone updating is performed by using the non-dominated solutions (*NDS*) to update the pheromone trails for each objective after all ants have constructed the solutions. The rule is given as follows:

$$\tau^o_{hj} = (1-\rho_g)\cdot\tau^o_{hj} + \rho_g \cdot \sum\nolimits_{sol\in NDS} \Delta\tau^o_{hj}, \tag{24}$$

$$\Delta\tau^o_{hj} = \begin{cases} \dfrac{1}{f^o_{hj}} & \text{if jobs } J_h \text{ and } J_j \text{ are in the same batch;} \\ 0, & \text{otherwise.} \end{cases} \tag{25}$$

where f^o_{hj} is the value of the oth objective function, $\rho_g \in (0, 1)$ is a parameter.

4.4. Local optimization strategies

After each solution is completed, we call the local optimization algorithms for the two objective respectively to improve the solution.

4.4.1. Local optimization for the makespan

In PACO, each ant chooses the first job randomly and groups it into a batch. However, if a batch with larger release time is processed earlier, the completion time of the machine will definitely increase, as well as the value of $C_{\max}$. Hence, we propose a local optimization algorithm for $C_{\max}$, called LOM algorithm. For a generated solution, we sort the batches on each machine in ascending order of their release times. Let MB_i denote the batch set of M_i and $|MB_i|$ is the number of batches on M_i. Algorithm LOM is described in Algorithm 1.

Algorithm 1. The LOM Algorithm

```
for i = 1, 2, ..., m do
    sort the batches on M_i in ascending order of the release times of the batches
end for
for i = 1, 2, ..., m do
    for k = 1, 2, ..., |MB_i| do
        if (k==1)
            ST_ki = RT_ki
        else
            ST_ki = max{RT_ki, CT_(k-1)i}
        CT_ki = ST_ki + PT_ki
    end for
end for
```

Here is an example of the LOM. The parameters of the five batches of the solution σ_0 are shown in Table 2. The first row denotes the batches, the second and the third rows are the release times and the processing times of the corresponding batches. The scheduling of the batches in σ_0 is shown in Fig. 1. The improved solution adjusted by the LOM algorithm is shown in Fig. 2.

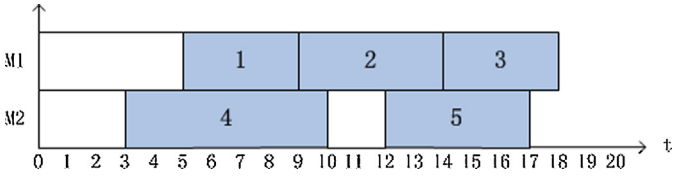

Fig. 1. Gantt chart of the initial solution σ_0.

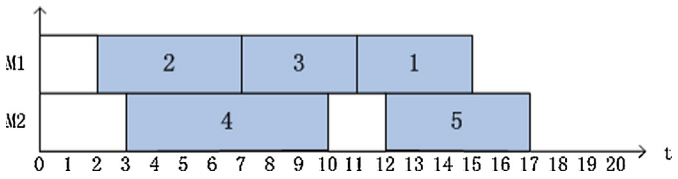

Fig. 2. Gantt chart of the solution σ_1 adjusted by the LOM.

Definition 1. Suppose there exists a feasible schedule σ and B_b is the current batch, ST_b is the starting time of B_b, R_b is the release time of B_b ($ST_b \geq R_b$). If ST_b and R_b satisfy the following condition, we say that B_b is processed on time; otherwise, it is delayed.

$$ST_b = R_b \quad \forall B_b \in B \tag{26}$$

According to Fig. 1, the original value of $C_{\max}$ is 18. On M_1, B_1 is processed on time, B_2 and B_3 are delayed by 7 and 11, respectively. Thus, the total delay of σ_0 is 18. Additionally, the release time of B_1 is the largest among all the batches but B_1 is processed first leading to both B_2 and B_3 being delayed. It is easy to observe that the total delay would be decreased if more batches are processed on time by scheduling as early as possible the batches that arrive earlier. Therefore, the algorithm LOM improves the initial solution σ_0 and generates a new solution σ_1, by replacing the original batch order $B_1 - B_2 - B_3$ by a new order $B_2 - B_3 - B_1$, as shown in Fig. 2. According to Fig. 2, B_2 is processed on time, B_3 and B_1 are delayed for 4 and 6, respectively. Thus, the total delay is 10. It is obvious that the total delay of the batches on M_1 is reduced by 8, and the completion time of M_1 decreases from 18 to 15. As for the batches on M_2, since they are all processed on time, the batch order is not adjusted. In all, compared to the initial solution σ_0, the value of $C_{\max}$ of σ_1 decreases from 18 to 17 by using LOM.

4.4.2. Local optimization for the EPC

Based on the fact that EPC is influenced by the electricity price as well as $C_{\max}$, we propose the second local optimization algorithm LOC to improve the EPC on the condition that the value of $C_{\max}$ is not increased. That is, LOC tries to move the processing of the batches from the period of high electricity price to that of low price. Algorithm LOC is described as Algorithm 2.

Algorithm 2. The LOC Algorithm

```
for i = 1, 2, ..., m do
    for k = |MB_i|, |MB_i| - 1, ..., 1 do
        if (k == |MB_i|)
            t_min = ST_ki
            t_max = max{C_max, CT_ki} - PT_ki
            ST_ki = arg min_{t_min ≤ t ≤ t_max} ∫_t^{t+PT_ki} f(q)·PW_i dq /* if there is more than one t
        that makes the value the same, then choose the maximal t. */
        else
            t_min = ST_ki
            t_max = ST_(k+1)i - PT_ki
            ST_ki = arg min_{t_min ≤ t ≤ t_max} ∫_t^{t+PT_ki} f(q)·PW_i dq /* if there is more than one t
        that makes the value the same, then choose the maximal t. */
        CT_ki = ST_ki + PT_ki /*update CT_ki*/
    end for
end for
```

The adopted electricity price function is a segmented periodic function of time t, illustrated in Fig. 3.

For solution σ_1 in Fig. 2, it can be observed that there is no space to make any objective value better by moving batch B_5 on M_2. However, B_4 is processed on the high-price period. Moreover, an idle period [10,12] between the completion of B_4 and the starting time of B_5 is of the low price. Therefore, B_4 can be delayed to reduce the electricity cost of M_2. For M_1, B_1 is completely processed

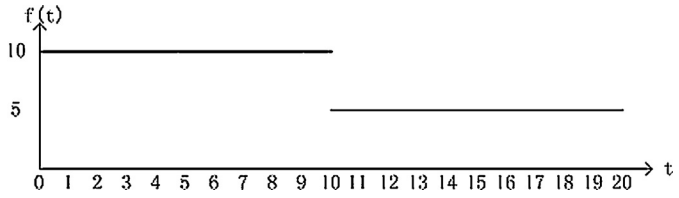

Fig. 3. The function of electricity price.

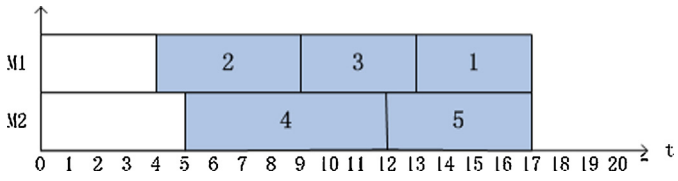

Fig. 4. The obtained solution $\sigma 2$ by LOC algorithm.

in the low-price period. B_2 is processed in the high-price period. During the processing period [7,10], B_3 is in high price and is in low price during [10,12]. There is also an idle period [15,17], the distance between the completion of B_1 and $C_{\max}$. Postponing the process of B_1 by two will not only reduce the electricity cost, but also reserve more space for adjusting the batches assigned before B_1. In this case, a free low-price period emerges between the completion time of B_3 and the starting time of B_1. Similarly, B_3 can be started from time nine, so that B_3 is mostly processed in the low-price period, which further reduces the electricity cost. Similar to B_1, B_2 is adjusted to begin processing from time four. Fig. 4 gives the scheduling, denoted by σ_2, after improvement by LOC. The total electricity cost of σ_2 is decreased from 1670 to 1530.

4.5. Description of the proposed algorithm

During the solution construction, a taboo list, denoted by *DE*, is used to record the processed states of the jobs. *DE* is a $1 \times n$ vector with initial value $(1, 2, 3, \ldots, n)$, corresponding to the job indexes. Once job $J_j (j = 1, \ldots, n)$ is assigned to a batch, then the value of the corresponding position in *DE* is changed to zero. Thus, $DE = (0, 0, \ldots, 0)$ means that all jobs have been grouped into batches. Additionally, a Pareto set, namely *DNS*, is used to store the non-dominated solutions during the search. *NDS* is initialized as $\emptyset$. The proposed algorithm, called PACO, is described as Algorithm 3.

Algorithm 3. The PACO Algorithm

1: Initialize the pheromone matrix; the number of jobs n; the number of machines m; the capacity of the machine C; the number of ants N_a; the maximum number of iterations $T_{\max}$; the initial value of pheromone τ_0; the evaporation rates ρ_l and ρ_g; the relative importance of pheromone trails and heuristics α and β;
2: $t = 1$;
3: if $t > T_{max}$, then output *NDS*;
4: $a = 1$;
5: generate the user's preference vector $V = (v_{mk}, v_{epc})$, randomly; initialize the taboo list $DE = (1, 2, 3, \ldots, n)$;
6: if $DE \neq (0, 0, \ldots, 0)$ then go to Step 7; else go to Step 10;
7: ant a chooses machine M_i according to Eq. (21); constructs a new empty batch B_b on machine M_i; chooses an unscheduled job at random and adds it into the current batch; updates *DE*;
8: ant a constructs the candidate lists CL_1 and CL_2;
9: if $CL_1 \neq \emptyset$, then ant a selects the next job with the largest probability according to Eq. (22) from CL_1, updates *DE* and the local pheromone according to Eq. (23) and go to Step 8; else if $CL_2 \neq \emptyset$, then ant a selects the next job with the largest probability according to Eq. (22) from CL_2, updates *DE* and the local pheromone according to Eq. (23) and go to Step 8; else go to Step 6;
10: call the algorithms LOM and LOC, respectively; update *NDS* after insert the generated solution into *NDS*;
11: a++; if $a > N_a$, update the global pheromone according to Eq. (24), then go to Step 12; else go to Step 5;
12: t++; go to Step 3.

The running time of the PACO algorithm mainly consists of five parts. That is, the initialization, the solution construction, the LOM algorithm, the LOC algorithm and the update of the global pheromone.

(1) The time complexity of the parameter initialization is determined by the initialization of the pheromone matrixes, i.e., $O(n^2)$.
(2) During the process of solution construction, the time complexity of initializing the taboo list and choosing the machine is $O(n)$ and $O(m)$, respectively. The time complexity of choosing one unscheduled job, constructing the candidate list and computing of the probability of jobs in candidate are all $O(n)$. The time complexity of updating the local pheromone is $O(1)$. Thus, the total running time of the above process is $O(n + mn + n^2)$. Since n is generally far more than m, the time complexity of solution construction is $O(n^2)$.
(3) In the LOM algorithm, the complexity of sorting the batches and computing the starting time and the completion time of the batches are $O(mn^2)$ and $O(mn)$, respectively. Thus, the complexity of LOM algorithm is $O(mn^2)$.
(4) The running time of the LOC algorithm is determined by recalculating the starting time of batches whose complexity is $O(mn)$. Hence the time complexity of LOC algorithm is $O(mn)$.
(5) The update of the global pheromone trails is $O(n^2)$.

The algorithms LOM and LOC will be executed after each ant completes its solution. Therefore, the time complexity of each iteration is $O(N_a mn^2)$. In all, the complexity of the proposed PACO algorithm is $O(T_{\max} N_a mn^2)$.

5. Computational experiments

To evaluate the performance of the proposed algorithm, we compare PACO with NSGA-II [42] and SPEA2 [43], two classical multi-objective optimization algorithms. Another compared algorithm is the ACO algorithm from [41], denoted by DACO, to minimize the makespan and the total energy consumption of the machines for the job shop scheduling problem. To extend the three algorithm to our problem and reduce the complexity of NSGA-II and SPEA2, we use the Best-Fit-LPT (BFLPT) rule to form batches first and then call the three algorithms to schedule the batches on the machines, respectively. In the DACO, once all the jobs are grouped into batches by the BFLPT, the batches are then sorted in non-descending order of their arrivals. The generated batch sequence is called the old sequence, namely *OBS*. The ant colony construct a new batch sequence, *NBS*, based on *OBS*. According to *NBS*, the batches are scheduled on the BPMs by the LS algorithm [41]. The sequence index of each batch $B_k 1$ in *OBS* is denoted by $POS(k1)$. The heuristic information $\eta_{k1,k2}$ is defined as follows:

$$\tau(k1, k2) = |B| - |POS(k1) - POS(k2)| \tag{27}$$

It means that the next batch whose release time is closer to that of the current batch is preferable to be selected. Similar to PACO, two pheromone trail matrixes are designed for the two objectives, separately, where $\tau^o_{k1,k2}, (k1, k2 = 1, \ldots, b)$ is defined as the the times that batch B_{k1} is scheduled before batch B_{k2} on the same machine for the oth objective. The initial value of $\tau_{k1,k2}$ is set to 1. Additionally, the users' preferences are also incorporated into generating *NBS* in the DACO. Moreover, the local optimization algorithm LOC

Table 3
Factors and levels.

Factors	Levels
n	N1 = 20, N2 = 50, N3 = 100
s_j	S1 = U[1,15], S2 = U[15,35]
p_j	U[8,48]
r_j	U[1,*LB*]
m	M1 = 2, M2 = 4
C	40
SW	1
PW	8

and LOM in PACO are both combined into DACO to ensure the fairness of the comparison.

5.1. Experimental settings

The random test instances were generated based on the method in [36]. To increase the diversity of the testing problem instances, we consider several factors with different levels, as shown in Table 3. Note that *LB* is calculated based on the method proposed in [18].

The electricity price function adopted here, as shown in Fig. 3, is formulated as follows:

$$f(t) = \begin{cases} 10, & 20d - 20 \le t < 20d - 10 \\ 5, & 20d - 10 \le t < 20d \end{cases} \quad (28)$$

where d is a natural number.

The detailed parameter setting of the four algorithms is shown in Table 4, where Q_a is the size of the archive set in SPEA2. The comparative experiments were programmed in C++ and executed on a PC with Intel Core 3 processor and 4G RAM.

5.2. Evaluation metrics

To measure the performance of the three multi-objective optimization algorithms, the following performance metrics are considered.

(1) Number of Pareto solutions metric (*NPS*): this metric presents the number of non-dominated solutions obtained by each algorithm.

(2) Coverage metric (C): this metric allows clear differentiation of two sets E and F [44]. The value of $C(E,F)$ presents the percentage of solutions in F dominated by at least one solution in E, which can be calculated by Eq. (29).

$$C(E,F) = \frac{|\{f \in F \mid \exists e \in E : e \succ f\}|}{|F|}. \quad (29)$$

The value of $C(E,F)$ is between 0 and 1. The closer the value of $C(E,F)$ to 1, the more solutions in F are dominated by the solutions in E, as well as E is more better than F. However, this metric is not symmetrical. That is, $C(E,F) = 1 - C(F,E)$ usually does not hold. Consequently, both the values of $C(E,F)$ and $C(F,E)$ need to be calculated.

(3) Hypervolume indicator (H): this metric describes the similarity between the non-dominated solution sets and the Pareto frontier [43]. This metric was proved to be Pareto compliant [45]. That is, for set E and set F, if E dominates F, the value of E, namely $H(E)$, must be more than that of F, namely $H(F)$. The bigger the value of $H(E)$, the closer E is to the Pareto frontier.

We selected a point $x^* = (x_{mk}, x_{epc})$, which is dominated by all the solutions of the four algorithms. It can be calculated as in Eq. (30) [46].

$$x_s = \max_s + \phi(\max_s - \min_s), \quad (30)$$

where $s \in \{mk, epc\}$, $\max_s$ and $\min_s$ are respectively the maximum and the minimum value of s among all the solutions found by the four algorithms; ϕ is set to be 0.1.

(4) Diversity (DVR): this metric represents the area covered by the solutions, which is calculated as follows:

$$DVR_\Omega = (\max_{\xi \in \Omega} C^\xi_{\max} - \min_{\xi \in \Omega} C^\xi_{\max}) \times (\max_{\xi \in \Omega} EPC^\xi - \min_{\xi \in \Omega} EPC^\xi) \quad (31)$$

(5) Spacing (SPC): this metric measures the spread of the solutions along the Pareto frontier, which is computed as follows:

$$SPC_\Omega = \frac{\left[1/|\Omega| \sum_{\xi \in \Omega} (d_\xi - \bar{d})^2\right]^{1/2}}{\bar{d}}, \quad (32)$$

where d_ξ is the Euclidean distance between solution ξ and its closest neighbor in the Pareto frontier Ω, $\bar{d} = \frac{1}{\Omega}\sum_{\xi \in \Omega} d_\xi$.

(6) Distance with the lower bound (DLB): this metric measures the solution quality of the algorithms. That is, how close the solutions on the obtained Pareto frontier are to the lower bound of the problem. The accuracy of the Pareto frontier Ω is measured by its distance to the lower bound, denoted by DLB_Ω, and formulated as follows:

$$DLB_\Omega = \frac{\sum_{\xi \in \Omega} \min\left\{\frac{C^\xi_{\max} - C^{LB}_{\max}}{C^{LB}_{\max}}, \frac{EPC^\xi - EPC^{LB}}{EPC^{LB}}\right\}}{|\Omega|}, \quad (33)$$

where $|\Omega|$ is the number of non-dominated solutions. $C^{LB}_{\max}$, the lower bound of $C_{\max}$ [47], is computed as follows:

Step 1. Let J^1 be the set of jobs in J that satisfy the following relation:

$$J^1 = \{J_j \in J \mid S - s_j < \min_{J_i \in J}\{s_i\}\}. \quad (34)$$

Each job in J^1 can be only assigned to one batch and will not accommodate any other job in the same batch. The release times of such a batch will be the release time of the job in that batch.

Step 2. Let J^2 include the jobs in J that do not belong to J^1 (*i. e.*, $J^2 = J \setminus J_1$). Divide each job J_j in J^2 into s_j jobs with unit size, each of which has the release time r_j and the processing

Table 4
Parameter settings.

PACO	DACO	NSGA-II	SPEA2
N_a :100(n=20)	N_a :100(n=20)	N_a :100(n=20)	N_a :100(n=20)
N_a :150(n=50)	N_a :150(n=50)	N_a :150(n=50)	N_a :150(n=50)
N_a :200(n=100)	N_a :200(n=100)	N_a :200(n=100)	N_a :200(n=100)
$T_{\max}$ =200	$T_{\max}$ =200	$T_{\max}$ =200	Q_a :50(n=20)
		Crossover probability 1.0	Q_a :80(n=50)
		Mutation probability 0.01	Q_a :100(n=100)
			$T_{\max}$ =200
			Crossover probability 1.0
			Mutation probability 0.01

time p_j. Sort these unit-size jobs in decreasing order of their processing times and group the first C unassigned jobs into an individual batch one by one until all the jobs are batched. Thus, the total number of the batches L can be calculated as follows:

$$L = \left\lceil \frac{\sum_{J_k \in J^2} S_k}{C} \right\rceil . \tag{35}$$

Step 3. A lower bound of the optimal makespan $C^{LB}_{\max}$ can be calculated as follows:

$$C^{LB}_{\max} = \max \left\{ \left\lceil \frac{\sum_{J_j \in J^1} p_j + \sum_{l=1}^{L} PT_l}{m} \right\rceil + \min_{J_j \in J}\{r_j\}, \max_{J_j \in J}\{r_j + p_j\} \right\}, \tag{36}$$

where m is the number of machines, C is the machine capacity. EPC^{LB} is calculated according to $C^{LB}_{\max}$ where all the jobs are processed in the low electricity price period.

(7) Run time (T): this metric is usually used to compare the time performance of the algorithms run in the same situation with the same population size and the same iteration number.

5.3. Experimental results

Table 5 presents the mean number of non-dominated solutions for each group of instances by using the four algorithms. The symbols in the first column, i.e., "Grp. Code", denote the group codes of the instances in the experiments corresponding to the levels in Table 3. Take the first group code in Table 5 for an example, "M1N1S1" denotes the group of ten random instances with 20 small-size jobs on two machines. Columns 2–4, 5–7, 8–10 and 11–13 denote the mean of the maximum "*MAX*", the minimum "*MIN*" and the average numbers "*AVG*" of the non-identical solutions obtained by PACO, DACO, NSGA-II and SPEA2, respectively. Specifically, for each instance group, we run each algorithm on each instance ten times, and calculate the maximum (minimum, or average) numbers of solutions of the ten runs first. Then, we compute the average maximum (minimum, or average) numbers of solutions over the ten instances in the same group. Note that in Table 5, the best average results of each groups, the largest value in each row, are shown in boldface. The more non-identical solutions, the better the performance of the multi-objective optimization algorithm is.

The top six rows of Table 5 show the results over small-size jobs, from which we can observe that PACO beats the other three algorithms on all the instances with small-size jobs except the minimum solution number for the group M1N2S1. The bottom six rows of Table 5 present the results on large-job instances. For large-job instances, the mean of the maximum and the minimum numbers of the solutions obtained by PACO are all more than those of the other three algorithms, except the maximum solution number for M2N1S2. Nevertheless, PACO finds the most average solutions on all the large-job instances among the four algorithms. Generally speaking, the more the jobs are, the more the non-dominated solutions are obtained by the algorithms. Moreover, the superiority of PACO to the other three algorithms on large-job instances is becoming obvious with the increasing number of jobs. In terms of the NPS metric, PACO is significantly superior to the other three algorithms and its average solution number could be up to 13.5 for the group M1N2S2.

Table 6 presents the mean coverage metric of the four algorithms. The final Pareto solution set of each algorithm is obtained by combining the solutions of ten runs for each instance. The first column is defined the same as that of Table 5. The numbers in each

Table 5
Comparison of the four algorithms using the NPS metric.

Grp. Code	PACO			DACO			NSGA-II			SPEA2		
	MAX	*MIN*	*AVG*	*MAX*	*MIN*	*AVG*	*MAX*	*MIN*	*AVG*	*MAX*	*MIN*	*AVG*
M1N1S1	**2.9**	**2.4**	**2.8**	2.7	2.3	2.5	1.6	1	1.24	2.1	1.4	1.7
M1N2S1	**9.2**	3.8	**6.72**	6.8	**5.3**	6	3.3	1	1.52	3.2	1	2.08
M1N3S1	**9.9**	**3.7**	**6.7**	6.6	3.2	4.7	3.6	1	1.82	4	1	1.8
M2N1S1	**2.4**	**2.3**	**2.31**	1	1	1	1	1	1	1	1	1
M2N2S1	**5.4**	**2**	**3.57**	4.2	1.9	3.2	2.2	1	1.46	2.4	1	1.51
M2N3S1	**9.1**	**3**	**5.97**	8.1	2.4	5.6	3	1	1.57	3.7	1	1.57
M1N1S2	**11**	**7.7**	**9.26**	9	4.3	8.3	3.1	1	1.6	2.7	1	1.37
M1N2S2	**13.5**	**5.9**	**9.35**	7.5	2.2	5.5	4.2	1	1.99	4.1	1	1.77
M1N3S2	**8.8**	**2.7**	**5.56**	6.1	1.4	4.1	3.5	1	2	4.2	1	1.69
M2N1S2	1.3	**1.3**	**1.3**	1.3	1.1	1.2	1.6	1	1.13	**1.8**	1	1.07
M2N2S2	**5.5**	**2.2**	**3.75**	4.7	1.4	3.5	3.2	1	1.65	2.6	1	1.35
M2N3S2	**6**	**1.9**	**3.88**	5.1	1.1	3.1	3.3	1	1.69	4.9	1	2.33

Table 6
Comparison of the three algorithms using the C metric.

Grp. code	C(PACO, NSGA-II)	C(PACO,SPEA2)	C(PACO,DACO)	C(DACO,PACO)	C(NSGA-II,PACO)	C(NSGA-II,SPEA2)	C(SPEA2,PACO)	C(SPEA2,NSGA-II)
M1N1S1	**0.75**	**0.7**	**0.49**	0.01	0.1	0.1	0	0.32
M1N2S1	**0.82**	**0.75**	**0.62**	0.04	0.02	0.13	0.04	0.67
M1N3S1	**0.7**	**0.78**	**0.32**	0.03	0.0125	0.208	0.025	0.75
M2N1S1	**1**	**1**	**0.5**	0.1	0	0.1	0	0
M2N2S1	**0.73**	**0.7**	**0.37**	0.17	0	0.19	0.06	0.59
M2N3S1	**0.8**	**0.7**	**0.18**	0.16	0.04	0.23	0.088	0.68
M1N1S2	**0.85**	**1**	**0.66**	0.07	0.018	0.23	0	0.64
M1N2S2	**1**	**1**	**1**	0	0	0.39	0	0.37
M1N3S2	**1**	**1**	**1**	0	0	0.75	0	0.08
M2N1S2	**1**	**1**	**0.3**	0	0	0.2	0	0.8
M2N2S2	**1**	**1**	**1**	0	0	0.31	0	0. 5
M2N3S2	**1**	**1**	**1**	0	0	0.55	0	0.4

Table 7
Comparison of the four algorithms using the *H* metric.

Grp. Code	PACO	DACO	NSGA-II	SPEA2
M1N1S1	**21,309**	6979.2	4013.8	4926.7
M1N2S1	**345,927.2**	151,580.2	73,434.2	66,132.6
M1N3S1	**606,371.9**	572,944.3	132,541.8	157,977.4
M2N1S1	**5610.6**	788.4	643.9	643.9
M2N2S1	**40,372.4**	34,095	10,779.3	11,607.3
M2N3S1	**399,452.7**	218,037.1	52,668.1	74,837.1
M1N1S2	**1,552,174**	883,946.7	211,817.3	34,584.4
M1N2S2	**3,588,616.8**	385,522.1	34,108.7	52,562
M1N3S2	**12,987,453.5**	1,021,087.9	261,909.1	110,222.6
M2N1S2	**52,052.9**	36,120	33,208.2	1097.2
M2N2S2	**623,750.9**	84,928.1	22,842.7	23,660.4
M2N3S2	**5,230,355.2**	426,957.3	87,939.1	151,548.3

column of Table 5 are the average of the *C* metric values on the ten instances of the corresponding instance group. The top and bottom six rows show the results on instances with small jobs and larger jobs, respectively. According to the average *C* values on each group, it can be observed that PACO outperforms the others, especially on instances with large job sizes. That is, the solutions of all groups found by DACO, NSGA-II and SPEA2 are almost dominated by those of PACO on the instances with large job sizes. Thus, in terms of the coverage, PACO is the best among all the algorithms.

Table 7 presents the results of the four algorithms in terms of the metric Hypervolume indicators over each group of instances. The top six rows show the results on small-job instances, where the average *H* values of PACO are all larger than those of the other three algorithms. According to the numbers in the bottom six rows, the average *H* values on each group of large-job instances, PACO is significantly superior to the other three algorithms. Obviously, PACO performs better than the other three algorithms on *H* metric, which means the solutions of PACO are closer to the Pareto frontier.

Table 8 presents the results of the four algorithms in terms of the two metrics DVR and SPC. Similarly, the real numbers in each row of Table 8 are the average values on the ten instances of each group. For each metric, the best result for each instance group is in boldface. The larger the value of DVR, and SPC, the better the solution is. Columns 2, 4, 6 and 8 of Table 8 present the average diversity metric of the algorithms. It can be observed that the average DVR values of PACO are larger than those of DACO, NSGA-II and SPEA2. According to the results on the instance groups with small-size jobs, as shown in the top six rows of Table 8, the average values of DVR of the DACO, NSGA-II and SPEA2, being zero on group M2N1S1, are worse than those of PACO. The bottom six rows of Table 8 show the average values on the large-job instance groups, where the average DVR values of PACO are all larger than those of DACO, NSGA-II and SPEA2. That is, PACO outperforms the other algorithms in terms of DVR over all instance groups.

The results of the SPC metric are shown as columns 3, 5, 7 and 9 of Table 8. The SPC values of PACO are more than those of DACO, NSGA-II and SPEA2, which means that the solutions found by PACO distribute more uniformly along the Pareto frontier than the others. Additionally, the top six rows, the average values of SPC over the small-job instances, show that PACO performs better than the others. The bottom six rows list the average results on the large-job instances. Consistent with the results on instances with small jobs, the performance of PACO in terms of the SPC is the best. Furthermore, on the groups M2N1S1 and M2N1S2, the SPC values of DACO, NSGA-II or SPEA2 are zero, due to the solutions found by the three algorithms being too closed. Especially, according to Table 4, it can be seen that DACO, NSGA-II and SPEA2 are only able to find one Pareto solution for M2N1S1.

Table 9 shows the average results of the DLB and the run time of the solutions of the four algorithms. Each run time of one algorithm is obtained as follows: first, average the run time of ten independent runs over every instance of each group; second, average the result of each group over its ten instances. For each metric, the best value of each instance group is in boldface. Note that the smaller the values of the DLB, as well as the runtime, the better performance the algorithm has.

Columns 2, 4, 6 and 8 present the results in terms of the DLB metric on all groups of the four algorithms. It can be observed that the average DLB value of PACO of each group are the smallest among the four algorithms. PACO outperforms the other three not only on the small-job instances but also on the large-job instances, which signifies that the solutions obtained by PACO are closer to the lower bound of the two objectives compared to those of the other algorithms. In addition, the results of the algorithms on several groups are smaller than the other groups. The reason may be that the solution space of the corresponding instance group is relatively small, as well as the *LB* being tight, especially on the instances with four machines.

The results of the average run time of the four algorithms on each group are shown in columns 3, 5, 7 and 9. The run time of all algorithms increases with the job numbers, as well as the machine numbers. According to the results of the small-job instances, the run time of PACO is more than that of the others, in that updating the pheromone trails and the local optimization increase its time complexity. Moreover, each ant constructs one feasible solution at every iteration. However, in DACO, NSGA-II or SPEA2, the batches are formed by a heuristic first and then scheduled by a meta-heuristic. The computation time of the heuristics is much less than that of the meta-heuristics. Additionally, in terms of the run time, PACO beats the other algorithms in 5 out of 6 groups of the big jobs except on M2N1S2 and M2N3S2.

Table 8
Comparison of the four algorithms using the DVR and SPC two metrics.

Grp. Code	PACO		DACO		NSGA-II		SPEA2	
	DVR	SPC	DVR	SPC	DVR	SPC	DVR	SPC
M1N1S1	**3689.5**	**0.38**	1554	0.27	4	0.08	653	0.07
M1N2S1	**42,340**	**1.10**	6623.5	0.37	1573	0.21	1374.5	0.23
M1N3S1	**83,051**	**0.86**	17,753	0.50	939.5	0.36	1631.5	0.49
M2N1S1	**2490.5**	**0.24**	0	0	0	0	0	0
M2N2S1	**6564.5**	**0.52**	1977.5	0.19	672	0.16	601.5	0.41
M2N3S1	**57,806**	**1.13**	8107	0.86	942.5	0.29	698.5	0.22
M1N1S2	**20,171.5**	**0.67**	15,384.5	0.52	874	0.37	872.5	0.21
M1N2S2	**73,263**	**0.72**	36,221.5	0.70	863	0.09	2101.5	0.29
M1N3S2	**135,289**	**0.84**	48,290.5	0.74	2651	0.28	4478	0.45
M2N1S2	**1097**	**0.12**	13.5	0	35	0	14.5	0
M2N2S2	**20,538**	**0.50**	12,997.5	0.43	519.5	0.20	1041	0.10
M2N3S2	**53,825.5**	**0.46**	22,928.5	0.41	1516	0.08	1641	0.28

Table 9
Comparison of the three algorithms using the DLB and time metrics.

Grp. Code	PACO		DACO		NSGA-II		SPEA2	
	DLB	Time	DLB	Time	DLB	Time	DLB	Time
M1N1S1	**0.15**	1.35	0.27	**0.59**	0.26	0.71	0.26	0.61
M1N2S1	**0.23**	7.99	0.40	**1.70**	0.38	3.36	0.37	2.89
M1N3S1	**0.34**	31.84	0.50	**4.79**	0.50	13.22	0.49	10.43
M2N1S1	**0.01**	1.92	0.06	0.78	0.06	0.86	0.06	**0.69**
M2N2S1	**0.02**	7.81	0.07	**2.12**	0.07	4.02	0.07	2.98
M2N3S1	**0.05**	33.28	0.13	**5.89**	0.13	13.97	0.12	11.17
M1N1S2	**0.21**	**1.66**	0.25	1.94	0.23	2.23	0.31	1.91
M1N2S2	**0.35**	**7.36**	0.50	9.94	0.51	18.07	0.51	16.63
M1N3S2	**0.30**	**27.15**	0.56	43.80	0.56	96.14	0.60	87.41
M2N1S2	**0.005**	2.98	0.016	2.91	0.01	2.47	0.036	**2.03**
M2N2S2	**0.02**	**13.02**	0.08	15.70	0.07	19.44	0.07	17.34
M2N3S2	**0.01**	**69.52**	0.14	73.93	0.13	108.72	0.14	92.28

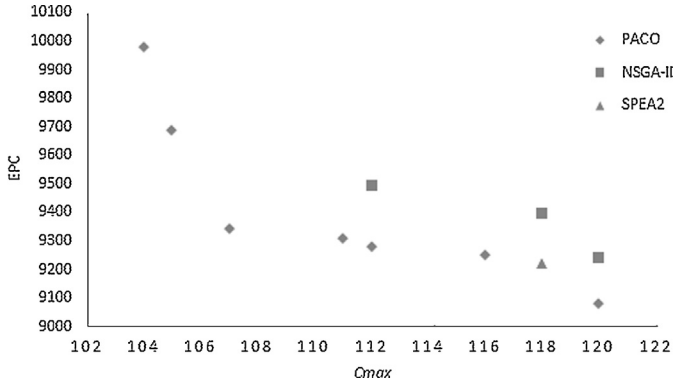

Fig. 5. Solution distribution of the four algorithms on Ins. 20-2-1-4.

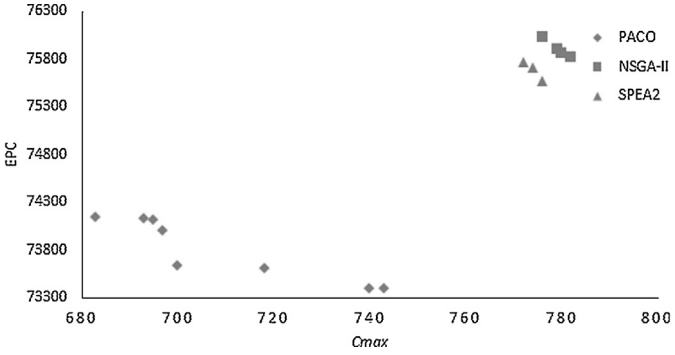

Fig. 8. Solution distribution of the four algorithms on Ins. 50-2-2-2.

To compare the solution quality of the four algorithms clearly, the solution distributions of some instances are illustrated as Figs. 5–10. The x-axis and the *y*-axis denote the values of C_{max} and the EPC, respectively. The four numbers linked by dashes, i.e., the code of an instance, in each caption represent the job number, the machine number, the type of the job size ("1" and "2" represent the small size and large size separately), and the instance index in each group, respectively. From these figures, it can be seen that not

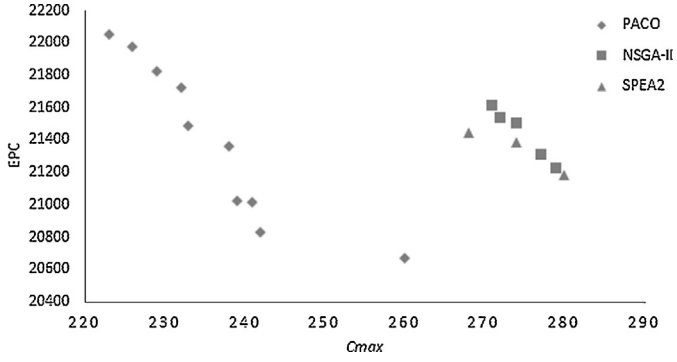

Fig. 6. Solution distribution of the four algorithms on Ins. 50-2-1-6.

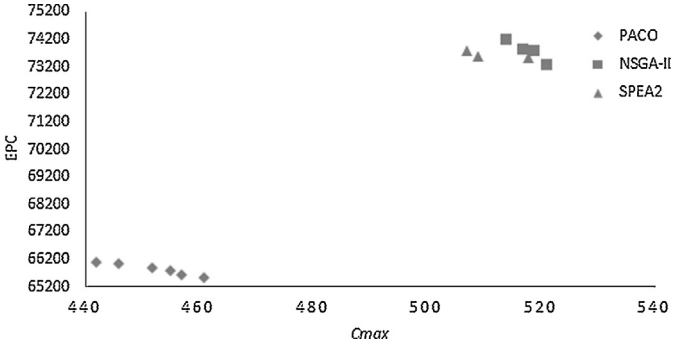

Fig. 9. Solution distribution of the four algorithms on Ins. 50-4-2-10.

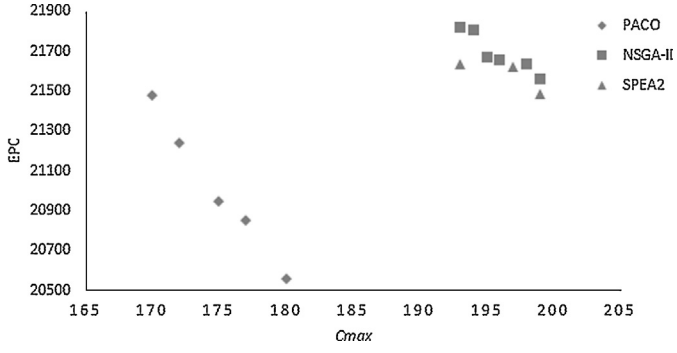

Fig. 7. Solution distribution of the four algorithms on Ins. 50-4-1-4.

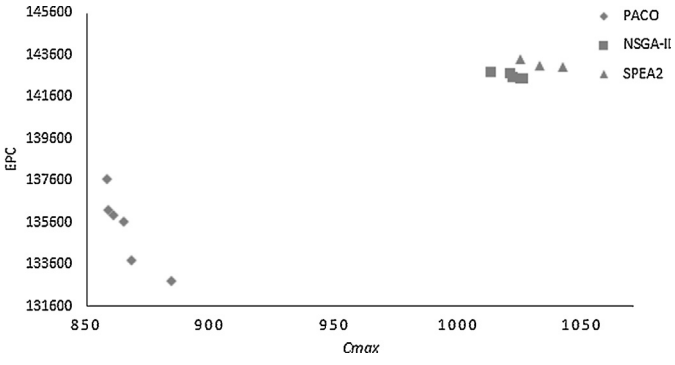

Fig. 10. Solution distribution of the four algorithms on Ins. 100-4-2-3.

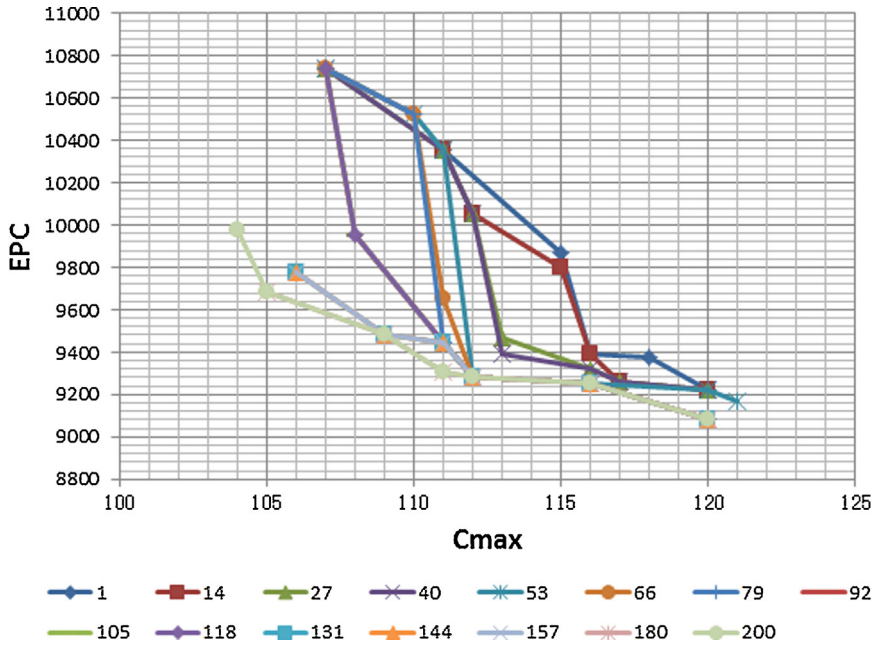

Fig. 11. Solution convergence of PACO algorithm on Ins. 20-2-1-4.

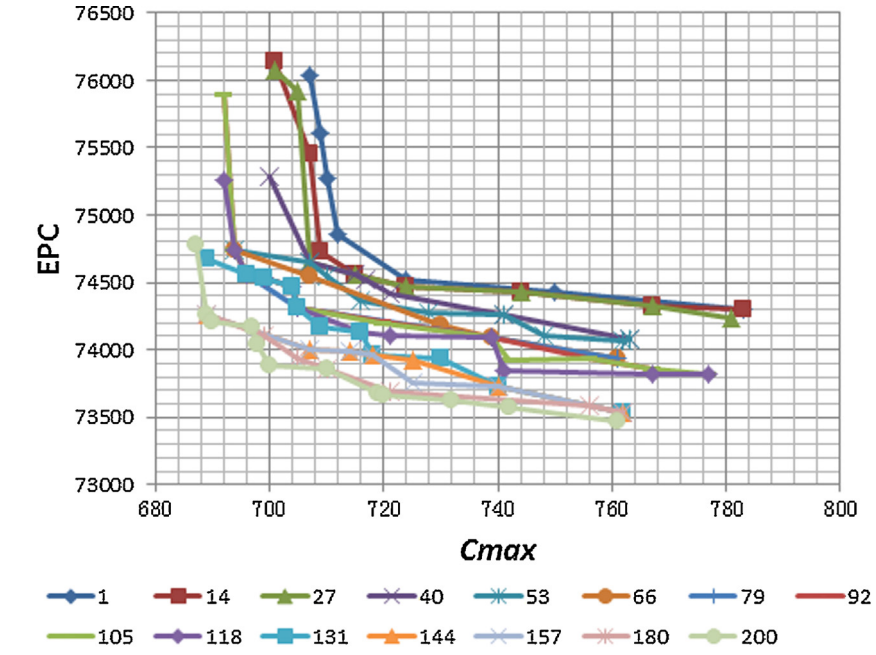

Fig. 12. Solution convergence of PACO algorithm on Ins. 50-2-2-2.

only the numbers of the Pareto-optimal solutions but also the distribution of the solutions obtained by PACO outperform those of the other algorithms.

To show the convergence of the proposed algorithm, we illustrate the Pareto solutions of every thirteen iterations for two different instances, i.e., Ins. 20-2-1-4 and Ins. 50-2-2-2, as Figs. 11 and 12. The non-dominated solutions found by the ant colony at each iteration are connected by the line segments. The lines with different colors correspond to different iterations which are shown in the below of the two figures. Each symbol in the line represents one non-dominated solution to the instance at the iteration. From the two figures, it can be observed that the non-dominated solution set found by the ant colony are both gradually closer to the two axes iteration by iteration.

6. Conclusions

In this paper, we addressed a bi-objective scheduling problem on parallel BPMs with dynamic job arrivals and non-identical job sizes for minimizing the makespan and total electricity cost by a proposed PACO algorithm. For the objective C_{max}, we construct two candidate lists to reduce the search space, and provide different dynamic heuristic information to guide the search. Finally, two object-oriented local optimization algorithms are incorporated

to improve the solutions, respectively. Taking into account different evaluation metrics, such as the diversity of the non-dominated solutions, the run time and the distribution of the Pareto-optimal solutions, the PACO was compared with three other algorithms. The simulated results showed that the proposed PACO algorithm significantly outperforms the compared algorithms, DACO, NSGA-II and SPEA2. As a result, PACO is effective and efficient since PACO is able to obtain the satisfactory Pareto solutions within reasonable computation time.

Future research can be extended to more complex manufacturing environments, such as flow shop or job shop, as well as the constraints related to the difference between the BPMs, such as the non-identical velocities and the unequal machine capacities. Additionally, other objectives could also be considered, such as the maximum tardiness, the total completion times, and so on.

Acknowledgements

The work of the first author is supported by the National Natural Science Foundation under Grants 71601001 and 71671168, the Humanity and Social Science Youth Foundation of Ministry of Education of China under Grant 15YJC630041, the Science Foundation of Anhui Province under Grant 1608085MG154, the Natural Science Foundation of Anhui Provincial Education Department under Grant KJ2015A062, the Foundation of Co-Innovation Center for Information Supply and Assurance Technology of Anhui University under Grant ADXXBZ201509. The work of the fourth author is supported by the Natural Science Foundation of China under Grant 71471052.

References

[1] Y. Ikura, M. Gimple, Efficient scheduling algorithms for a single batch processing machine, Oper. Res. Lett. 5 (2) (1986) 61–65.
[2] C.Y. Lee, R. Uzsoy, L.A. Martin-Vega, Efficient algorithms for scheduling semiconductor burn-in operations, Oper. Res. 40 (4) (1992) 764–775.
[3] J.H. Ahmadi, R.H. Ahmadi, S. Dasu, C.S. Tang, Batching and scheduling jobs on batch and discrete processors, Oper. Res. 40 (4) (1992) 750–763.
[4] M. Mathirajan, A.I. Sivakumar, A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor, Int. J. Adv. Manuf. Technol. 29 (9–10) (2006) 990–1001.
[5] H. Luo, G.Q. Huang, Y.F. Zhang, Q.Y. Dai, X. Chen, Two-stage hybrid batching flowshop scheduling with blocking and machine availability constraints using genetic algorithm, Robot. Comput.-Integr. Manuf. 25 (6) (2009) 962–971.
[6] H. Luo, G.Q. Huang, Y.F. Zhang, Q.Y. Dai, Hybrid flowshop scheduling with batch-discrete processors and machine maintenance in time windows, Int. J. Prod. Res. 49 (6) (2011) 1575–1603.
[7] V. Chandru, C.Y. Lee, R. Uzsoy, Minimizing total completion time on batch processing machines, Int. J. Prod. Res. 31 (9) (1993) 2097–2121.
[8] F.J. Ghazvini, L. Dupont, Minimizing mean flow times criteria on a single batch processing machine with non-identical jobs sizes, Int. J. Prod. Econ. 55 (3) (1998) 273–280.
[9] R. Uzsoy, Scheduling a single batch processing machine with non-identical job sizes, Int. J. Prod. Res. 32 (7) (2007) 1615–1635.
[10] R. Uzsoy, Y.Y. Yang, Minimizing total weighted completion time on a single batch processing machine, Prod. Oper. Manag. 6 (1) (2009) 57–73.
[11] S. Melouk, P. Damodaran, P.Y. Chang, Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing, Int. J. Prod. Econ. 87 (2) (2004) 141–147.
[12] F.D. Chou, P.C. Chang, H.M. Wang, A hybrid genetic algorithm to minimize makespan for the single batch machine dynamic scheduling problem, Int. J. Adv. Manuf. Technol. 31 (3) (2006) 350–359.
[13] S.G. Koh, P.H. Koo, J.W. Ha, W.S. Lee, Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families, Int. J. Prod. Res. 42 (42) (2004) 4091–4107.
[14] P.Y. Chang, P. Damodaran, S. Melouk, Minimizing makespan on parallel batch processing machines, Int. J. Prod. Res. 42 (19) (2004) 4211–4220.
[15] P. Damodaran, P.Y. Chang, Heuristics to minimize makespan of parallel batch processing machines, Int. J. Adv. Manuf. Technol. 37 (9) (2008) 1005–1013.
[16] C.Y. Lee, Minimizing makespan on a single batch processing machine with dynamic job arrivals, Int. J. Prod. Res. 37 (1) (1999) 219–236.
[17] P. Damodaran, N.S. Hirani, M.C. Vélez-Gallego, Scheduling identical parallel batch processing machines to minimise makespan using genetic algorithms, Eur. J. Ind. Eng. 3 (2) (2009) 187–206.
[18] P. Damodaran, M.C. Vélez-Gallego, Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times, Int. J. Adv. Manuf. Technol. 49 (49) (2010) 1119–1128.
[19] S.G. Li, G.J. Li, S.Q. Zhang, Minimizing makespan with release times on identical parallel batching machines, Discrete Appl. Math. 148 (1) (2005) 127–134.
[20] S.H. Chung, Y.T. Tai, W.L. Pearn, Minimising makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes, Int. J. Prod. Res. 47 (18) (2009) 5109–5128.
[21] H.M. Wang, F.D. Chou, Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics, Expert Syst. Appl. 37 (2) (2010) 1510–1521.
[22] X. Zhang, X.T. Li, J.A. Wang, Local search algorithm with path relinking for single batch-processing machine scheduling problem, Neural Comput. Appl. 5 (2016) 1–14.
[23] J.E.C. Arroyo, Y.T. Leung, Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times, Comput. Oper. Res. 78 (2016) 117–128.
[24] A.H. Kashan, B. Karimi, F. Jolai, An Effective Hybrid Multi-Objective Genetic Algorithm for Bi-Criteria Scheduling on a Single Batch Processing Machine With Non-Identical Job Sizes, Williams & Wilkins, 1995.
[25] R. Xu, H.P. Chen, X.P. Li, A bi-objective scheduling problem on batch machines via a Pareto-based ant colony system, Int. J. Prod. Econ. 145 (1) (2013) 371–386.
[26] S.J. Wang, M. Liu, F. Chu, C.B. Chu, Bi-objective optimization of a single machine batch scheduling problem with energy cost consideration, J. Clean. Prod. 137 (2016) 1205–1215.
[27] C.N. Potts, M.Y. Kovalyov, Scheduling with batching: a review, Eur. J. Oper. Res. 120 (2) (2000) 228–249.
[28] L. Mönch, J.W. Fowler, S. Dauzère-Pérès, S.J. Mason, O. Rose, A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations, J. Sched. 14 (6) (2011) 583–599.
[29] M. Dorigo, L.M. Gambardella, Ant colonies for the travelling salesman problem, Biosystems 43 (2) (1997) 73–81.
[30] B. Bullnheimer, R.F. Hartl, C. Strauss, A new rank based version of the ant system—a computational study, Central Eur. J. Oper. Res. (1) (1999) 25–38.
[31] T. Stützle, H.H. Hoos, Max–min ant system, Future Gen. Comput. Syst. 16 (8) (1999) 889–914.
[32] M.L. Pilat, T. White, Using genetic algorithms to optimize ACS-TSP, in: Ant Algorithms, Third International Workshop, ANTS 2002, Brussels, Belgium, September 12–14, 2002, Proceedings, 2002, pp. 282–287.
[33] A.H. Kashan, B. Karimi, Scheduling a single batch-processing machine with arbitrary job sizes and incompatible job families: an ant colony framework, J. Oper. Res. Soc. 59 (9) (2008) 1269–1280.
[34] B.Y. Cheng, K. Li, B. Chen, Scheduling a single batch-processing machine with non-identical job sizes in fuzzy environment using an improved ant colony optimization, J. Manuf. Syst. 29 (1) (2010) 29–34.
[35] P. Damodaran, P.K. Manjeshwar, K. Srihari, Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms, Int. J. Prod. Econ. 103 (2) (2006) 882–891.
[36] R. Xu, H.P. Chen, X.P. Li, Makespan minimization on single batch-processing machine via ant colony optimization, Comput. Oper. Res. 39 (3) (2012) 582–593.
[37] B.Y. Cheng, Y.T. Leung, K. Li, Integrated scheduling of production and distribution to minimize total cost using an improved ant colony optimization method, Comput. Ind. Eng. 83 (1) (2015) 217–225.
[38] Z.H. Jia, C. Wang, Y.T. Leung, An ACO algorithm for makespan minimization in parallel batch machines with non-identical job sizes and incompatible job families, Appl. Soft Comput. 38 (2015) 395–404.
[39] B. Yagmahan, M.M. Yenisey, Ant colony optimization for multi-objective flow shop scheduling problem, Comput. Ind. Eng. 54 (3) (2008) 411–420.
[40] L. Li, F. Qiao, Q.D. Wu, ACO-based multi-objective scheduling of parallel batch processing machines with advanced process control constraints, Int. J. Adv. Manuf. Technol. 44 (9–10) (2010) 985–994.
[41] B. Du, H.P. Chen, G.Q. Huang, H.D. Yang, Preference Vector Ant Colony System for Minimising Makespan and Energy Consumption in a Hybrid Flow Shop, Springer, London, 2011.
[42] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[43] E. Zitzler, M. Laumanns, L. Thiele, Spea2: improving the strength Pareto evolutionary algorithm Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems, vol. 3242, 2001, pp. 95–100.
[44] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: empirical results, Evol. Comput. 8 (2) (2000) 173–195.
[45] M. Fleischer, The measure of Pareto optima applications to multi-objective metaheuristics, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2003, pp. 519–533.
[46] J. Knowles, Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems, IEEE Trans. Evol. Comput. 10 (1) (2006) 50–66.
[47] P. Damodaran, M.C. Vélez-Gallego, J. Maya, A grasp approach for makespan minimization on parallel batch processing machines, J. Intell. Manuf. 22 (5) (2011) 767–777.