Research Article

Xiao Xin, Muhammad Ijaz Khan, and Shuguang Li*

Scheduling equal-length jobs with arbitrary sizes on uniform parallel batch machines

https://doi.org/10.1515/math-2022-0562
received August 5, 2022; accepted January 31, 2023

Abstract: We consider the problem of scheduling jobs with equal lengths and arbitrary sizes on uniform parallel batch machines with different capacities. Each machine can only process the jobs whose sizes are not larger than its capacity. Several jobs can be processed as a batch simultaneously on a machine, as long as their total size does not exceed the machine's capacity. The objective is to minimize makespan. Under a divisibility constraint, we obtain two efficient exact algorithms. For the general problem, we obtain an efficient 2-approximation algorithm. Previous work has shown that the problem cannot be approximated to within an approximation ratio better than 2, unless P = NP, even when all machines have identical speeds and capacities.

Keywords: scheduling, uniform parallel batch machines, job sizes, equal job lengths, makespan

MSC 2020: 90B35, 68Q25

1 Introduction

In today's competitive environment, batch processing is a very common procedure in many manufacturing fields such as metalworking, and wafer fabrication for the avoidance of setups and/or facilitation of material handling [1,2]. A batch is defined as a group of jobs that have to be processed jointly and a batch scheduling problem consists of grouping jobs on each machine into batches that are scheduled either in serial (called serial batch) or in parallel (called parallel batch). For the serial batch, the processing time of a batch equals to the sum of the processing times of its jobs, while for the parallel batch, the processing time of a batch equals to the largest processing time of its jobs. In this article, we consider the parallel batch scheduling model.

There are numerous applications of parallel batch mentioned in the literature, such as aircraft manufacturing, automobile gear manufacturing and healthcare. However, the bulk of the literature on parallel batch scheduling deals with the semiconductor industry [2–4], which has already become one of the largest industries in the world. Nowadays microprocessors, memory chips and other semiconductor-related devices appear in electronics as a part of real daily life ranging from personal computers to cellular phones. Semiconductor manufactures need to utilize their resources effectively to confront the huge demand and severe competition in the marketplace. Consequently, efficient scheduling is of great concern to overall performance.

* **Corresponding author: Shuguang Li,** College of Computer Science and Technology, Shandong Technology and Business University, Yantai 264005, China, e-mail: sgliytu@hotmail.com
Xiao Xin: College of Foreign Studies, Shandong Technology and Business University, Yantai 264005, China; Shandong Co-Innovation Center of Future Intelligent Computing, Shandong Technology and Business University, Yantai 264005, China
Muhammad Ijaz Khan: Department of Mechanical Engineering, Lebanese American University, Beirut 362060, Lebanon; Department of Mechanics and Engineering Science, Peking University, Beijing 100871, China

Real-world production systems usually require batch processing machines arranged in parallel in order to prevent the system from being blocked by the unavailability (e.g., breakdown) of a single machine. To make a scheduling problem closer to the real-world situation, the constraints of non-identical job sizes should be considered. Moreover, batch processing machines often have different capacities and may run at different speeds.

1.1 Problem formulation

In this article, we consider the following problem of scheduling n independent jobs on m uniform parallel batch machines. Let $\mathcal{J} = \{1, 2, \ldots, n\}$ denote the set of all jobs and $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ the set of all machines. Job $j \in \mathcal{J}$ $(j = 1, 2, \ldots, n)$ has a *length* $p_j \geq 0$ and a *size* $s_j > 0$. All jobs are available from time zero onwards. Machine M_i $(i = 1, 2, \ldots, m)$ has a *speed* v_i and a *capacity* K_i. The *processing time* of job j is p_j / v_i if j is processed on machine M_i. The machines are indexed such that $K_1 \leq K_2 \leq \cdots \leq K_m$. For any job j, $s_j \leq K_m$ holds. However, it is possible that $s_j > K_i$ for some i and j. Machine M_i can process several jobs as a batch simultaneously as long as the total size of these jobs does not exceed K_i. The *length* $p(B_g)$ of batch B_g is determined by the longest length of all the jobs in B_g. The *processing time* of batch B_g is $p(B_g)/v_i$ if B_g is processed on machine M_i. The goal is to find a schedule to minimize the *makespan* which is defined as $C_{\max} = \max_j C_j$, where C_j denotes the completion time of job j in the schedule. Using the notation proposed in [5,6], this problem can be denoted as $Q|s_j, p\text{-batch}, K_i|C_{\max}$. We focus on the special case where all jobs have equal lengths, i.e., $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$. The special case of $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ where all machines have identical speeds is denoted as $P|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$.

Note that $1|s_j, p_j = p, p\text{-batch}, B|C_{\max}$ (the special case of $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ where $m = 1$) is just the standard bin-packing problem [7]. It is strongly NP-hard and cannot be approximated to within an approximation ratio better than 3/2, unless P = NP [8]. On the other hand, Dosa et al. [9] proved that $P|s_j, p_j = p, p\text{-batch}, B|C_{\max}$ cannot be approximated to within an approximation ratio better than 2, unless P = NP. Hence, $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ cannot be approximated to within an approximation ratio better than 2, unless P = NP.

1.2 Literature review

In recent years, a body of research has started to address the parallel batch scheduling problems. For a detailed review of the existing results, please refer to [1–4].

This article is motivated by [10,11]. Wang and Leung [10] studied $P|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$, and presented a 2-approximation algorithm. They also obtained an algorithm with asymptotic approximation ratio 3/2. Ozturk et al. [11] presented a 2-approximation algorithm for $P|r_j, s_j, p_j = p, p\text{-batch}, B|C_{\max}$ (jobs have unequal release times and machines have identical capacities B). Inspired from a special case of the bin-packing problem [12], they also presented an efficient exact algorithm for a special case of $P|r_j, s_j, p_j = p, p\text{-batch}, B|C_{\max}$ where job sizes form a strongly divisible sequence (to be defined in Section 3), denoted as $P|r_j, s_j(\textit{strongly divisible}), p_j = p, p\text{-batch}, B|C_{\max}$.

Although the problem of scheduling parallel batch machines has been studied extensively in the last few decades, relatively little research has been done on the problem of scheduling uniform parallel batch machines with non-identical capacities [2,3]. Also, some research considered non-identical job sizes [13,14]. Li et al. [13] proposed several heuristics for the problem of scheduling uniform parallel batch machines with non-identical job sizes, unequal release times, identical capacities and makespan minimization. Zhou [14] proposed an effective discrete differential evolution (DDE) algorithm for the problem of scheduling uniform parallel batch machines with non-identical job sizes, unequal release times, non-identical capacities and makespan minimization. Both Li et al. [13] and Zhou et al. [14] assumed that largest job size fits in the

machine with the smallest capacity, i.e., $s_j \le K_1$ for all jobs j. Li et al. [15] proposed several heuristics for the problem of scheduling unrelated (the processing time of job j is p_{ji} if j is processed on machine M_i) parallel batch machines with non-identical job sizes, equal release times, identical capacities and makespan minimization. Arroyo and Leung [16] proposed several heuristics for the problem of scheduling unrelated parallel batch machines with non-identical job sizes, unequal release times, identical capacities and makespan minimization.

Several papers are concerned with the problem of scheduling parallel batch machines (all machines have identical speeds) with non-identical job sizes, non-identical capacities and makespan minimization [10,17–22]. Among them, [17,18] assumed that any job can fit in any machine, which is different from the problem we study in this article; while in [10,19–22] there is no such restriction.

Many recent research results contain complexity, mathematical models, as well as heuristic/meta-heuristic algorithms and are also related to different production areas. Some of them but not all, please see [23–28]. Note that in [28], Li discussed the problem of scheduling equal-length jobs with processing set restrictions on uniform parallel batch machines, where all jobs have the same size. However, in this article, we focus on the case of arbitrary job sizes.

1.3 Contribution and organization

For the case of equal release times, we can extend the results obtained in [10,11] to uniform parallel batch machines, allowing non-identical capacities. To the best of our knowledge, $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ has not been studied to date. Only its special cases $P|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ and $P|s_j, p_j = p, p\text{-batch}, B|C_{\max}$ have been studied in [10,11].

The organization of the article is as follows. Section 2 provides preliminaries and notations. In Section 3, we consider the special case of $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ where job sizes form a divisible sequence (to be defined in Section 3), denoted as $Q|s_j(\textit{divisible}), p_j = p, p\text{-batch}, K_i|C_{\max}$. We present two efficient exact algorithms for this case. In Section 4, we present an efficient 2-approximation algorithm for $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$. In Section 5, we conclude this article and discuss future research directions.

2 Preliminaries and notations

Before we proceed, we introduce some useful notations. Let $K_0 = 0$. Let $\mathcal{J}_i = \{j \in \mathcal{J} | K_{i-1} < s_j \le K_i\}$, $i = 1, 2, \ldots, m$. It is possible that $\mathcal{J}_i = \varnothing$ for some i. We have $\mathcal{J} = \bigcup_{i=1}^{m} \mathcal{J}_i$. Call $\{M_i, M_{i+1}, \ldots, M_m\}$ the *processing set* of job $j \in \mathcal{J}_i$. Call $M_i, M_{i+1}, \ldots, M_m$ the *eligible machines* of job $j \in \mathcal{J}_i$. Job $j \in \mathcal{J}_i$ is called *eligible* for each machine in $\{M_i, M_{i+1}, \ldots, M_m\}$.

Throughout this article, let OPT denote the makespan of an optimal schedule for the problem under consideration. Clearly, there is an optimal schedule in which the batches assigned to the same machine are processed successively, without any idle time between them. There are at most n batches on each machine, and the kth batch on M_i starts at time $(k-1)p/v_i$, $k = 1, 2, \ldots$, $i = 1, 2, \ldots, m$. Thus, as in [29], we consider at most mn possible values for OPT. These values can be sorted in ascending order in $O(mn \log m)$ time by merging m-ordered lists of length $O(n)$ using a divide-and-conquer approach (see [30], Section 2.7).

3 Scheduling with divisible job sizes

In this section, we consider a special case where the job sizes form a *divisible sequence*, i.e., for any two jobs j_1 and j_2 in $\mathcal{J}$ such that $s_{j_1} \ge s_{j_2}$, s_{j_2} exactly divides s_{j_1}. We also say that $\mathcal{J}$ has *divisible job sizes*. For a given

batch capacity B, the pair $(\mathcal{J}, B)$ is called *weakly divisible* if $\mathcal{J}$ has divisible job sizes and *strongly divisible* if in addition the largest job size exactly divides B [12].

Divisible job sizes are of interest because they arise naturally in certain applications and because some NP-hard problems produce substantially better solutions for such sets of jobs, see, e.g., [11,12,31–34].

Ozturk et al. [11] presented an exact algorithm running in $O(n^2 \log n)$ time for $P|r_j, s_j$ (strongly divisible), $p_j = p$, p-batch, $B|C_{\max}$ (job sizes and the common machine capacity are strongly divisible). For the case of equal release times, we can generalize the result to uniform parallel batch machines with non-identical capacities and require only the weakly divisible job sizes. The problem can be denoted as $Q|s_j(\textit{divisible}), p_j = p, p\text{-batch}, K_i|C_{\max}$.

Lemma 1. [12] *If $\mathcal{J}$ has divisible job sizes, then any set of jobs from $\mathcal{J}$ that individually do not exceed s_j and that in total sum to at least s_j must contain a subset that sums exactly to s_j, where s_j denotes the size of job j.*

At the beginning, we do the following preprocessing in $O(n \log n)$ time: Sort all the jobs in $\mathcal{J}_i$ ($i = 1, 2, \ldots, m$) in non-increasing order of their sizes, and let J_i denote the ordered list.

We use a binary search to determine OPT in $O(\log(mn))$ iterations. In each iteration, we select a target makespan T and attempt to construct a schedule with makespan at most T by the following Assignment A procedure, or determine that it is impossible to do so. The procedure handles the machines in increasing order of their indices. (Recall that the machines are indexed in the non-decreasing order of their capacities.) Suppose that the procedure is handling machine M_i. In the procedure, U_i represents the ordered list of unassigned eligible jobs for M_i. The jobs in U_i are sorted in non-increasing order of their sizes. From among the jobs in U_i which can be accommodated in a batch (ties broken in favor of the earliest batch) on M_i, the procedure selects and assigns the job which has the largest size into this batch. Repeat this process until any job in U_i cannot be filled into any batch on M_i. Then the procedure proceeds to handle machine M_{i+1}.

Assignment A(T):

Step 1. Let $U_0 = \varnothing$, $i = 1$.

Step 2. While $i \le m$, do:

(i) Let $U_i = J_i || U_{i-1}$, where $||$ denotes the concatenation of the ordered lists J_i and U_{i-1}. Imagine there are $\min\{n, \lfloor T \cdot v_i / p \rfloor\}$ empty batches of length p and capacity K_i processed successively on machine M_i, with the first one started at time zero. (Do not construct these batches explicitly.)

(ii) If $U_i \neq \varnothing$, then we assign the first job in U_i into a batch on M_i such that the total size of the jobs in this batch does not exceed K_i (ties broken in favor of the batch which is started earliest). Delete this job from U_i. Repeat this process until $U_i = \varnothing$ or each of the $\min\{n, \lfloor T \cdot v_i / p \rfloor\}$ batches on M_i has an unused capacity less than the size of the first job in U_i. In the latter case, we do a binary search in U_i to find the largest job which can be filled into a non-empty batch on M_i without exceeding K_i. Assign this job to such a batch that is started earliest and delete the job from U_i. We repeat the process until any job in U_i cannot be filled into any batch on M_i without exceeding K_i (i.e., each of the $\min\{n, \lfloor T \cdot v_i / p \rfloor\}$ batches on M_i has an unused capacity less than any size of the jobs in U_i).

(iii) Let $i = i + 1$.

Lemma 2. *If* OPT $\le T$, *Assignment A will generate a feasible schedule in $O(m + n \log n)$ time for $Q|s_j$ (divisible), $p_j = p$, p-batch, $K_i|C_{\max}$ whose makespan is at most T.*

Proof. We need to prove that if OPT $\le T$, then $U_m = \varnothing$ when Assignment A terminates. Thus, a feasible schedule will be generated.

Let Σ^* be an optimal schedule. Let Σ be the schedule generated by Assignment A. Let k_i^* denote the number of batches processed on machine M_i in Σ^*. Let k_i denote the number of actually generated (i.e., non-empty) batches processed on machine M_i in Σ.

Label the batches on M_1 in Σ in increasing order of their start times as $B_1, B_2, \ldots, B_{k_1}$. Label the batches on M_2 in Σ in increasing order of their start times as $B_{k_1+1}, B_{k_1+2}, \ldots, B_{k_1+k_2}$. Repeat this label process for the

batches on the other machines. Finally, label the batches on M_m in Σ in increasing order of their start times as $B_{k_1+k_2+\cdots+k_{m-1}+1}, \ldots, B_{k_1+k_2+\cdots+k_{m-1}+k_m}$. Accordingly, label the batches in Σ^* on the corresponding positions as $B_1^*, B_2^*, \ldots, B_{k_1+k_2+\cdots+k_{m-1}+k_m}^*$. Since OPT $\le T$, some batches among $B_1^*, B_2^*, \ldots, B_{k_1+k_2+\cdots+k_{m-1}+k_m}^*$ may be empty.

We will modify Σ^* in such a way that batches $B_1^*, B_2^*, \ldots, B_{k_1+k_2+\cdots+k_{m-1}+k_m}^*$ become $B_1, B_2, \ldots, B_{k_1+k_2+\cdots+k_{m-1}+k_m}$ one by one. In the end, Σ^* is modified into Σ. It follows that $U_m = \varnothing$ when Assignment A terminates.

For simplicity of notation, assume for the moment that batch B_1 consists of jobs $1, 2 \ldots, n'$ such that $s_1 \ge s_2 \ge \cdots \ge s_{n'}$. Consider batch B_1^*. Suppose that B_1^* contains jobs $1, 2 \ldots, h$, but does not contain job $h+1$, where $h+1 \le n'$. Let B^* denote the batch containing job $h+1$ in Σ^* (B^* may be processed on a machine other than M_1). Let A denote the set of jobs in B_1^* but not in B_1. By Step 2(ii) of Assignment A, each job in A has size at most s_{h+1}. Now, by Lemma 1, either the total size of the jobs in A is less than s_{h+1} or some subset of those jobs sums exactly to s_{h+1}. In either case, we can exchange the corresponding jobs in B_1^* and job $h+1$ in B^*, such that job $h+1$ appears in (modified) B_1^*. Repeat the process until batch B_1 appears in (modified) Σ^*.

Repeat the aforementioned process to modify batches $B_2^*, B_3^*, \ldots, B_{k_1+k_2+\cdots+k_{m-1}+k_m}^*$ in order until Σ^* becomes Σ. Hence, Assignment A will generate a feasible schedule whose makespan is at most T.

It remains to offer an implementation of Assignment A, which gives the required time complexity. Clearly, the overall running time of the procedure is dominated by Step 2(ii). Note that we handle an empty batch on M_i only when the first job in U_i (which has the largest size among the jobs in U_i) cannot be accommodated in any non-empty batches on M_i. If an empty batch is handled, at least one job will be filled in it. It follows that $\sum_{i=1}^{m} k_i \le n$.

Each iteration of Step 2(ii) can be executed by a naive quadratic time implementation. Also, there are easy implementations such that each iteration of Step 2(ii) can be executed in sub-quadratic time, leading to an overall running time $O(mn \log n)$ of Assignment A. However, Step 2(ii) behaves very much like the First Fit algorithm for the bin-packing problem [7,35], in that it assigns the jobs depending predominantly on the order of the batches in the original left to right sequence. We can adapt a binary tree data structure used for the First Fit algorithm [35] to execute (all iterations of) Step 2(ii) in sub-quadratic time, thus leading to an overall running time $O(m + n \log n)$ of Assignment A.

Precisely, in the i-iteration of Step 2(ii), we will construct a binary tree of depth $O(\log k_i)$ with k_i leaves corresponding to the actually generated batches $B_{k_1+k_2+\cdots+k_{i-1}+1}, \ldots, B_{k_1+k_2+\cdots+k_{i-1}+k_i}$ on M_i, in sequence from left to right. The k_i leaves are labeled with the unused capacities of the corresponding actually generated batches on M_i. Each internal node is labeled with the larger of the labels of its (at most) two sons, and hence the largest unused capacity over all the actually generated batches corresponding to the leaves which descend from that node.

The tree is constructed incrementally, which is different from [35]. Let *leaves* denote the number of its current leaves. Initially, *leaves* = 1 and the tree consists of only one leaf whose label is K_i. It grows in two phases, but the second phase may possibly not exist. Moreover, the tree grows new leaves only in the first phase. In the second phase, the labels of the nodes may decrease, but no new leaf will appear.

In the first phase, we scan U_i and every job encountered must be assigned to M_i. The first phase ends when $U_i = \varnothing$, or *leaves* $= \min\{n, \lfloor T \cdot v_i / p \rfloor\}$ but the first job in U_i cannot be filled into a non-empty batch on M_i without exceeding K_i. Only in the latter case, the second phase begins. In the second phase, we stop scanning U_i. Instead, we do a binary search in U_i to find the largest job which can be filled into a non-empty batch on M_i without exceeding K_i (ties broken in favor of the first such job in U_i). Assign this job to such a batch that is started earliest and delete it from U_i. We repeat the process until any job in U_i cannot be filled into a non-empty batch on M_i without exceeding K_i, and the second phase ends.

Below, let us illustrate the implementation details.

In the first phase, we scan U_i from the first job to the last job. For each job encountered, we check the label of the tree root. There are two different cases to consider:

Case 1. The label of the root is not less than the size of the job.

In this case, the job can be filled into a batch on M_i which corresponds to a leaf of the tree, without exceeding K_i. To assign the job, we traverse the tree, from the root to a leaf, by always taking the leftmost

node whose label is not less than the size of the job. Fill the job into the batch corresponding to the destination leaf.

The path from the tree root to the destination leaf is an *updating path*. Update the labels of the nodes on the updating path as follows. First, let the label of the destination leaf be equal to its value minus the size of the job. Then, proceed back up the updating path and relabel the label of each internal node with the larger of the labels of its (at most) two sons. Keep *leaves* unchanged. Delete this job from U_i.

Case 2. The label of the root is less than the size of the job.

In this case, the job cannot be filled into a non-empty batch on M_i without exceeding K_i. There are two different cases to consider:

Subcase 2.1. $leaves < \min\{n, \lfloor T \cdot v_i / p \rfloor\}$.

We fill this job into an empty batch which starts earliest on M_i. Delete this job from U_i. If the tree has no node of degree one, then insert a new node as the tree root whose left son is the old root. Insert a new leaf u in the tree as the right son of the lowest node w of degree one. Leaf u corresponds to the batch accommodating the job and its label is K_i minus the size of the job. If u is not on the same depth as all the other leaves, then insert several new nodes $w_1, w_2, \ldots, w_b$ on the edge $\langle w, u \rangle$ such that leaf u is on the same depth as all the other leaves. Node w_1 is the right son of w, while node w_{a+1} is the left son of w_a, $a = 1, 2, \ldots, b-1$, and leaf u becomes the left son of w_b. The path from the tree root to u is an updating path. Update the labels of the nodes back up the updating path such that each internal node is labeled with the larger of the labels of its (at most) two sons. Let $leaves = leaves + 1$.

Subcase 2.2. $leaves = \min\{n, \lfloor T \cdot v_i / p \rfloor\}$.

It indicates that the first phase ends, and we have to enter the second phase.

Constructing the binary tree used in the i-iteration can be done in $O(k_i \log k_i)$ time, since the tree has only k_i leaves and its depth is $O(\log k_i)$. Using the tree, in the first phase, we can assign a job in Q_i into a batch on M_i (Case 1 and Subcase 2.1) in $O(\log k_i)$ time. Since $\sum_{i=1}^{m} k_i \le n$, constructing the binary trees in all m iterations can be accomplished in $O(m + n \log n)$ time. Assigning jobs into batches in the first phase in all m iterations can also be accomplished in $O(m + n \log n)$ time. (Each machine has to be checked once, though some machines may process no batches at all.)

In the second phase, using a binary search in U_i instead of scanning it, we find the largest job in U_i whose size is not larger than the label of the root (ties broken in favor of the first such job in U_i). Assign this job into a non-empty batch on M_i without exceeding K_i via the constructed binary tree (as described in Case 1), and delete it from U_i. We update the labels of the tree accordingly. Repeat this process until any job in U_i has a size larger than the label of the root, indicating that no job can be filled into a non-empty batch on M_i without exceeding K_i. The second phase ends.

Assigning each job to a batch in the second phase can be done in $O(\log n)$ time. Assigning jobs into batches in the second phase in all m iterations can be accomplished in $O(n \log n)$ time.

Hence, the time complexity of Assignment A is $O(m + n \log n)$. □

Example 1. Here is an example to illustrate how the binary tree grows. Fix a particular machine M_i. In U_i, there are six jobs of size 8, six jobs of size 4, two jobs of size 2 and two jobs of size 1, sorted in non-increasing order of their sizes. The job sizes form a divisible sequence. Let $K_i = 13$ and $\min\{n, \lfloor T \cdot v_i / p \rfloor\} = 3$. Each of the first three jobs in U_i will be filled into an empty batch. For ease of explanation, during the illustration we will not delete the assigned jobs from U_i. The state of the binary tree after the first three jobs of size 8 are assigned is shown in Figure 1. The assigned jobs (represented by their sizes) are provided at the leaves of the tree, and the labels of all the nodes are also provided.

The next three jobs of size 8 cannot fit in any non-empty batch and have to be left over (to be assigned in the next iterations). Among them, the first one tells us that the second phase begins. Then, the first three jobs of size 4 can be assigned, and we assign each of them to a different non-empty batch. The state of the binary tree after the first three jobs of size 4 are assigned is shown in Figure 2.

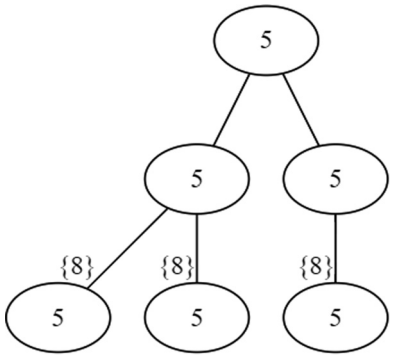

Figure 1: Tree directory for the Implementation of Step 2(ii), after assigning the first three jobs of size 8.

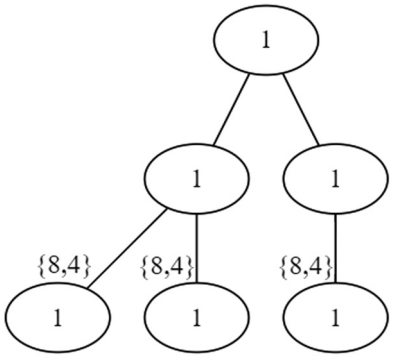

Figure 2: Tree directory for the Implementation of Step 2(ii), after assigning the first three jobs of size 4.

Now, the unused capacity of each existing batch is 1. Neither the next three jobs of size 4 nor the two jobs of size 2 can be filled in. These jobs have to be left over. Only the two jobs of size 1 can be assigned, and we assign them in the first two batches. The state of the binary tree after the two jobs of size 1 are assigned is shown in Figure 3. In final U_i, there are three jobs of size 8, three jobs of size 4 and two jobs of size 2, sorted in non-increasing order of their sizes.

We can use the following Assignment A1 procedure instead of Assignment A. It runs in $O(m + n \log n)$ time, too. The analysis of its correctness and running time is similar to that of Lemma 2 and thus omitted.

The procedure handles the machines in decreasing order of their indices. In the procedure, U represents the ordered list of all unassigned jobs which are sorted in non-increasing order of their sizes. Initially, $U = J$,

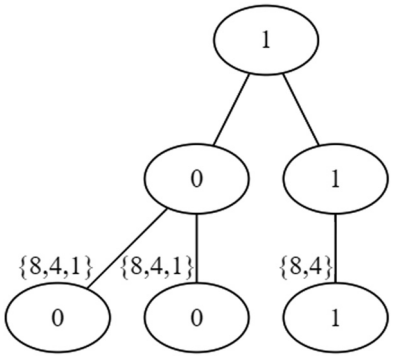

Figure 3: Tree directory for the Implementation of Step 2(ii), after assigning the two jobs of size 1.

where $J = J_m||J_{m-1}||\cdots||J_1$. (Recall that J_i denotes the ordered list of the jobs in $\mathcal{J}_i$ which are sorted in non-increasing order of their sizes, $i = 1, 2, \ldots, m$.) Suppose that the procedure is handling machine M_i. In contrast with Assignment A, Assignment A1 knows the exact number of batches processed on M_i beforehand. Consequently, this procedure is slightly simpler than Assignment A, since we can determine the binary tree structure statically. From among the jobs in U which can be accommodated in a batch (ties broken in favor of the earliest batch) on M_i, the procedure selects and assigns the job which has the largest size into this batch. Repeat this process until any job in U cannot be filled into any batch on M_i. Then the procedure proceeds to handle machine M_{i-1}.

Assignment A1 (T):

Step 1: Let $U = J$. Let $k_{m+1} = 0$. Let $k_i = \min\{n - \sum_{l=i+1}^{m+1} k_l, \lfloor T \cdot v_i / p \rfloor\}$, $i = m, m-1, \ldots, 1$.

Step 1: For $i = m, m-1, \ldots, 1$ (this ordering is used crucially), if $U \neq \varnothing$ do:

(i) Assign k_i empty batches of length p and capacity K_i to machine M_i. Process these empty batches on M_i successively, with the first one started at time zero.

(ii) Construct a binary tree of depth $O(\log k_i)$ with exactly k_i leaves corresponding to the empty batches processed on M_i, in sequence from left to right. The k_i leaves are labeled with K_i. Each internal node is also labeled with K_i. Use this binary tree to assign the jobs in U to the batches processed on M_i and delete the assigned jobs from U, and update the labels of the tree accordingly, until $U = \varnothing$ or we find a job which cannot be filled into any batch on M_i without exceeding K_i. In the latter case, if the job belongs to J_i, then the procedure fails and terminates.

(iii) If $U \neq \varnothing$, then we do a binary search to find the largest job in U which can be filled into a batch on M_i without exceeding K_i (ties broken in favor of the first such job in U), assign this job to M_i via the binary tree and delete the job from U. Update the labels of the tree accordingly. Repeat this process until any job in U cannot be filled into a batch on M_i without exceeding K_i.

We obtain the following theorem.

Theorem 1. *There is an exact algorithm* (*the binary search together with Assignment A or Assignment A1*) *for* $Q|s_j$ (*divisible*), $p_j = p$, p*–batch*, $K_i|C_{\max}$ *that runs in* $O(mn \log m + n \log n \cdot \log(mn))$ *time*.

Proof. The correctness of the algorithm follows from the aforementioned analysis. Procedure Assignment A (or Assignment A1) runs in $O(m + n \log n)$ time for any given value of T. Since there are $O(\log(mn))$ iterations to determine OPT, the whole algorithm runs in $O(mn \log m + n \log n \cdot \log(mn))$ time. (The first term comes from the sorting procedure performed once at the beginning.) □

4 An algorithm for the general case

We now turn to the general case of $Q|s_j, p = p, p\text{-batch}, K_i|C_{\max}$, eliminating the assumption of divisible job sizes. We will provide a 2-approximation algorithm for it. As illustrated in Section 1, 2 is the best possible ratio for this problem, unless P = NP [9].

Again, we restrict our attention on an optimal schedule where the kth batch on M_i starts exactly at time $(k-1)p/v_i$, $k = 1, 2, \ldots$, $i = 1, 2, \ldots, m$. We consider at most mn possible values for OPT. These values can be sorted in ascending order in $O(mn \log m)$ time.

We can obtain all $\mathcal{J}_i$ in $O(n + m)$ time: Distribute the jobs with the same processing set into the same subset. It is worth pointing out that the jobs in $\mathcal{J}_i$ are unsorted, $i = 1, 2, \ldots, m$.

We use a binary search to determine OPT in $O(\log(mn))$ iterations. Use the following Assignment B procedure within the binary search. For each value T selected, Assignment B searches for a schedule with makespan at most T which allows one-job-overfull batches. A batch is a one-job-overfull batch if the total size of the jobs in it exceeds its capacity, but after removing one particular job from it, the total size of the left jobs does not exceed its capacity. The procedure handles the machines in increasing order of their

indices. Suppose that the procedure is handling machine M_i. In the procedure, U_i represents the unordered set of unassigned eligible jobs for M_i. The procedure repeatedly assigns the jobs in U into each batch on M_i until the batch becomes a one-job-overfull batch, or until there are no jobs in U_i, whichever occurs first. Then the procedure proceeds to handle machine M_{i+1}.

Assignment B (T):

Step 1. Let $U_0 = \emptyset$, $i = 1$.

Step 2. While $i \leq m$, do:

(i) Let $U_i = \mathcal{J}_i || U_{i-1}$. Let $k_i = 0$.

(ii) While $U_i \neq \emptyset$ and $k_i < \min\{n, \lfloor T \cdot v_i / p \rfloor\}$, do:

Open a batch of length p and capacity K_i. Fill the batch by repeatedly picking a job out of U_i and putting it into the batch until the total size of the jobs in the batch exceeds K_i. Assign this batch to machine M_i. Let $k_i = k_i + 1$.

(iii) Let $i = i + 1$.

Lemma 3. *If* $\mathrm{OPT} \leq T$, *Assignment B will generate a schedule allowing one-job-overfull batches in* $O(m + n)$ *time for* $Q|s_j, p_j = p, p\text{–batch}, K_i|C_{\max}$ *whose makespan is at most* T.

We obtain:

Theorem 2. *There is a 2-approximation algorithm for* $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ *that runs in* $O(mn \log m + n \log(mn))$ *time.*

Proof. The binary search equipped with Assignment B will return a schedule allowing one-job-overfull batches for $Q|s_j, p_j = p, p\text{-batch}, K_i|C_{\max}$ whose makespan is at most OPT. In the last step of the algorithm, for each one-job-overfull batch in the schedule, we open a new batch for the last packed job in it and process the new batch immediately after it on the same machine. Therefore, the final schedule has makespan at most 2OPT. □

Note that a bottleneck operation in the algorithm is the sorting procedure performed once at the beginning.

5 Conclusion

In this article, we investigated the problem of minimizing makespan for scheduling jobs with equal lengths and arbitrary sizes on uniform parallel batch machines with different capacities. Each machine can only process the jobs whose sizes are not larger than the machine's capacity. We presented two exact algorithms under a divisibility constraint, and a 2-approximation algorithm for the general problem. These algorithms are of practical significance, since they are efficient and easy to implement. A future research topic is to investigate some other objective functions for parallel batch scheduling problems with equal job lengths and arbitrary job sizes.

Acknowledgments: The authors would like to thank the editor and the referees for their helpful comments in improving the quality of the article.

Funding information: This work was supported by Natural Science Foundation of Shandong Province China (No. ZR2020MA030).

Author contributions: All authors contributed equally to the writing of this article. All authors read and approved the final manuscript.

Conflict of interest: The authors state no conflict of interest.

Data availability statement: No data, models, or code are generated or used during the study.

References

[1] C. N. Potts and M. Y. Kovalyov, *Scheduling with batching: a review*, European J. Oper. Res. **120** (2000), 228–249.

[2] J. W. Fowler and L. Mönch, *A survey of scheduling with parallel batch* (*p-batch*) *processing*, European J. Oper. Res. **298** (2022), 1–24.

[3] M. Mathirajan and A. Sivakumar, *A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor*, Int. J. Adv. Manuf. Technol. **29** (2006), 990–1001.

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

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Ann. Discr. Math. **5** (1979), 287–326.

[6] P. Brucker, *Scheduling Algorithms*, 5th edition, Springer, Berlin, Heidelberg, New York, 2007.

[7] E. G. Coffman, M. R. Garey, and D. S. Johnson, Approximation algorithms for bin packing: A survey, *Approximation Algorithms for NP-hard Problems*, PWS Publishing Co., Boston, MA, USA, 1996, pp. 46–93.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, New York, 1979.

[9] G. Dosa, Z. Tan, Z. Tuza, Y. Yan, and C. S. Lanyi, *Improved bounds for batch scheduling with nonidentical job sizes*, Naval Res. Logist. **61** (2014), 351–358.

[10] J.-Q. Wang and J. Y.-T. Leung, *Scheduling jobs with equal-processing-time on parallel machines with non-identical capacities to minimize makespan*, Int. J. Prod. Econ. **156** (2014), 325–331.

[11] O. Ozturk, M.-L. Espinouse, M. D. Mascolo, and A. Gouin, *Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates*, Int. J. Prod. Res. **50** (2012), 6022–6035.

[12] E. G. Coffman, M. R. Garey, and D. S. Johnson, *Bin packing with divisible item sizes*, J. Complexity **3** (1987), 406–428.

[13] X. Li, H. Chen, B. Du, and Q. Tan, *Heuristics to schedule uniform parallel batch processing machines with dynamic job arrivals*, Int. J. Comput. Integr. Manuf. **26** (2013), 474–486.

[14] S. Zhou, M. Liu, H. Chen, and X. Li, *An effective discrete differential evolution algorithm for scheduling uniform parallel batch processing machines with non-identical capacities and arbitrary job sizes*, Int. J. Prod. Econ. **179** (2016), 1–11.

[15] X. Li, Y. Huang, Q. Tan, and H. Chen, *Scheduling unrelated parallel batch processing machines with non-identical job sizes*, Comput. Oper. Res. **40** (2013), 2983–2990.

[16] J. E. C. Arroyo and J. Y.-T. Leung, *Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times*, Comput. Oper. Res. **78** (2017), 117–128.

[17] S. Xu and J. C. Bean, *A genetic algorithm for scheduling parallel non-identical batch processing machines*, IEEE Symposium on Computational Intelligence in Scheduling, Honolulu, HI, USA, 2007, pp. 143–150.

[18] 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 Syst. Appl. **37** (2010), 1510–1521.

[19] P. Damodaran, D. A. Diyadawagamage, O. Ghrayeb, and M. C. Velez-Gallego, *A particle swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines*, Int. J. Adv. Manuf. Technol. **58** (2012), 1131–1140.

[20] Z.-H. Jia, K. Li, and J. Y.-T. Leung, *Effective heuristic for makespan minimization in parallel batch machines with non-identical capacities*, Int. J. Prod. Econ. **169** (2015), 1–10.

[21] Z.-H. Jia, T. T. Wen, J. Y.-T. Leung, and K. Li, *Effective heuristics for makespan minimization in parallel batch machines with non-identical capacities and job release times*, J. Ind. Manag. Optim. **13** (2017), 977–993.

[22] S. Li, *Approximation algorithms for scheduling jobs with release times and arbitrary sizes on batch machines with non-identical capacities*, European J. Oper. Res. **263** (2017), 815–826.

[23] A. Gürsoy, *Optimization of product switching processes in assembly lines*, Arab. J. Sci. Eng. **2022** (2022), 1–16.

[24] F. Zheng, Y. Chen, M. Liu, and Y. Xu, *Competitive analysis of online machine rental and online parallel machine scheduling problems with workload fence*, J. Comb. Optim. **44** (2022), 1060–1076.

[25] A. Gürsoy and N. K. Gürsoy, *On the flexibility constrained line balancing problem in lean manufacturing*, Textile Apparel. **25** (2015), 345–351.

[26] R. Zhang, P.-C. Chang, S. Song, and C. Wu, *A multi-objective artificial bee colony algorithm for parallel batch-processing machine scheduling in fabric dyeing processes*, Knowl. Based Syst. **116** (2017), 114–129.
[27] A. Gürsoy, *An integer model and a heuristic algorithm for the flexible line balancing problem*, Textile Apparel. **22** (2012), 58–63.
[28] S. Li, *Efficient algorithms for scheduling equal-length jobs with processing set restrictions on uniform parallel batch machines*, Math. Biosci. Eng. **19** (2022), 10731–10740.
[29] C. L. Li, *Scheduling unit-length jobs with machine eligibility restrictions*, European J. Oper. Res. **174** (2006), 1325–1328.
[30] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publ. Comp. London-Amsterdam-Don Mills-Sydney, 1974.
[31] J. Kang and S. Park, *Algorithms for the variable sized bin packing problem*, European J. Oper. Res. **147** (2003), 365–372.
[32] A. Bar-Noy, R. E. Ladner, and T. Tamir, *Windows scheduling as a restricted version of bin packing*, ACM Trans. Algorithms **3** (2007), 28-es.
[33] P. Detti, *A polynomial algorithm for the multiple knapsack problem with divisible item sizes*, Inform. Process. Lett. **109** (2009), 582–584.
[34] G. Wang and L. Lei, *Polynomial-time solvable cases of the capacitated multi-echelon shipping network scheduling problem with delivery deadlines*, Int. J. Prod. Econ. **137** (2012), 263–271.
[35] D. S. Johnson, *Near-optimal Bin Packing Algorithms*, Ph.D. thesis, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973.