Makespan minimisation on parallel batch processing machines with non-identical job sizes and release dates

Onur Ozturk, Marie-Laure Espinouse, Maria Di Mascolo, Alexia Gouin

International Journal of Production Research
Vol. 00, No. 00, 00 Month 2009, 1–18

RESEARCH ARTICLE

Makespan minimization on parallel batch processing machines with non-identical job sizes and release dates

Onur Ozturk[a], Marie-Laure Espinouse[a]*, Maria Di Mascolo[a], Alexia Gouin[b]

[a] *Grenoble-INP / UJF-Grenoble 1 / CNRS, G-SCOP UMR5272 Grenoble, F-38031, France (Grenoble-Science pour la Conception, l'Optimisation et la Production)*;
[b] *Grenoble-INP / UJF-Grenoble 1 / CNRS, GIPSA-lab UMR 5216 St. Martin D'Heres, F-38402, France (Laboratoire Grenoblois de l'Image, de la Parole, du Signal et de l'Automatique)*

(*v3.5 released octobre 2011*)

The problem we study in this paper arises from the washing step of hospital sterilization services. Washers at the washing step are capable of handling more than one medical device set as long as their capacity is not exceeded. The medical device set sizes and arrival times to the sterilization service may be different, but they all have the same washing duration. Thus, we model the washing step as a batch scheduling problem where medical device sets are treated as jobs with non-identical sizes and release dates, but equal processing times. The main findings we present in this paper are the following: First, we study two special cases for which polynomial algorithms are presented. Afterwards, we develop a 2-approximation algorithm for the general problem. Finally, we develop a MILP model and compare it to another MILP model from the literature. Computational results show that our MILP model outperforms the other MILP model.

Keywords: Parallel batch scheduling; mixed integer linear programming; approximation algorithm; sterilization service

1. Introduction

Batch processing machines are encountered in many different industries. Some examples of such machines are diffusion or oxidation tubes in semiconductor wafer fabrication, burn-in ovens in semiconductor testing, etc.. The batch processing problem we treat has its origins from the washing step of hospital sterilization services.

*Corresponding author. Email: marie-laure.espinouse@g-scop.grenoble-inp.fr

ISSN: 1478-6451 print/ISSN 1478-646X online

DOI: 10.1080/1478645YYxxxxxxx
http://www.informaworld.com

A hospital sterilization service is responsible for sterilizing medical devices after utilization in surgical operations. A medical device is an instrument, apparatus, appliance, or any other article, which is used for medical purposes on patients, in diagnosis, therapy or surgery. In fact, it is more appropriate to refer to these instruments as reusable medical devices (or RMDs) as they are reused after sterilization. Note that all RMDs used in a surgical operation constitute the RMD set of this surgical operation.

After utilization for a surgical operation, RMDs are directly placed in a substance, allowing pre-disinfection, and are transferred to the sterilization service. There, they are firstly rinsed and washed in automatic washers. Rinsing is performed either manually or automatically in automatic washers. After washing, RMDs are checked and packed into appropriate boxes. All items must be packed individually or grouped into boxes prior to sterilization. They are then sterilized in so-called "autoclaves", transferred to operating theaters and stored before reutilization.

There can be a large number of different RMD sets in a hospital. Moreover, for a typical hospital, there may be hundreds of RMD references. Because each surgical operation may require different numbers and types of RMDs, RMD sets may be of different sizes. For different reasons (surgery start times and durations, pre-disinfection procedure, etc.), RMD sets are ready for washing at different moments within the same day. However, washing duration is the same for all RMD sets.

The washing step of a sterilization service is composed of automatic washers. These washers have a fixed capacity and they can handle more than one RMD set at the same time as long as their capacity is not exceeded. Note that in the washing step, RMD sets are not usually allowed to be split among several washers due to organizational and traceability reasons.

The washing steps of sterilization services are generally the bottleneck of the whole sterilization process (Ngo Cong (2009)). Thus, a good utilization of automatic washers is crucial in order to minimize the total washing time at this step and to increase the performance of the sterilization process. In this paper we thus address the problem of minimizing makespan (C_{max}) on parallel batch processing machines in presence of non-identical job sizes, different release dates and equal processing times. This problem has newly been investigated in the literature considering different job processing times. However, studying equal processing times provides us some interesting avenues. In this study, we first develop exact algorithms for two special cases of our problem where job sizes are not arbitrary. Afterwards for the general case, we present an approximation algorithm[1] and show that it has a performance guarantee equal to 2. According to our knowledge, heuristics presented in the literature for the scheduling problem considering different job sizes, release dates and processing times do not have polynomial complexity due to batch creation procedures. Finally, we present an exact resolution method based on a MILP model. We test its efficiency in terms of resolution time and also compare it to another MILP model from the literature.

In the following section we present the basic assumptions and notations used throughout this paper. Section 3 reviews previous related work on scheduling batch processing machines. Section 4 presents two exact algorithms for two special cases. In section 5, we first give an algorithm having a polynomial complexity and prove its performance guarantee. Then, we propose a MILP model and test its efficiency for the resolution time on several instances.

[1]An approximation algorithm is a heuristic which has a performance guarantee and a polynomial complexity.

2. Assumptions and notations

In this paper, we model the washing step of sterilization services as a parallel batch scheduling problem. Hence, automatic washers at the washing step are treated as batch processing machines and RMD sets as jobs. The assumptions we make are the following.

- There are N jobs to be processed. The release date and the size of a given job j are denoted by r_j and w_j, respectively. Job processing times are the same for all jobs and are denoted by p.
- All machines have the same capacity B, and the size of a job cannot be greater than machine capacity.
- Several jobs can be batched together, complying with the machine capacity constraint.
- Since all jobs have the same processing time, p, the processing time of any batch is p.
- We are not allowed to split a job into several batches.

Inspired by Graham's notation (Graham et al. (1979)), we propose the following notation for our problem: $P|p-batch, r_j, p_j = p, w_j, B|C_{max}$. In this notation, P stands for identical parallel machines, $p-batch$ for parallel batching; r_j and w_j denote job release date and size, respectively, $p_j = p$ stands for equal processing time, and B for machine capacity. Finally, C_{max} refers to makespan.

3. Previous related work

The batch processing literature is really vast. In this section, we will be mostly speaking about parallel batch processing problems where jobs have non-identical sizes and release dates. But first, let us give some explanations about parallel batch scheduling.

For parallel batching problems with different job sizes, the sum of job sizes which are put in a batch defines the size of that batch. Batch sizes should not exceed machine capacity. Each job should be assigned to just one batch. Moreover, the processing time of a batch is equal to the longest processing time of jobs in that batch (Potts and Kovalyov (2000)). If job families are considered, jobs are partitioned into different sets according to their processing times, machine setup times, etc. There are two types of family scheduling models: compatible and incompatible job families. In compatible job family model, jobs from different families may be batched together (Neale and Duenyas (2003)). If incompatible job families are considered however, only jobs from the same family can be batched together (Azizoglu and Webster (2001)). Jobs within a family may have different sizes, release dates and processing times.

Papers on batch scheduling with non-identical job sizes can be classed in four groups as follows: 1- single machine and identical release dates, 2- parallel machines and identical release dates, 3- single machine and unequal release dates, 4- parallel machines and unequal release dates.

In table 1, we give a brief classification of the literature on parallel batch scheduling problems with non-identical job sizes. We see that most of the papers study the makespan minimization considering equal job release dates. According to our knowledge, Uzsoy (1994) is the first one who studied parallel batching considering different job sizes. He shows that minimizing makespan on a single machine with identical job processing times and equal release dates but different job sizes is strongly NP-hard. The NP-hardness of this special case implies that our problem is also NP-hard.

Table 1 shows that there are 8 articles considering unequal release dates and thus

more related to our problem. Among them, two papers work on minimizing makespan on a single machine where approximation algorithms are presented. Another paper takes into account job rejection penalty beside makespan minimization and develops also an approximation algorithm. The fourth group presented on table 1 contains papers closer to our study since we consider parallel machines and unequal job release dates. Chung et al. (2009) are the first ones who study the scheduling problem presented by the fourth group on table 1. They develop an exact method and two heuristic approaches. The exact method they propose is a MILP model. Because that MILP model requires long computation times, they proposed also heuristics inspired from the heuristic developed by Lee and Uzsoy (1999). The heuristics of Chung et al. (2009) are composed of two phases. The first phase forms batches and the second phase schedules batches on machines. The first phase is common to both heuristics and is inspired from the "DELAY" algorithm presented in Lee and Uzsoy (1999). It uses two parameters: α, for determining the time window in which jobs are batched, β, for determining the fullness of batches. They run the algorithm with different combinations of parameter values and select the best solution. It is reported that the computational burden of batch creation procedure is low enough. But as cited in Lee and Uzsoy (1999), the performance of batch formation is sensitive to the values of the parameters α and β (Note that Lee and Uzsoy (1999) consider equal job sizes). Damodaran et al. (2011) develop a meta-heuristic called Greedy Randomized Adaptive Search Procedure (GRASP is a metaheuristic method introduced by Feo and Resende (1989)). They report that the GRASP approach guarantees the optimal solution for small instances and is more effective than the heuristics proposed by Chung et al. (2009). Damodaran and Velez Gallego (2010) propose a constructive heuristic. This heuristic operates first by determining a time horizon, after which it solves a 0-1 knapsack problem to select the jobs to be batched. It is compared to the MILP model and to the heuristics given by Chung et al. (2009) as well as to the GRASP approach developed by Damodaran et al. (2011). It is reported that their heuristic outperforms the heuristics of Chung et al. (2009) and gives similar results to those of the GRASP method. This heuristic is pseudo-polynomial, since it solves a 0-1 knapsack problem using the dynamic algorithm proposed by Martello and Toth (1990). Chen et al. (2010) develop a genetic algorithm and an ant colony optimization. For the batch assignment procedure, they propose a heuristic (ERT-LPT: earliest ready time-longest processing time) which is used in common in both meta-heuristics. For computational experimentations, they develop another heuristic considering the batch creation procedure proposed by Dupont and Jolai Ghazvini (1998) where ERT-LPT is applied afterwards. Their results indicate that both meta-heuristics outperform the heuristic approach. Wang and Chou (2010) consider machines with different capacities. They develop a genetic and a simulated annealing algorithm, and test their algorithms on the instances defined by Chung et al. (2009). It is reported that the proposed meta-heuristics are more efficient than the heuristics of Chung et al. (2009).

Because we consider equal job processing times, our problem is a special case of the problem treated in the fourth group on table 1. However, our problem is also strongly NP-hard. Moreover, the consideration of equal job processing times allows us to develop exact algorithms for two special cases which are explained in the next section. Afterwards, we return to our original problem and propose a heuristic algorithm with performance guarantee. Finally, we develop an exact method, which is in fact a MILP model, and test its efficiency in terms of resolution time.

Table 1. Literature related to parallel batch scheduling problems with different job sizes

Group	Reference	Solution approach	Criteria
Single machine and identical release dates	Uzsoy (1994)	Heuristics, B&B	C_{max}, $\sum C_j$
	Jolai Ghazvini and Dupont (1998)	Heuristics	$\sum C_j$
	Dupont and Jolai Ghazvini (1998)	Heuristics	C_{max}
	Kempf et al. (1998)	MILP, heuristics	C_{max}, $\sum C_j$
	Azizoglu and Webster (2000)	B&B	$\sum w_j C_j$
	Azizoglu and Webster (2001)	B&B	$\sum w_j C_j$
	Zhang et al. (2001)	Approximation algorithm	C_{max}
	Dupont and Dhaenens-Flipo (2002)	B&B	C_{max}
	Melouk et al. (2004)	MILP, Simulated annealing	C_{max}
	Kashan et al. (2006)	Genetic algorithms	C_{max}
	Zhang et al. (2007)	Approximation algorithms	$\sum C_j$
	Kashan et al. (2009)	Approximation algorithms	C_{max}
	Parsa et al. (2010)	Branch and price algorithm	C_{max}
	Kashan et al. (2010)	Genetic algorithms	C_{max}
Parallel machines and identical release dates	Chang et al. (2004)	Simulated annealing	C_{max}
	Kashan et al. (2008)	Genetic algorithms	C_{max}
Single machine and unequal release dates	Li et al. (2005)	Approximation algorithm	C_{max}
	Nong et al. (2008)	Approximation algorithm	C_{max}
	Lu et al. (2010)	Approximation algorithm	C_{max}+job rejection
Parallel machines and unequal release release dates	Chung et al. (2009)	MILP, heuristics	C_{max}
	Damodaran et al. (2011)	meta-heuristic	C_{max}
	Damodaran and Velez Gallego (2010)	heuristics	C_{max}
	Chen et al. (2010)	Genetic algo, ant colony heuristic	C_{max}
	Wang and Chou (2010)	Genetic algo, simulated annealing	C_{max}

4. Polynomially solvable cases

In this section, we present polynomial time algorithms for two special cases. The first one, which we shall refer to as $P|p-batch, r_j, p_j = p, w_j(strongly\ divisible), B|C_{max}$, considers a special order for job sizes. In the second one, job splitting is allowed. That problem, which we shall refer to as $P|p-batch, r_j, p_j = p, w_j(split), B|C_{max}$, will be used in obtaining a lower bound on the optimal value for the original problem.

4.1. *Case 1: Job sizes forming a strongly divisible sequence*

This case is inspired from a special case of a bin-packing problem where job sizes form a strongly divisible sequence. The Grahams notation can be modified as follows in order to represent this special case: $P|p-batch, r_j, p_j = p, w_j(strongly\ divisible), B|C_{max}$.

In the standard one-dimensional bin packing problem, we are given a pair (J, B), more precisely, a capacity B and a list of items $J = j_1, j_2, ..., j_N$, and are asked to partition the items into a minimum number of subsets such that the item sizes in each subset sum to no more than B. Coffman et al. (1987) showed that if item sizes form a strongly divisible sequence, the algorithms first fit (FF) and the first fit decreasing (FFD) are optimal for the bin packing problem. Let us first remind these algorithms and give the definition for the strongly divisible sequence.

Algorithm First Fit (FF) (Coffman et al. (1997))
Step 1: Arrange items in some arbitrary order
Step 2: Select the item at the head of the list and place it in the first bin with enough space to accommodate it. If it fits in no existing bin, create a new bin.
(The complexity of FF is $0(NlogN)$)

Algorithm First Fit Decreasing (FFD) (Coffman et al. (1997))
This algorithm sorts jobs in non-increasing order of sizes and then applies FF. The complexity of FFD is also $0(NlogN)$.

Strongly divisible sequence (Coffman et al. (1987))
Let W be a list of item sizes such that $w_1 > w_2 > ... > w_i > w_{i+1} > ...$ The sizes of items form a divisible sequence if w_{i+1} exactly divides w_i. This sequence is strongly divisible if in addition the largest item size, w_1, exactly divides the batch capacity.

In analysis of our scheduling problem, let us give some properties about the job sizes forming a strongly divisible sequence, and about the processing time of batches, respectively.

Lemma 1. (Coffman et al. (1987)) In case of jobs forming a strongly divisible sequence, FFD starts a new bin only when all previous bins are completely full.

Lemma 2. For the strongly divisible pair (J, B), let b_1 be a partially filled batch. If a job j with size w_j does not completely enter this batch, then in b_1, there is at least one job whose size is smaller than w_j.

Let b_1 be a partially filled batch. Suppose that there are only jobs whose sizes are bigger or equal to w_j in batch b_1. Let w_{sum} be the sum of sizes of these jobs. Because each bigger job size can be expressed as a multiple of smaller job sizes with an integer number, then $w_{sum} = coeff_1 * w_j$ where $coeff_1$ is an integer. Moreover, each job size can divide exactly the batch capacity. Let B be the batch capacity. Then, $B = coeff_2 * w_j$ where $coeff_2$ is an integer. Because, $w_{sum} \leq B$, $coeff_1 \leq coeff_2$. Then, if $coeff_1 = coeff_2$, the batch is full. Otherwise, there is enough space for job j in batch b_1. Thus, if b_1 is a partially filled batch in which job j can not be accommodated, then there should be at least one job whose size is smaller then w_j.□

Lemma 3. The aforesaid batch, b_1, in lemma 2, can be completely filled by removing some jobs having smaller sizes than w_j in order to accommodate the job j in b_1.

The jobs of b_1 can be divided into two groups according to their sizes: group 1 for jobs whose sizes are bigger or equal to w_j and group 2 for jobs whose sizes are smaller than w_j. These jobs can easily be arranged in non-increasing order of sizes. Then, the first group of jobs occupies a space which is an exact multiple of w_j. Suppose that the batch b_1 is composed of sub-batches having sizes equal to w_j. Regarding the second group, these jobs can be seen as a subgroup of the strongly divisible sequenced jobs for which the size of a batch is equal to w_j. Moreover, arranging these jobs in non-increasing order of sizes is the same as applying FFD on these jobs where the batch capacity is equal to w_j. Thus, according to lemma 1, only the last sub-batch is partially filled which is in fact the last part of the batch b_1, equal to w_j in size. Hence, removing the jobs of the

partially filled sub-batch from the batch b_1 lets accommodate job j completely in batch b_1 which becomes completely filled. □
The lemma 3 is going to be used in the algorithm in order to have fully complete batches.

Lemma 4. If all batches have the same processing time, then in the optimal solution for the makespan criterion, batches are placed consecutively on machines in non-decreasing order of batch ready times where the ready time of a batch is equal to the greatest release date of jobs contained.

It is clear that batch assignment procedure given in lemma 4 yields to the optimal makespan since equal batch processing times are considered. □

The notations used in the algorithm are:
L: list containing jobs sorted in non-decreasing order of release dates
nb_{old}: minimum number of batches that can be formed with the jobs in L
j_{first}: first job in L
b_b: currently open batch
nb_{new}: the new minimum number of batches that can be formed with the jobs in L after removing j_{first} from L
space: in case a job does not entirely fit a non-full batch, *space* represents the space needed in order that the job can be fully put in the batch
j_{last}: last job in batch b_b after sorting jobs in non-increasing according to their sizes
w_{last}: size of j_{last}

Algorithm for Strongly Divisible Sequence (SDS)
Step 1: Sort jobs in non-decreasing order of job release dates: L
Step 2: While L is not empty, apply FF to L in order to calculate the minimum number of batches to form: nb_{old}. Put the first job, j_{first}, of L into a new batch, b_b. Erase that job from L and re-apply FF to L in order to calculate the new minimum number of batches: nb_{new}. If $nb_{old} > nb_{new}$, close the batch. Else,
  Step 2.1: While $nb_{old} = nb_{new}$, put the first job, j_{first}, of L into the batch b_b.
  Update L and re-apply FF to L in order to calculate nb_{new}.
    Step 2.1.1: If the job, j_{first}, does not completely enter the batch, calculate the space needed: *space* to accommodate the job completely into the batch. Sort jobs in batch b_b in non-increasing order of sizes.
      Step 2.1.1.1: While $space > 0$, remove the last job, j_{last}, from b_b.
      Set $space = space - w_{last}$. Put j_{last} back into L.
      Step 2.1.1.2: Put j_{first} in b_b and update L.
      Step 2.1.1.3: Set $nb_{new} = nb_{new} - 1$.
      Step 2.1.1.4: Sort jobs of L in non-decreasing order of job release dates.
    Step 2.2: Close the batch.
Step 3: Set ready times of batches equal to the greatest release date of jobs they contain. Sort batches consecutively on machines in non-decreasing order of ready times starting from the first machine.

The time complexity of SDS can be defined as $0(N^2 logN)$. In the worst case, a number of jobs, let us say q, is firstly placed in a batch then they all may be removed from the batch for a large size job (step 2.1.1). If q is sufficiently small according to the total number of jobs, N, the number of times that q jobs are put into a batch and then removed from that batch approaches to N. Each time a job is put in a batch,

the new minimum number of batches is calculated with the un-batched jobs by applying FF. Thus, the time complexity of SDS can be defined as $0(N^2 logN)$ for the worst case.

Theorem 1. SDS is optimal for the problem:
$P|p-batch, r_j, p_j = p, w_j(strongly\ divisible), B|minimizing\ the\ number\ of\ batches$

Proof. The number of batches formed by SDS is equal to the number of batches formed by applying FF. As proven by Coffman et al. (1987), FF minimizes the number of batches if job sizes form a strongly divisible sequence. Hence, SDS minimizes the number of batches. □

Before showing the optimality of SDS for the makespan criterion, let us give another property about the minimum completion time of a problem.

Lemma 5. Consider a problem with N jobs. The minimum completion time after job j, where $1 \leq j \leq N$, is equal to the sum of the release date of j and the execution time for the minimum number of batches that can be created with jobs having release dates greater or equal to the release date of job j.

For any problem with N jobs and M machines, a lower bound on the number of batches, say nb, can be given by: $nb = \left\lceil \sum_{j=1}^{N} w_j / B \right\rceil$. Without lost of generality, let jobs be sorted in non-decreasing order of release dates. Consider a job j such that $1 \leq j \leq N$ and $r_j \leq r_N$. It is also possible to calculate a lower bound on the number of batches for jobs whose release dates are greater or equal to r_j. Let nb_j be a lower bound on the number of batches that can be formed with jobs whose release dates are greater or equal to r_j. Then, $nb_j = \left\lceil \sum_{k \in S} w_k / B \right\rceil$ where $S = \{k | r_k \geq r_j\}$. This way, we do not take into account jobs which are released before job j.

Let M be the number of machines. We know by lemma 4 that these nb_j batches should be placed consecutively on machines in non-decreasing order of batch ready times where the ready time of a batch is equal to the greatest release date of jobs contained. Then, the maximum number of batches placed on machines can be calculated as $\lceil nb_j / M \rceil$. Thanks to placing batches consecutively on machines, each machine contains a minimum number of batches. Moreover, in the best case, the execution of the first batch can start at r_j. Thus, the smallest completion time after job j can be given as: $r_j + \lceil nb_j / M \rceil * p$ where p is the processing time of jobs. In other words, the minimum completion time after job j is equal to the sum of the release date of j and the execution time of the minimum number of batches that can be created with jobs whose release dates are greater or equal to r_j. □

Theorem 2. SDS is optimal for the problem:
$P|p-batch, r_j, p_j = p, w_j(strongly\ divisible), B|C_{max}$

Proof. SDS closes a batch if and only if, after placing a job in that batch, the total number of batches to form with the un-batched jobs decreases or the batch is completely full. In fact if a batch is completely full after the addition of a job, since the algorithm minimizes also the number of batches, the total number of batches to form with the un-batched jobs necessarily decreases by 1. Now, we would like to show that the minimum completion time stated in lemma 5 is reachable with the SDS algorithm.

Suppose that for a given problem with N jobs and M machines, the minimum number of batches is equal to nb. Let s_m be the processing start time for batch b_m. Let us denote by s_{nb} the processing start time of the last batch and r_N the release date of the last job. If $s_{nb} = r_N$, then the makespan is already optimal. But, if $s_{nb} > r_N$, then there is no idle time between batches b_{nb} and the batch before b_{nb} on the same machine. In fact, as batches are placed consecutively on machines according to ready times, the batch before the last one on the same machine is the batch $b_{(nb-M)}$. Let us consider the maximal chain of batches without idle times on the machine that contains the last batch: $b_{(nb-q*M)}, ..., b_{nb}$ where $q \geq 1$. Now, consider a job k whose release date is equal to the processing start time of the batch $b_{(nb-q*M)}$ i.e. $r_k = s_{(nb-q*M)}$. We have seen by lemma 5 that the minimum completion time after any job, say j, could be $r_j + \left\lceil nb_j/M \right\rceil * p$.

Thanks to SDS, the condition to close a batch is that the total number of batches to form with the un-batched jobs should decrease. Thus, after the last job of each batch, the algorithm forms always a minimum number of batches. By construction, the minimum completion time after job k is reached and we have the optimal makespan: $C^*_{max} = r_k + \left\lceil nb_k/M \right\rceil * p$. □

4.2. *Case 2: Job splitting is allowed*

We develop here a polynomial time algorithm in case jobs are allowed to be split. Indeed, even if job splitting is not generally allowed, it may take place in some sterilization services. We show the problem by $P|p - batch, r_j, p_j = p, w_j(split), B|C_{max}$.

In fact, if a job can be split, it can then be considered as the composition of smaller jobs where the sum of their sizes is equal to the size of the original job. Or, it is possible to consider the job as if it is composed of jobs having very small and equal sizes. Thus, the problem becomes equivalent to the case where jobs have unit sizes, equal processing times and different release dates. Ikura and Gimple (1986) study that case in presence of a single machine to minimize makespan. They develop an exact algorithm. We inspire from their algorithm and show that it is also optimal for the case of parallel machines.

The proposed algorithm starts by creating a list, L, of jobs sorted in non-increasing order of job release dates. Then, a lower bound on the number of batches is calculated and thanks to the job splitting property, the algorithm forms a minimum number of batches. As we are allowed to split jobs, the minimum number of batches is found by rounding up to the smallest integer value when dividing the sum of all job sizes by the machine capacity. After finding the minimum number of batches, we start filling the batches with the first job of the previously created list, L. A batch is closed when it is completely filled or when there are no more jobs to be batched. In case a job does not entirely fit a batch, then the job is split. The first part of the job is put into the batch and the second part is treated as a new job having the same release date as the original job. Finally, batches are sorted consecutively on machines in non-decreasing order of ready times.

The algorithm sorts jobs in non-increasing order of job release dates and then forms batches with successive jobs. Thus, the complexity of the algorithm can be defined as $0(NlogN)$.

Split Job algorithm
Step 1. Sort jobs in non-increasing order of job release dates $r_j : L$
Step 2. Calculate the minimum number of batches needed: $nb = \left\lceil \sum_{j=1}^{N} w_j/B \right\rceil$
Step 3. While $nb > 0$, open a batch: b_b
Step 3.1. Put the first job of L into the batch b_b. Update L. If the batch is 100% filled or if there is no more elements in L, close the batch and set $nb = nb - 1$.
Step 3.1.1. If the job does not completely fit the batch, split the job. Put the first part of the job in order to fill the batch entirely, then close the batch and set $nb = nb - 1$. Update the size of the split job.
Step 4. Set ready times of batches equal to the greatest release date of jobs they include. Sort batches consecutively on machines in non-decreasing order of ready times starting from the first machine.

Theorem 3. *Split Job* algorithm is optimal for the problem: $P|p-batch, r_j, p_j = p, w_j(split), B|C_{max}$

Proof. We can give a proof similar to that of theorem 2. After assigning batches on machines, consider a chain of batches on the machine that sets the makespan value. The ready time of the first batch in that chain is equal to the release date of a job. Moreover, after closing each batch, we create always a minimum number of batches with the un-scheduled job. Thus, the condition cited in lemma 5 is satisfied and the algorithm is optimal. □

5. Solution approaches for $P|p-batch, r_j, p_j = p, w_j, B|C_{max}$

We now return to our original problem where jobs can not be split while having arbitrary sizes. In this section, we first define how a lower bound can be obtained for the problem. Then, we give an algorithm and prove that its performance guarantee is equal to 2. Afterwards, we propose a MILP model and test its efficiency on several instances.

5.1. *Lower bound for the problem*

In section 4.2, we treated the problem with job splitting and showed that in each batch there was always a last job after which a minimum number of batches were created. This way, the minimum completion time is obtained and the makespan value gets the best possible value. Therefore, we use the *Split Job* algorithm as a lower bound algorithm for our problem.

5.2. *2-approximation algorithm*

The proposed algorithm first finds the lower bound of the given problem, using the algorithm *Split Job*. Then, from each batch, the split jobs are removed and a list is created with these jobs sorted in non-increasing order of sizes. We try to accommodate these jobs into the previously formed batches respecting the batch ready times (i.e. if a job can be inserted into a batch, its release date must be smaller than the ready time of the batch). Finally, all un-batched jobs are batched using the FFD and scheduled after

the previously formed batches. We name this algorithm as *Combine Job*.

Combine Job algorithm
Step 1. Apply the *Split Job* algorithm.
Step 2. Remove split jobs from the batches.
Step 3. Create a list, L, of jobs in non-increasing order of sizes with jobs found in step 2.
Step 4. For all previously created batches and for all jobs, j_l, in L
Step 4.1. If j_l can be accommodated in a batch and if the release date of j_l is smaller than the ready time of the batch, place j_l in the batch and erase it from L.
Step 5. For all the un-batched jobs, apply the FFD algorithm and schedule the so-formed batches in non-decreasing order of ready times following the batches formed in step 1.

The time complexity of the algorithm depends on step 4. The number of batches created by the *Split Job* algorithm is bounded by the number of jobs which is N. If each batch contains a split job, we can have at most $N-1$ split jobs. Then, the number of iterations performed in step 4 is bounded by N^2. Thus, the final complexity of *Combine Job* algorithm is $0(N^2)$.

Theorem 5. *Combine Job* is a 2-approximation algorithm for the problem $P|p-batch, r_j, p_j = p, w_j, B|C_{max}$.

Proof. Let us denote by C_{max}^{LB} the makespan found by *Split Job* algorithme (which is in fact the lower bound of the problem) and C_{max}^{*} the optimal solution. It is obvious that $C_{max}^{LB} \leq C_{max}^{*}$. Let C_{max} be the completion time found by *Combine Job*. Note that the algorithm finds first the lower bound of the problem, removes the split jobs from batches and tries to insert these jobs (without splitting) into the existing batches without changing the C_{max}^{LB} value. Then, if there are still un-batched jobs, they are batched using FFD and scheduled after the batches formed by *Split Job* algorithm.

Let us suppose that the lower bound algorithm forms nb batches. Then, in the worst case, each batch splits a job and there are nb-1 split jobs. Suppose that those split jobs have large sizes. Then, they can neither be inserted in any existing batch nor batched together. Hence, these $nb-1$ jobs are scheduled one by one following the batches created in step 1.

Let us suppose that there are M machines, and in the lower bound solution each machine handles nb_m batches, where $1 \leq m \leq M$. Then, $\max_{m=1}^{M}\{nb_m\} * p \leq C_{max}^{LB}$. Note that after scheduling the previously split nb-1 jobs one by one and consecutively following these batches, in the new schedule, each machine can have at most $\max_{m=1}^{M}\{nb_m\}$ more batches. Thus, the relation between C_{max} and C_{max}^{LB} becomes: $C_{max} \leq C_{max}^{LB} + \max_{m=1}^{M}\{nb_m\} * p$. Moreover, $C_{max}^{LB} + \max_{m=1}^{M}\{nb_m\} * p \leq 2C_{max}^{LB} \leq 2C_{max}^{*}$. Thus, we get: $C_{max} \leq 2C_{max}^{*}$.□

5.3. *Mixed integer linear programming model (MILP model)*

For the case where job processing times are different, Chung et al. (2009) proposed a MILP model. Thus, that MILP model could be used for our problem in order to find the optimal solution. However, considering equal job processing times allows us to formulate

another MILP model which can find the optimal solution faster than the MILP given by Chung et al. (2009). Below, we explain our MILP model and test its efficiency in the next section.

Let us start by introducing the indexes, parameters and variables used in the model.

Indexes:

$j : 1, ..., N$ for jobs
$k : 1, ..., N$ for batches (since at most N batches can be created)
$m : 1, ..., M$ for machines

Parameters:

w_j: size of job j
r_j: release date of job j
N: number of jobs
M: number of machines
B: machine capacity
p: job processing time
nb: lower bound on the number of batches ($nb = \left\lceil \sum_{j=1}^{N} w_j/B \right\rceil$)

Decision variables:

x_{jkm}: 1 if job j is executed in batch k and on machine m, 0 otherwise
b_{km}: 1 if batch k is created on machine m, 0 otherwise
S_{km}: ready time of batch k on machine m
C_{max}: makespan

Mathematical formulation:

$$\text{Minimize } C_{max}$$

$$\text{subject to}$$

$$\sum_{k=1}^{N}\sum_{m=1}^{M} x_{jkm} = 1 \qquad j = 1, ..., N \tag{1}$$

$$\sum_{j=1}^{N} w_j * x_{jkm} \leq B * b_{km} \qquad k = 1, ..., N,\ m = 1, ..., M \tag{2}$$

$$\sum_{m=1}^{M} b_{km} \leq 1 \qquad k = 1, ..., N \tag{3}$$

$$S_{km} \geq x_{jkm} * r_j \quad j = 1, ..., N,\ k = 1, ..., N,\ m = 1, ..., M \tag{4}$$

$$S_{km} \geq S_{k-1,m} + p * b_{k-1,m} \qquad k = 2, ..., N;\ m = 1, ..., M \tag{5}$$

$$C_{max} \geq S_{Nm} + p * b_{Nm} \qquad m = 1, ..., M \tag{6}$$

$$b_{k,(k mod M)+1} \geq 1 \qquad k = 1, ..., nb \tag{7}$$

$$b_{km} = 0 \qquad k = nb + 1, ..., N,\ \forall m \neq (k\ mod\ M) + 1 \tag{8}$$

$$x_{jkm} \in \{0, 1\}, b_{km} \in \{0, 1\}, S_{km} \geq 0, C_{max} \geq 0$$

Our objective is to minimize the makespan. Constraint set (1) ensures assignment of all jobs to a batch and to a machine. Constraint set (2) is the capacity constraint in case that batch k is created on machine m. Constraint set (3) assigns a batch at most to one machine. Constraint set (4) sets the ready time of a batch equal to the largest release date of jobs in that batch. In case more than one batch is assigned to a machine, constraint set (5) ensures a difference at least equal to the execution time p, between the processing of these batches. If b_{km} is null, then batch k on machine m is a dummy batch,

and its ready time is directly given to the next indexed batch on machine m and the processing time of batch k is 0 (i.e. $p * b_{km} = 0$). Another function of constraint set (5) is to assign a ready time to all batches, from 1 to N, even if they are not created, i.e. dummy batches also obtain a ready time with (5). Finally, constraint set (6) sets the C_{max} value. Constraint sets (7) and (8) are valid inequalities used to improve the resolution time of the model. With these constraints, we aim to reduce the number of equivalent solutions, and thus the resolution space of the MILP. To understand these constraints properly, let us now explain how we consider the assignment procedure of batches on machines. It is clear that a batch can be assigned to only one machine. We know that at least nb batches are created. We can thus pre-assign batches starting from any machine in the problem. Constraint set (7) places the first nb batches consecutively on machines. k *mod* M determines the machine on which batch k will be processed. We add 1 to (k *mod* M) to prevent having 0 as a machine index and, without loss of generality, we place the first batch on the second indexed machine. Since we have identical machines and equal batch processing times, this placement is acceptable. Once these batches have been assigned to the machines, we can continue pre-assignment starting from nb+1 up to N. However, we cannot be sure if these batches will be created. Thus, the binary variable $b_{k[(kmodM)+1]}$ (for k larger than nb) can be 0 or 1. As a batch can be assigned at most to one machine, the binary variables b_{km} (for k larger than nb and m different from (k *mod* M) +1) are necessarily equal to zero. We thus introduce the constraint set (8) to define the binary variables which are equal to zero.

When M is equal to 1, constraint set (8) is replaced by:

$$b_{km} \geq b_{k+1,m} \qquad k = nb + 1, ..., N - 1 \tag{9}$$

With this constraint set, we assume that low indexed batches have a higher priority than other batches.

The number of variables in the model is $N^2M + 2NM + 1$. The number of constraints for the case where $M > 1$ is $N^2M + 3NM + N + nb(2 - M)$. If $M = 1$, the model contains $N^2 + 5N - 1$ constraints.

5.4. *Computational results for the MILP model*

In this section, we test the effectiveness of the proposed MILP model and the behavior of the lower bound algorithm according to optimal solutions. We compare our MILP model to the MILP proposed by Chung et al. (2009).

5.4.1. Test instances

The test instances are inspired from real data given by a French private hospital that we refer to as the "real case". More precisely, the machine capacity, batch processing time, job sizes and arrival times are inspired from the real case. There are 4 automatic washers in the sterilization service. Washer capacities are the same and equal to 6 DIN (DIN is a standard measurement type for the volume of automatic washers), and washing time (i.e. batch processing time) is 60 minutes. In the hospital sterilization service investigated, RMD set sizes are multiples of 1/36 of machine capacity. There are thus 36 different RMD set sizes, ranging from 1/6 DIN to 6 DIN. We observed the occurrence frequencies of different RMD set sizes for data over a 5-day period, and noticed that they were uniformly distributed. Consequently, the job sizes in our problem are sampled from a

discrete uniform distribution, U[1;36], and are estimated by U[1;36]/6. Arrival times were also observed over a 5-day data period. We noticed that inter-arrival times between two RMD sets may take any value between 0 and 40 minutes. Therefore, we sampled job arrival times from a uniform distribution such that two consecutive arriving jobs may have an inter-arrival time equal to X minutes, where X∼U [0; 40]. We denote this type of arrival as a "random RMD arrival". However, in some other sterilization services, regular collection of RMD sets may take place in operating theaters. In this case, someone is in charge of collecting RMD sets from operating theaters at fixed intervals all day long, thus, RMD sets arrive at the sterilization service regularly. We consider 2 different values for the regular inter-arrival times: 20 minutes and 40 minutes, and assume that the number of jobs released in a collecting tour is sampled from a uniform distribution which is U[0;2] for 20 minutes of regular collecting and U[1;3] for 40 minutes of regular collecting. We thus define 3 instance types, according to RMD set arrivals. Let us refer to them as 1^{st}, 2^{nd} and 3^{rd} instance types for irregular arrivals, 20 minutes of regular collecting, and 40 minutes of regular collecting, respectively. We group our experiments according to the number of jobs and number of machines. The number of machines varies from 1 to 4 machines, while job numbers are 10, 15, 20 and 25. For each job number/machine number combination, we tested 90 different instances. Thus, for each type of instance, i.e. 1^{st}, 2^{nd} and 3^{rd} types, 30 instances are tested in any job number/machine number combination. An Intel Corel 2 Duo, 3 Ghz CPU computer with 3.25 GB Ram is used for all computational experiments. CPLEX version 10.2 is used to implement the MILP models. The resolution time limit with CPLEX is set to one hour.

5.4.2. Performance of the proposed MILP model and quality of the lower bound algorithm

We compare our MILP model to the model proposed by Chung et al. (2009). However, in their problem, batch processing times are not equal. So, they have proposed a constraint set in order to calculate the batch processing times. Since all job/batch processing times are equal in our problem, we removed this constraint set from their MILP model in order to adapt it better to our problem. We refer to our MILP model as $MILP$ and to the MILP of Chung et al. (2009) as $MILP_{lit}$. In table 2, we show the average resolution times and the percentage of optimally solved instances for each of the machine number/job number combination.

It is clear that our MILP model is much more efficient than the other one in terms of resolution time. Starting from the 25 jobs/3-4 machines instances, we observe some instances for which our MILP model can not find the optimal solution in the given time limit. Thus, in order to have a better idea of the resolution limit with our MILP model, we tested some more instances containing more than 25 jobs. Numerical experiments show that some instances are not solved within one hour starting from instances containing 28 jobs for one machine case, and, 26 jobs for two machines case. For 3 and 4 machine cases, first instances which are not solved within one hour contain 23 (or more) jobs.

Remember that the last two constraint sets in our MILP are valid inequalities. If these constraints are removed, the optimal value will not change. But, the time needed to find the optimal solution may increase. For that purpose, we removed these constraints from the model and tested some instances containing 15 jobs. We saw that without these valid inequalities, computational times exceeded 1 hour for all 2, 3 and 4 machines instances. As for one machine instances, instances are optimally solved but the average resolution time increases to 300 seconds.

Concerning the quality of *Split Job* as a lower bound algorithm, we compare the

Table 2. Resolution limits for the MILP models

Nb. of jobs	Nb. of mach.	$MILP$ Av. Resol. time	$MILP$ % of opt. sol. inst.	$MILP_{lit.}$ Av. Resol. time	$MILP_{lit.}$ % of opt. sol. inst.
10	1	<1 sec	100%	≃232 sec	100%
10	2	<1 sec	100%	≃341 sec	100%
10	3	<1 sec	100%	≃473 sec	100%
10	4	<1 sec	100%	≃178 sec	100%
15	1	<1 sec	100%	> 3600 sec	0%
15	2	<1 sec	100%	> 3600 sec	0%
15	3	≃10 sec	100%	> 3600 sec	≃40
15	4	≃7 sec	100%	≃711 sec	≃81
20	1	≃1 sec	100%	> 3600 sec	0%
20	2	≃242	100%	> 3600 sec	0%
20	3	≃450	100%	> 3600 sec	0%
20	4	≃341	100%	≃1379 sec	≃66
25	1	≃4.5 sec	100%	> 3600 sec	0%
25	2	≃568 sec	100%	> 3600 sec	0%
25	3	≃1147 sec	≃89%	> 3600 sec	0%
25	4	≃933 sec	≃88%	> 3600 sec	0%

Table 3. Comparison between the lower bound and the optimal results

Nb. of jobs	Nb. of mach.	Avg. gap in terms of solution quality
10	1	≃13%
10	2	≃4.5%
10	3	≃4.5%
10	4	≃0.6%
15	1	≃19.5%
15	2	≃9.3%
15	3	≃3.17%
15	4	≃0.45%
20	1	≃14%
20	2	≃6%
20	3	≃3.5%
20	4	≃0.6%
25	1	≃14%
25	2	≃6.3%

optimal results to those found by the *Split Job* algorithm. Therefore, in table 3, we show the average difference between the solutions found by the lower bound algorithm (i.e. *Split Job* algorithm) and the optimal solutions for each job number/machine number combination. The reported gap is calculated by $(C^*_{max} - C^{LB}_{max}) * 100/C^{LB}_{max}$ where C^*_{max} is the optimal solution and C^{LB}_{max} is the lower bound solution.

We experimented until 25 jobs and 2 machines instances, because beyond this machine number/job number combination, the instances are not all optimally solved. Table 3 shows that the quality of the lower bound algorithm is quite good for 2, 3 and especially 4 machines instances. For the case of a single machine, job splitting property allows smaller number of batches which are sequenced consecutively on just one machine. Thus,

the makespan value for the lower bound is much smaller than the optimal one.

6. Conclusion

In this paper we modeled the washing step of a sterilization service as a batch scheduling problem. As the washing step is generally a bottleneck of the overall sterilization process, we aimed to minimize the total duration time of washing operations.

The batch scheduling problem we tackled has the following specifications: parallel batching machines, job release dates, job sizes, limited machine capacity and equal job processing times. First, we developed polynomial time algorithms for two special cases where job sizes are not arbitrary. Afterwards, we gave an approximation algorithm for our original problem and showed that it has a performance guarantee equal to 2. Finally, we developed an exact resolution method which is a MILP model and tested its efficiency in terms of resolution time. Remember that a general case of our problem is obtained when job processing times are different. Chung et al. (2009) developed a MILP model for that general case. However according to numerical tests in the final section, consideration of equal processing times allowed us to develop a more efficient MILP model which runs faster than the MILP given by Chung et al. (2009).

In future work, this study can be extended considering uncertainties for RMD set/job arrivals. Moreover, some other objective functions (ex: sum of job completion times) may be studied.

References

Azizoglu, M. and Webster, S. 2000. Scheduling a batch processing machine with non-identical job sizes. *International Journal of Production Research*, 38 (10), 2173–2184.

Azizoglu, M. and Webster, S. 2001. Scheduling a batch processing machine with incompatible job families. *Computers and Industrial Engineering*, 39 (3-4), 325–335.

Chang, Y., Damodaran, P. and Melouk, S., 2004. Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, 42 (19), 4211–4220.

Chen, H., Du, B. and Huang, G.Q., 2010. Metaheuristics to minimise makespan on parallel batch processing machines with dynamic job arrivals. *International Journal of Computer Integrated Manufacturing*, 23 (10), 942–956.

Chung, S.H., Tai, Y.T., and Pearn, W.L., 2009. Minimizing makespan on parallel batch processing machines with non-identical ready time and arbitrary job sizes. *International Journal of Production Research*, 47 (18),5109–5128.

Coffman Jr, E.G., Garey, M.R. and Johnson, D.S. 1987. Bin packing with divisible item sizes. *Journal of Complexity*, 3 (4), 406–428.

Coffman Jr, E.G., Garey, M.R. and Johnson, D.S. 1997. Approximation algorithms for bin packing: A survey. *in Approximation Algorithms for NP-Hard Problems* , D. Hochbaum (ed.), PWS Publishing, Boston, 46–93.

Damodaran, P. and Velez Gallego, M.C., 2010. Heuristics for makespan minimization on parallel batch processing machines with unequal job ready times. *International Journal of Advanced Manufacturing Technology*, 49 (9-12), 1119–1128.

Damodaran, P., Velez-Gallego, M.C. and Maya, J., 2011. A GRASP approach for

makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, 22 (5), 767–777.

Dupont, L. and Jolai Ghazvini, F., 1998. Minimizing makespan on a single batch processing machine with non-identical job sizes. *European Journal of Automation*, 32 (4), 431–440.

Dupont, L. and Dhaenens-Flipo, C., 2002. Minimizing the makespan on a batch processing machine with non-identical job sizes: an exact procedure. *Computers and Operations Research*, 29 (7), 807-819.

Feo, T. and Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8 (2), 67-71.

Graham, R., Lawler, E., Lenstra, J., and Kan, A.,1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey *Annals of Discrete Mathematics*, 5, 287–326.

Jolai Ghazvini, F. and Dupont, L. 1998. Minimizing mean flow times criteria on a single batch processing machine with non-identical job sizes. *International Journal of Production Economics*, 55 (3), 273–280.

Ikura, Y. and Gimple, M. 1986. Efficient scheduling algorithms for a single batch processing machine. *EOperations Research Letter*, 5, 61–65.

Kashan, A.H., Karimi, B., and Jolai, F., 2006. Minimizing Makespan on a Single Batch Processing Machine with Non-identical Job Sizes: A Hybrid Genetic Approach. *EvoCOP'2006*, 135–146.

Kashan, A.H., Karimi, B., and Jenabi, M., 2008. A hybrid genetic heuristic for scheduling parallel batch processing machines with arbitrary job sizes. *Computers and Operations Research*, 35 (4), 1084–1098.

Kashan, A.H., Karimi, B., and Fatemi Ghomi, S.T.M., 2009. A note on minimizing makespan on a single batch processing machine with nonidentical job sizes. *Theoretical Computer Science*, 410 (27-29), 2754–2758.

Kashan, A.H., Karimi, B., and Jolai, F., 2010. An effective hybrid multi-objective genetic algorithm for bi-criteria scheduling on a single batch processing machine with nonidentical job sizes. *Engineering Applications of Artificial Intelligence*, 23 (6), 911–922.

Kempf, K.G., Uzsoy, R. and Wang, C., 1998. Scheduling a single batch processing machine with secondary resource constraints. *Journal of Manufacturing Systems*, 17 (1), 37–51.

Lee, C.Y., and Uzsoy, R., 1999. Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research*, 37 (1), 219–236.

Li, S., Li, G., Wang, X. and Liu, Q., 2005. Minimizing makespan on a single batching machine with release times and non-identical job sizes. *Operations Research Letters*, 33 (2), 157-164.

Lu, S., Feng, H. and Li, X., 2010. Minimizing the makespan on a single parallel batching machine. *Theoretical Computer Science*, 411 (7-9), 1140–1145.

Martello, S. and Toth, P., 1990. Knapsack problems: algorithms and computer implementations. *John Wiley and Sons*, ChichesterNew York.

Melouk, S., Damodaran, P. and Chang, P., 2004. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *International Journal of Production Economics*, 87 (2), 141–147.

Neale, J.J. and Duenyas, I., 2003. Control of a Batch Processing Machine Serving Compatible Job Families. *IIE Transactions*, 38 (8), 699–710.

Ngo cong, K., 2009. Etude et amelioration de l'organisation de la production de dispositifs medicaux steriles. Thesis (PhD). Universite Joseph Fourrier, FR.

Nong, Q.Q., Ng, C.T. and Cheng T.C.E., 2008. The bounded single-machine parallel-batching scheduling problem with family jobs and release dates to minimize makespan. *Operations Research Letters*, 36 (1), 61–66.

Parsa, N.R., Karimi, B., Lee, and Kashan, A.H., 2010. A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers and Operations Research*, 37 (10), 1720–1730.

Potts, C.N., and Kovalyov, M.Y., 2000. Scheduling with batching: A review. *European Journal of Operational Research*, 120 (2), 228–249.

Standard AFNOR FD S98-135. Sterilisation des dispositifs medicaux. *Guide pour la maitrise des traitements appliques aux dispositifs medicaux reutilisables*

Uzsoy, R., 1994. Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32 (7), 1615–1635.

Wang, H. and Chou, F-D. 2010. Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics. *Expert Systems with Applications: An International Journal*, 37 (2), 1510-1521.

Zhang, G., Cai, X., Lee, C.Y. and Wong, C.K., 2001. Minimizing makespan on a single batch processing machine with nonidentical job sizes. *Naval Research Logistics*, 48 (3), 226–240.

Zhang, X., Zhang, Y., Cao, Z. and Cai, Z., 2007. Approximation Schemes for Scheduling a Batching Machine with Nonidentical Job Size. *Journal of Systems Science and Complexity*, 20 (4), 592–600.