EFFECTIVE HEURISTICS FOR MAKESPAN MINIMIZATION IN PARALLEL BATCH MACHINES WITH NON-IDENTICAL CAPACITIES AND JOB RELEASE TIMES

ZHAO-HONG JIA* AND TING-TING WEN

Key Lab of Intelligent Computing and Signal Processing of Ministry of Education
Anhui University
Hefei, Anhui, 230039, China

JOSEPH Y.-T. LEUNG AND KAI LI

School of Management, Hefei University of Technology
Hefei, Anhui, 230009, China
Department of Computer Science, New Jersey Institute of Technology
Newark, NJ 07102, USA

(Communicated by Kok Lay Teo)

Abstract. We consider the problem of scheduling a set of n jobs with arbitrary job sizes, processing times and release times on a set of m parallel batch machines with non-identical capacities; the objective is to minimize the makespan. We first present an algorithm to compute a lower bound for the optimal makespan. Based on different rules of batching the jobs and assigning the batches to the machines, several heuristics are proposed to solve the problem. The performance of the proposed heuristics is evaluated by computational experiments. The proposed heuristics are compared against the lower bound and against each other. Our results show that the one of the proposed algorithms outperforms all the other heuristics.

1. Introduction. Scheduling on batch processing machines (BPMs) has wide applications in many industries, such as semiconductor manufacturing, transportation, steel casting, and food processing. It is originally motivated by the burn-in operation in semiconductor fabrication. Due to the long processing time compared with other operations (120 hours versus 4-5 hours), the burn-in operation is often the bottleneck in the semiconductor production [24]. Hence, scheduling the burn-in operation efficiently is crucial in obtaining high productivity. Different from the classical scheduling model, in batch scheduling, a machine can process a group of jobs at the same time. That is to say, the jobs are grouped into batches and the batches are then scheduled on the machines. The processing time of a batch is equal to the longest processing time of the jobs in that batch [29]. In practical application, the sizes of the jobs are non-identical. The total size of all the jobs in a batch cannot exceed the capacity of the machine that processes the batch.

2010 Mathematics Subject Classification. Primary: 68T20, 68T05; Secondary: 05-04.

Key words and phrases. Parallel batch machines, non-identical machine capacities, release times, non-identical job sizes, makespan, heuristics.

* Corresponding author: Zhao-Hong Jia.
There are two types of batch scheduling problems: serial-batching (s-batching) and parallel-batching (p-batching). In the former type, the jobs in a batch are processed in serial, while in the latter type the jobs in a batch are processed in parallel. P-batch scheduling is important not only in semiconductor manufacturing [32], but also in other modern manufacturing industries, such as chemical and mineral processing, pharmaceutical and metalworking industries, as well as environmental stress screening chamber fabrication [47].

In this paper, we consider a dynamic p-batch scheduling problem on parallel BPMs, where the machine capacities are non-identical. A set of jobs with non-identical job sizes and processing times have to be grouped into batches such that the total size of the jobs in the batch cannot exceed the capacity of the machine that processes it. The jobs have non-identical release times. The batches are then scheduled on the machines so as to minimize the makespan. Once a batch is being processed, it cannot be interrupted and no job can be added into or removed from the batch. The problem of minimizing the makespan on a single BPM with non-identical job sizes and identical release times has been shown to be NP-hard [38]. Therefore, our problem is also NP-hard.

The problem can be resolved by solving two independent subproblems: grouping the jobs into batches and then scheduling the batches on the parallel BPMs. We propose two batching algorithms to group the jobs into batches; i.e., heuristics based on the First-Fit and the Best-Fit rules. Then two heuristics based on the First-Fit and the Best-Fit rules are applied to schedule the batches on the machines. In all, there are four heuristics proposed to solve the problem.

The rest of the paper is organized as follows. In Section 2, we review related work on BPM scheduling problems. Section 3 formally defines the problem. In Section 4, we present a method to compute a lower bound for the optimal makespan. The proposed algorithms are described in Section 5. With elaborate experimental design, the proposed heuristics are compared with the lower bound and against each other in Section 6. Finally, in Section 7, we draw some concluding remarks and give some possible directions for future studies.

2. Literature review. In classical machine scheduling, parallel machine scheduling have been extensively studied over the last three decades [44, 18, 28]. Jobs with machine eligibility restrictions have also been extensively studied [26]. Ou et al. [33] give a heuristic for inclusive processing set restrictions with worst-case performance ratio at most 4/3. Huo and Leung [16] provide a faster algorithm for a more general case with the same worst-case bound. The difference between our paper and previous studies is that we are scheduling parallel-batch jobs, rather than traditional jobs. Although significant studies have been devoted to batch scheduling problems, we only focus on reviewing those work that have commonalities in their assumptions with ours, especially those of non-identical job sizes.

The earliest study in this area can be traced back to Ikura and Gimple [17], who propose an algorithm to minimize the makespan on a single BPM with identical processing time, unit job size, and dynamic job arrivals. After that, more research has been done on this problem. Uzsoy [38] proves that scheduling jobs with non-identical job sizes on a single BPM so as to minimize makespan is strongly NP-hard. He then present several heuristics and a branch-and-bound algorithm. Among the heuristics he presented, the First-Fit-Longest-Processing-Time (FFLPT) heuristic shows the best performance. The p-batch scheduling problem can be partitioned
into two subproblems; i.e., grouping the jobs into batches and scheduling the batches on the machines. The first subproblem shares some similarities with the bin-packing problem [21]. Dupont and Jolai Ghazvini [12] give two heuristics; i.e., the Best-Fit-Longest-Processing-Time (BFLPT) and the Successive Knapsack (SK). The former heuristic is based on the Best-Fit rule for the bin-packing problem, while the latter constructs a solution batch by batch to minimize the unoccupied space of the batches. According to experimental results, the BFLPT method is better than the SK method. Uzsoy and Yang [40] present several heuristics and a branch-and-bound algorithm to minimize the total weighted completion time. For minimizing the makespan on a single BPM, Dupont and Dhaenens-Flipo [11] propose a branch-and-bound algorithm. Zhang et al. [49] are the first ones to give worst-case performance bounds for several heuristics for the makespan minimization on a single BPM. Li et al. [27] develop an approximation algorithm for the general problem with non-identical release times. Tang et al. [37] propose a heuristic algorithm to solve the coordinated scheduling problem of hybrid batch production on a single batching machine and two-stage transportation connecting the production. Fan et al. [13] provide the pseudo-polynomial-time algorithms to solve the bi-criteria scheduling on a single batch machine to minimize two regular scheduling criteria, i.e., the total weighted completion time and the weighted total number of tardy jobs, that are non-decreasing in the job completion times. Recently, metaheuristics, such as simulated annealing (SA) [10], genetic algorithms (GA) [20], ant colony optimization (ACO) [6] and max-min ant system (MMAS) [18] have been applied to solve the p-batch scheduling problem on a single BPM.

Considering that jobs in practice often arrive at different times, Lee and Uzsoy [25] present polynomial and pseudo-polynomial time algorithms for some special cases and heuristics for makespan minimization on a single BPM. Sung and Choung [36] study the same problem and develop several heuristics. For scheduling on a single BPM with job release times, Wang and Uzsoy [42] present a GA combined with dynamic programming to minimize the maximum lateness, while Kruz and Mason [23] propose a heuristic to minimize the total tardiness. By using the features of the problem, Xu et al. [46] give an ACO algorithm to minimize the makespan on a single BPM with job release times and non-identical job sizes.

P-batch problems on parallel machines are also considered because parallel machines are closer to real-world situations. Lee et al. [24] suggest using the Longest-Processing-Time (LPT) algorithm for parallel BPMs with identical job sizes. Uzsoy [39] address the problems of minimizing the makespan, maximum lateness, and total weighted completion time for parallel BPMs with job release times and incompatible job families. To schedule on two identical BPMs with a common deadline, unit processing time and unit setup time, Brucker et al. [3] present a dynamic programming algorithm. Mönch and Almeder [31] develop an ant colony system to minimize the total weighted tardiness (TWT) on parallel BPMs with incompatible families, identical job size and arbitrary job weights. Using the structural properties of the problem, Venkataramana and Srinivas Raghavan [41] design three ACO-based algorithms to solve the same problem. For the same problem, Almeder and Mönch [1] propose an efficient Variable Neighborhood Search (VNS) method combined with a decomposition heuristic and a local search scheme. Considering the precedence constraints among the jobs, Bilyk et al. [2] provide two metaheuristics to schedule the incompatible jobs with ready times on parallel batch machines for TWT minimization. The VNS-based approach slightly outperforms the one based on greedy
randomized adaptive search procedure. Klemmt et al. [22] study the problem of unrelated parallel batch machines with incompatible job families and unequal job ready times. They also propose a mixed integer programming decomposition based on time windows and a VNS-based method to minimize TWT, respectively.

Since job sizes are generally different in many applications, researchers turn their attentions to scheduling jobs with non-identical job sizes. To minimize the makespan, Chang et al. [4] present a SA algorithm for scheduling jobs with non-identical sizes on parallel BPMs. Damodaran and Chang [8] propose heuristics to minimize the makespan on parallel BPMs. First, two heuristics are applied to batch the jobs; i.e., the FFLPT and BFLPT. Then, the batches are scheduled by either the LPT or the Multifit (MF) rule. Meta-heuristics have also been studied. Kashan et al. [19] propose a hybrid genetic heuristic (HGH) to minimize the makespan on parallel BPMs with arbitrary job sizes. Shao et al. [35] develop a Neural Network (NN) method to solve the problem. Cheng et al. [7] give a polynomial time algorithm to minimize the makespan and total completion time for scheduling on parallel BPMs with non-identical job sizes. Malve and Uzsoy [29] provide an efficient GA for minimizing the maximum lateness on parallel BPMs with dynamic job arrivals. Chen et al. [5] apply an ACO and a GA algorithm to the problem on parallel BPMs with dynamic job arrivals.

The above research on parallel BPMs assume that all machines have the same capacity. In real-world production environments, there are machines that are older models and machines that are newer models. Newer machines usually have larger capacities than the older ones. Moreover, some jobs may have sizes larger than the capacity of the older machines. Thus, it is important to study this scheduling problem with non-identical machine capacities and jobs can only fit in a subset of the machines. To address batch scheduling with non-identical machine capacities, Xu and Bean [45] propose a GA based on random keys encoding (RKGA). Comparing with commercial solver (CPLEX), the RKGA finds a better solution in a reasonable amount of time. Wang and Chou [43] present a metaheuristic based on GA and SA to minimize the makespan, with non-identical job sizes and job release times as well as non-identical machine capacities. Damodaran et al. [9] provide a particle swarm optimization (PSO) algorithm for the problem with non-identical job sizes and machine capacities. The results of the comparative experiment validate the effectiveness and efficiency of the PSO algorithm. However, the problems studied in these three papers are different from ours. Xu and Bean [45] and Wang and Chou [43] both assume that all jobs can fit in any machine. That is, the largest job size is smaller than the smallest machine capacity. In addition, the problem studied in Damodaran et al. [9] assumes that all jobs arrive at time zero. Our problem extends the previous research by involving parallel machines with non-identical capacities, non-identical job sizes and dynamic job arrivals.

We refer the readers to the articles [34], [30], and [32] for an extensive review of the BPM problems.

3. Problem statement and notation. Using the three-field notation proposed in [15], the problem under study can be denoted by $P \mid p \text{-batch}, M_j(\text{inclusive}), s_j, r_j \mid C_{\text{max}}$, where $P$ and $p \text{-batch}$ denote parallel p-batching processing machines, $M_j(\text{inclusive})$ denotes the machines for processing each job is a subset of the machine set, $s_j$ and $r_j$ denote the jobs have unequal sizes and unequal arrival times, and
**Effective Heuristics for Makespan Minimization**

\( C_{\text{max}} \) is the optimization objective, i.e., the makespan. The problem is described as follows:

1. A job set, denoted by \( J = \{1, 2, \ldots, n\} \), is to be grouped in batches, denoted by \( B_k \) \((k = 1, 2, \ldots, b)\), which are then scheduled on a set of \( m \) parallel batch machines with non-identical capacities. The number of batches cannot be determined until all the jobs have been grouped. Each job \( j \in J \) \((j = 1, 2, \ldots, n)\) has a processing time \( p_j \), a size \( s_j \) and a release time \( r_j \).

2. The machine set is denoted by \( M = \{1, 2, \ldots, m\} \). Each machine \( i \) \((i = 1, 2, \ldots, m)\) has a capacity denoted by \( S_i \). Without loss of generality, we assume that \( S_1 \leq S_2 \leq \cdots \leq S_m \) and the size of each job does not exceed the largest machine capacity \( S_m \). The total size of all the jobs in a batch \( B_k \) cannot exceed the capacity of the machine \( i \) where the batch is processed, i.e., \( \sum_{j \in B_k} s_j \leq S_i \). The completion time of a machine is the completion time of the last batch on the machine.

3. The batch set generated by grouping the jobs is denoted by \( B \). The processing time of batch \( B_k \), denoted by \( P_k \), is given by the longest processing time of all the jobs in the batch, i.e., \( P_k = \max \{p_j | j \in B_k\} \). The release time of batch \( B_k \), denoted by \( R_k \), is the latest arrival time of all the jobs in this batch, i.e., \( R_k = \max \{r_j | j \in B_k\} \).

4. After the batch sequence is determined, each batch \( B_k \) has a start time \( S_k \) and a completion time \( C_k \). If \( B_l \) is the batch that is processed immediately before \( B_k \), then \( S_k = \max \{R_l, C_l\} \). If \( B_k \) is the first batch scheduled, then \( S_k = R_k \). In both cases, \( C_k = S_k + P_k \).

The objective is to minimize the makespan \( C_{\text{max}} \), which is the maximum of the completion times of all the machines.

4. **A lower bound.** In this section we propose a method to compute a lower bound for a given instance of the problem. The lower bound will be used to evaluate the performance of our heuristics. Since the problem is strongly NP-hard, it is computationally infeasible to obtain optimal solutions for large-scale instances. This is the reason why a lower bound is needed.

In order to explain our heuristics and the lower bound, it will be more convenient to assume that there are only three machine capacities and four job release times. The generalization to more than three machine capacities and four job release times is obvious. Let the three machine capacities be denoted by \( S_1, S_2 \) and \( S_3 \), where \( S_1 < S_2 < S_3 \), and the four job release times be denoted by \( r_1, r_2, r_3 \) and \( r_4 \), where \( r_1 < r_2 < r_3 < r_4 \). Without loss of generality, we may assume that \( r_1 = 0 \).

The number of machines whose capacity is \( S_i \) \((i = 1, 2, 3)\) is denoted by \( m_i \). Note that \( m = \sum_{i=1}^{3} m_i \). According to the machine capacities, the machine set can be partitioned into three subsets, denoted by \( M^1, M^2 \) and \( M^3 \). Similarly, the job set can be divided into three subsets, denoted by \( J^1, J^2 \) and \( J^3 \), where \( J^1 = \{ j | s_j \leq S_1 \} \), \( J^2 = \{ j | S_1 < s_j \leq S_2 \} \) and \( J^3 = \{ j | S_2 < s_j \leq S_3 \} \). The jobs in \( J^1 \) can be scheduled on any machine, while the jobs in \( J^2 \) and \( J^3 \) can be scheduled on any machine in \( M^2 \cup M^3 \), respectively.

Let the job set \( J \) be divided into four sets: \( J_i \), \( 1 \leq i \leq 4 \), where \( J_i \) are all the jobs with release time \( r_i \). We first take \( J_4 \) and assign it to \( J_4 \). We then compute a lower bound (to be described later) for \( J_4 \) as if these jobs are all released at time \( r_4 \). Then we merge \( J_3 \) into \( J_4 \) to obtain \( J_5 \), and compute a lower bound for \( J_5 \) as if all these jobs are released at time \( r_3 \). We then merge \( J_2 \) into \( J_5 \) to obtain
Finally, we merge all these jobs are released at time $r_2$. To compute the lower bound for the jobs in $J_i$, we divide $J_i$ into three sets, $J_1^i$, $J_2^i$ and $J_3^i$, where $J_1^i = \{ j \in J_i \mid s_j \leq S_1 \}$, $J_2^i = \{ j \in J_i \mid S_1 < s_j \leq S_2 \}$ and $J_3^i = \{ j \in J_i \mid S_2 < s_j \leq S_3 \}$. Clearly, the length of the optimal schedule for $J_i$ is at least the longest processing time of the jobs in $J_i$. So, we let $LBB_1 = \max \{ p_j \mid j \in J_i \}$. Another lower bound can be computed by scheduling all the jobs in $J_3^i$ on a machine in $M^3$. Here, we split each job $j$ into $p_j * s_j$ unit-size and unit processing time jobs. Therefore, we let $LBB_2 = [\sum_{j \in J_3^i} (p_j * s_j)/(m_3 * S_3)]$. Another lower bound can be computed by scheduling all the jobs in $J_2^i \cup J_3^i$ on a machine in $M^2 \cup M^3$. So, we let $LBB_3 = [\sum_{j \in J_2^i \cup J_3^i} (p_j * s_j)/(m_2 * S_2 + m_3 * S_3)]$. The final possibility is to schedule all the jobs in $J_i$ on any machine. So, we let $LBB_4 = [\sum_{j \in J_i} (p_j * s_j)/(m_1 * S_1 + m_2 * S_2 + m_3 * S_3)]$. The lower bound for the jobs in $J_i$ is $LBB = \max \{ LBB_1, LBB_2, LBB_3, LBB_4 \}$.

Shown below is the algorithm to compute the lower bound for $J$. The algorithm employs a function $LBB(\cdot)$ to compute the minimum length schedule for $J_i$. The function $LBB(\cdot)$ will be shown in the next algorithm.

**Algorithm 1. The LB Algorithm**

1. According to the job arrival times, divide the job set $J$ into four subsets, denoted by $J_1$, $J_2$, $J_3$ and $J_4$.

   $$J = J_1 \cup J_2 \cup J_3 \cup J_4,$$

   \hspace{1cm} (1)

   where the release times of $J_1$, $J_2$, $J_3$ and $J_4$ are $r_1$, $r_2$, $r_3$ and $r_4$, respectively.

2. Let $J_4 = J_4$. Call $LBB(\cdot)$ to compute the minimum length schedule for $J_4$.

   $$LB_1 = LBB(J_4).$$

3. Merge $J_3$ into $J_4$ to obtain $J_3$; i.e., $J_3 = J_4 \cup J_3$. Call $LBB(\cdot)$ to compute the minimum length schedule for $J_3$.

   $$LB_3 = LBB(J_3).$$

4. Merge $J_2$ into $J_3$ to obtain $J_2$; i.e., $J_2 = J_3 \cup J_2$. Call $LBB(\cdot)$ to compute the minimum length schedule for $J_2$.

   $$LB_2 = LBB(J_2).$$

5. Merge $J_1$ into $J_2$ to obtain $J_1$; i.e., $J_1 = J_2 \cup J_1$. Call $LBB(\cdot)$ to compute the minimum length schedule for $J_1$.

   $$LB_1 = LBB(J_1).$$

6. The lower bound of the problem is given as follows:

   $$LB = LB_k + r_k,$$

   \hspace{1cm} (6)

   where $k = \arg \max \{ LB_1 + r_1, LB_2 + r_2, LB_3 + r_3, LB_4 + r_4 \}$.

Therefore, the upper bound of the problem, denoted by $UB$, is given by the following equation.

$$UB = 2 * LB_k + r_k$$

**Algorithm 2. The LBB Function**

Input: A job set $J$
Output: The minimum length schedule for $J$
1. According to the machine capacities, divide the job set \( J \) into three subsets, denoted by \( J^1, J^2 \) and \( J^3 \).

\[
J = J^1 \cup J^2 \cup J^3, \tag{8}
\]
where \( J^1 = \{ j \mid s_j \leq S_1 \}, J^2 = \{ j \mid S_1 < s_j \leq S_2 \}, J^3 = \{ j \mid S_2 < s_j \leq S_3 \} \).

2. Calculate and output the minimum length schedule for \( J \).

\[
LBB = \max\{LBB_1, LBB_2, LBB_3, LBB_4\}, \tag{9}
\]
where \( LBB_1 = \max\{p_j \mid j \in J^1\}, LBB_2 = \lceil \sum_{j \in J^2} (p_j \ast s_j)/(m_3 \ast S_3) \rceil, \]
\[
LBB_3 = \lceil \sum_{j \in J^3} (p_j \ast s_j)/(m_2 \ast S_2 + m_3 \ast S_3) \rceil, \text{ and } LBB_4 = \lceil \sum_{j \in J} (p_j \ast s_j)/(m_1 \ast S_1 + m_2 \ast S_2 + m_3 \ast S_3) \rceil. \]

5. **Description of the heuristics.** There are two steps involved in solving the problem. In the first step, we group all the jobs into batches to obtain a batch set. In the second step, schedule the batch set on the parallel machines. The scheduling is done by conducting a binary search in the interval \( [LB, UB] \). For each value \( L \) obtained in the binary search, we check if we can schedule all the batches so that the makespan is at most \( L \). If we are successful, then search the lower half by setting \( UB \) to be \( L \); otherwise, search the upper half by setting \( LB \) to be \( L \). This process is repeated until \( UB \leq LB \). The makespan obtained by the heuristic is given by \( UB \).

For the first step, we can apply two heuristics based on the First-Fit [38] and the Best-Fit [14] rules, respectively. The First-Fit rule constructs the batches by first sorting the jobs in descending order of their processing times, and then putting the first job in the sequence into the first batch that can accommodate the job. If no batch can accommodate the job, construct a new batch and put the first job in the new batch. The Best-Fit rule works in the same manner as the First-Fit rule, except that the first job is put into the batch with the smallest unoccupied space. For the second step, the First-Fit rule schedules the batches by first sorting the batches in descending order of their processing times, and then assigning the first batch in the sequence to the first machine that can process it. The Best-Fit rule assigns the first batch to the best fit machine, i.e., the machine with the smallest unoccupied space based on a bound. In all, there are four heuristics by combining the two batching rules and the two scheduling rules. They are denoted by FF, FB, BF, and BB, respectively. Note that the first letter denotes the batching rule and the second letter denotes the scheduling rule. Since the differences between the four heuristics are only in the way of batching the jobs and scheduling the batches, we will describe only the first heuristic, i.e., FF, in detail. The other three heuristics can be obtained by replacing the First-Fit rule by the Best-Fit rule.

5.1. **The FF heuristic.** To make the heuristic easy to understand, we assume that there are three different machine capacities and four job release times as described in Section 4. The generalization to more than three machine capacities and more than four job release times is easy to see.

The FF heuristic works as follows. First, it computes \( LB \) and \( UB \) by Algorithm 1 in Section 4. Then, according to the job release times, it divides the job set \( J \) into four subsets, denoted by \( J_1, J_2, J_3, J_4 \). The arrival times of the jobs in the subsets are \( r_1, r_2, r_3, r_4 \), respectively, where \( r_1 < r_2 < r_3 < r_4 \). Then, it sets \( L = (UB + LB)/2 \), and attempts to schedule the jobs in \( J \) so that the makespan is at most \( L \). The scheduling is done step by step. In the first step, it tries to schedule
the jobs in $J_4$ between $L$ and $r_4$ by the algorithm VIAB (to be described later). The scheduling starts from $L$ and works towards $r_4$. VIAB returns a value 1 if $J_4$ can be scheduled between $L$ and $r_4$; otherwise, it returns 0. If VIAB returns 0, this means that it is impossible to schedule the jobs in $J_4$ between $L$ and $r_4$. In this case it searches the upper half of the range by setting $LB$ to be $L$. On the other hand, if VIAB returns 1, then it records the completion times $T = (T_1, T_2, \cdots, T_m)$, where $T_i$ is the completion time of machine $i$. Then it calls VIAB again to see if the jobs in $J_3$ can be scheduled between $T$ and $r_3$. It repeats the same process for $J_2$ and $r_2$, as well as $J_1$ and $r_1$. If VIAB returns 1 in all of these calls, the scheduling is successful and it searches the lower half of the range by setting $UB$ to be $L$. On the other hand, if VIAB returns 0 in one of these calls, the scheduling is unsuccessful and we set $LB$ to be $L$.

The algorithm VIAB has three input parameters: $X$, $P$ and $r$. $X$ is a set of jobs, $P = (P_1, P_2, \cdots, P_m)$ is a vector of completion times of the $m$ machines, and $r$ is a release time. VIAB will heuristically decide if we can schedule the jobs in $X$ between $P$ and $r$. It works as follows. Partition the jobs in $X$ into three subsets $X^1$, $X^2$ and $X^3$, where $X^1 = \{j \in X \mid s_j \leq S_1\}$, $X^2 = \{j \in X \mid S_1 < s_j \leq S_2\}$, and $X^3 = \{j \in X \mid S_2 < s_j \leq S_3\}$. Similarly, divide the machines into three sets $M^1$, $M^2$ and $M^3$, where the machines in $M_i$, $1 \leq i \leq 3$, have capacity $S_i$. Sort the jobs in $X^1$ in descending order of their processing times, and batch them by the First-Fit rule. Sort the batches in descending order of their processing times, and schedule the batches on the machines of $M^1$ by the First-Fit rule. The scheduling starts from $P$ and works towards $r$. No job is allowed to complete before $r$. If there are some batches that cannot be scheduled, then merge the jobs in the unscheduled batches into $X^2$. We then proceed to schedule the jobs in $X^2$ on the machines in $M^2$ in the same manner as in $X^1$. After the jobs in $X^2$ are scheduled, we proceed to schedule the jobs in $X^3$ on the machines in $M^3$ in the same manner as in $X^2$. If all the batches can be scheduled on the machines in $M^3$; then we record the completion time of each machine in $P$ and return 1; otherwise, we return 0.

The flow of the proposed heuristic, referred to as the FF algorithm, is described as follows.

Algorithm 3. The FF Algorithm

1. In ascending order of the job release times, partition the job set into four subsets, denoted by $J_1, J_2, J_3, J_4$.
2. Compute $LB$ and $UB$ according to Algorithm 1 in Section 4.
3. $L \leftarrow (LB + UB)/2$. /* The median. */
4. $T = (L, L, \cdots, L)$.
5. Schedule the job set $J_4$ by the VIAB algorithm; i.e., $FLAG \leftarrow$ VIAB$(J_4, T, r_4)$. /* Call VIAB$(J_4, T, r_4)$ to test whether all the jobs can be scheduled on the machines backwards from time $T$ to $r_4$. */
6. If $FLAG = 0$ then $LB \leftarrow L$, goto Step 3. /* Narrow the range of search. */
7. Schedule the job set $J_3$ by the VIAB algorithm; i.e., $FLAG \leftarrow$ VIAB$(J_3, T, r_3)$. /* Call VIAB$(J_3, T, r_3)$ to test whether all the jobs can be scheduled on the machines backwards from time $T$ to $r_3$. */
8. If $FLAG = 0$ then $LB \leftarrow L$, goto Step 3. /* Narrow the range of search. */
9. Schedule the job set $J_2$ by the VIAB algorithm; i.e., $FLAG \leftarrow$ VIAB$(J_2, T, r_2)$. /* Call VIAB$(J_2, T, r_2)$ to test whether all the jobs can be scheduled on the machines backwards from time $T$ to $r_2$. */
10. If $FLAG = 0$ then $LB \leftarrow L$, goto Step 3. /* Narrow the range of search. */
11. Schedule the job set \( J_1 \) by the VIAB algorithm; i.e., \( FLAG \leftarrow VIAB(J_1, T, r_1) \). /* Call VIAB\((J_1, T, r_1)\) to test whether all the jobs can be scheduled on the machines backwards from time \( T \) to \( r_1 \). */

12. If \( FLAG = 1 \) then \( UB \leftarrow L \); else \( LB \leftarrow L \).

13. If \( UB \leq LB \) then stop; else goto Step 3.

Algorithm 4. VIAB\((X, P, r)\)

1. According to the machine capacities, divide the job set \( X \) into three subsets, denoted by \( X_1, X_2, X_3 \), and the machine set into three subsets, denoted by \( M_1, M_2, M_3 \).

2. \( UNSCHED \leftarrow \emptyset \). /* Initialize \( UNSCHED \), the set of unscheduled jobs. */

3. For \( l = 1 \) to \( 3 \)
   
   (a) \( JOBS \leftarrow UNSCHED \cup X^l \). /* The job set to be scheduled. */
   
   (b) Sort the jobs in \( JOBS \) in descending order of their processing times.
   
   (c) Sort the batches in \( B^l \) in descending order of their processing times.
   
   (d) Schedule the batches on \( M^l \) by the First-Fit rule. The scheduling is done backwards from \( P \) towards \( r \). No job can complete earlier than \( r \).
   
   (e) If all the batches can be scheduled, then \( UNSCHED \leftarrow \emptyset \); else \( UNSCHED \leftarrow \) all the jobs in the unscheduled batches.

4. If \( UNSCHED = \emptyset \) then record the completion times of all the machines in \( P \) and return “1”; else return “0”.

In Steps 3(b) and 3(d) of the VIAB algorithm, the methods of batching and scheduling are both based on the First-Fit rule.

5.2. The other three heuristics. There is another algorithm of batching the jobs, as well as scheduling the batches, that is based on the Best-Fit rule. Thus, besides FF, there are three other heuristics, depending on whether we use the First-Fit or the Best-Fit rule in Steps 3(b) and 3(d) of the VIAB algorithm. In the FB heuristic, the First-Fit rule is used in Step 3(b) and the Best-Fit rule is used in Step 3(d). In the BF heuristic, the Best-Fit rule is used in Step 3(b) and the First-Fit rule is used in Step 3(d). Finally, in the BB heuristic, the Best-Fit rule is used in both Step 3(b) and Step 3(d).

6. Computational experiments. To evaluate the performance of the four heuristics, a series of computational experiments are performed. As stated in Section 2, the heuristics proposed by Xu and Bean [45] and Wang and Chou [43] do not work for our model, since they assumed that all the jobs fit in all the machines. Additionally, since Damodaran et al. [9] assumed that all the jobs arrived at time zero, their PSO algorithm cannot apply to our problem either. Therefore, our experiments will compare the proposed heuristics to the lower bound as well as against each other.

6.1. Parameter settings. The number of machines is set to be 10; i.e., \( m = 10 \). Three machine capacities are chosen; \( S_1 = 10, S_2 = 25 \) and \( S_3 = 65 \). The larger machine capacity is approximately 2.5 times that of the smaller machine capacity. In real-life applications, the machines with larger capacity are newer and more expensive than machines with smaller capacity. Therefore, the number of machines with larger capacity is usually less than the number of machines with
smaller capacity. Hence, we choose five, three, and two machines with capacity $S_1$, $S_2$ and $S_3$, respectively. There are six distinct job numbers ($n$) chosen from 90 to 180, increasing by 18 jobs every time; i.e., $n = \{90, 108, 126, 144, 162, 180\}$. For each $n$, we randomly generate 10 instances. For each instance, we set three groups of jobs, denoted by $J^1$, $J^2$ and $J^3$. The jobs in $J^1$ can be scheduled on all 10 machines, the jobs in $J^2$ can only be scheduled on five machines and the jobs in $J^3$ can only be scheduled on two machines. The numbers of jobs in the three groups are chosen to be $2n/3$, $2n/9$ and $n/9$, respectively. We choose four different job release times, $r_1 = 0, r_2 = 20, r_3 = 40, r_4 = 60$. The release time of each job is randomly selected from $\{0, 20, 40, 60\}$. After the release times are generated, we partition the jobs according to their arrival times. Each partition is further divided into three subgroups; i.e., $J^1, J^2, J^3$.

The processing times of the jobs are uniformly distributed in $[5, 15]$. The parameter setting of the job sizes is a key point in the experiment. If the job sizes are generated by uniform distribution, the probability of obtaining a small-size job is the same as that of a large-size job. Consequently, the number of jobs in a batch is relatively small, which makes the problem relatively easy to solve. If there are more small-size jobs than large-size jobs, then the heuristics will be able to assign small-size jobs to machines with larger capacities, which will complicate the process of scheduling. Therefore, in the experiment, the job sizes are generated by Poisson distribution. That is, $\{s_j \mid j \in J^1\} \sim P(\lambda_1), \{s_j \mid j \in J^2\} \sim P(\lambda_2)$ and $\{s_j \mid j \in J^3\} \sim P(\lambda_3)$, where $\lambda_1 = S_l/2 l = 1, 2, 3$. Specifically, $\lambda_1 = 5, \lambda_2 = 12.5$ and $\lambda_3 = 32.5$. If job $j$ belongs to class $l$ and the size generated is smaller than $S_{l-1}$, then we let $s_j$ to be $S_{l-1}$. Similarly, if it is larger than $S_l$, then we let it be $S_l$. The value assigned to $s_j$ is given as follows:

$$s_j = \begin{cases} S_{l-1} & s_j < S_{l-1}, \\ s_j, & S_{l-1} \leq s_j \leq S_l, \\ S_l, & s_j > S_l, \end{cases} \tag{10}$$

where $S_0 = 1$. Furthermore, since it is desirable to have more small-size jobs than large-size jobs, we randomly select 70% of jobs from the interval $(S_{l-1}, S_l/2]$ and 30% from the interval $(S_l/2, S_l]$. Table 1 shows the parameter setting chosen for the experiment.

| factors                   | categories and values             |
|---------------------------|----------------------------------|
| machine capacities        | $S_1 = 10, S_2 = 25, S_3 = 65$   |
| machine numbers           | $m_1 = 5, m_2 = 3, m_3 = 2$      |
| job numbers $n$           | $n = \{90, 108, 126, 144, 162, 180\}$ |
| processing times of jobs  | $U[5,15]$                        |
| job arrival times         | $r_0 = 0, r_1 = 20, r_3 = 40, r_4 = 60$ |

6.2. **Experimental results and analysis.** Each algorithm is programmed in C++ and executed on a PC with an Intel Core 2 processor and 2G RAM. To compare the algorithms objectively and accurately, we present the results of each instance as well as the average results of each group of 10 instances. We use the relative distance between the solution and the lower bound as the measure of the
The percentage of the relative difference between the objective value of algorithm A and the lower bound, denoted by $R_A$, is computed by

$$R_A = \left( \frac{C_{\text{max}}^A}{LB} - 1 \right) \times 100$$

where $C_{\text{max}}^A$ is the makespan obtained by algorithm A and LB is the lower bound of the instance computed by Algorithm 1 in Section 4. The smaller the value of $R_A$, the better the algorithm.

Tables 2 to 7 present the results for different numbers of jobs. In each table, column 1 represents the index number of the instance. Column 2 gives the LB value. Columns 3 to 6 report the percentage of relative difference of the FF, FB, BF and BB algorithms, respectively. The numbers in the last row give the average results of the 10 instances.

Table 2 presents the results obtained by the algorithms for instances with 90 jobs. The $R$ values of the BB algorithm are better than the FF, FB and BF algorithms in every instance; it is better than 3% on the average. There are four instances in which BB obtains the same makespan as LB. The FB algorithm is the second best. FF and BF are the worst and their average performances are about the same.

Table 3 summarizes the results for instances with 108 jobs. Except a tie in one instance, the $R$ values of the BB algorithm outperform the FF algorithm in every instance, with a wide margin. The $R$ values of the BB algorithm also outperform the FB algorithm, except in three instances. The BB algorithm outperforms the BF algorithm in eight instances, loses in one instance and ties in one instance. However, the average $R$ value of the BB algorithm is better than that of the FB algorithm, which in turn is better than the BF and FF algorithms.

Table 4 shows the results for instances with 126 jobs. The $R$ value of the BB algorithm outperforms the FF, FB and FB algorithms in every instance. The $R$ value of the FB algorithm outperforms FF in every instance; with the exception of two instances, it also outperforms BF in every instance.

Table 5 gives the results for instances with 144 jobs. The $R$ value of the BB algorithm outperforms the BF and FF algorithms in every instance. With the exception of two instances, the BB algorithm outperforms the FB algorithm in every instance. The BB algorithm has the best average $R$ value, followed by FB, BF and FF.

Table 6 presents the results for instances with 162 jobs. The $R$ value of the BB algorithm outperforms the other three algorithms in all instances. The FB algorithm outperforms BF in every instance, with the exception of one instance; it also outperforms FF in every instance, with the exception of two instances. Accordingly, the average $R$ value of BB is the best, followed by FB, BF and FF.

Table 7 shows the results for instances with 180 jobs. The $R$ value of the BB algorithm outperforms the FF algorithm in every instance. The BB algorithm beats the FB algorithm in nine instances and loses in one instance. It also beats the BF algorithm in nine instances and ties in one instance. Moreover, the average $R$ value of the BB algorithm over the ten instances is the best. The BF algorithm is the second best algorithm, while the FF algorithm shows the worst performance.

Fig. 1 depicts the comparison of the average values of each group of 10 instances for different numbers of jobs. The □, ▼, × and + symbols are for the $R$ values of the FF, FB, BF and BB algorithms, respectively. As can be seen from Fig. 1, the performance of the BB algorithm is the best among the four algorithms. The FB and the BF algorithms have more or less the same performance, and both of them
are better than the FF algorithm. It can also be observed that the performances of the four algorithms become worse when the number of jobs increases from 90 to 144.

![Figure 1. Comparison of solution quality and robustness on the problems with different numbers of jobs.](image)

| Ins. No. | LB | $R_{FF}$ | $R_{FB}$ | $R_{BF}$ | $R_{BB}$ |
|---------|----|---------|---------|---------|---------|
| 1       | 75 | 1.33    | 1.33    | 1.33    | 0.00    |
| 2       | 75 | 5.33    | 4.00    | 5.33    | 0.00    |
| 3       | 75 | 4.00    | 2.67    | 4.00    | 0.00    |
| 4       | 75 | 4.00    | 4.00    | 4.00    | 2.67    |
| 5       | 75 | 6.67    | 4.00    | 5.33    | 2.67    |
| 6       | 75 | 13.33   | 10.67   | 13.33   | 8.00    |
| 7       | 75 | 10.67   | 8.00    | 10.67   | 1.33    |
| 8       | 75 | 10.67   | 10.67   | 10.67   | 6.67    |
| 9       | 75 | 1.33    | 1.33    | 4.00    | 0.00    |
| 10      | 75 | 8.00    | 8.00    | 8.00    | 4.00    |
| Average | 75 | 6.53    | 5.47    | 6.67    | 2.53    |

As can be observed from the experimental results, the overall performance of the BB algorithm is better than that of the other three algorithms, which demonstrates the effectiveness and efficiency of the BB algorithm. This is due to the effective mechanisms used by the BB algorithm. That is, the Best-Fit batching approach is better than the First-Fit batching approach. Moreover, it is better to schedule the batches on the machines by the Best-Fit rule than the First-Fit rule.
Table 3. Results for instances with 108 jobs.

| Ins. No. | LB | R_FF | R_FB | R_BF | R_BB |
|----------|----|------|------|------|------|
| 1        | 75 | 13.33| 9.33 | 14.67| 6.67 |
| 2        | 75 | 10.67| 6.67 | 10.67| 4.00 |
| 3        | 75 | 14.67| 10.67| 12.00| 6.67 |
| 4        | 75 | 21.33| 13.33| 20.00| 8.00 |
| 5        | 75 | 14.67| 10.67| 17.33| 6.67 |
| 6        | 75 | 14.67| 12.00| 14.67| 13.33|
| 7        | 75 | 10.67| 8.00 | 10.67| 10.67|
| 8        | 75 | 21.33| 12.00| 8.00 | 13.33|
| 9        | 75 | 20.00| 16.00| 21.33| 13.33|
| 10       | 75 | 17.33| 14.67| 17.33| 10.67|

Average 75 | 15.87| 11.33| 14.67| 9.33 |

Table 4. Results for instances with 126 jobs.

| Ins. No. | LB | R_FF | R_FB | R_BF | R_BB |
|----------|----|------|------|------|------|
| 1        | 80 | 21.25| 16.25| 20.00| 12.50|
| 2        | 75 | 34.67| 30.67| 32.00| 20.00|
| 3        | 75 | 28.00| 24.00| 24.00| 16.00|
| 4        | 75 | 36.00| 32.00| 34.67| 22.67|
| 5        | 83 | 21.69| 18.07| 20.48| 16.87|
| 6        | 75 | 29.33| 28.00| 28.00| 21.33|
| 7        | 78 | 24.36| 20.51| 23.08| 14.10|
| 8        | 79 | 27.85| 22.79| 26.58| 16.46|
| 9        | 77 | 27.27| 18.18| 25.97| 11.69|
| 10       | 77 | 22.08| 20.78| 22.08| 19.48|

Average 77.4 | 27.25| 23.13| 25.69| 17.11 |

Table 5. Results for instances with 144 jobs.

| Ins. No. | LB | R_FF | R_FB | R_BF | R_BB |
|----------|----|------|------|------|------|
| 1        | 86 | 38.37| 36.05| 38.37| 30.23|
| 2        | 89 | 23.60| 20.23| 20.23| 14.61|
| 3        | 86 | 32.56| 29.07| 31.40| 26.74|
| 4        | 76 | 35.53| 30.26| 34.21| 27.63|
| 5        | 79 | 40.51| 32.91| 37.98| 24.05|
| 6        | 75 | 34.67| 29.33| 36.00| 32.00|
| 7        | 83 | 30.12| 24.10| 28.92| 21.69|
| 8        | 82 | 25.61| 19.51| 23.17| 20.73|
| 9        | 88 | 25.00| 23.86| 25.00| 17.05|
| 10       | 78 | 41.03| 37.18| 41.03| 32.05|

Average 82.2 | 32.70| 28.25| 31.63| 24.68 |

7. Conclusions. In this paper, we study the batch scheduling of jobs with non-identical sizes and release times on machines with non-identical capacities, so as to minimize the makespan. We give a method to compute a lower bound. Based on the First-Fit and the Best-Fit rules used in bin-packing, we present four heuristics to solve the problem. Computational experiment shows that the BB algorithm (which batches the jobs and schedule the batches by the Best-Fit rule) outperforms the other heuristics.

Future research will focus on extending the proposed algorithms to more complex problems, such as the jobs are incompatible, or other constraints. Also, another
interesting direction is to design meta-heuristics with better performance to solve the studied problem. Other optimization objectives such as maximum lateness and mean weighted flow time are unexplored either.

Acknowledgments. The authors would like to thank the anonymous referees for the helpful suggestions on the presentation of this paper. The work of the First author is supported by the National Natural Science Foundation of China under grants 71471052, 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.

REFERENCES

[1] C. Almeder and L. Mönch, Metaheuristics for scheduling jobs with incompatible families on parallel batch machines, *Journal of the Operational Research Society*, 62 (2011), 2083–2096.
[2] A. Bilyk, L. Mönch and C. Almeder, Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics, *Computers & Industrial Engineering*, 78 (2014), 175–185.
[3] P. Brucker, A. Gladky, H. Hoogeveen, M.-Y. Kovalyov, C.-N. Potts, T. Tautenhahn and S.-L. Van de Velde, Scheduling a batching machine, *Journal of Scheduling*, 1 (1998), 31–45.
[4] P.-Y. Chang, P. Damodaran and S. Melouk, Minimizing makespan on parallel batch processing machines, *International Journal of Production Research*, 42 (2004), 4211–4220.
[5] H.-P. Chen, B. Du and G. Huang, Metaheuristics to minimize makespan on parallel batch processing machines with dynamic job arrivals, *International Journal of Computer Integrated Manufacturing*, 23 (2010), 942–956.

[6] B.-Y. Cheng, K. Li and B. Chen, Scheduling a single batch-processing machine with non-identical job sizes in fuzzy environment using an improved ant colony optimization, *Journal of Manufacturing Systems*, 29 (2010), 29–34.

[7] B.-Y. Cheng, S.-L. Yang, X.-X. Hu and B. Chen, Minimizing makespan and total completion time for parallel batch processing machines with non-identical job sizes. *Applied Mathematical Modelling*, 36 (2012), 3161–3167.

[8] P. Damodaran and P.-Y. Chang, Heuristics to minimize makespan of parallel batch processing machines. *International Journal of Advanced Manufacturing Technology*, 37 (2008), 1005–1013.

[9] P. Damodaran, D.-A. Diyadawagamage, O. Ghelayeb and M.-C. Véliz-Gallego, A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines. *International Journal of Advanced Manufacturing Technology*, 58 (2012), 1151–1140.

[10] P. Damodaran, P.-K. Maneshwar and K. Srihari, Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms. *International Journal of Production Economics*, 103 (2006), 882–891.

[11] L. Dupont and C. Dhaenens-Flipo, Minimizing the makespan on a batch machine with non-identical job sizes: An exact procedure. *Computers & Operations Research*, 29 (2002), 807–819.

[12] L. Dupont and F. Jolai Ghazvini, Minimizing makespan on a single batch processing machine with non identical job sizes, *European Journal of Automation System*, 32 (1998), 431–440.

[13] B.-Q. Fan, J.-J. Yuan and S.-S. Li, Bi-criteria scheduling on a single parallel-batch machine. *Applied Mathematical Modelling*, 36 (2012), 1338–1346.

[14] F.-J. Ghazvini and L. Dupont, Minimizing mean flow time criteria on a single batch processing machine with non-identical job sizes. *International Journal of Production Economics*, 55 (1998), 273–280.

[15] R.-L. Graham, E.-L. Lawler, J.-K. Lenstra and A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5 (1979), 287–326.

[16] Y. Huo and J.-Y.-T. Leung, Fast approximation algorithms for job scheduling with processing set restrictions. *Theoretical Computer Science*, 411 (2010), 3947–3955.

[17] Y. Ikura and M. Gimple, Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letter*, 5 (1986), 61–65.

[18] Z.-H. Jia and J.-Y.-T. Leung, An improved meta-heuristic for makespan minimization of a single batch machine with non-identical job sizes. *Computers & Operations Research*, 46 (2014), 49–58.

[19] A.-H. Kashan, B. Karimi and M. Jenabi, A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers & Operations Research*, 35 (2008), 1084–1098.

[20] A.-H. Kashan, B. Karimi and F. Jolai, Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non-identical job sizes. *International Journal of Production Research*, 44 (2006), 2357–2360.

[21] A.-H. Kashan, M.-H. Kashan and S. Karimiyan, Particle swarm optimizer for grouping problems. *Information Sciences*, 252 (2013), 81–95.

[22] A. Klemmt, G. Weigert, C. Almeder and L. Mönch, A Comparison of MIP-based Decomposition Techniques and VNS Approaches for Batch Scheduling Problems. *Proceedings of the Modeling and Analysis of Semiconductor Manufacturing Conference (MASM)*, 2009, 1686–1694.

[23] M.-E. Kurz and S.-J. Mason, Minimizing total weighted tardiness on a batch-processing machine with incompatible job families and job ready times. *International Journal of Production Research*, 46 (2008), 131–151.

[24] C.-Y. Lee, R. Uzsoy and L.-A. Martin-Vega, Efficient algorithms for scheduling semi-conductor burn-in operations. *Operations Research*, 40 (1992), 764–775.

[25] C.-Y. Lee and R. Uzsoy, Minimizing makespan on a single batch minimizing makespan on a single burn-in oven in semiconductor manufacturing. *European Journal of Operational Research*, 120 (2000), 559–574.
[26] J.-Y.-T. Leung and C.-L. Li, Scheduling with processing set restrictions: A survey, *International Journal of Production Economics*, 116 (2008), 251–262.

[27] S.-G. Li, G.-J. Li, X.-L. Wang and Q.-M. Liu, Minimizing makespan on a single batching machine with release times and non-identical job sizes, *Operations Research Letters*, 33 (2005), 157–164.

[28] M. Liu, Parallel-machine scheduling with past-sequence-dependent delivery times and learning effect, *Applied Mathematical Modelling*, 37 (2013), 9630–9633.

[29] S. Malve and R. Uzsoy, A genetic algorithm for minimizing maximum lateness on parallel identical batch processing machines with dynamic job arrivals and incompatible job families, *Computers & Operations Research*, 34 (2007), 3061–3028.

[30] M. Mathirajan and A.-I. Sivakumar, A literature review, classification and simple metaanalysis on scheduling of batch processors in semiconductor, *International Journal of Advanced Manufacturing Technology*, 29 (2006), 990–1001.

[31] L. Mönch and C. Ahneder, Ant colony optimization approaches for scheduling jobs with incompatible families on parallel batch machines, *Proceedings 4th Multi-disciplinary International Conference on Scheduling: Theory and Applications (MISTA)*, Dublin, Ireland, 2009, 106–114.

[32] L. Mönch, J.-W. Fowler, S. Dauzere-Peres, S.-J. Mason and O. Rose, A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations, *Journal of Scheduling*, 14 (2011), 583–599.

[33] J. Ou, J.-Y.-T. Leung and C.-L. Li, Scheduling parallel machines with inclusive processing set restrictions, *Naval Research Logistics*, 55 (2008), 328–338.

[34] C. Potts and M. Kovalyov, Scheduling with batching: A review, *European Journal of Operational Research*, 120 (2000), 228–249.

[35] H. Shao, H.-P. Chen and G. Huang, Minimizing makespan for parallel batch processing machines with non-identical job sizes using neural nets approach, *Proceedings of the 23rd IEEE Conference on Industrial Electronics and Applications, Singapore*, (2008), 1921–1924.

[36] C.-S. Sung and Y.-I. Choung, Minimizing makespan on a single burn-in oven in semiconductor manufacturing, *European Journal of Operational Research*, 120 (2000), 559–574.

[37] L.-X. Tang and H. Gong, A hybrid two-stage transportation and batch scheduling problem, *Applied Mathematical Modelling*, 32 (2008), 2467–2479.

[38] R. Uzsoy, Scheduling a single batch processing machine with non-identical job sizes, *International Journal of Production Research*, 32 (1994), 1615–1635.

[39] R. Uzsoy, Scheduling batch processing machines with incompatible job families, *International Journal of Production Research*, 33 (1995), 2685–2708.

[40] R. Uzsoy and Y. Yang, Minimizing total weighted completion time on a single batch processing machine, *Production and Operations Management*, 6 (1997), 57–73.

[41] M. Venkataramana and N.-R. Srinivasa Raghavan, Ant colony-based algorithms for scheduling parallel batch processors with incompatible job families, *International Journal of Mathematics in Operational Research*, 2 (2010), 73–98.

[42] C.-S. Wang and R. Uzsoy, A genetic algorithm to minimize maximum lateness on a batch processing machine, *Computers & Operations Research*, 29 (2002), 1621–1640.

[43] H.-M. Wang and F.-D. Chou, Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics, *Expert Systems with Applications*, 37 (2010), 1510–1521.

[44] X.-Y. Wang and J.-J. Wang, Scheduling deteriorating jobs with a learning effect on unrelated parallel machines, *Applied Mathematical Modelling*, 38 (2014), 5231–5238.

[45] S.-B. Xu and J.-C. Bean, A genetic algorithm for scheduling parallel non-identical batch processing machines, *IEEE Symposium on Computational Intelligence in Scheduling, SCIS'07*, (2007), 143–150.

[46] R. Xu, H.-P. Chen and X.-P. Li, Makespan minimization on single batch-processing machine via ant colony optimization, *Computers & Operations Research*, 39 (2012), 582–593.

[47] R. Xu, H.-P. Chen and X.-P. Li, A bi-objective scheduling problem on batch machines via a pareto-based ant colony system, *International Journal of Production Economics*, 145 (2013), 371–386.

[48] N. Yin, L.-Y. Kang, T.-C. Sun, C. Yue and X.-R. Wang, Unrelated parallel machines scheduling with deteriorating jobs and resource dependent processing times, *Applied Mathematical Modelling*, 38 (2014), 4717–4755.
[49] G. Zhang, X. Cai, C.-Y. Lee and C.-K. Wong. Minimizing makespan on a single batch processing machine with nonidentical job sizes. *Naval Research Logistics*, 48 (2001), 226–240.

Received August 2015; 1st revision March 2016; 2nd revision June 2016.

E-mail address: zhjia@mail.ustc.edu.cn
E-mail address: wentingtingahu@163.com
E-mail address: leung@oak.njit.edu
E-mail address: hfutlk@139.com